IBM

Oracle to DB2 Conversion Guide
for Linux, UNIX, and Windows

Step-by-step guide to migrating from Oracle to DB2 9.1

Conversion examples - including XML conversion

Step-by-step guide to MTK tool usage

Whei-Jen Chen
An Na Choi
Marina Greenstein
Scott J Martin
Fraser McArthur
Carlos Eduardo Abramo Pinto
Arthur V Sammartino
Nora Sokolof

Redbooks

IBM

International Technical Support Organization

Oracle to DB2 Conversion Guide for Linux, UNIX, and Windows

August 2007

Second Edition (August 2007)

This edition applies to DB2 Version 9 for Linux, UNIX, and Windows, Oracle 9i, Oracle 10g.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™	IBM®	Rational®
AIX®	IMS™	Redbooks®
AS/400®	iSeries®	Redbooks (logo) ®
Cloudscape™	i5/OS®	REXX™
DB2 Connect™	Lotus®	Tivoli®
DB2 Universal Database™	OS/390®	WebSphere®
DB2®	PowerPC®	z/OS®
developerWorks®	POWER™	zSeries®
DRDA®	pureXML™	1-2-3®
Informix®	QMF™	

The following terms are trademarks of other companies:

SAP R/3, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Snapshot, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

Enterprise JavaBeans, Java, JavaBeans, JDBC, JRE, J2EE, J2SE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Microsoft, SQL Server, Visual C++, Visual Studio, Windows NT, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM® DB2 has long been known for its technology leadership. This IBM Redbooks® publication, intended for technical staff who are involved in an Oracle® to DB2 conversion project, is an informative guide that describes how to migrate the database system from Oracle to DB2 Version 9 on Linux®, UNIX®, and Microsoft® Windows® platforms.

This book provides conversion methodology and step-by-step instructions for installing and using IBM Migration Toolkit (MTK) to port the database objects and data from Oracle to DB2. It illustrates, with examples, how to convert the stored procedures, functions, and triggers. Application programming and conversion considerations are discussed, along with the differences in features and functionality of the two products.

In addition, you can find script conversion samples for data loading, database administration, and reports that are useful for DBAs. The testing section provides procedures and tips for conversion testing and database tuning. The laboratory examples are performed under Oracle 10g and DB2 Version 9. However, the migration process and examples can be applied to Oracle 7, 8, and 9i.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Whei-Jen Chen is a Project Leader at the International Technical Support Organization, San Jose Center. She has extensive experience in application development, database design and modeling, and DB2 system administration. Whei-Jen is an IBM Certified Solutions Expert in Database Administration and Application Development, as well as an IBM Certified IT Specialist.

An Na Choi is a Field Technical Sales Specialist with the Information Management department of SWG in South Korea. After graduating from university, she worked for a local IT company for two years developing an encyclopedia application. An Na has supported DB2 and Informix® products including DB2 Database server, Tools, WebSphere® Information Integrator and Informix Dynamic Server, and DB2 for SAP® for eight years. As a presales engineer, An Na has performed benchmark tests and participated in Oracle to DB2 conversion projects. She has extensive experience in deploying IBM

Information Management solutions to many customers with DB2, Informix Dynamic Server, DB2 on SAP, WSII and many other tools.

Marina Greenstein is a Senior Certified Consulting IT Software Specialist with the IBM Database Migration Team. She is an IBM Certified Solutions Expert who joined IBM in 1995 with experience in database application architecture and development. During the 11 years Marina has been with the DB2 Migration Team, she has assisted customers in their migrations from Microsoft SQL Server™, Sybase, or Oracle databases to DB2. She has presented migration methodology at numerous DB2 technical conferences and at SHARE. She is also the author of multiple articles and a white paper about DB2 migration.

Scott J Martin is a DB2 Technology Consultant with the IBM Innovation Center (IIC) for Business Partners in Waltham, MA, which is part of IBM Developer Relations. He has 25 years of IT experience with IBM, and is a certified Advanced Database Administrator for DB2 LUW V8.1, a certified Application Developer for the UDB Family, and a certified Database Administrator for DB2 for z/OS®. For the past four years with the IIC, Scott has helped business partners convert their applications from Oracle, SQL Server, and MySQL to DB2 for LUW and DB2 for z/OS. In this role, Scott has also managed DB2 performance enablements and performed a variety of architecture and database administration tasks supporting business partners. Prior to this, he spent 13 years with IBM Global Services performing Application Development, Database Administration, and Database Tuning for DB2 mainframe environments.

Fraser McArthur is a DB2 Technical Consultant with the Information Management Partner Enablement organization at the IBM Toronto Lab, where he has worked for the last seven years. He focuses on assisting IBM Data Services Business Partners, performing application migrations and performance tuning, which can involve anything from low-level and detailed application development and troubleshooting to high-level database design and administration. Recently, his focus has been on DB2 pureXML™. Fraser also conducts DB2 technical workshops and publishes the occasional article to IBM developerWorks® for the DB2 community.

Carlos Eduardo Abramo Pinto is an IT Specialist for Database Administration in IBM Global Services in Brazil, supporting IBM local and international customers. He has more than 11 years of IT experience in a wide range of client and server platforms, including technical and system support of the Windows operating system, Oracle, DB2, and Microsoft SQL Server databases in UNIX, Linux and Windows platforms. He is also specialized in Oracle Real Application Clusters implementation and support. Carlos is an IBM DB2 Certified Database Administrator on DB2 UDB V8.1 for Linux, UNIX and Windows, an Oracle Certified Professional on Oracle 8i, 9i and 10g databases, a Microsoft Certified IT Professional Database Administrator and Database Developer for Microsoft SQL

Server 2005, a Microsoft Certified Database Administrator for Microsoft SQL Server 7 and 2000, and a Microsoft Certified System Engineer on Windows NT® 4 and 2000.

Arthur V Sammartino is a Certified Consulting I/T Specialist with the IBM Database Migration Team (also known as SMPO). He is responsible for assisting customers who are considering a database migration from competitive RDBMS products (Oracle, Sybase, or Microsoft SQL Server) to IBM DB2 residing on Linux, UNIX, Windows, or the z/OS platforms. In addition to his conversion responsibilities, his experience includes supporting clients with application development environment and setup concerns, as well as migration tool installation and education. Art is certified as both an IBM Database Administrator and an IBM Application Developer. He has been a contributing author for the IBM Redbooks *DB2 UDB v7.1 Porting Guide*, *Oracle to DB2 UDB Conversion Guide*, and *DB2 9 pureXML Guide*. He also assisted with *Microsoft SQL Server to IBM DB2 UDB Conversion Guide*, and was a co-author of the article "Using DB2 routines to ease migration."

Nora Sokolof is a Certified Consulting Brand Sales IT Specialist with the IBM DB2 Migration Team (also known as the SMPO). She holds a Master of Science degree in Software Engineering from Pace University. She has been with IBM for more than 20 years, and has held positions as a DB2, Informix, Oracle and PeopleSoft® development DBA. She also designed the first IBM flexible benefits enrollment database in 1993. Nora is an IBM Certified Database Administrator. She is also the author of several white papers including "Transitioning from IBM Informix to DB2 - Database Comparisons and Migration Topics", and has co-authored the following IBM Redbooks publications: *Planning for a Migration of PeopleSoft 7.5 from Oracle/UNIX to DB2 for OS/390*, *Database Transition: Informix Dynamic Server to DB2 Universal Database*, and *Database Strategies: Using XPS and DB2 Universal Database*.
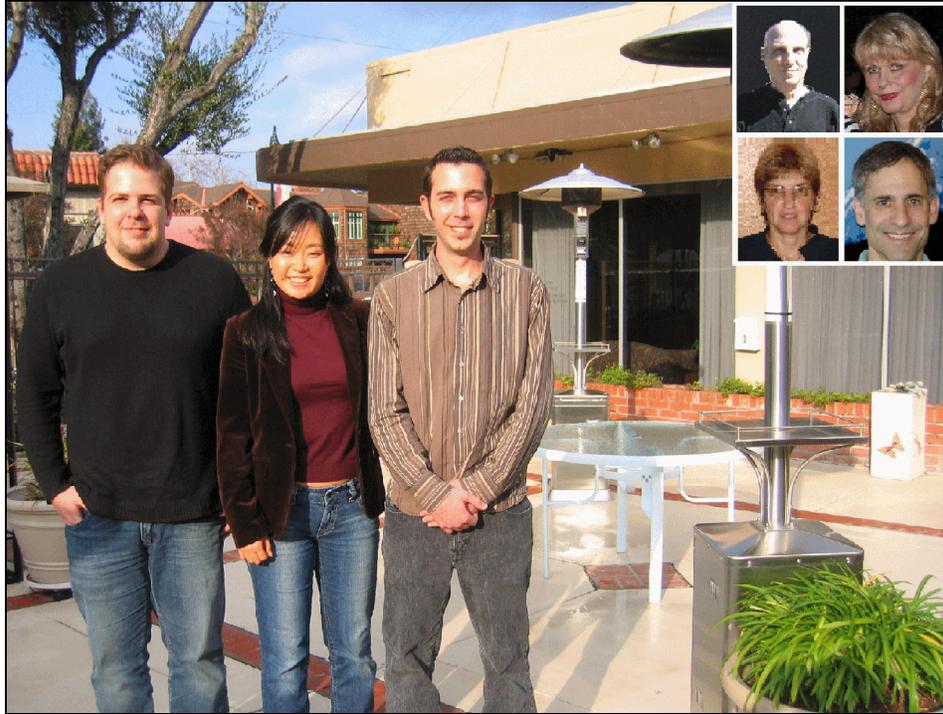
Figure 1   Left to right: Carlos, An Na, and Fraser. Upper right corner, left to right, row 1: Art and Nora, row 2: Marina and Scott

Acknowledgments

Thanks to the following people for their contributions to this project:

Deb Jenson
IBM Software Group, Information Management, Competitive Technology

Michael Gao
Sam Poon
IBM Software Group, Information Management Enablement Support

Matthias Nicola
Miso Cilimdzic
George Lapis
Ted Wasserman
Priti Desai
IBM Silicon Valley Laboratory

Barry Faust
IBM Software Migration Office

Takashi Tokunaga
Information Management Technical Sales and Services, IBM Japan

Fadel Fiani
Stefan Hummel
Ranjit K. Kalidasan
Ken Leonard
Artur Wronski
Authors of *Oracle to DB2® UDB Conversion Guide*, SG24-7048

Emma Jacobs, Sangam Racherla
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

► Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

► Send your comments in an E-mail to:

redbooks@us.ibm.com

► Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7048-01
for Oracle to DB2 Conversion Guide for Linux, UNIX, and Windows
as created or updated on August 8, 2007.

August 2007, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information.

New information
- ► Migration methodology
- ► Conversion planning technical consideration
- ► XML conversion
- ► Database administration and management

Changed information
- ► IBM Migration Toolkit
- ► DB2 product family
- ► Migration examples

1

Introduction

Since migrating from Oracle to DB2 for Linux, UNIX, and Windows requires a certain level of knowledge in both environments, the purpose of this chapter is to introduce the architectural overview of both Oracle and DB2. This is meant to facilitate the understanding of both architectures, taking into consideration that the reader may be an Oracle or a DB2 DBA.

This chapter includes the following topics:

- ► DB2 9 family of products
- ► Terminology
- ► Architectural overview including:
  - – Processes architecture
  - – Memory architecture
  - – Directory structure
  - – Oracle data dictionary and DB2 catalog
  - – Database connectivity
  - – Replicating
  - – Partitioned database architecture

1

1.1 DB2 family of products

In the era of *Information On Demand*, IBM Information Management software offers a wide range of products to accommodate different business needs and technical requirements in order to provide customers with a robust and scalable enterprise wide solution.

Beginning with DB2 9, the DB2 Universal Database™ for Linux, UNIX, and Windows product name has been simplified by removing "Universal Database" and "UDB". This change has been implemented on user interfaces, in documentation, and in packaging materials. Previous versions of DB2 database products and documentation retain "Universal Database" and "UDB" in the product naming. Also starting in DB2 9, the term *data server* is introduced to describe the product. A data server provides software services for the secure and efficient management of structured information. DB2 9 is a hybrid relational and XML data server.

DB2 offers database solutions that run on all platforms including Microsoft Windows, AIX®, Solaris™, HP-UX, Linux, AS/400®, OS/390® and z/OS. Furthermore, DB2 technologies support both 32-bit and 64-bit environments, providing support for 32-bit operating systems on Linux on x86 and Windows, and 64-bit operating systems on UNIX, Linux and Windows. The DB2 product family comprises a variety of packages that provide customers with choices based on business need. The following lists the DB2 product offerings for Linux, UNIX, and Windows:

► **DB2 Enterprise 9**
  DB2 Enterprise 9 is the ideal data server for the most demanding workload. It easily scales to handle high-volume transaction processing, multi-terabyte data warehouses, and mission critical applications from vendors such as SAP. It is also designed to provide 24x7x365 availability, including: High Availability Disaster Recovery (HADR), Tivoli® System Automation, Table Partitioning, Multidimensional Data Clustering (MDC), Materialized Query Tables (MQTs), Full intra-query parallelism, and the Connection Concentrator. Platforms supported are Linux, UNIX, and Windows.

► **DB2 Workgroup 9**
  DB2 Workgroup 9 is the ideal data server for deployment in a departmental, workgroup, or medium-sized business environment, suitable for transaction processing or complex query workloads on servers with up to four processors and 16 GB of memory. Platforms supported are Linux, UNIX, and Windows.

► **DB2 Express 9**
  DB2 Express 9 is the ideal entry level data server, which provides very attractive entry-level pricing. Suitable for transaction processing or complex query workloads on servers with up to two processors, it can address up to

four GB of memory. Platforms supported are Linux, Solaris x86, and Windows.

DB2 Express-C

DB2 Express-C is a version of DB2 Express 9 for the *community*. It is a no-charge data server for use in development and deployment of applications including: XML, C/C++, Java™, .NET, PHP, and more. DB2 Express-C also includes pureXML for free, while it is a purchasable feature for all other DB2 products. DB2 Express-C can be run on up to two dual-core CPU servers, with up to 4 GB of memory, any storage system setup, and with no restrictions on database size or any other artificial restrictions. Platforms supported are Linux and Windows. DB2 Express-C can be seamlessly upgraded to any of the other DB2 9 products, without modifying your database or your application.

► **DB2 9 value-added features**

While DB2 9 includes capabilities that serve the needs of most deployments, additional capabilities are required for certain application types, workloads, or environments that are not needed by every deployment. Rather than build a one-size-fits-all offering, IBM makes these capabilities available as optional features to give you the flexibility to purchase only what you need. The following are the available value-added features, along with the required DB2 product in parentheses:

– **pureXML (Enterprise, Workgroup, Express)**

The DB2 pureXML feature seamlessly integrates XML and relational data and unlocks the latent potential of XML by providing simple efficient access to XML with the same levels of security, integrity, and resiliency taken for granted with relational data. DB2 9 stores XML data in a hierarchical structure that naturally reflects the structure of XML. This structure, along with innovative indexing techniques, allows DB2 to efficiently manage this data and eliminate the complex and time-consuming parsing typically required for XML.

– **Storage Optimization (Enterprise)**

The DB2 Storage Optimization feature gives you the ability to compress data on disk in order to decrease disk space and storage infrastructure requirements and can save up to 80% in storage space. Since disk storage systems can often be the most expensive components of a database solution, even a small reduction in the storage subsystem can result in substantial cost savings for the entire database solution.

– **Advanced Access Control (Enterprise)**

The Advanced Access Control feature increases the control you have over who can access your data using label-based security. Label Based Access Control (LBAC) lets you decide exactly who has write access and who has read access to individual rows and individual columns. LBAC controls access to table objects by attaching security labels to them. Users

attempting to access an object must have its security label granted to them. When there's a match, access is permitted; without a match, access is denied.

– **Performance Optimization (Enterprise, Workgroup, Express)**
The DB2 Performance Optimization feature includes two critical components (DB2 Performance Expert and DB2 Query Patroller) that can significantly improve the overall responsiveness of your data server and database applications. DB2 Performance Expert and DB2 Query Patroller are complementary tools to improve data server performance, response times, and throughput. While DB2 Query Patroller enables you to focus on queries (with the ability to hold, schedule, cancel, fix, and prioritize queries), DB2 Performance Expert allows you to focus on overall DB2 system, operating system, and application performance, which can be monitored and analyzed over time.

– **Database Partitioning (Enterprise)**
The DB2 Database Partitioning feature (DPF) can be used to manage a large database better by dividing it into multiple partitions that are physically placed on one or more servers offering a great deal of flexibility in scalability. This requires no changes at all from an application's or user's perspective—everything still looks and acts like a regular database. Most often, DPF has been used by customers with very large databases who have partitioned the database across a cluster of multiple inexpensive servers instead of undertaking the overhead of a large higher-cost server.

– **Geodetic Data Management (Enterprise)**
The Geodetic Data Management feature provides the ability to store, access, manage, or analyze location-based round earth information for weather, defense, intelligence, or natural resource applications for commercial or government use. It provides the ability to manage and analyze spatial information with accuracies in distance and area by treating the earth as a continuous spherical coordinate system.

– **Real Time Insight (Enterprise)**
The DB2 Real-Time Insight feature is powered by the DB2 Data Stream Engine, which enables organizations to store and forward high volumes of data from multiple data streams. The data messages from the feed can be aggregated, filtered, and enriched in real time before being stored or forwarded. DB2 Data Stream Engine can load high volumes of data into the DB2 data server and make that data available to queries in real-time through SQL.

– **Homogenous Federation (Enterprise, Workgroup, Express)**
The DB2 Homogeneous Federation feature delivers the ability to easily manage and access remote DB2 (mainframe and distributed) and Informix data servers as local tables. Homogeneous federation meets the needs of

customers that require unified access to data managed by multiple data servers.

– **High Availability (included in Enterprise, available in Workgroup, Express)**
The DB2 High Availability feature provides 24 x 7 availability for your DB2 data server through replicated failover support and data recovery modules. The three packages that comprise this feature bring a unique aspect of high availability to the data server environment. This feature consists of the High Availability Disaster Recovery (HADR), the Online Reorganization feature, and IBM Tivoli System Automation for Multiplatforms (TSA MP).

– **Workload Management (included in Enterprise, available in Workgroup, Express)**
The DB2 Workload Management feature leverages the Connection Concentrator in conjunction with either Query Patroller (QP) or the DB2 Governor to provide a more proactive, fail-safe workload environment for your customers. Connection Concentrator allows for fail-safe operation and load balancing of a workload, and also allows reallocation of work with every new transaction. QP is a powerful query workload management offering that proactively and dynamically controls submission and execution of queries to better manage DB2 data server workloads to meet business needs. The DB2 Governor monitors and changes the behavior of applications that run against the DB2 data server.

► **Complimentary DB2 9 software**
For application development on DB2 9, the following software is available for download free of charge:

– **DB2 Developer Workbench**
The DB2 Developer Workbench is an Eclipse-based tool that replaces the Development Center in DB2 V8. Developer Workbench is a comprehensive development environment for creating, editing, debugging, deploying, and testing DB2 stored procedures and user-defined functions. You can also use Developer Workbench to develop SQLJ applications, and to create, edit, and run SQL statements and XML queries.

– **DB2 Information Center**
The DB2 Information Center is where you can find information that you need to use the DB2 family of products and features. The DB2 Information is available as a local installation, or viewable online. The online version always contains the most up-to-date information.

– **DB2 Runtime Client**
The DB2 Runtime Client provides a means for applications to connect to remote DB2 databases. It provides support for common database access interfaces, such as: JDBC™, ADO.NET, OLE DB, ODBC, and DB2

Command Line Interface (CLI), and includes the drivers and capabilities to define data sources. The DB2 Runtime client provides base client support to handle database connections, SQL statements, XQuery statements, and DB2 commands. It can be freely distributed with your application.

– **DB2 Client**
The DB2 Client includes all the functionality of the DB2 Runtime Client plus functionality for client-server configuration, database administration, and application development.

– **DB2 Driver for ODBC and CLI**
The DB2 Driver for ODBC and CLI provides runtime support for the DB2 CLI application programming interface (API) and the ODBC API. Though the DB2 Client and DB2 Runtime Client both support the DB2 CLI and ODBC APIs, this driver is not a part of either DB2 client. It is available separately, installed separately, and supports a subset of the functionality of the DB2 clients. The driver has a much smaller footprint than the DB2 Client and the DB2 Runtime Client and you can have multiple installations of the driver on a single machine. You can include the driver in your database application installation package, and redistribute the driver with your applications.

– **DB2 Driver for JDBC and SQLJ**
The DB2 Driver for JDBC and SQLJ is a single application driver to support the most demanding Java applications. This agile driver can be used in type 4 or type 2 mode (with the appropriate client environment). The driver is JDBC 3.0 compliant and supports XML and XQuery. You can include the driver in your database application installation package, and redistribute the driver with your applications.

– **DB2 Net Search Extender (NSE)**
The DB2 Net Search Extender is now available at no extra charge. To be compatible with the new DB2 XML functions, DB2 NSE fully supports the XML data type, and all text search functions can be used on XML documents that are stored natively in the database. DB2 NSE continues to support text search on XML documents stored in BLOB and CLOB data types.

– **DB2 Spatial Extender**
The DB2 Spatial Extender allows you to store, manage, and analyze spatial data (information about the location of geographic features) in DB2 along with traditional data for text and numbers. With this capability, you can generate, analyze, and exploit spatial information about geographic features, such as the locations of office buildings or the size of a flood zone.

– **DB2 Runtime Client merge modules (Windows)**
Using the DB2 Runtime Client merge modules, you can easily add DB2

Runtime Client functionality to any product that uses the Windows Installer.

– **DB2 9 National Language Pack**
National language (NL) users need to download this National Language Pack in order to receive the NL version of DB2 9 (applies to Data Server, DB2 Client, DB2 Runtime Client, and the Spatial Extender). If you are installing a non-English version of DB2 9, you must use one or more additional CDs called a National Language Package. This package contains national language support files (product files that are specific to a language).

► The following DB2 Application Development suite is available for purchase:

DB2 Edition for Application Development Deployment
This edition offers a package for a single application developer to design, build, and prototype applications for deployment on any of the IBM Information Management client or server platforms. This comprehensive developer offering includes DB2 Workgroup 9 and DB2 Enterprise 9, IDS Enterprise Edition V10.0, Cloudscape™ V10.1, DB2 Connect™ Unlimited Edition for zSeries®, and all the DB2 V9.1 features, allowing customers to build solutions that utilize the latest data server technologies. The software in this package cannot be used for production systems. You must acquire a separate user license for each authorized user of this product.

IBM also provides federated technologies to extend the functionality of DB2. With WebSphere Federation Server, you can access objects in many different databases, such as Oracle, with a single query.

1.1.1 DB2 9 Autonomic computing features

Autonomics were first introduced in DB2 Version 8.2. to help provide a computing environment that has the ability to sense and dynamically respond to situations that occur in accordance with business policies and objectives. The goal is to shift the burden and reduce the complexity of managing computing environments from people to technology, to help reduce the total cost of ownership (TCO), and increase business productivity. Simply put, an autonomic computing environment is self-configuring, self-healing, self-optimizing, and self-protecting. This initiative is not to replace the DBA, but instead make their job easier as data being stored, analyzed, and metricized increases.

DB2 Version 8.2 introduced the following autonomic computing features:

► Design Advisor has been enhanced to recommend indexes, materialized query tables (MQTs), multidimensional clustering tables (MDCs), and partitions.

- ► Configure Automatic Maintenance wizard for automating maintenance activities such as BACKUP, REORG, and RUNSTATS.

- ► Health Center Recommendation advisor, which monitors the state of the database environment and can send out notifications if alarm or warning thresholds are exceeded and can also perform scripted actions.

- ► Automated log file management.

- ► Automatic RUNSTATS profiling, which allows DB2 to monitory queries over time and choose the best statistics collection options for optimal query performance.

- ► BACKUP and RUNSTATS throttling to decrease impact on database users.

- ► BACKUP and RESTORE self-tuning.

- ► The RECOVER DATABASE command to simplify database recovery for the DBA.

- ► The DFT_PREFETCH_SZ configuration parameter now supports an automatic setting for prefetch size, which allows DB2 to choose an optimal setting.

Building on the autonomic features and enhancements introduced in DB2 Version 8.2, some of the key DB2 9 features include:

- ► **Adaptive, self-tuning memory allocation**
  This feature makes the task of DB2 server configuration easier, by continuously and intelligently updating configuration parameters and buffer pool sizes.

- ► **Automatic storage support**
  This feature, which is enabled by default, allows for the size of your database to automatically grow across disk and file systems and eliminates the need to manage storage containers. Since automatic storage uses Database Managed Storage (DMS) for regular and large data, there is improved performance over that of System Managed Storage (SMS). This feature also supports DPF database storage.

- ► **Automated statistics collection**
  This feature, which is enabled by default, enables the automatic collection of statistics using runstats. The process runs as a background process whenever DB2 identifies it as being needed for query optimization.

- ► **Automatic configuration of prefetchers and page cleaners**
  This feature, which is enabled by default, allows the number of prefetchers and page cleaners to be automatically determined by DB2 based on the server environment characteristics.

► **Automatic table and index reorganization enhancements**
This feature provides new policy options for automated table and index reorganization.

1.2 Terminology

Before getting into the conversion process, a clear understanding of the terminologies used in Oracle and DB2 helps you map each terminology between Oracle and DB2. This section discusses some of these terminologies, and a mapping between them is given.

1.2.1 Terminology mapping

Table 1-1 provides a quick reference of commonly used terminologies in Oracle and DB2. More information about the terminology mapping is provided in Appendix B, "Terminology mapping" on page 607.

Table 1-1   Mapping of Oracle terminology to DB2

Oracle	DB2
Instance	Instance
Database	Database
Initialization File	Database Manager Configuration File
Table spaces	Table spaces
Data blocks	Pages
Extents	Extents
Datafiles	DMS containers
Redo Log Files	Transaction Log Files
PL/SQL	SQL/PL
Data Buffers	Buffer Pool
SGA	Database Manager and Database shared memory
Data Dictionary	Catalog
Library Cache	Package Cache
Large Pool	Utility heap
Data Dictionary cache	Catalog cache

Oracle	DB2
SYSTEM table space	SYSCATSPACE table space

1.3 Architecture overview

It is useful to understand the differences between Oracle's architecture and that of DB2 before attempting the Oracle to DB2 migration process. Both products include their own memory architecture, background processes, database related files, and different configuration files. Both Oracle and DB2 consist of an instance and the database(s) attached to that instance. This section provides a general description of the architectures of each vendor.

Figure 1-1 is an overview of the Oracle architecture. The upper level shows the memory architecture, the middle level is the process component, and the bottom level shows the database component.
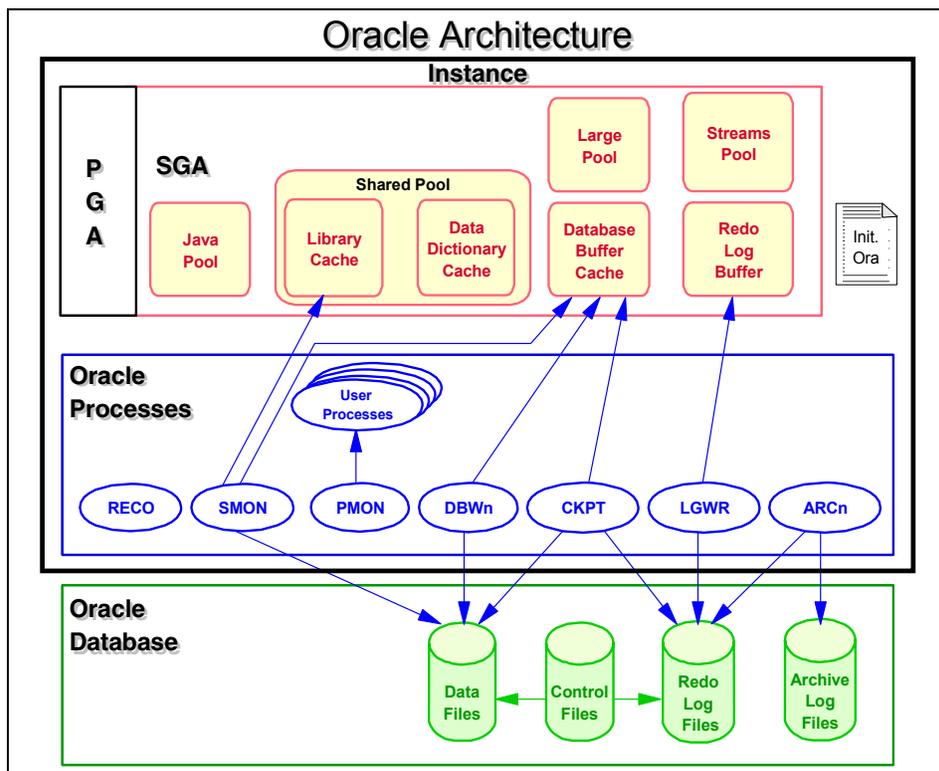
Figure 1-1   Oracle architecture overview

Figure 1-2 shows the DB2 architecture overview. DB2 implements a dedicated process architecture. From a client-server view, the client code and the server code are separated into different address spaces. The application code runs in the client process, while the server code runs in separate processes. The client process can run on the same machine as the database server or a different one, accessing the database server through a programming interface. The memory units are allocated for database managers, database, and application.

The following section discusses both architectures, detailing memory components and background processes of both databases.
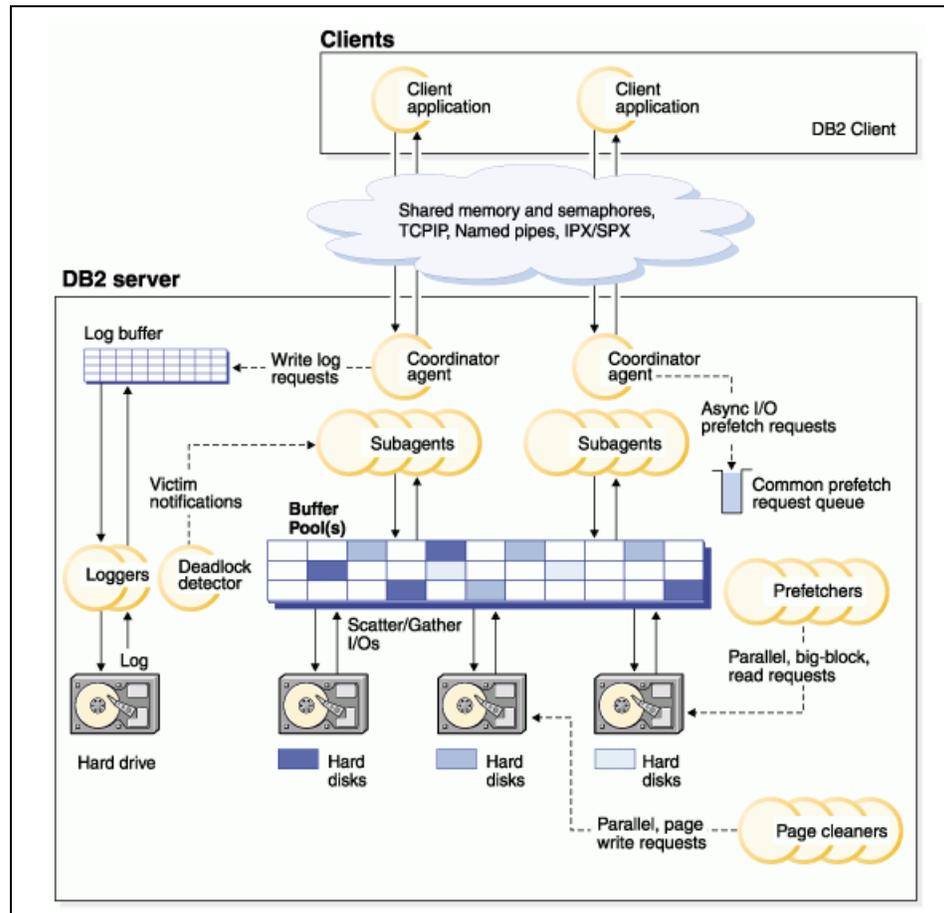
Figure 1-2   DB2 architecture overview

1.3.1 Memory architecture

This section discusses the memory architecture in Oracle and DB2. Oracle and DB2 allocate and use memory for instance and database operation. There are various memory structures used for different processes. This section gives a broader overview about how memory is allocated and used in a simple Oracle and DB2 server.

The memory architecture of an Oracle database consists of the memory area allocated to the Oracle instance and database upon startup. The amount of memory allocated is controlled by parameters defined in the Oracle configuration file.

The memory architecture of DB2 is slightly different from Oracle's. Unlike Oracle, the DB2 server can run multiple databases under one instance and hence has configuration files at both the instance level (Database Manager configuration file) and at the database level (Database configuration file).

Oracle

Oracle uses memory to run the code and share data among users. The two basic components of the Oracle memory structure are the Program Global Area (PGA) and the System Global Area (SGA). Figure 1-3 shows the primary memory architecture of an Oracle server.
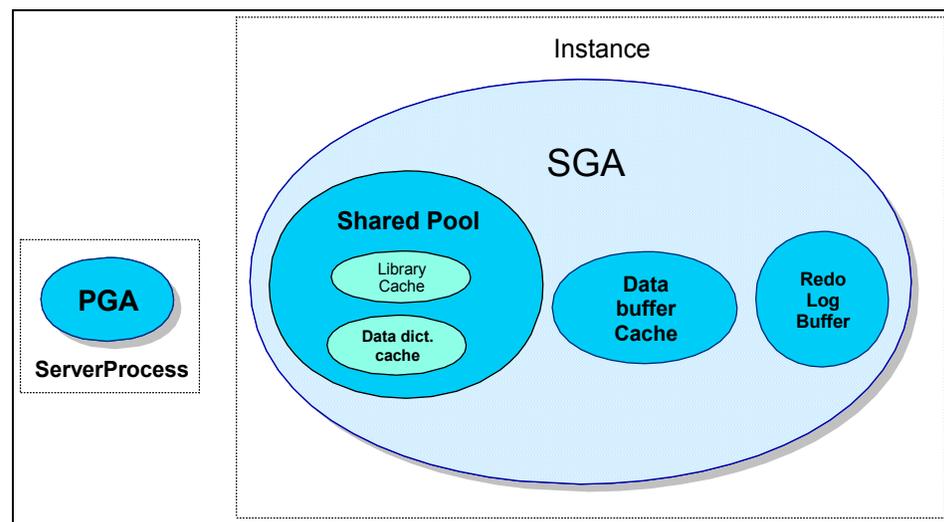
Figure 1-3   Oracle memory architecture

The PGA is associated with the server process and contains the data and control information. For the dedicated server configuration, the primary contents of the

PGA are the sort area, session information, cursor state and stack space. This is non-sharable memory, which is writable only by the server process. The PGA is allocated whenever a server process starts. The total size of the PGA is controlled by the PGA_AGGREGATE_TARGET initialization parameter in version 10g.

The SGA is the shared memory region allocated for the entire Oracle Instance. The SGA is a group of shared memory structures in which the basic components are the shared pool, data buffer cache, and the redo log buffer. The shared pool contains the library cache, data dictionary cache, along with buffers for parallel execution messages, and control structures. The library cache holds the SQL statement text, the parsed SQL statement, and the execution plan. The data dictionary cache contains reference information about tables, views, object definitions, and object privileges.

The shared pool size is controlled by the SHARED_POOL_SIZE initialization parameter. The data buffer cache stores the most recently used Oracle data blocks. Oracle reads the data blocks from the datafiles and places them in data buffers before processing the data. The number of buffers allocated is controlled by DB_CACHE_SIZE. The redo log buffer is a circular buffer that contains redo entries of the change information made to the database. These redo log buffers are written into the redo log files and are required for the database recovery. The sizes of the redo log buffers are controlled by the LOG_BUFFER initialization parameter. The other memory structures of the SGA include the large pool, used for backup purposes; the Java pool, used for Java objects; and the streams pool, used for streams memory. For a shared server configuration in version 10g the session information and the sort areas are in SGA instead of PGA.

DB2

The three primary memory structures in DB2 are the Instance Shared Memory (also known as Database Manager Shared Memory), the Database Shared Memory (also known as Database Global memory), and the Application Shared Memory (also known as Application Global Memory). Figure 1-4 shows the basic memory architecture of a DB2 data server.
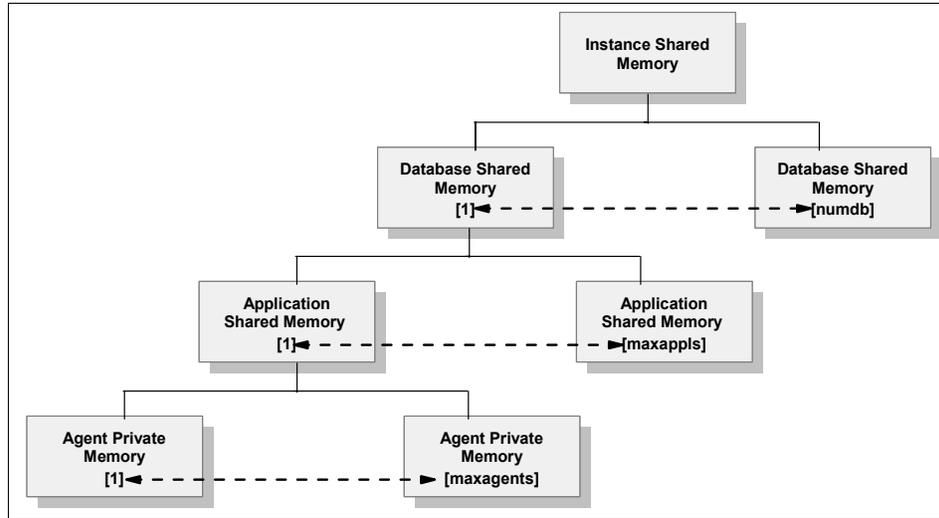
Figure 1-4   DB2 memory architecture

Instance Shared Memory is allocated when the instance is started. All other memory is attached or allocated from the Instance Shared Memory, which is controlled by the INSTANCE_MEMORY database manager (DBM) configuration parameter. By default, this parameter is set to *automatic,* which enables the DB2 server to allocate the necessary memory for the instance.

Database Shared Memory is allocated when the database is first activated or connected to for the first time. This memory is shared by all the applications that might connect to the database as well as the database Engine Dispatchable Units (EDUs) or database process that runs within each database. The memory allocated for the database process includes:

► Buffer pools - equivalent to Data Buffers in Oracle
► Lock list
► Database heap (includes log buffer)
► Utility heap - equivalent to Large Pool in Oracle
► Package cache - equivalent to library cache in Oracle
► Catalog cache - equivalent to data dictionary cache in Oracle

The buffer pools can be compared to the database buffers in Oracle, and the package cache and catalog cache can be compared to library cache and data dictionary cache in Oracle, respectively. Database Shared Memory is controlled by DATABASE_MEMORY database (DB) configuration parameter. By default, this parameter is set to *automatic,* which enables DB2 to calculate and allocate the memory for the database. Table 1-2 shows the DB2 database memory segments and the associated parameters.

Table 1-2   Database memory segments and parameters

Database memory	Parameter
Buffer pools	BUFFPAGE
Lock list	LOCKLIST
Database heap (includes log buffer)	DBHEAP
Utility heap size	UTIL_HEAP_SZ
Package cache	PCKCACHESZ
Catalog cache - equivalent to data	CATALOGCACHE_SZ

Application Shared Memory is allocated when an application connects to a database. This happens only in partitioned database environments, or in a non-partitioned database environment where intra-partition is enabled, or if the connection concentrator is enabled. This memory is used by the connecting agents to execute the work requested by the clients. The database manager configuration parameter MAX_CONNECTIONS limits the maximum number of applications that connect to the database, which in turn sets the upper limit for the maximum Application Shared Memory allocated.

Note: For more information about DB2 Memory Management, refer to Chapter 11 "Configuring DB2 instances and databases: Configuring DB2 memory allocation" in the Performance Guide, SC10-4222.

1.3.2 Process architecture

Any database instance is nothing but a collection of processes and memory structures. This section discusses the processes in Oracle and DB2.

Oracle

There are two major types of Oracle processes: the user processes and the background processes (see Figure 1-5).
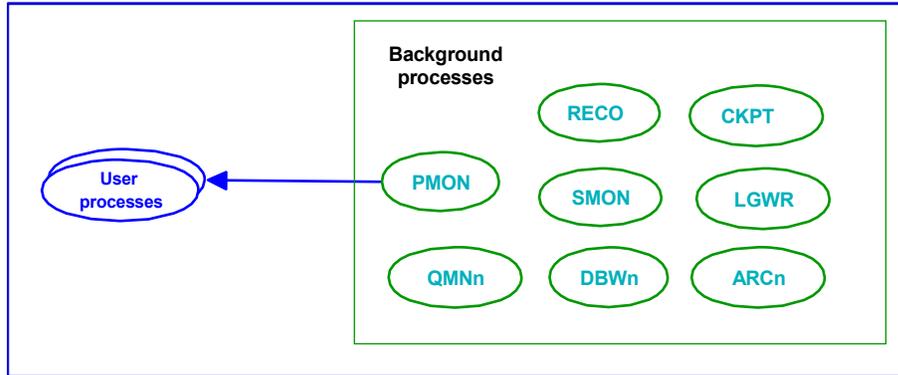
Figure 1-5   Oracle process architecture

User processes

Oracle creates a user process when the user or application connects to the database. For each user process, a server process is created by Oracle to handle the user process request to an Oracle instance. This architecture works when the client is on a different machine. When the client and the server are on the same machine, the user process and server process are combined into a single process. The function of the server process is to parse the SQL statement, read the Oracle data blocks from the datafile to the data buffer, and return the result set to the client.

Oracle background processes

Oracle requires a number of processes to be running in the background, in order to be operational and open to users. These processes are:

► **Database writer (DBWR)** - This background process writes all dirty data blocks from the database buffer cache to the datafiles on disk. The DBA can configure multiple DBWR processes in order to improve performance.

► **Log writer (LGWR)** - This is the process that handles writing data from the redo log buffer cache onto the redo log files.

► **System monitor (SMON)** - This process has two functions: It performs an instance recovery when the Oracle instance fails, and it coalesces smaller fragments of disk space together.

► **Process Monitor (PMON)** - This process cleans up any remaining Oracle processes resulting from a failing user process. Furthermore, it rolls back any uncommitted transactions that were performed by the user.

► **Checkpoint (CKPT)** - This process writes log sequence numbers to the database headers and control files.

- ► **Recoverer Process (RECO)** - This process automatically resolves failures in distributed transactions when using the distributed database configuration.

- ► **Archiver Processes (ARCn)** - This process is used for ARCHIVELOG mode when automatic archiving is enabled, to copy redo log files to a designated storage device after a log switch.

- ► **Queue Monitor Processes (QMNn)** - This optional process monitors message queues when using Oracle Streams Advanced Queing.

DB2

For a DB2 instance to start and run, several processes are created and interact with each other. These processes maintain the database created on the instance and the applications connected to the database. There are several background processes in DB2 that are pre-started, and some start on a need-only basis. This section explains some of the important background processes.

DB2 background processes

The DB2 server activities are performed by Engine Dispatchable Units (EDUs) that are defined on a Windows environment as threads and as background processes on both UNIX and Linux systems.

Like Oracle, there are many background processes dedicated to the operation of the DB2 instance. As mentioned in the previous paragraph, some DB2 background processes are started with the instance, and others are initialized when the database is activated by a connection. Figure 1-6 shows the necessary background processors of the DB2 server at the instance, application, and database level. In the following sections, we discuss some of the important processes in each level of DB2.
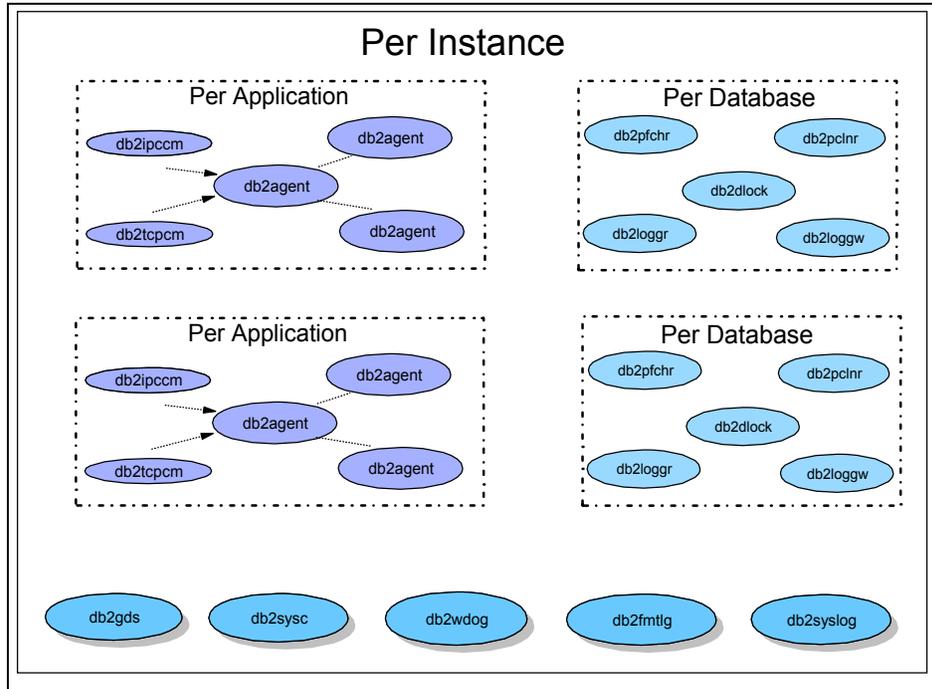
Figure 1-6   DB2 processor architecture

Instance level processes

The following background processes start as soon as the DB2 server is started with the **db2start** command:

- ► **DB2 daemon spawner (db2gds)** - This is a global daemon processor started for each instance. This process starts all the EDUs (process) in UNIX.

- ► **DB2 system controller (db2sysc)** - This is the system controller processor. This is the main process. Without this process, the instance cannot function.

- ► **DB2 watchdog (db2wdog)** - This process is required only on UNIX platforms. This process is the parent process for all the processes.

- ► **DB2 format log (db2fmtlg)** - This preallocates log files in the log path when the LOGRETAIN database configuration parameter is set to ON, and the USEREXIT parameter is set to OFF. This is similar to the optional Archiver log (ARCn) process in Oracle and is enabled when the database is set in ARCHIVELOG mode.

- ► **DB2 system logger (db2syslog)** - This is the system logger process responsible for writing the operating system error log.

Database level processes

The following background processes are started when an active connection to the database is established:

- ▶ **DB2 log reader (db2loggr)** - This process reads the log files during transaction rollback, restart recovery, and rollforward operations. It does part of the functions that the Oracle PMON process does.

- ▶ **DB2 log writer (db2loggw)** - This is the log writer process that flushes the database log from the log buffer to the transaction log files on disk. This is equivalent to the LGWR process in Oracle.

- ▶ **DB2 page cleaner (db2pclnr)** - This process is presented to make room in the buffer pool before prefetchers read pages on disk storage and move them into the buffer pool. Page cleaners are independent of the application agents that look for and write out pages from and to the buffer pool to ensure that there is room in the buffer pool. This is equivalent to the DBWR process in Oracle.

- ▶ **DB2 prefetcher (db2pfchr)** - This process retrieves data from disk and moves it into the buffer pool before the application needs the data. This does part of the functions of the Oracle server process.

- ▶ **DB2 deadlock detector (db2dlock)** - This is the database deadlock detector process. This process scans the lock list (the lock information memory structure of DB2) and looks for deadlocked connections.

Application level processes

These processes are started for each application connecting to the database:

- ▶ **DB2 communication manager (db2ipccm)** - This is the interprocess communication (IPC) process started for each application connecting locally. This process communicates with the coordinating agent to perform database tasks. This can be thought of as an Oracle user process connecting locally.

- ▶ **DB2 TCP manager(db2tcpcm**) - This is the TCP communication manager process. This process is started when the remote client or application connects to the database using TCP/IP communication. This process communicates with the coordinating agent to perform database tasks. This is equivalent to a user process in Oracle.

- ▶ **DB2 coordinating agent (db2agent)** - This process handles requests from applications or connections and performs all database requests on behalf of the application. There will be one *db2agent* per application unless the connection concentrator is established. If intra-partition parallelism is enabled, the db2agent will call DB2 subagents to perform the work.

- ▶ **DB2 subagent (db2agnta)** - This is an idle subagent, which works with the db2agent process when intra-partition parallelism is enabled.

► **Active subagent (db2agntp)** - This is the active subagent that is currently performing work. This process is used when enabling SMP parallelism, which means having more processes achieving the same task. In order to enable this feature in DB2, you must set the intra-parallelism database parameter to true.

The db2agent process, with or without the combination subagents, performs a similar function to that of the Oracle server process.

1.3.3 Files and directory structure

This section discusses important files and the common directory structure used in Oracle and DB2. An instance and database requires a number of files such as datafiles, configuration files, log files, etc. to operate and store data. The directory structure gives an idea of how a product is installed and how some files are placed on this structure.

Oracle

Every Oracle instance needs a set of files to comprise itself and operate. These files include the datafiles, redo log files, control file, parameter file, the alert and trace log files, and the password file, as shown in the Figure 1-7. The physical files to mount a table space in Oracle are called *datafiles*. The datafiles store the data, index and rollback segments of the Oracle database. Oracle maintains the database transactions in a transactional log files called *redo log files*. There should be at least one set of redo log files created for a database to operate. Every Oracle database has a *control file*. The control file contains the entries that describes the physical structure of the database. Every time a instance is started, the control file is used to identify the datafiles and redo log files to start the database.

The *parameter* file is used by the Oracle instance during startup. The file contains the values for many initialization parameters used to allocate memory and start the process for the instance to run. The *password* file is a security file used for authenticating which users are permitted to start up or shut down an instance or perform other privileged maintenance on a database with SYSDBA or SYSOPER privileges and additionally OSDBA or OSOPER privileges. The *alert and trace log files* are the diagnostics files used by the Oracle instance to record all the dump information of the database such as internal errors, block corruption errors, and so forth.
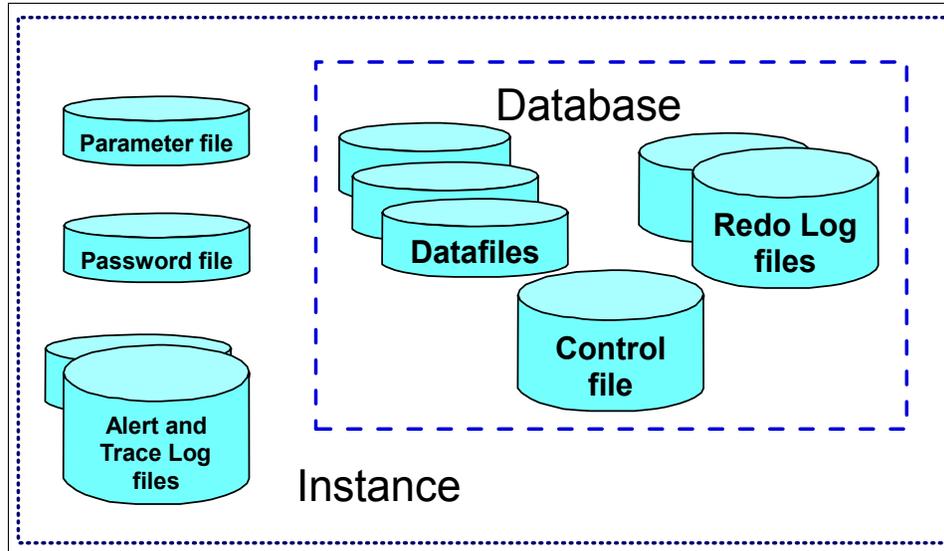
Figure 1-7   Oracle database files

Oracle installation follows Optimal Flexible Architecture (OFA) standards in creating the directories and placing the files. The OFA is a set of file naming and placement standards. Oracle recommends following OFA standards. Using OFA, the Oracle installation process places the Oracle software in $ORACLE_BASE\ORACLE_HOME and database files in the $ORACLE_BASE\oradata directory. Figure 1-8 shows a sample installation directory structure on the 10g version of Oracle. The initialization file and the password file reside under the dbs path in the UNIX server and database directory in Windows server. The bin directory contains all the executable binary files.

The $ORACLE_HOME\rdbms\admin directory contains the DDL scripts to create the data dictionary tables and views, the administration procedures, and package scripts. These scripts are run when creating the database manually. The $ORACLE_HOME\network\admin directory contains the listener.ora and tnsnames.ora files for communication process. Section 1.3.5, "Communication" on page 27 explains more about these files.

```
                Windows                                    UNIX

c:\oracle\product\10.2.0          /u01/app/oracle
            \admin                             /admin
                \ora_emp                       /product/10.2.0/db_1
                     \adump                         /bin
                     \bdump                         /dbs
                     \cdump                         /install
                     \create                        /lib
                         ...                        /network
            \db_1                                   /sqlplus
                 \BIN                               /rdbms
                 \database                              ...
                 \LIB                          /oradata
                 \sqlplus
                     ...
            \oradata
```

Figure 1-8   Oracle directory structure using OFA

DB2

The primary files and directories for a DB2 instance and database include the DMS containers, SMS containers (directory or path), DBM CFG file, DB CFG file, Transaction log files, and the db2diag.log file. This structure is shown in Figure 1-9. The DBM CFG file is created per DB2 instance and contains the configuration parameters and values. This file resides under the sqllib directory of the instance home named db2systm. This can be related to the initialization parameter file in Oracle, but unlike Oracle, this is not a text file, but rather a binary file and can only be updated using the UPDATE DBM CFG command.

The DB CFG file is the configuration file for each database; it stores the database configuration values. This file is stored with the name SQLDBCON under the database directory SQLnnnnn, where nnnnn is the database number assigned. This file is also a binary file and can only be updated using the UPDATE DB CFG command.

Each database contains the table space containers, which can be either DMS containers as a physical file or partition, or SMS containers, which is a directory in Windows and a file system path in the UNIX environment. Under SMS, a number of different files are created to store the data and the index. The transactional log files record the database transactions, which is required for database recovery. The NEWLOGPATH database configuration parameter identifies the log path if the log files are stored in a location other than the default log path. The db2diag.log file, which is like the Oracle Alert log file, records the

DB2 error dump information. The DIAGPATH DBM configuration parameter
identifies the location of the db2diag.log file.
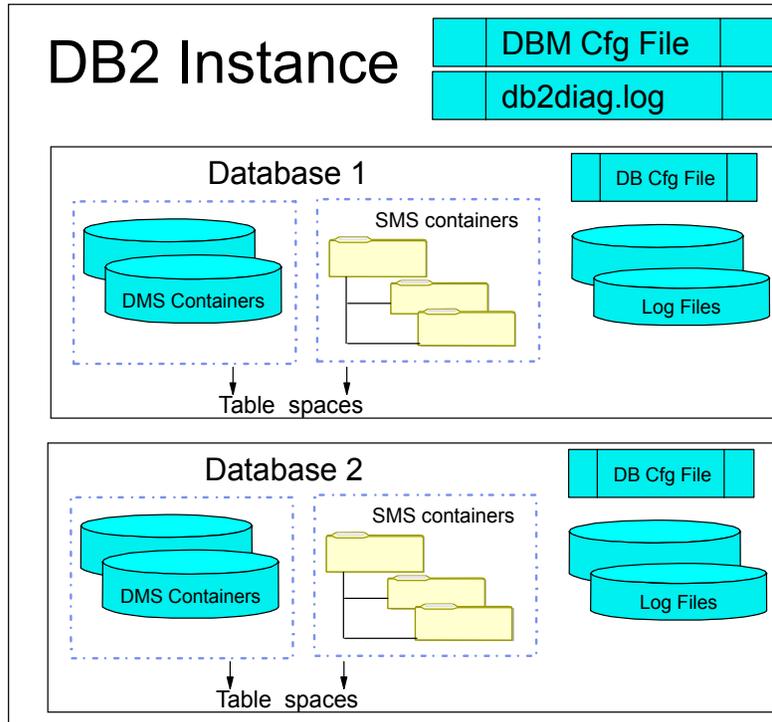
Figure 1-9   DB2 Instance and database files

Figure 1-10 shows the default directory structure for a simple CREATE
DATABASE command with no TABLESPACE options specified. By default, in
DB2 9, the table spaces will be created using automatic storage. You can
optionally specify storage locations for the automatic storage on the CREATE
DATABASE command, however, if no locations are specified, the automatic
storage location uses the value specified in the DBM CFG parameter
DFTDBPATH. The catalog and user table space will then be created as DMS
containers, while the system temporary table space will be created as an SMS
container. The log files will be created in the SQLLOGDIR directory, which can
be changed by updating the NEWLOGPATH DB CFG parameter.
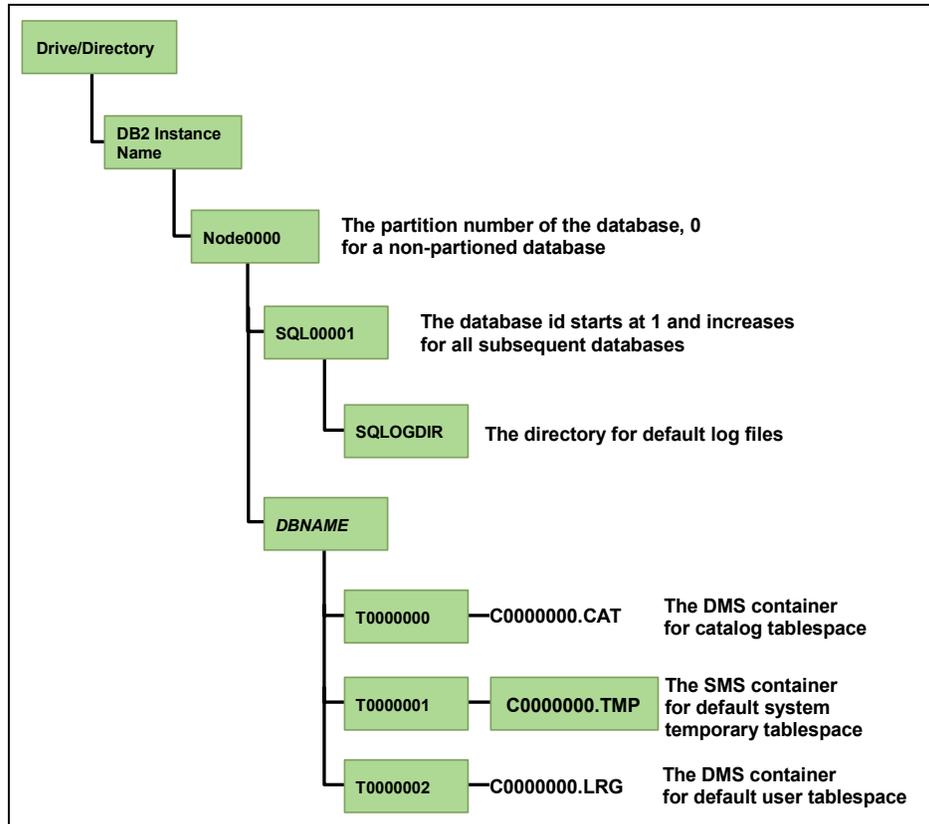
Figure 1-10   DB2 directory structure for a simple create database command

The DB2 Installation path depends on the operating system in which it is installed. On the Windows operating system, the default installation goes under the C:\Program Files\IBM\SQLLIB directory. This path can also be changed during the installation. On UNIX systems, the default installation path is /opt/IBM/db2/V9.1, while on Linux the default installation path is /opt/ibm/db2/V9.1. On UNIX systems, a sqllib directory is created under the instance home directory and has symbolic links to actual files under the installation directory. Figure 1-11 shows some of the installation directory structure for the Windows environment, and the sqllib directory structure for UNIX environments.

On UNIX, the adm directory consists of instance administration commands, license management commands, and other commands. The backup directory consists of the DBM configuration file backup and node configuration backup. The bin directory consists of all DB2 command binaries. The bnd directory contains various database bind files. The db2dump directory holds the

db2diag.log file and other trace files. All the external stored procedures and executable programs are stored under the function directory. The Java directory contains the JDBC driver files. The samples directory contains all the program samples that come shipped with DB2 software.

```
              Windows                              UNIX

C:\Program Files\IBM\sqllib          /$DB2_HOME/sqllib
     \adsm                                 /adm
     \BIN                                  /adsm
     \bnd                                  /backup
     \cfg                                  /bin
     \DB2                                  /bnd
     \FUNCTION                             /cfg
     \include                              /db2dump
     \instance                             db2nodes.cfg
     \java                                 db2profile
     \lib                                  /function
     \license                              /include
     \samples                              /java
                                           /lib
                                           /lib32
                                           /lib64
                                           /samples
```

Figure 1-11   DB2 directory structure

Configuration files

Since the Oracle instance can only support one database, its background processes are enabled as soon as the instance is started. Therefore, Oracle has one configuration file that is used to configure and tune the database.

The DB2 instance, however, can support multiple databases, and therefore, consists of an instance level shared memory and a database shared memory, running on the server side. Starting the DB2 instance will only start the instance level processes. Database level processes, such as those that control transactional processing tasks, logging, and writing to containers on disk are only enabled when the database itself is activated by a user or an application connection.

Therefore, there are two files controlling the configuration and tuning of the DB2 server and database. The first file is used to configure and tune the DB2 server at the instance level is called the Database Manager Configuration (DBM CFG) file. The second is a database level configuration file (DB CFG) used to control database level parameters.

1.3.4 Data Dictionary and Catalog

Every RDBMS has a form of metadata that describes the database and its objects. Essentially, the metadata contains information about the logical and physical structure of the database, integrity constraints, user and schema information, authorization, privilege information, and so on.

In the Oracle database, this metadata is stored in a set of read-only tables and views called the Data Dictionary. These tables and views are updated by the Oracle server. The Data Dictionary is owned by the user SYS and stored in the SYSTEM table space. The base tables are all normalized and are seldom accessed directly, hence, user accessible views are created using the catalog.sql script. The Data Dictionary is organized under three qualifiers: the USER_xxx views, the ALL_xxx views, and the DBA_xxx views.

The USER_xxx views show the object information owned by the current user; the ALL_xxx views show all the object information that can be accessed by the current user; and the DBA_xxx view is the database administrator view and contains information on all the objects in the database. Apart from the Data Dictionary, Oracle maintains another set of virtual tables called the dynamic performance views; the views created on them are prefixed by V$. These views are called the fixed views, and are available when the instance is started, without the need of the database to be opened.

In DB2, the metadata is stored in a set of base tables and views called the Catalog. The Catalog contains information about the logical and physical structure of the database objects, object privileges, Integrity information, etc.

The DB2 database Catalog is automatically created when the database is created. The base tables are owned by the SYSIBM schema and stored in the SYSCATSPACE table space. On top of the base tables, the SYSCAT and SYSSTAT views are created. SYSCAT views are the read-only views that contain the object information, and SYSSTAT are the updateable views, which contain statistical information. Users should view the catalog information through the SYSCAT views, as the base tables are more complex and not as reader friendly.

Unlike Oracle, DB2 does not maintain any dynamic performance views, but uses commands to get the information from the system directory, such as LIST DATABASE DIRECTORY, LIST TABLESPACES, LIST APPLICATIONS. Table 1-3 shows some of the commonly used views available in the Oracle Data Dictionary and DB2 Catalog.

Table 1-3   Data Dictionary and Catalog

Oracle Data Dictionary	DB2 Catalog
DBA_TABLES	SYSCAT.TABLES
DBA_TAB_COLUMNS	SYSCAT.COLUMNS
DBA_TABLESPACES	SYSCAT.TABLESPACES
DBA_INDEXES	SYSCAT.INDEXES
DBA_TAB_PRIVS	SYSCAT.TABAUTH
DBA_TRIGGERS	SYSCAT.TRIGGERS
DBA_VIEWS	SYSCAT.VIEWS
DBA_SEQUENCES	SYSCAT.SEQUENCES
DBA_PROCEDURES	SYSCAT.ROUTINES

The complete list of DB2 Catalog views can be found in *SQL Reference, Volume 1,* SC10-4249.

1.3.5 Communication

This section gives an overview of the communication architecture that enables simple client-server communication in Oracle and DB2 environments and some of the tools used to communicate from the client.

Database accessing

Both DB2 and Oracle support dynamic and embedded static SQL interfaces. Oracle provides the SQL*Plus tool for command line access to the database server. SQL*Plus also comes with a GUI version. DB2 provides the Command Line Processor (CLP) for command line access to the database server. The GUI version for this tool is called the Command Editor. The Oracle client installation installs the SQL*Plus tool, Oracle Net Services software, ODBC drivers, and other tools. This software provides basic client-server communication to access the database server. The DB2 client installation provides the DB2 runtime client, Command Line Processor, Configuration Assistant, Command Editor, ODBC drivers, etc. for basic client-server communication.

Oracle communication

The client-server communication in Oracle Server is handled by Oracle Net Services. Oracle Net Services support communications on all major protocols. The Oracle Net Services provide a communication interface between the client user process and the Oracle server process, enabling the data transmission and

message exchange between Oracle server and client. The Oracle Net Services use a technology called Transparent Network Substrate (TNS) to perform these tasks. The TNS enables peer-to-peer application connectivity, where the two nodes communicate with each other directly.

The Oracle Net Services provides the listener process that resides in the Oracle server, which listens for incoming client connection requests, and maps it to the Oracle instance. The listener is configured with one or more protocol addresses; the client is configured with one of these protocol address and can send connection requests to listener. A configuration file, listener.ora, is maintained in the Oracle server that contains the protocol address, database service information, and listener configuration parameters. The listener process is controlled by the LSNRCTL utility; the LSNRCTL utility reads the listener.ora file and starts the listener process. The server services information in the client is maintained in a file called tnsnames.ora. Oracle Net Configuration Assistant and Net Manager are graphical utilities used to configure the Oracle Net Services like listener, service naming, and so on.

DB2 communication

DB2 supports several communication protocols for client-server communication such as TCP/IP, NPIPE, and so on. Most protocols are automatically detected and configured during an instance creation. The DB2COMM registry variable identifies the protocol detected in a server. To enable a specific protocol, use the db2set DB2COMM command. For TCP/IP, a unique port address has to be specified in the database manager configuration. This port is registered in the services file. For example, to reserve port 50000 with the service name db2c_DB2, the entry in the services file is:

```
db2c_DB2 50000/tcp
```

Update this information in the database manager using the command:

```
db2 UPDATE DBM CFG USING SVCENAME db2c_DB2
```

These tasks can also be performed using the DB2 Configuration Assistant utility. At the client, the database information is configured using either the CATALOG command or using the Configuration Assistant. The database are configured under a node which describes the host information like protocol and port, etc. To configure a remote TCP/IP node, the command used is:

```
db2 CATALOG TCPIP NODE node-name REMOTE host-name SERVER
service-name
```

The service name registered in the server or the port number can be specified in the *SERVER* option. To catalog a database under this node, the command is:

```
db2 CATALOG DATABASE database-name AS alias-name AT NODE node-name
```

The Configuration Assistant is the GUI tool used in the client to configure a database. Figure 1-12 shows how the Configuration Assistant is used to add a database connection. The option "Search the network" is used to add a database using the DB2 Discovery Method. Using this option, the DB2 servers installed in the entire network can be searched and used to add database connection. This is possible when the DB2 Administration Server (DAS) process is created on the server and enabled for discovery.

Note: The DB2 Discovery method is enabled at the instance level using the DISCOVER_INST parameter, and at database level using the DISCOVER_DB parameter.
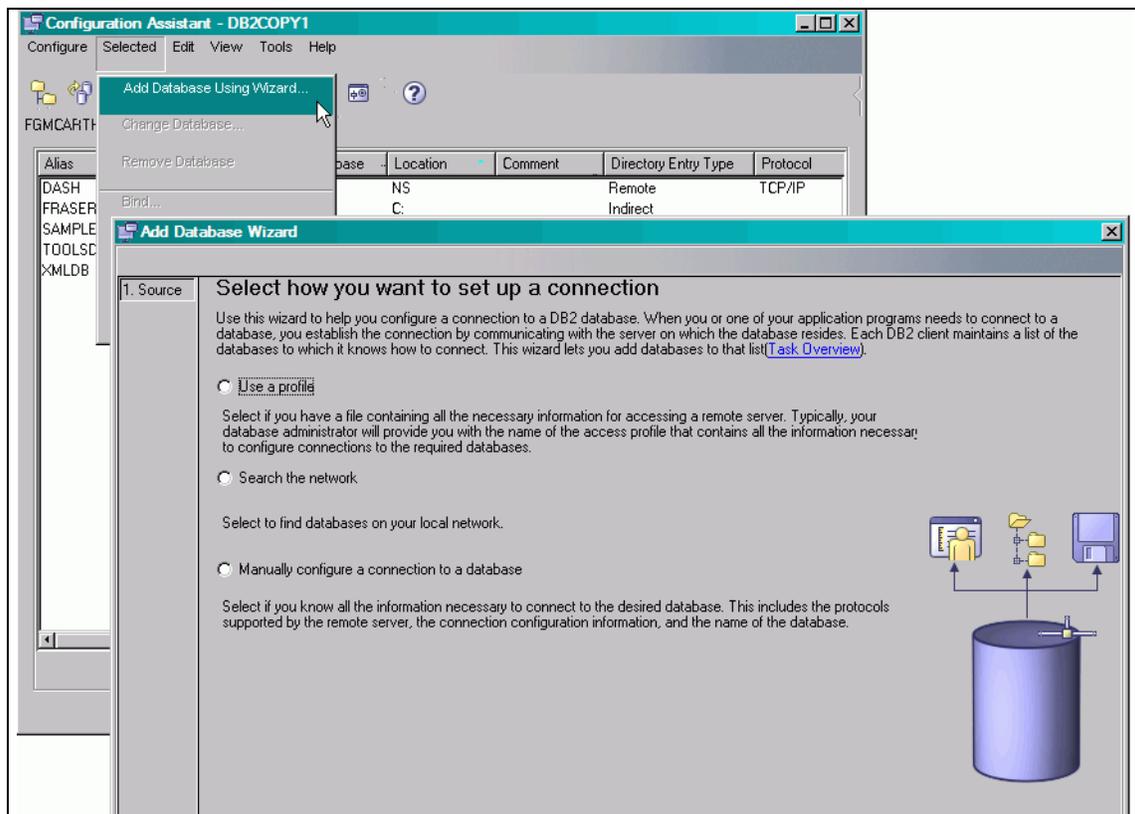
Figure 1-12   Configuring the database connection using Configuration Assistant

1.3.6 Data replication

Both Oracle and DB2 provide replication capabilities in their databases. The main purpose of enabling replication is to have the same set of date records in more than one location. Having two copies of the same database can be part of a high availability solution. This section discusses the replication approach by Oracle and DB2.

Oracle replication

Oracle has the ability to replicate data from one Oracle database to another. Changes to an Oracle database are replicated to another Oracle database through the *capture* and *apply* processes. Oracle replication uses triggers in order to capture transactional changes and stores them in a local queue. Oracle then uses packages in order to apply the replicated changes to the target database. Oracle Enterprise Manager (OEM) may be used to perform replication tasks.

DB2 replication

DB2 replication is an asynchronous process. The frequency of replication can be set to minimize any delay. There are two administration interfaces through which we can set up replication. The first is the DB2 Replication Center, which are GUI tools provided by DB2 to define, manage, and monitor replication. The second is the *asnclp* tool, which provides command line definition of replication objects.

The replication process in DB2 typically consists of identifying and setting up three databases:

► **Source database** - Contains source tables that need to be replicated.

► **Target database** - A database server where the target tables will reside and the apply process will take place.

► **Control database** - Contains the control tables storing the necessary information for the apply program and which can reside on either the source or target database.

To replicate data to or from a non-IBM relational database, you use the replication and federated functions in WebSphere Replication Server for Linux, UNIX, and Windows. The corresponding source or target replication function can be provided by DB2 for Linux, UNIX, and Windows, or another WebSphere Replication Server. WebSphere Replication Server supports replication to and from the following non-DB2 targets and sources:

► Informix
► Microsoft SQL Server
► Oracle
► Sybase

► Teradata (target only)

The three types of replication provided by DB2 are: SQL replication, Q replication, and Event publishing.

SQL replication

In SQL replication, changes to sources are captured, and the committed transactional data is stored temporarily in staging tables. The changes are then read from the staging tables and replicated to corresponding target tables. With staging tables, data can be captured and staged once for delivery to multiple targets, in different formats, and at different delivery intervals.

You can use SQL replication to replicate data from DB2 sources to targets by using the Capture and Apply programs. The Capture program runs on the source system. The Capture program reads DB2 recovery logs for changed source data and saves the committed changed data to staging tables. The Apply program typically runs on the target system. The Apply program retrieves captured data from staging tables and delivers the data to targets.

Q replication

In Q replication, large volumes of data are replicated at very low levels of latency. Q replication captures changes to source tables and converts committed transactional data to messages. This data is sent as soon as it is committed at the source and read by the Q replication server. The data is not staged in tables. The messages are sent to the target location through WebSphere MQ message queues, where the messages are read from the queues and converted back into transactional data. The transactions are then applied to target tables. There is one transmission queue for each target.

With Q replication, you can replicate committed transactional data from DB2 sources to targets by using the Q Capture and Q Apply programs. The Q Capture program runs on the source system. The Q Capture program reads DB2 recovery logs for changed source data and writes the changes to WebSphere MQ queues. The Q Apply program runs on the target system. The Q Apply program retrieves captured changes from queues and writes the changes to targets.

Event publishing

In event publishing, changes to source tables are translated into XML messages and sent over WebSphere MQ queues to a user application of your choice. Event publishing uses only the Q Capture program, not the Q Apply program. The Q Capture program captures changes that you specify when you create an object called an XML publication. These transactions or row-level changes are then sent to a queue. You specify which queue to use when you create an object called a publishing queue map.

Event publishing gives you the flexibility to use transactional data that is published in XML format for a wide variety of uses. You determine what happens to the published changes. You can use them to feed data to your web applications, trigger events, and more. If you want to replicate changes to a target using the Q Apply program, use Q replication rather than event publishing.

1.4 Parallel database architecture

Both Oracle and IBM offer parallel architecture or clustering environments for their databases in order to provide customers with the ability to support very large databases (VLDBs). This is achieved by partitioning the database over multiple nodes or servers. Oracle offers Real Application Cluster (RAC), formerly known as Oracle Parallel Server, and IBM offers DB2 Enterprise with the Database Partitioning Feature (DPF), formerly known as DB2 Extended Enterprise Edition (EEE).

There are three major architectures used to implement a partitioned environment: Shared memory, Shared disk, and Shared nothing. This book briefly discusses both Shared disk (Figure 1-13) and Shared nothing (Figure 1-14) architectures. Table 1-4 shows the differences between the technologies.

Table 1-4 Shared disk architecture vs. Shared nothing architecture

Shared disk architecture	Shared nothing architecture
Requires special hardware	Does not require special hardware
Non-linear scalability	Provides near linear scalability
Balanced CPU or node fail-over	Balanced/Unbalanced CPU or node fail-over
Requires CPU level communication at disk access	Minimal communication
Non-disruptive maintenance	Non-disruptive maintenance

1.4.1 Real Application Clusters

Real Application Clusters (RAC) is Oracle 10g's clustering technology, which provides an environment capable of supporting large databases. RAC is based on a shared disk architecture aimed at achieving high availability of a distributed environment.
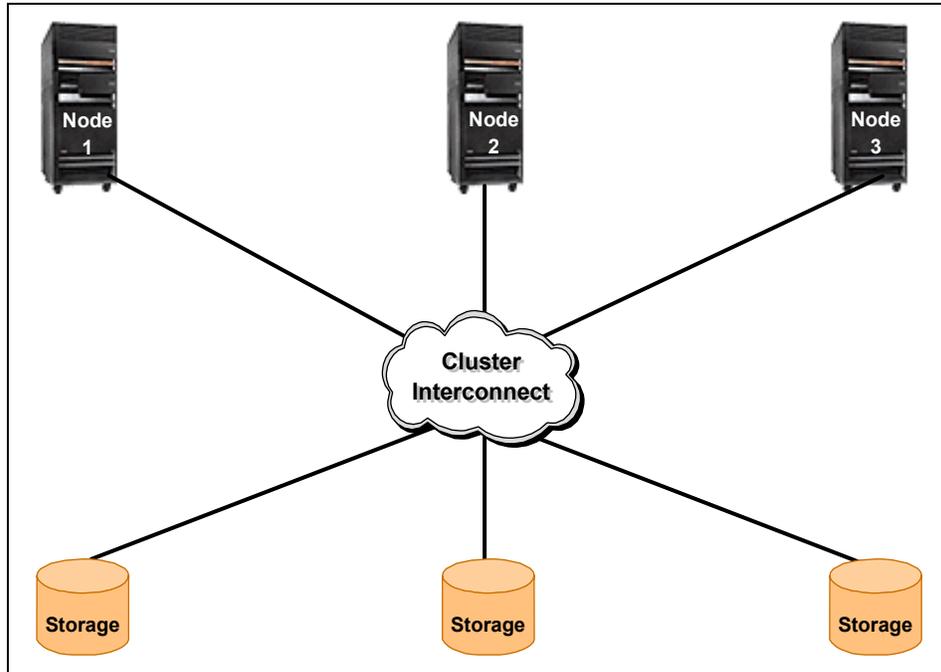
Figure 1-13   Shared disk architecture

RAC is an extension to the Oracle database, which enables building a multi-node database environment. RAC requires an Oracle database, and the clustering technology provided by the platform vendor in order to achieve successful installation and implementation.

RAC consists of three major components: nodes containing CPUs, cluster interconnect, and storage units. The nodes are processing nodes. Typically, every node is a symmetric multiprocessing node (SMP). As shown in Figure 1-13, in a shared disk environment every processing node has access to every storage unit. However, every node in the cluster has its own memory, operating system, and database instance. The nodes do not share memory among each other.

1.4.2 DB2 Enterprise with the Database Partitioning Feature (DPF)

IBM Information Management software extends DB2 to the parallel multi-node (multi-partition in DB2 terminology) environment in order to provide a scalable solution capable of supporting large amounts of data.

The database partitioning feature in DB2 is based on a shared nothing architecture. As shown in Figure 1-14, every partition in the cluster has its own

dedicated memory, operating system, and storage units. An application of a shared nothing architecture is aimed at achieving high scalability and improving performance. The DPF option of DB2 Enterprise does not require any clustering technologies to run. However, a high availability solution can be implemented in conjunction with the clustering technology provided by the platform vendor.
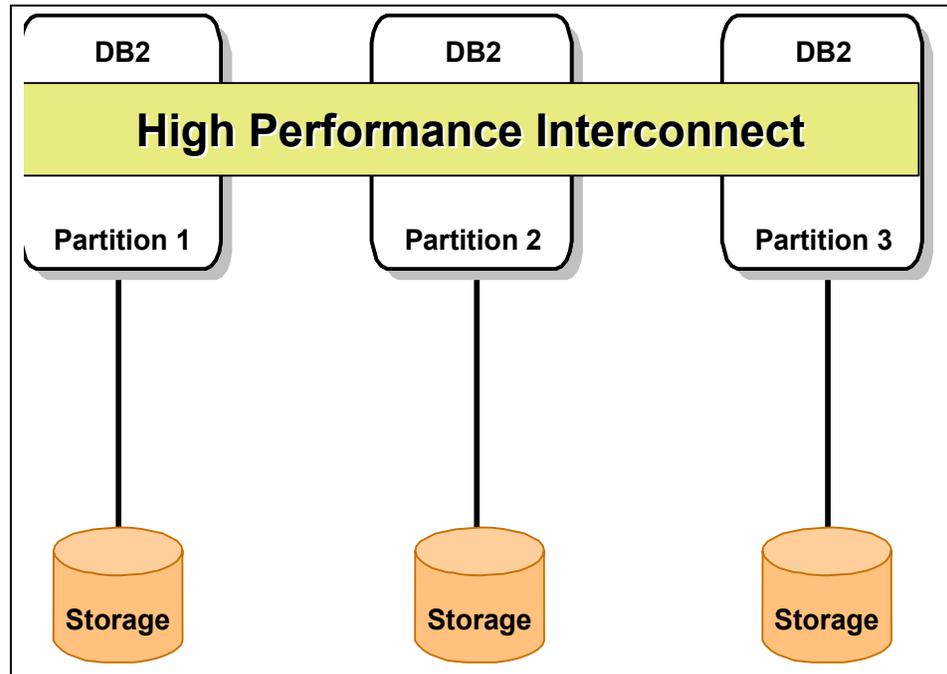
Figure 1-14   Shared nothing architecture

DB2 Enterprise DPF uses two levels of parallelism in order to achieve good performance:

▶ **Intra-partition parallelism,** which is the ability to have multiple processors process different parts of an SQL query, index creation, or a database load within a database partition. This level of parallelism can be specified in the DBM configuration file by setting the INTRA_PARALLEL parameter to ON.

▶ **Inter-partition parallelism**, which provides the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one server or multiple database servers. This can be accomplished on both SMP servers and massively parallel processing (MPP) clustered servers.

2

Conversion methodology

This chapter describes the IBM migration strategy and the resources that are available to assist you towards this goal. We also discuss some database features for which reengineering can be considered during the planning time due to the implementation differences between Oracle and DB2.

We discuss the following topics:

- ► IBM conversion strategy
- ► Additional migration resources
- ► Conversion planning technical considerations

2.1 Pre-conversion tasks

Before undertaking a migration project, there are several planning activities that should be performed.

The following list summarizes those areas and the type of information that you need to gather and consider:

► Perform a hardware and software architectural assessment:

– Decide on the target hardware platform for the production system.

– Understand the workload characteristics and requirements.

– Know the number of concurrent users.

– Take an inventory of software and hardware upgrades.

– Decide what machine the migration will be developed and tested on.

► Investigate the scope, duration, and cost of the project:

– Which application areas will be migrated?

– How complex is the migration?

– How many database objects and applications will be migrated?

– What are some of the differences between the source and target databases?

– How long will the migration take?

– Obtain a ballpark estimate or proposal for work.

► Identify business dependencies:

– Are there timeline or business cycle requirements?

– Do renewal of licensing agreements impact schedules?

► Identify the people resources and skills required:

– Will resources be in-house, outsourced, or a combination of both?

– Do in-house resources have skills for both the source and target databases?

– Are in-house resources available to perform the migration?

– Are in-house resources available to support an outsourced migration?

► Identify the services and tools that can be used during the migration.

2.1.1 IBM Software Migration Project Office team

The Software Migration Project Office team (SMPO) at IBM provides free assistance for many pre-migration tasks. The team consists of technical specialists whose mission is to facilitate and assist with all phases of a migration to DB2. The team has assisted hundreds of customers with their migrations and has database administration and application development skills for the source databases (Oracle, SQL Server, and Sybase) that will be migrated to DB2.

Some of the tasks that the SMPO provides assistance with are:

- ► Selection of application areas to migrate
- ► Assessment of migration complexity
- ► Ballpark migration estimates delivered in hours
- ► Sample database and code conversions
- ► Migration tool selection and demonstrations
- ► Implementation of DB2 features
- ► Obtaining technical collateral
- ► Problem resolution related to the migration
- ► Selection of migration services
- ► Database administration and SQL application development comparison presentations between the source database and DB2

In addition, the SMPO has a direct line of communication to the DB2 development teams. The benefits of this relationship are several: it provides a conduit that assists in resolving migration related issues; it is used to advise the developers of features that, if added to DB2, can facilitate future migrations; it relays customer wish-lists for future DB2 features.

The SMPO ballpark estimate

A type of assistance that is frequently requested from the SMPO is a ballpark estimate for the conversion of database objects and applications. The SMPO uses prior experiences to deliver an estimate ranging from the least to the maximum number of hours of effort expected. To deliver the ballpark, the SMPO provides a questionnaire and the customer collects and returns metrical information for the objects to be converted.

The following is an example of the types of metrics collected:

- ► Number of database objects including tables, indexes, and views

- Number of database application objects including packages, procedures, triggers and user defined functions; average lines of code and SQL statements for each
- Number and language of application programs including lines of code and average number of SQL statements per module
- Volume of production data to be migrated

Although the metrics of a source system are an indicator of the size and complexity of a migration, there are other activities that are also part of most migration projects.

Some of these activities are:
- Database physical design
- Testing
- Performance tuning
- Cutover to production
- Project management
- Technical education

Note: The SMPO ballpark estimate does not include the hours required to perform the aforementioned activities.

If the migration project is outsourced to IBM services, a proposal for work is delivered along with an estimate in time and dollar amount. The services' estimate uses the information collected by the SMPO questionnaire as input (when available) and includes all activities that are part of a migration project.

In addition to metrical information and project tasks, there are additional factors that influence the size and complexity of a migration.

Some of these factors are:
- Amount and type of proprietary SQL used
- Quality of data
- Existence of system documentation
- Database design requirements such as high availability and replication
- Third party software dependencies
- Operating system and hardware platform change
- Availability of a dedicated machine for migration development
- Extent of code freeze enforcement during migration
- Length of the cutover window
- Skill level of resources performing the migration

These issues are examined more closely and taken into account during the assessment phase of a migration project.

For more information about the SMPO, visit the following Web site:

http://www.ibm.com/software/solutions/softwaremigration/

You may also contact your local IBM Sales Representative for information about the SMPO.

2.2 IBM conversion strategy

The IBM worldwide DB2 Migration Center has developed a "best-practices" methodology to help customers develop a strategy of how to best migrate their databases to DB2. You can use this methodology to conduct your own migration project or can contract the fee-based services of the DB2 Migration Center.

The DB2 Migration Center consists of a team of migration specialists that can perform the entire migration or partner with your team for a more cost effective solution. Alternatively, the DB2 Migration Center can also simply provide direction and advice on an as-needed basis at an hourly rate. The DB2 Migration Center team can be contacted through your IBM Sales Representative.

If you select the DB2 Migration Center to perform or participate in the migration, you can expect a team that has completed successful migrations for over twenty years. The result of these experiences has led to the development of a planned approach that can minimize the risk and cost of a migration. The Migration Center takes full advantage of all skills available at IBM including the lab development teams. In fact, a "lab advocate" is assigned to each project to ensure success.

Note: The SMPO differs from the DB2 Migration Center in that the SMPO is a pre-sales team that offers non-billable migration assistance.

IBM's migration methodology consists of the following primary phases:
- ► Assessment
- ► Conversion
- ► Test
- ► Implementation and cutover

Generally a migration project is an iterative process that consists of multiple rounds of conversion, testing, and refinement. Each phase has specific objectives and deliverables and is described in the following sections.

> **Note:** We recommend that a dedicated machine be used during development and testing of the migrated system.

2.2.1 Assessment phase

In this phase, it is essential to focus on system architecture analysis and information gathering to make decisions about how the migration will be performed. The primary objectives of this phase are to develop an overall migration strategy, perform initial project planning, and assess the benefits and risks of various migration options.

Other objectives include:

► Analyze characteristics and size of the source environment

   – Inventory the database objects and code to be converted.
   – Validate project scope.

► Devise a strategy for key issues

   – Coexistence of source and target databases
   – Ongoing application development
   – Change management
   – Performance requirements
   – Tool selection
   – Naming conventions
   – Database physical design
   – Define standards
   – Data migration strategy
   – Security planning
   – Cutover strategy

► Project plan development

   – Task assignments
   – People resources are identified
   – Project milestones defined

> **Note:** If the project is outsourced to the DB2 Migration Center, this phase is also used to validate the estimates delivered by the Migration Center prior to starting the project. At the end of this phase, adjustments to the initial estimates may be made.

2.2.2 Conversion phase

This phase is the core of the migration project and is divided into three sub-phases:

► Database conversion and design
► Calibration
► Application migration

Database conversion and design

During this phase, a test database is set up and used for migration development and functional testing of converted objects.

In addition, in this phase the final DB2 physical database design for the production system is planned and designed. The physical layout of the DB2 database is critical to the success of the project. It must be implemented according to best practices to ensure optimal performance. It must be structured to support future maintenance and to provide maximum flexibility to meet future requirements—all without compromising existing business processes and program logic.

A good portion of the database object conversion is automated. The IBM Migration Toolkit (MTK), an IBM product and free download, is the tool of choice for converting database objects such as tables, data types, constraints, indexes, views, triggers, stored procedures, user-defined functions, built-in functions, and sequences. However, the tool does not migrate database security features such as user IDs, authorizations or privileges (GRANTs and REVOKEs). In addition, the creation of table spaces and the placement of tables and indexes within table spaces along with disk layout is a manual part of the physical design process.

While the MTK is the tool of choice to convert procedures, triggers, and user-defined functions to DB2, the conversion of these application objects usually requires a certain amount of manual intervention depending on the SQL constructs used. The MTK reports those statements that cannot be converted automatically.

Lastly, during this phase a small amount of test data is loaded in support of functional testing of the converted objects.

> **Note:** A migration typically does not include logical redesign, although some amount of reengineering may be done to take advantage of DB2-specific features.

Calibration

The calibration subphase is designed to validate the planned strategy for the conversion including quality assurance review, standards compliance, and performance. During the calibration subphase, a few selected programs are converted as samples to ensure that the code produced complies with requirements and possesses characteristics that ensure the future maintainability of the applications. Typically, 20 to 60 programs of a selected group are converted.

There are several aspects in defining the system to be used in the calibration. The primary requirement, however, is that the chosen programs should be representative of how the whole application portfolio uses the source database and application languages. They are also expected to provide baseline performance and data usage that can be easily generalized. The generalized results will then be used to predict the convertibility of each of the other application subsystems as well as their likely post-conversion performance characteristics.

During this phase there is a review of the data transfer requirements and design of data transfer strategy. The data transfer strategy must ensure that all the production data can be transferred into the production system within the cutover window.

In addition, detailed resource and schedule planning is finalized. The test strategy for user acceptance and parallel testing is also finalized.

Application migration

Insights gained during the calibration phase are incorporated into the migration strategy. In this phase, conversion of the entire portfolio of objects and applications is completed.

Depending on the project, the following types of SQL applications may be converted:

► Embedded SQL programs such as those written in C, C++, COBOL, or Java
► ODBC and JDBC programs
► SQL scripts
► .NET programs

> **Note:** If the migration is performed by the DB2 Migration Center, then this phase may be performed at an IBM location.

Since this phase is mostly executed using manual techniques, expertise with both the source and DB2 platform is essential. Not all SQL may need to be converted because the same SQL syntax often runs on both the source and DB2

databases. However, all SQL must be examined to determine whether conversion is needed or not. Besides syntax, the semantics of the SQL needs to be examined to ensure that the SQL statement behaves the same way and returns the same results on DB2 as on the source database and performs as well if not better.

The MTK has an SQL translator, an interactive component, which can be used to convert a single SQL statement that has been extracted from the source code. The translator converts the statement into an equivalent statement for DB2. Before translating an SQL statement, the tables that the statement references must first be converted by the MTK.

2.2.3 The test phase

The test phase consists of three subphases:

- ► Conversion refresh
- ► Data migration
- ► Testing

Conversion refresh

Since most customers are unable to freeze their application for the duration of a migration, and since the calibration programs are typically converted early in the process, a migration refresh is required just prior to customer testing and production implementation. Based on change control logs, those programs, applications, and data structures that changed during the conversion phase are reconverted. This process relies heavily upon the inventory and analysis activities that took place during the assessment phase as that effort cross-referenced programs, procedures, functions, data structures, and application code. Using the baseline as a guide, the migration team reconverts those converted objects that were impacted by ongoing development.

Data migration

The process of data migration consists of four activities: unload, transform, cleanse, and reload. This subphase is accomplished by a combination of automated and manual methods.

In this phase, the MTK can be used to automate the migration of the data or to generate scripts that can be used to migrate the data manually. The MTK can be used to unload the data from the source database and load the data into DB2. However, most importantly, the MTK automates the transformation of the source data into a format that DB2 will accept during load without errors. For example, DB2 requires that character data is delimited by double quotation marks and that date and time values are in a specific format. Tools other than the MTK may be

used to unload the data, however, the format of the data should be examined to insure that it is in a format that can be loaded into DB2. If it is not, the data will have to be manipulated by an intermediate process into a format that can be loaded.

As mentioned previously, several test iterations of data migration are performed to ensure that all data can be migrated within the cutover window. Data migration is also needed to support performance testing.

Testing

The next step is the testing of the converted applications. This includes:

► Integration testing
► Parallel testing
► Performance testing
► System testing

In support of this phase, test scenarios need to exist or be developed. In addition, the production-like database environment is built, configured, tuned, and populated with a full volume of data. To assist with configuring DB2 parameters properly, the Configuration Advisor, a tool provided with DB2, should be used.

In this phase, integration, parallel, performance, and system testing are performed. Parallel testing consists of capturing output from the source environment for comparison with output and performance from the DB2 environment. The goal of this phase is to identify migration-caused defects and performance differences and issues.

If performance issues are found, it may be necessary to revisit the physical database design and implement some changes. The DB2 Design Advisor is a useful tool to assist with this task. The Design Advisor can recommend indexing and other physical database structures such as materialized query tables (MQTs), multidimensional clustering tables (MDC), and database partitioning features (used with DPF). Of course, any physical design changes may also result in data movement changes and perhaps even changes to the application code.

Other DB2 tools packaged with DB2 that can be used to tune performance, particularly SQL statements, are the Activity Monitor, the Snapshot™ Monitor, the Event Monitor, and the Explain facility (including Visual Explain).

2.2.4 Implementation and cutover phase

After the testing phase has completed, final testing is performed to validate implementation of the new production environment and to gain user acceptance.

Upon acceptance of the system, an additional "dry run" of the cutover procedure is performed, usually over a "dress-rehearsal" weekend prior to the final cutover weekend. Any issues discovered during the dry run are corrected. On the cutover weekend, the database is refreshed with production data and the system goes live.

In preparation of this phase, a backup of the source production system should be taken in case there is an unexpected need to back out the new system. In addition, all database maintenance procedures such as backups are put into production.

In some cases, the final cutover phase does not allow for downtime of the production system. In these cases, special planning and procedures are developed to accommodate the phasing in of the new system without impacting the production environment.

2.2.5 Migration project skills, roles, and responsibilities

During a migration project a variety of skills are required. In order for the migration to be successful, adequate skill levels within the following areas are recommended:

- ► System architect (part-time)
- ► System administration (part-time)
- ► Database administrator with source database and DB2 skills (full-time)
- ► Migration specialists with source database and DB2 application skills (full-time)
- ► DB2 tuning expertise (part-time)
- ► Subject matter expertise (part-time)
- ► Project management (full time)
- ► System testing (part-time)

You should carefully assess the skills of in-house staff and determine whether the above skills are available. If not, then the project should be either entirely or partially outsourced to fill the gaps.

If the DB2 Migration Center has been contracted to perform the migration, your team still plays an important role. Starting early in the project, your staff can "shadow" the IBM team as they perform their tasks. The IBM team also makes recommendations to your staff on how they can perform their tasks during the migration project. In this way, your staff can benefit from the expertise of the IBM team and gain a deep understanding of the migrated objects that you will

ultimately have responsibility for. In addition, during the project your staff takes primary responsibility for the following tasks:

► Project management

This person manages all tasks that are assigned to your team and reports the status of the project to your management.

► Management of the physical machine environment

The IBM team will rely on your staff to maintain physical machine resources.

► Testing

Except for functional testing, performed by the IBM team, your staff has the primary responsibility for the testing phase. It is also your responsibility to provide test plans, test scripts, and test data.

2.3 Additional migration resources

IBM provides the following additional migration resources:

► IBM Migration Toolkit (MTK)

The IBM Migration Toolkit is a free conversion tool that can be downloaded from:

http://www.ibm.com/software/data/db2/migration/mtk/

For additional information about the MTK, you can send an e-mail to the following address:

mtk@us.ibm.com

► IBM DB2 education

The following course is recommended for experienced Oracle DBAs:

Fast Path to DB2 for experienced relational DBAs (CF281)

For information on this course and other IBM education, see:

http://www.ibm.com/services/learning/

► DB2 manuals can be found at:

http://www-306.ibm.com/software/data/db2/udb/support/manualsv9.html

► IBM Redbooks can be found at:

http://ibm.com/redbooks

► IBM Press books on DB2 and other products can be found at:

http://www.redbooks.ibm.com/ibmpress/

- The DB2 Express-C Edition is a free version of DB2 and can be downloaded from:

  http://www.ibm.com/developerworks/downloads/im/udbexp/?S_TACT=105AGX 01&S_CMP=HP

- The IBM developerWorks Web site offers white papers and other technical information on DB2 development:

  http://www.ibm.com/developerworks

- The IBM Porting zone Web site offers general porting information:

  http://www.ibm.com/developerworks/db2/zones/porting/index.html

- The IBM DB2 MigrateNow Web site offers information on migrations to DB2:

  http://www.ibm.com/software/data/db2/migration/dbmigteam.html

- The following Task IDs can be used to contact IBM about migrations to DB2:
  - db2mig@us.ibm.com (for United States, Canada, and Latin America only)
  - emeadbct@uk.ibm.com (for Europe, Middle East, and Africa only)
  - dungj@hk1.ibm.com (for Asia Pacific only)

2.4 Conversion planning technical considerations

Although a migration typically does not include logical database redesign, in the conversion phase some amount of reengineering can be considered to take advantage of DB2-specific features.

In this section, we provide a brief discussion of some of the Oracle features commonly found in Oracle environments that you may want to consider reengineering during conversion planning or database design phases. We introduce the DB2 features that provide similar functionality and the resources for learning these DB2 features and functions.

2.4.1 Task scheduling

In an Oracle environment, tasks can be automated using either a native operating system tool or a database tool. The common operating system tool used in UNIX systems is crontab, and in Windows systems is Scheduled Tasks. Oracle 10g introduces Database Scheduler, a collection of programs using the DBMS_SCHEDULER package to provide scheduling functionality. In releases prior to 10g, Oracle has the DBMS_JOBS package to provide similar functionality.

DB2 job scheduling capabilities are implemented using four main components:

► Client Tools - the tools used for administration of scheduled tasks.

► Tools Catalog Database - a repository for tasks and related information about tasks.

► Scheduler - the mechanism that starts the jobs.

► DB2 Administration Server (DAS) - a central point of control for tasks in DB2 instances and databases that controls the Scheduler and the execution of tasks.

The DB2 Task Center is the Client Tool used to schedule and execute unattended jobs. The DB2 Journal is used to view historical information about tasks, database actions and operations, and messages and notifications. These tools can be directly accessed, or can be accessed from within the DB2 Control Center. Task Center can run DB2 command scripts as well as OS command scripts, enable or disable schedules, and send e-mail notification containing information about the execution of tasks to designated contacts.

Just as for Oracle, operating system tools can also be used to schedule tasks in DB2 databases.

There is not a direct path to migrate Oracle scheduled tasks to DB2. In order to achieve the same functionality in a DB2 environment, you must do the following before the DB2 database goes into production:

► Identify the existing scheduled tasks in your environment.

► Analyze what tasks will be necessary in the new DB2 environment and discard unnecessary ones.

► Modify, or adapt, the tasks or scripts to work with DB2.

► Implement the tasks using DB2 Task Scheduler or operating system tool.

► Analyze output logs to verify if the conversion is producing desired results.

For more information on DB2 scheduled tasks, refer to chapter 7 "Using the DB2 administration tools" in *Administration Guide: Implementation,* SC10-4221*.*

2.4.2 Auditing

Oracle 10g provides audit functionality through the use of standard and fine-grained auditing options. It allows for system privilege, object privilege, and SQL statement auditing. Audit trails can be stored in either the operating system's audit system or in an internal database table. LogMiner can be used for analyzing redo log data for suspicious queries.

DB2's audit facility is independent of the database server and operates even when the database instance is stopped. It must be started or stopped explicitly with the db2audit tool. SYSADMIN authority is required to use this tool. The audit facility acts at the instance level, recording both instance and database activities. The audit log file, db2audit.log, is stored in instance's security subdirectory. This audit log file has a proprietary format, and security administrators must extract audit records to a flat file or ASCII delimited files in order to analysis the data.

DB2's audit facility can monitor different categories of database events, and can audit successes, failures or both operations for each event. The number of records generated for a given database operation depends on the number of categories of events recorded. DB2 can implement synchronous auditing, where the event generating the audit record will wait until the record is written to disk, or asynchronous auditing, where the auditing records are buffered and written to disk when they reach a limit specified by the audit_buf_sz database configuration parameter.

Oracle audit events cannot be automatically converted to DB2 audit events. It is necessary to manually implement audit policies in the DB2 database after database conversion.

Before implementing auditing in a DB2 database, it is good practice to set it up in a development server to measure how much overhead auditing tasks will cause and how much storage will be necessary to hold all audit information.

For more information on DB2 audit functionality, refer to chapter 9 "Auditing DB2 database activities" in *Administration Guide: Implementation,* SC10-4221-00.

2.4.3 National Language support

Oracle provides support for single-byte (SBCS), multi-byte (MBCS), and Unicode character sets. At the server side, the character set is defined during the database installation, while at the client side the character set is defined by using the environment variable. If a client character set is different from a server character set, Oracle NET automatically converts the data between the two encoding schemes. Oracle can store Unicode characters either into a CHAR column created with a UTF-8 database or into an NCHAR data type regardless of the database character set.

DB2 provides support for single-byte, multi-byte, and Unicode character sets. On the server side, the code page for a database is specified using CODESET and TERRITORY clauses of the CREATE DATABASE command, as shown in Example 2-1. If not specified, DB2 creates the database using the codepage of the operating system where the database is being created.

Example 2-1   Code set and territory specification during database creation

```
CREATE DATABASE TESTDB9 ON C:
    USING CODESET UTF-8 TERRITORY US
```

At the client side, the code set and territory parameters can be specified using the DB2CODEPAGE and DB2TERRITORY environment variables, as shown in Example 2-2.

Example 2-2   Client code set and territory specification

```
db2set DB2CODEPAGE=1208
db2set DB2TERRITORY=1
```

A DB2 database can only store Unicode data if a Unicode character set has been specified as the database code set, or if the table is created with the CCSID clause. To use the CCSID clause, it is necessary to activate the database configuration parameter ALT_COLLATE. Once set, this parameter cannot be changed or reset. If you plan to use DB2 XML features, the database codeset must be a Unicode character set, such as UTF-8.

Choosing the right code set for the database is critical for a successful database conversion. You must choose a compatible DB2 code set with an existing Oracle character set to avoid problems during data conversion. If the Unicode character set is used in an existing Oracle database, or if there are columns of NCHAR data types, or if XML support is required, the DB2 database code set must be Unicode compatible.

For more information on DB2 National Language Support, refer to *National Language Support Guide and Reference,* SC10-4380-00.

2.4.4 Authentication and authorization

Oracle can authenticate users by four different methods:

► Database authentication - The database performs user authentication and authorization.

► External authentication - The operating system or a network service performs authentication.

► Global authentication and authorization - The user is globally authenticated by an enterprise directory using Secure Sockets Layer (SSL).

► Proxy authentication and authorization - The user is authenticated by a middle tier server.

The authentication method is specified during the creation of the user, and it is possible to have different users with different types of authentication on the same database. Such user information is stored in several Data Dictionary tables. Administrative connections to the database can be made using either operating system authentication or a password file.

DB2 authentication security is based on two security levels: *Authentication and Authorization*.

► Authentication
Authentication is the verification of user name and password identity. DB2 does not directly authenticate users, it relies on a security facility outside the database, often part of the operating system or a separate product, or may not exist at all. DB2 does not keep any login information stored in catalog tables. DB2 provides authentication both at the server side or at the client side, and can use or not use encryption for authentication. Also, it is possible to use Kerberos security protocol for authentication.

There is only one authentication type per instance, which will be used for all databases under its control.

► Authorization
Authorization is used to determine what authenticated users can perform inside the database, and what objects they can access. Authorization can be broken down into two categories: *authorities* and *privileges*.

DB2 authorities

An authority in DB2 is defined as a group in the operation system and granting a specific user this authority simply means that this user is assigned to this group. DB2 provides administrative authorities to allow administrators to perform high-level instance and database management tasks and operations. Authority information is stored in the database configuration manager file. The levels of authorities in DB2 are classified as follows:

► **SYSADM** - Administrative authority. System administrators are given full privileges over the entire DB2 instance. SYSADM cannot be granted with a SQL statement.

► **SYSCTRL** - System control authority. System controllers are given full privileges for managing the system, but are not allowed access to data. SYSCTRL cannot be granted with a SQL statement.

► **SYSMAINT** - System maintenance authority. System supports are given a subset of privileges to manage the system. SYSMAINT cannot be granted with a SQL statement.

► **SYSMON** - System monitor authority. System monitors are given authority required to use the database system monitor. SYSMON cannot be granted with a SQL statement.

- **SECADM** - Security administrator. Security administrators are given authority to perform database security administration. SECADM can be granted with a SQL statement.

- **DBADM** - Administrative authority. Database administrators have control over an individual database. DBADM can be granted with an SQL statement.

- **LOAD** - The LOAD authority is granted on the database level. Users with LOAD authority can load data to a table, Quiesce the table space for the table, perform `Runstats` and `List Tablespaces` commands. To load data to a table, the INSERT privilege on the table is also required. Depending on the load activity, the UPDATE and DELETE privilege on the table may also be needed.

DB2 privileges

DB2 privilege concept is similar to Oracle privilege concept. Database privileges are granted in the database through the SQL command GRANT. Privileges are stored in the system catalog tables within the database There are three types of privileges: ownership, individual, and implicit:

- **Ownership or CONTROL privileges** - In most cases the database user who creates a database object is automatically granted the CONTROL privilege. This privilege permits the user to grant other database users certain privileges on this object. The GRANT privilege can be granted through the GRANT statement.

- **Individual privileges** - A classic example of this type of privileges is the SELECT, INSERT, UPDATE, and DELETE privileges.

- **Implicit privilege** - This is a sub privilege, which is automatically granted to a user when this user is granted a high level privilege.

When converting Oracle database users to DB2, you need to create user IDs in the operating system or in the external product for those users who are authenticated using the database authentication method. If the existing users are authenticated by any method other than database authentication, these user names can be used in the DB2 database. Since DB2 only supports one type of authentication per instance, it is necessary to properly choose one authentication type that fits the access requirements to all databases controlled by the instance.

DB2 MTK does not automatically convert Oracle privileges to DB2. During the conversion planning phase, it is necessary to identify all existing privileges in the Oracle database, and then map these privileges to DB2. The privileges required for data and application conversion should be implemented in the DB2 database first. After all data and application conversions are completed, other users and privileges can be implemented in DB2 to meet your data security requirements.

For more information on DB2 Security, refer to chapter 8 "Controlling database access" in *Administration Guide: Implementation,* SC10-4221.

2.4.5 Data partitioning

Oracle 10g has the following partitioning methods:

► Range partitioning - Data is partitioned based on ranges of column values.

► Hash partitioning - Data is partitioned evenly across a specified number of partitions.

► List partitioning - Data is partitioned based on a list of column values.

► Composite Range-Hash partitioning - Data is partitioned using the Range method and then, within each partition, it is partitioned using the Hash method.

► Composite Range-List partitioning - Data is partitioned using the Range method and then, within each partition, it is partitioned using the List method.

Indexes can also be partitioned, and can be of type Global, which allows the index partitions to be different from the underlying table, or Local, where Oracle creates a separate index for each partition in the table.

DB2 can partition the data in the following ways:

► Database partitioning
  The concept of database partitioning is to spread data among several database partitions stored on different servers. It is enabled using DB2 Database Partitioning Feature (DPF). It divides the data between database partitions using hash distribution keys to balance the workload evenly between all database partitions. The tables are created with a DISTRIBUTED BY clause to identify database partitioning.

► Table partitioning
  Introduced in DB2 9. Data is locally partitioned across multiple storage objects according to values in one or more rows. The tables are created with a PARTITIONED BY clause to identify table partitioning. DB2 table partitioning is similar to Oracle Range partitioning.

► Multidimensional clustering (MDC)
  DB2 groups rows with similar values on multiple keys, or dimensions, in the same table extent. The tables are created with a ORGANIZED BY clause to identify multidimensional clustering.

None of these partitioning methods are mutually exclusive; all of them can be combined to provide a tailored solution for business needs, as illustrated in Figure 2-1.
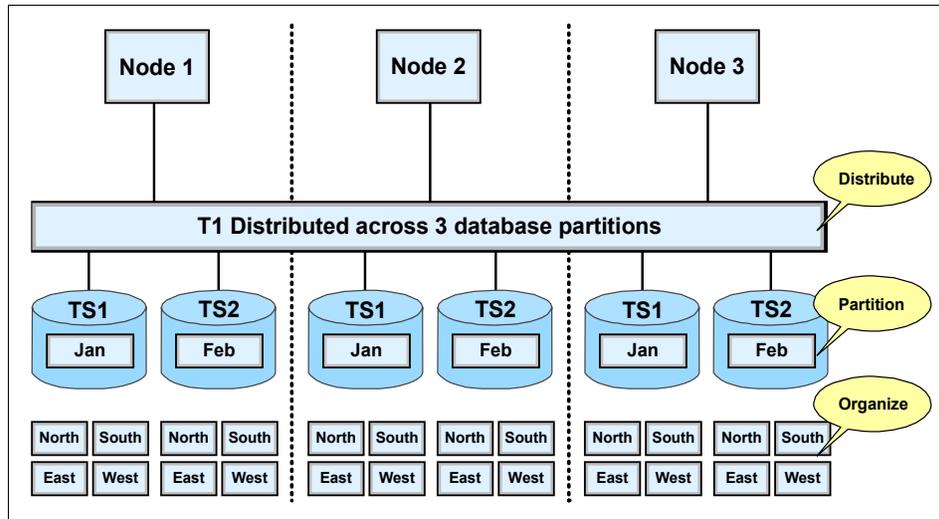
Figure 2-1   Three DB2 data organization schemes

Choosing the right partitioning method plays a major role during the database planning stage. Several factors must be analyzed in order to define what partitioning scheme is right for each situation, such as existing hardware, application and data design, and database workload. Because Oracle and DB2 have completely different structures, there is not a general rule for Oracle partitioning migration to DB2 partitioning. The data should be partitioned for performance reasons, for administrative reasons, or even to expand the capacity of data storage. For each of these situations, a different partitioning scheme approach should be necessary.

While complete coverage of DB2 DPF is beyond the scope of this book, the following Web sites provide resources to learn about DB2 data partitioning features and the best approach to implement each of them:

► http://www.ibm.com/developerworks/db2/library/techarticle/dm-0608mcine rney/

► http://www.ibm.com/developerworks/db2/library/techarticle/dm-0605ahuja 2/

Also, Chapter 6 "Designing partitioned databases" in *Administration Guide: Planning,* SC10-4223.

2.4.6 Oracle External tables

Oracle External tables are mainly used for data movement between Oracle databases. Oracle External tables have only metadata stored inside the

database dictionary tables, and the data itself resides outside the database in a proprietary format operating system files. External tables can be accessed using new ORACLE_DATAPUMP driver for loading and unloading operations. The command CREATE TABLE AS SELECT with the ORGANIZATION EXTERNAL clause is used to create and populate an external table, and after the table has been created and populated, no DML commands will be allowed. In releases prior to 10g, external tables were created using the ORACLE_LOADER driver, allowing only loading operations. This driver is still supported in Oracle 10g. The Oracle Data Pump unloading operation produces a binary, proprietary format file only readable by Data Pump utilities, so it is not possible to use external tables to move data extracted on an Oracle 10g database to previous releases or to heterogeneous databases.

DB2 provides three utilities for data movement:

► EXPORT
  Extracts data from a DB2 database or view to a file, using the delimited ASCII format (DEL), worksheet format (WSF), or PC information exchange format (IXF).

► IMPORT
  Reads data directly from a external file and stores it into a DB2 table. The DB2 IMPORT utility can read non-delimited ASCII format (ASC), DEL, WSF, or IXF format. Import performs SQL INSERTs to load data from external files to a DB2 table and may fire triggers defined on tables being imported to.

► LOAD
  LOAD is another utility used to read data directly from an external file and store into a DB2 table. The LOAD utility is much faster than the IMPORT utility because instead of performing SQL INSERTs to load the data, it writes formatted pages directly into the database. The LOAD utility can read files of format type ASC, DEL, IXF, or CURSOR (a cursor declared against a SELECT or VALUES statement).

In order to implement Oracle external table functionality in the DB2 database, we recommend the EXPORT utility for unloading operations, and the LOAD or IMPORT utility for loading operations.

For more information on DB2 data movement utilities, refer to *Data Movement Utilities Guide and Reference,* SC10-4227.

2.4.7 Oracle bigfile table spaces

Oracle 10g introduces the bigfile table space. A bigfile table space is a table space containing only one very large data file. Its addressing scheme is different from regular table spaces, now called smallfile table spaces. Bigfile and smallfile table spaces can coexist within a database.

Converting a bigfile table space to a DB2 table space is not a concern. The information stored into a bigfile table space can be moved to DB2 table spaces in the same way as smallfile table spaces.

2.4.8 Table space design

A database table space design is directly related to the type of database workload, grouping database objects according to their data access pattern in order to achieve higher performance or easier administration. For example, the following are some commonly used techniques:

► Separate table spaces for data, indexes, and LOB objects.

► Table spaces should have different block sizes according to objects stored in them.

► Read-only objects should be placed within the same table space.

► Table space containers should be spread among several disks or file systems to improve performance.

DB2 provides two storage options: *non-automatic storage* and *automatic storage.* In a non-automatic storage database, you must perform all database storage allocations and decisions; in an automatic storage database DB2 automatically performs it for you.

When a database is created using automatic storage management, you associate *storage paths* within the database using the clause ON in the CREATE DATABASE command. The containers and space management options are automatically chosen by the DB2 database manager. If you do not specify the storage paths, DB2 will use the value of the *dftdbpath* database manager configuration parameter. You can also associate *database paths* to specify location for various control files for the database. If you do not specify it, DB2 will use either the value of the ON clause on the CREATE DATABASE command or the value of the dftdbpath database manager configuration parameter. Example 2-3 shows examples of how to create an Automatic Storage-managed database.

Example 2-3   Automatic Storage-managed database examples

```
CREATE DATABASE DB1
CREATE DATABASE DB2 AUTOMATIC STORAGE YES
CREATE DATABASE DB3 ON /db2/db1/data1 DBPATH ON /db2/db1/control
```

Note: In DB2 9, if you omit the management clause on the CREATE DATABASE command, the database will be managed by Automatic Storage.

A database can only be enabled for automatic storage management during its creation. You cannot enable or disable automatic storage management for a database. Automatic storage management supports single-partition and multi-partition databases.

With automatic storage managed databases, it is not necessary to specify containers for a table space during its creation; the containers will be created in the database storage path. Example 2-4 shows a few examples of creating table spaces.

Example 2-4   Automatic storage management table space creation

```
CREATE TABLESPACE TBS1
CREATE LONG TABLESPACE LONG_TBS
CREATE TEMPORARY TABLESPACE TBSTMP
CREATE USER TEMPORARY TABLESPACE USERTMP
```

Although, the first time, table spaces created with automatic storage management can appear as a new type of table space, DB2 uses System Managed Space (SMS) table spaces for user and temporary table spaces, and Database Managed Space (DMS) table spaces for regular and large table spaces.

SMS and DMS are the two table space management types in DB2. Both types of table spaces have containers or data files associated with them, and they can coexist within a database.

SMS table space

This type of table space stores its containers in the form of operating system directories. Since this type of table space cannot be resized manually, enlarging the underlying file system would then increase the size of the table space. SMS table spaces acquire more space only when needed.

There are few advantages associated with creating SMS table spaces, such as ease of creation and maintenance. The main disadvantage of an SMS table space is that it cannot separate out table indexes and table data into their own table spaces.

DMS table space

The containers associated with a DMS table space are either operating system files or raw devices. A DMS table space can be resized manually with the ALTER TABLESPACE command using the RESIZE option. The database administrator decides the location of containers belonging to the table space and when to add containers. A DMS table space may be defined as regular, large, or temporary.

Table 2-1 shows the differences between DMS and SMS.

Table 2-1   Differences between SMS and DMS table spaces

Table space type	SMS	DMS
Can dynamically increase the number of containers in a table space	No	Yes
Can store index data for a table stored in a separate table space	No	Yes
Can store long data for a table stored in a separate table space	No	Yes
One data partitioned table can span multiple table spaces	Yes	Yes
Space allocated only when needed	Yes	No
Table space can be placed on different disks	Yes	Yes
Extent size can be changed after creation	No	No

There are three categories of table space based on the data stored in them:

► Regular table space - can store regular, index, and long data. Nevertheless, this type of table space is not optimized for long type data.

► Large table space - designed to store long character or LOB type data.

► Temporary table space - designed to store temporary tables. A user cannot define a permanent table in a temporary table space.

Note: Only users with SYSADM or SYSCTRL authority can create table spaces.

When planning for table spaces, you should consider the table space size, type, and the placement on the physical drive. Migration time is a good time to redesign the table spaces of your database if you have been considering it. Oracle data files are similar to the DB2 DMS table space container. You should also consider to take advantage of the DB2 automatic storage manager feature to simplify database storage administration.

For more information on DB2 table space design, refer to chapter 5, "Physical Database Design" in *Administration Guide: Planning,* SC10-4223.

2.4.9 Data encryption

Oracle 10g introduces DBMS_CRYPTO API for data encryption of sensitive data, and keeps the DBMS_OBFUSCATION_TOOLKIT API for backward

compatibility with previous releases. It provides DES, 3DES and AES cryptographic algorithms.

In addition to these APIs, Oracle offers, for an additional license fee, the Oracle Advanced Security option. It provides network encryption, database encryption, and strong authentication for Oracle databases.

The DB2 database manager configuration parameter AUTHENTICATION controls not only the encryption of user IDs and passwords during connection authentication, but also the encryption of communication data transferred between server and client. In order to enable data communication encryption, the database configuration manager AUTHENTICATION parameter can be set as DATA_ENCRYPT, as shown in Example 2-5.

Example 2-5   Enabling data encryption for DB2 communication

```
db2 => UPDATE DATABASE MANAGER CONFIGURATION USING AUTHENTICATION DATA_ENCRYPT
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
```

Note: After the authentication method is changed, it is necessary to restart DB2 instance for the change to take effect.

DB2 also provides built-in functions to enable data encryption: ENCRYPT, which encrypts the data using a password-base encryption method; DECRYPT_BIN and DECRYPT_CHAR, which decrypt the encrypted data; and GETHINT, which returns an encapsulated password hint.

DB2 Migration Tool Kit cannot directly extract data stored in an Oracle-encrypted column. To convert an Oracle-encrypted column to DB2, data must be manually decrypted using Oracle APIs, transferred from the Oracle to the DB2 database using MTK or any other tool, then encrypted in the DB2 database.

For more information on DB2 audit functionality, refer to chapter 9, "Auditing DB2 database activities" in *Administration Guide: Implementation,* SC10-4221.

2.4.10 Disaster recovery solutions

The Oracle data protection and disaster recovery solution is named Data Guard. It is included in Enterprise Edition and creates one or more standby databases transnationally identical to the production (or primary) database to protect data from errors, failures, disasters, and corruptions. These standby databases can be located at a recovery site for disaster recovery purposes. The data is transferred between production and standby databases by applying redo log data generated in a production database into the standby databases. With Data

Guard, it is possible to open the standby databases in read-only mode for reporting operations by temporarily suspending archived log file application on a secondary server. However, this can lead to a higher failover time because it might be necessary to apply pending archived log files after a primary server crash.

Although not designed for disaster recovery, it is possible to use replication products, such as Oracle Streams and Advanced Replication to protect sensitive data by storing a copy of them at a remote site.

The DB2 High Availability Disaster Recovery (HADR) feature provides disaster and recovery capability. HADR is included in all DB2 editions and is used to ensure database availability in the event of either planned or unplanned downtime, providing ultra fast failover with simple deployment and management procedures. HADR replicates data changes from a production (or primary) database to a standby database directly from the primary database log buffer, maintaining a fault tolerant replica. DB2 also provides the Automatic Client Reroute feature that enables client applications to recover from a primary server failure, automatically reconnecting to the standby server and continuing work with minimal interruption.

Figure 2-2 shows the HADR concept.
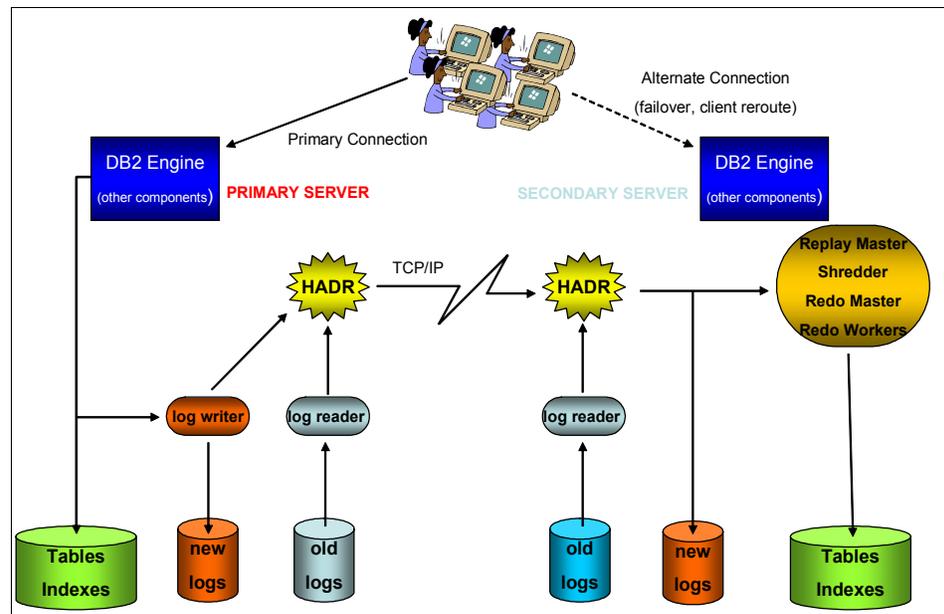
Figure 2-2   HADR concept

For a demonstration of DB2's HADR implementation, refer to:

http://demos.dfw.ibm.com/on_demand/Demo/IBM_Demo_DB2_HADR-Jan05.html

If reading data at the standby server is a higher priority, or if you want to use replication instead of HADR to store sensitive information at a disaster recovery site, IBM Replication Server provides a Q Replication feature. Using the Q Replication bidirectional or peer-to-peer replication feature makes it possible to replicate sensitive data from a production server to a standby server on a remote site. It is possible also to combine Automatic Client Reroute functionality with Q Replication to create a warm standby database.

There is no migration tool to convert Data Guard or replication configurations to a DB2 platform. You must first convert the Oracle database to DB2 and then implement a DB2 disaster recovery solution. DB2 HADR and IBM Replication Server solutions can be adopted for this situation, providing similar or superior disaster recovery protection to the new environment.

For more information about HADR concepts and implementation, refer to chapter 7, "High Availability disaster recovery (HADR)" in *Data Recovery and High Availability Guide and Reference,* SC10-4228.

For more information on Automatic Client Reroute, refer to chapter 1 "Before creating a database" in *Administration Guide: Implementation,* SC10-4221.

For more information about IBM Information Server features, refer to:

http://www-306.ibm.com/software/data/integration/

For more information about IBM Replication Server Q Replication, refer to:

► *WebSphere Information Integrator Q Replication: Fast Track Implementation Scenarios,* SG24-6487*.*

► http://www-128.ibm.com/developerworks/db2/roadmaps/qrepl-roadmap-v8.2. html

2.4.11 Oracle Database Resource Manager

Oracle Database Resource Manager permits database administrators to control the distribution of resources among database users, ensuring that the necessary resources will be available to business critical operations.

DB2 Query Patroller can be used for similar functionality. With Query Patroller you can proactively and dynamically control the flow of queries against the DB2 database, regulating the workload and permitting that small and high-priority queries have preference over heavy ones. Also, Query Patroller permits collection of database usage and statistics information for trend analysis.

For a demonstration of how DB2 Query Patroller can help to control database resource utilization, refer to

http://demos.dfw.ibm.com/on_demand/Demo/IBM_Demo_DB2_Query_Patroller-Jan05.html

For more information on Query Patroller, refer to *Query Patroller Administration and User Guide,* GC10-4241.

2.4.12 Replication considerations

There are several factors to be considered for conversion of an Oracle database involved in a replication topology:

- ▶ **Existing database environment** - Will all databases involved in the replication be converted? Will it be necessary to support heterogeneous replication between Oracle and DB2 databases?
- ▶ **Replication topology** - What replication topology will be converted? Will the replication be unidirectional, where the replicated data is read only at the target table, or will it be multidirectional, where data can be updated in both source and target tables?
- ▶ **Storage requirements** - How much additional space and hardware utilization is necessary to support the new replication topology?

DB2 replication capabilities are provided through the IBM Information Server platform. This platform is a collection of technologies that combines database management systems, Web services, replication, federated systems, and warehousing functions into a common platform. IBM Information Server solution for replication is IBM Replication Server, which can replicate data to or from non-IBM databases.

Heterogeneous access between different databases is called *federation.* The IBM Information Server solution for federated systems is IBM WebSphere Federation Server, with which you can integrate heterogeneous distributed databases, providing a common interface for accessing, replicating, and manipulating data.

If all databases involved in the replication topology are DB2 databases, you will use the IBM Replication Server features only. However, if it is necessary to access any heterogeneous database, such as Oracle, SQL Server, My SQL, etc., you will need to integrate the WebSphere Federation Server and IBM Replication Server features to design the topology.

To convert an Oracle database that is part of a replication topology to DB2, and configure replication between Oracle and DB2 databases, you should follow these steps:

1. Stop Oracle replication before database conversion to DB2.

2. Convert the Oracle database to DB2.

3. Install and configure the IBM Information Server tools (IBM WebSphere Federation Server and IBM Replication Server).

4. Install Oracle client tools into IBM Information Server server machine.

5. Configure replication between Oracle and DB2 through IBM Information Server products.

While complete coverage of IBM Replication Server features is beyond the scope of this book, the following Web sites provide a quick start to learn about these features, and the best approach to implement them:

http://www-306.ibm.com/software/data/integration/
http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com
.ibm.websphere.ii.product.overview.doc/dochome/iiypohom_wiihome.html
http://www.ibm.com/developerworks/db2/library/techarticle/dm-0505burner
/index.html

2.4.13 Data Warehouse considerations

A conversion of an Oracle Data Warehouse (DW) database to DB2 should be treated in the same way as an OLTP conversion. As in an OLTP environment, some topics must be considered before a DW migration:

► Will all Data Warehouse data be converted to DB2 in a single moment, or will it be broken into smaller pieces of data and migrated separately?

► How much time will be necessary to convert the data? How will business be affected during the conversion process?

► Are the existing hardware resources adequate for Data Warehouse needs? Can the existing storage hardware support Oracle and DB2 data, or will it be necessary to use additional storage hardware during the conversion?

► Will both Data Warehouse and base OLTP databases be migrated to DB2 at the same time\? Will it be necessary to federate, or replicate, data between heterogeneous databases?

► How will the extract, load and transform processes (ETL) be converted to DB2? Are the existing ETL tools compatible with the new environment, or will it be necessary to migrate these tools during the database conversion project?

► Is partitioning used? If yes, how should the existing partitioning scheme be converted to the DB2 scheme?

Due to the fact that Data Warehouse databases usually store terabyte of data, the physical storage requisites must be carefully designed in order to achieve high performance levels and easier administration. DB2 Data Partitioning Feature (DPF) is commonly used in DW databases. Federation of servers is also commonly used, and can be performed in DB2 databases using the IBM Information Server product WebSphere Federation Server.

Besides DPF and WebSphere Federation Server, DB2 provides a scalable, reliable, and robust suite of Data Warehouse tools to help you build, load, and manage your Data Warehouse environment. DB2 provides tools to easily design and implement Cubes, OLAP, data mining analysis, reports, and ETL in a DW environment.

For more information on DB2 Data Warehouse features, refer to:

http://www.ibm.com/software/data/db2/dwe/features.html

3

MTK

IBM Migration Toolkit (MTK) is a free tool designed to simplify and improve migration to DB2 for Linux, UNIX, and Windows (DB2) from other RDBMS. With MTK, database objects such as tables, views, data types, stored procedures, triggers, etc. can be automatically converted into equivalent DB2 database objects. MTK provides database administrators (DBAs) and application programmers with the tools needed to automate previously inefficient and costly migration tasks. With MTK, it is possible to reduce downtime, eliminate human error, and cut back on person-hours and other resources and tasks that are normally associated with database migration.

In this chapter we introduce some basic information regarding the IBM Migration Toolkit in these topics:

- ► MTK overview
- ► MTK planning
- ► MTK setup

3.1 MTK overview

The IBM Migration Toolkit (MTK) (Figure 3-1) is available free of charge from IBM at the following Web site:
http://www.ibm.com/software/data/db2/migration/mtk

MTK was developed by the IBM Silicon Valley Laboratory in San Jose, California with assistance and contributions from the Watson Laboratory in Hawthorne, NY. MTK is now being maintained and developed by the IBM team in Lenexa, KS. Some driving factors in creating MTK were:

► The need to develop a tool that is closely linked, and kept pace, with the technical development of DB2 and the databases with which it interacts.

► The need to address "real world" migration concerns. Because of IBM's significant experience in this area, the developers created a tool that meets the requirements of IBM migration teams while addressing the significant issues involved in customer migrations.

► The need for a tool that would convert as much, and as accurately, as possible.

► The need for a tool that would be available, free of charge, to those interested in doing a migration.
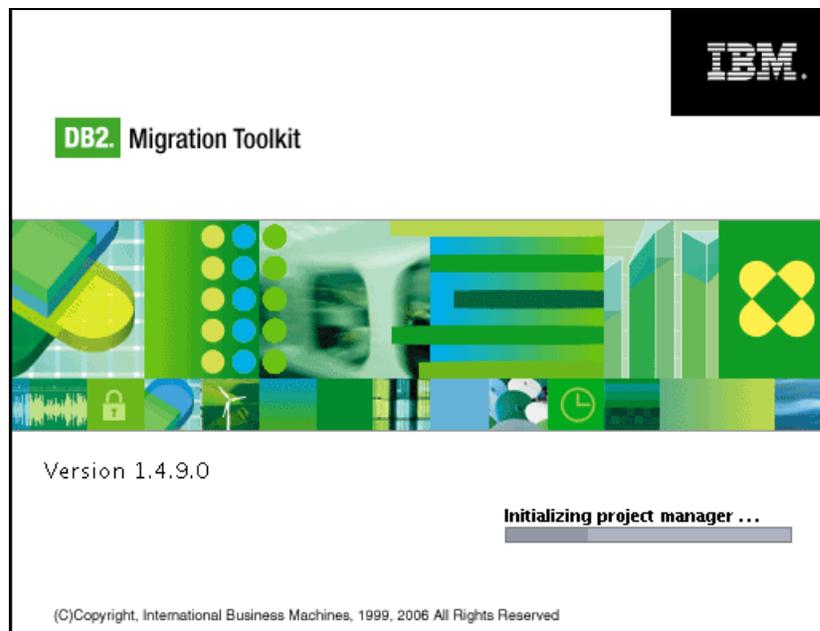
Figure 3-1   The IBM Migration Toolkit

3.1.1 MTK facts

In this section, we introduce MTK versions and features.

MTK versions

The latest version of MTK (as of the publication date of this document) is V1.4.9.0. This is the *recommended* version for all conversions from Oracle, Microsoft SQL Server, Sybase, and IDS sources at the time of writing. Since the MTK development team releases fixes and updates to the product at several times during the year, check the following Web site to download the latest version of MTK:

http://www-306.ibm.com/software/data/db2/migration/mtk/

> **Important:**
> In addition to the recommended version of MTK, there is an MTK *prototype*. This prototype, version 2.0.1.1 has been extended to support MySQL versions 4 and 5 as the source DBMS, and DB2 and IDS as the target DBMS. The emphasis of this prototype is to provide MySQL support; support for other source DBMS in this prototype is *not recommended*, as the support in MTK 1.4.x is more current for other sources. The 2.0.1.1 release contains a fair degree of migration support for MySQL, but is not nearly as complete at the time of the publication of this book. See the release notes for that version to obtain detailed information about the SQL constructs supported.

As of the date of this publication, the following operating systems, conversion sources, and conversion targets are supported by MTK V1.4.9.0:

▶ Supported operating systems

 – Windows 2000 and Windows XP
 – Linux RHEL 3
 – AIX 5L™ 5.2
 – Sun Solaris 2.9/9
 – HP HP-UX B.11.11

▶ Supported databases (as conversion sources)

 – Oracle 8i, 9i, 10g
 – Microsoft SQL Server, versions 7, 2000, and 2005
 – Sybase ASE, Versions 11,12, 12.5, and 15
 – Sybase SQL Anywhere version 9
 – Informix Dynamic Server 7.3, and 9

▶ Supported versions of DB2 (as conversion targets)

 – DB2 for Linux, UNIX and Windows V8.1 (Fix pack 3 and later), V8.2, and 9
 – DB2 for Linux, UNIX and Windows i5/OS®, v5R2, V5R3, V5R4

– DB2 for z/OS Version 8 and later

3.1.2 MTK features

MTK converts the following Oracle source database constructs into equivalent DB2 database objects:

► Data types
► Tables
► Columns
► Views
► Indexes
► Constraints
► Packages
► Stored procedures
► Functions
► Triggers
► Sequences

MTK enables the following tasks:

► Obtaining source database metadata (DDL) by EXTRACTING information from the source database system catalogs through JDBC or ODBC

► Obtaining source database metadata (DDL) by IMPORTING DDL scripts created by SQL*Plus or third-party tools

► Automating the conversion of database object definitions, including stored procedures, triggers, packages, tables, views, indexes, and sequences

► Deploying SQL and Java compatibility functions that permit the converted code to "behave" functionally similar to the source code

► "On the fly" conversion of PL/SQL statements using the SQL Translator tool. This tool is also effective as a DB2 SQL PL learning aid for PL/SQL developers.

► Viewing conversion information and messages

► Deployment of the converted objects into a new or existing DB2 database

► Generating and running data movement (unload/load) scripts or performing the data movement online

► Tracking the status of object conversions and data movement, including error messages, error location, and DDL change reports using the detailed migration log file and report

3.1.3 MTK GUI interface

The MTK GUI interface (Figure 3-2 on page 70) consists of five tabs, each of which represents a specific task in the conversion process. The tabs are organized from left to right as follows:

► Specify Source
► Convert
► Refine
► Generate Data Transfer Scripts
► Deploy to Target

The menu bar contains Application, Project, Tools, and Help:

► Application - When **Application** → **User Preferences** is selected from the menu bar, such preferences as MTK environment, project, and editor can be set.

► Project - When **Project** is selected from the menu bar, a new project can be created; an existing project can be opened, closed, modified, saved, dropped, backed up or restored; or an SQL source file can be imported.

► Tools - When **Tools** is selected from the menu bar, the SQL Translator can be launched or the Migration reports, Changes Report, or the log file can be examined.

► Help - When **Help** is selected from the menu bar, Help Content or About... can be viewed.
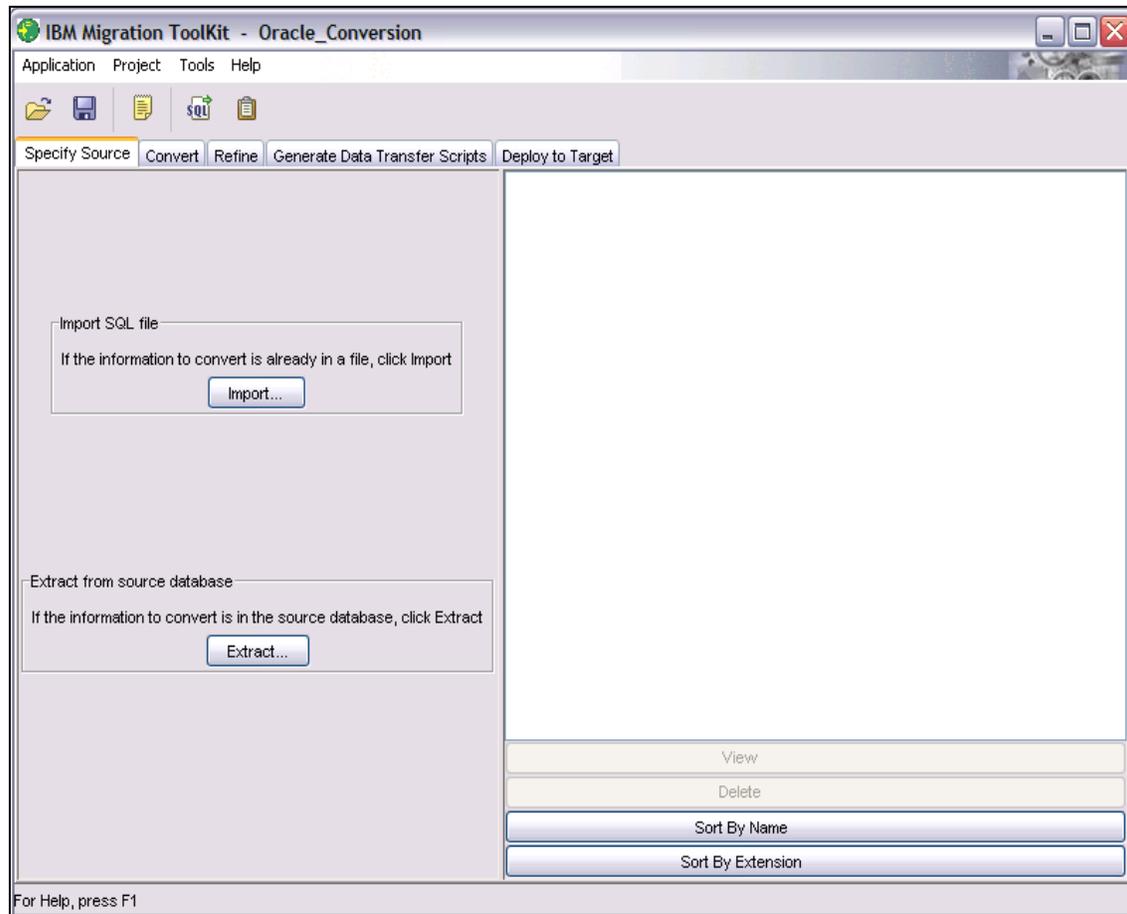
Figure 3-2   MTK GUI interface

3.1.4 Migration tasks

The five tabs in the initial pane of the MTK user interface represent the five phases of the MTK migration process. These are:

► Task 1: Specify source

The SPECIFY SOURCE task (Figure 3-3) focuses on Extracting or Importing database metadata (DDL) into the tool. The database objects defined in this DDL will then be used as the source code for conversion to DB2 equivalent objects. When **Extract...** is selected, a connection to the source database through ODBC or JDBC is required. Once the ODBC/JDBC connection is established, MTK will 'read' the system catalogs of the source database and extract the definitions for use in the conversion process. If **Import...** is

selected, an existing file, or files, which contain database object DDL must exist. The Import task copies the existing DDL file or files from the file system into the MTK project directory for use in the database structure conversion process.
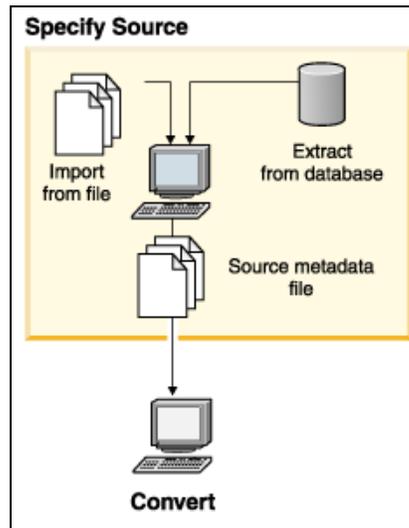
Figure 3-3   Specify source

Note: If the source DDL for the MTK project is obtained through the Import option, the ability of MTK to perform data movement is limited.

► Task 2: Convert

During the CONVERT task (Figure 3-4 on page 72) several *optional* tasks may be completed before the actual conversion of the source code. These are:

– Selecting format options for the converted code. Examples of options are: including the source code as comments in the converted code; including DROP before create object statements, among others.

– Making changes to the default mapping between a source data type and its target DB2 data type.

Once the optional tasks are completed, **Convert** may be selected to perform the conversion of the DDL statements into the corresponding DB2 DDL.

Each conversion generates two files:

– .db2 - this file contains all of the source code converted to DB2 target code.

– .rpt - this file can be opened and viewed from this pane. It is best, however, to examine it during the REFINE task, which succeeds the Convert task.
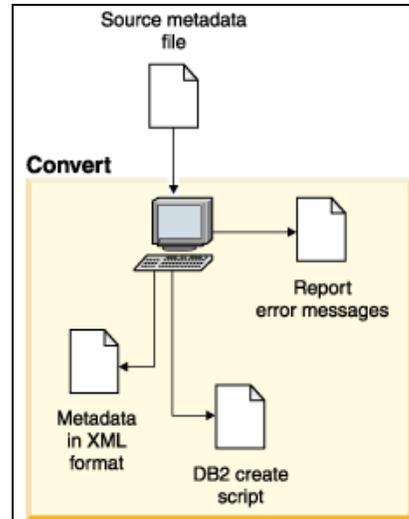
Figure 3-4   Overview of the Convert process

► Task 3: Refine

During the REFINE task (Figure 3-5 on page 73) the following are possible:

– The results of the conversion may be examined.
– Various types of messages generated by the tool may be viewed and, if necessary, specific changes may be made to the converted DDL.

Note: If any changes are made to the *converted* DDL, the Convert step must be re-run in order to apply the changes.

Other tools such as the SQL Translator, Log, and Reports can be used to assist in the Refine task. Once the Refine process has been completed, it is possible to move to the "Generate data transfer scripts" step to prepare the data transfer scripts, or to the "Deploy to Target" step to execute the DB2 DDL statements.
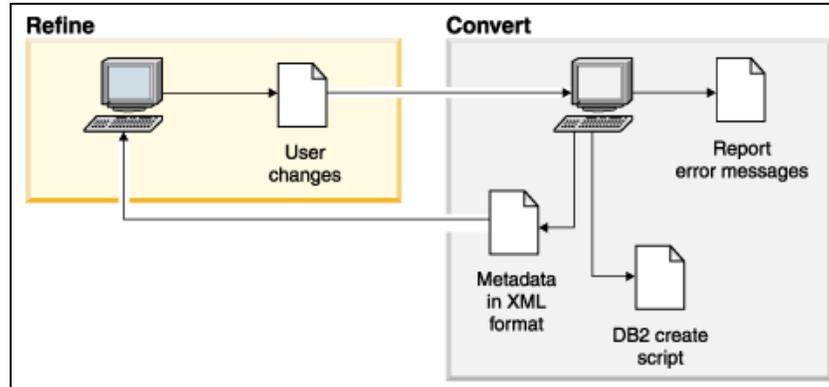
Figure 3-5   Refine

► Task 4: Generate data transfer scripts

In the GENERATE DATA TRANSFER task (Figure 3-6), scripts are generated that will be used to:

– Unload data from the source environment
– Load or Import data into DB2

Before creating the scripts, some advanced options may be selected that will affect how the IMPORT or LOAD utility operates. This will allow additional refinement of the Load or Import specifications that may correspond with the requirements of specific data and environments.
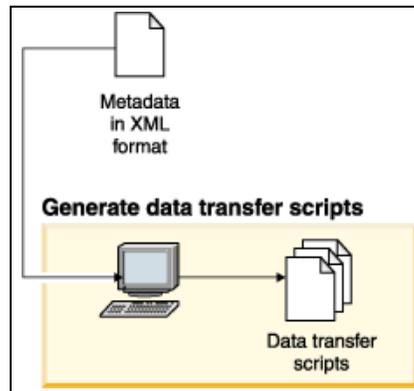
Figure 3-6   Overview of the Generate Data Transfer script task

► Task 5: Deploy to Target

The Deploy to Target task (Figure 3-7) is used to install database objects and Import/Load data into the target DB2 database. In this task, it is possible to:

– Deploy the converted objects into a previously existing database *or* create a new database into which the objects may be deployed.
– Execute the DDL to create the database objects.
– Extract data from the source database.
– Load/Import the source data into the target DB2 tables or choose any combination of the above three.
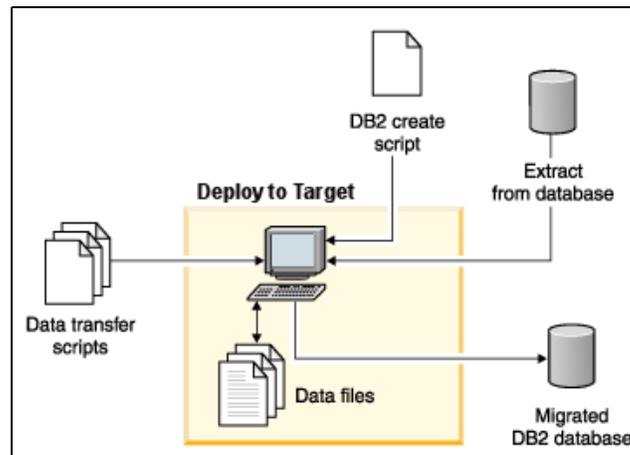
Figure 3-7   Deploy to Target

An overview of all the tasks in the MTK conversion process is shown in Figure 3-8.
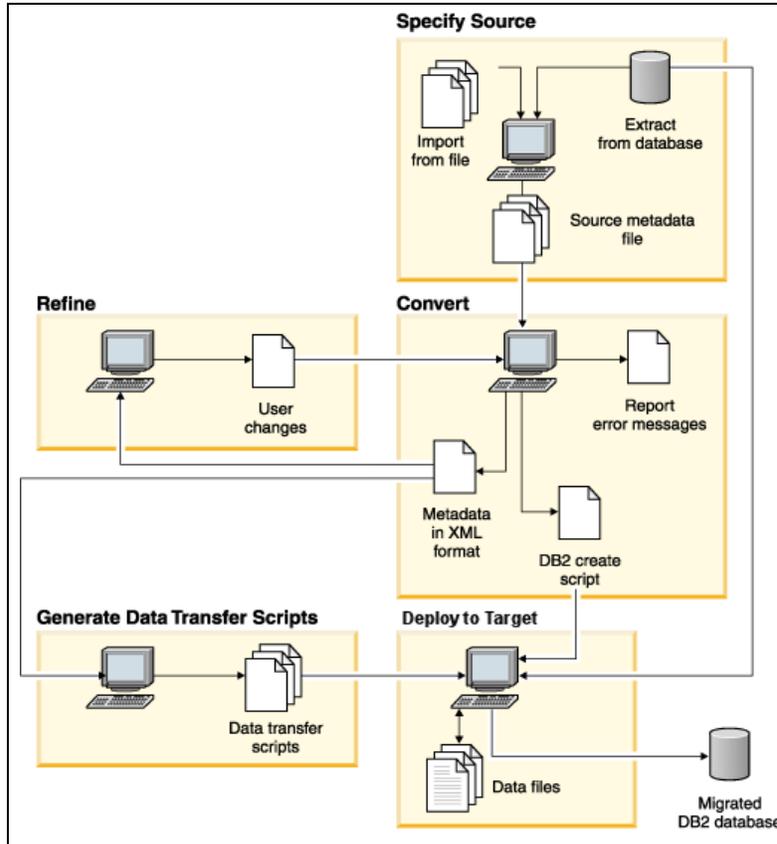
Figure 3-8 MTK conversion tasks overview

3.1.5 The MTK SQL Translator

The MTK SQL Translator (Figure 3-9 on page 76) enables "on-the-fly" conversion of individual statements, a series of statements, or stored procedures. The translator requires that all of the dependent objects for the SQL that you wish to convert are available to MTK. This may be accomplished in either of two ways:

► The current project already contains all of the *converted* objects on which the desired SQL depends (tables, views, etc.).

► The objects on which the SQL statement depends *will be created* in the SQL Translator window by placing them before the SQL that is to be converted.
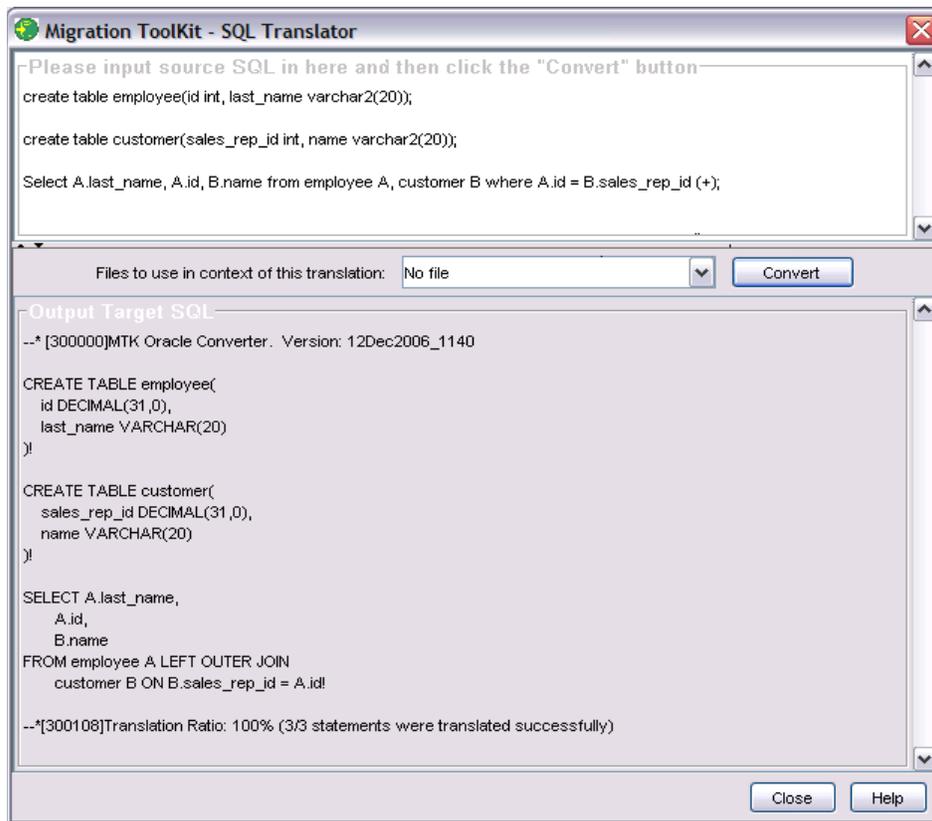
Figure 3-9   The MTK SQL Translator

3.2 MTK planning

MTK runs on a variety of operating system platforms including AIX, Linux, Sun Solaris, HP, and Windows. Before installing MTK, verify the hardware and software requirements provided in this section.

3.2.1 Operating system and version requirements

MTK runs on the following operating systems and versions:

- ▶ Windows 2000 and Windows XP
- ▶ Linux RHEL 3
- ▶ AIX 5L 5.2
- ▶ Sun Solaris 2.9/9
- ▶ HP HP-UX B.11.11

3.2.2 MTK hardware requirements

The hardware requirements for installation of MTK on all platforms are:

- ► 50 MB of disk space for installation
- ► 5 MB per project plus space to unload data
- ► 1 GB of memory is recommended (more memory is required if large source files will be converted).

3.2.3 MTK software requirements

Installation of MTK is supported on various platforms. The following information outlines the software requirements for installation of MTK for these platforms:

Windows

In earlier versions of MTK for the Windows environment a Java Runtime Environment (JRE™) was included as part of the installation package. Currently, the install of the MTK product no longer includes a JRE. As a result, the following is now a requirement:

Java Runtime Environment 1.4.2 *or greater* installed and accessible through the PATH environment variable.

Note: To deploy SQL stored procedures to DB2 Version 8.1 or earlier, Microsoft Visual C++® version 5 or later is required to compile the procedures. DB2 Version 8.2 and later and DB2 9 do not require a compiler.

UNIX and Linux

- ► Java(TM) Runtime Environment 1.4.2 *or greater* installed and accessible through the $PATH environment variable.
- ► For Linux, increase the message queue number to at least 128:

  ```
  sysctl -w kernel.msgmni=128
  ```

- ► To view the HTML reports that MTK generates, include the browser directory in the $PATH variable. If the browser cannot be found, MTK will launch an internal JAVA web browser, which can display HTML files, but does not handle frames or format the tables well.
- ► When extracting DDL from a data source using ODBC or Java, configure the client connection.

3.2.4 MTK requirements for data extraction

The data extraction requirements vary depending on the source RDBMS from which the data is to be extracted.

General data extraction requirements

The following are required for extracting data from a database in any RDBMS:

► When migrating to a DB2 database on UNIX and Linux platforms, use JDBC to connect to the source database.

On UNIX and Linux platforms, MTK does *not* support connecting to the source using ODBC.

► When migrating to a DB2 database on the Windows platform, you can use ODBC *or* JDBC to connect to the source database.

> **Important:** MTK ODBC extraction support will be discontinued on all platforms in subsequent MTK releases.

Oracle data extraction requirements

To extract data from an Oracle database, the following are required:

► To connect to an Oracle database, either of the following JDBC drivers can be used:

– ojdbc14.jar
– classes12.zip

> **Note:** To install ojdbc14.jar, refer to the installation instructions at:
>
> http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc_10201.html

► Edit the MTKMain.bat (Windows) or the MTKMain.sh (UNIX and Linux) file to add the path to *either* the ojdbc14.jar or the classes12.zip file. Example 3-1 shows the MTKMain.sh file before and after the necessary adjustments have been made.

Example 3-1   Editing the MTKMain.sh file to include the path to the ojdbc14.jar file

```
This is the unedited MTKMain.sh file:

#!/bin/sh
java -classpath
".:antlr.jar:common.jar:mtk.jar:ifxjdbc.jar:ifxtools.jar:ifxlang.jar:if
```

```
xlsupp.jar:jt400.jar:jlpex11.jar:help.jar:jhall.jar:xmistore.jar:cwm.ja
r:orainfxtUDFs.jar:$CLASSPATH" com.ibm.mtk.MTKMain $1 $2 $3 $4
```

This is the edited **MTKMain.sh** file with the path to the ojdbc14.jar
file included:

```
#!/bin/sh
java -classpath
".:antlr.jar:common.jar:mtk.jar:ifxjdbc.jar:ifxtools.jar:ifxlang.jar:if
xlsupp.jar:jt400.jar:jlpex11.jar:help.jar:jhall.jar:xmistore.jar:cwm.ja
r:orainfxtUDFs.jar:<PATH_TO_OJDBC14_JAR_FILE>/ojdbc14.jar:$CLASSPATH"
com.ibm.mtk.MTKMain $1 $2 $3 $4
```

► Include the following library in the $CLASSPATH:

   ${ORACLE_HOME}/jdbc/lib/classes12.zip

► Depending on your OS, include the following entries in the
   $LD_LIBRARY_PATH (Linux and Solaris), the $LIBPATH (AIX), or
   $SHLIB_PATH (HP-UX):

   – ${ORACLE_HOME}/lib
   – ${ORACLE_HOME}/lib32

> **Tip:** When migrating from Oracle databases, run statistics on the
> SYS.DEPENDENCY$, SYS.OBJ$, and SYS.USER$ tables before
> extracting. You can use the Oracle DBA Studio or the DBMS_STAT
> package to run statistics.

Microsoft SQL Server data extraction requirements

To extract data from a Microsoft SQL Server database, the following are
required:

► MTK supports connections for the JDBC Type 2 and Type 4 drivers. Either of
   the following Microsoft SQL Server drivers can be used:

   – SQL Server 2000 Driver for JDBC
   – Microsoft SQL Server 2005 JDBC Driver 1.0

> **Note:** These drivers are available for download at:
>
>   http://msdn2.microsoft.com/en-us/data/aa937724.aspx
>
> Refer to the accompanying instructions in the download to install the driver.

► Edit the MTKMain.bat (Windows) file to add the path to *either* the sqljdbc.jar
   (MS SQL 2005) or, in the case of MS SQL 2000, the msbase.jar, msutil.jar,

and the mssqlserver.jar files. Example 3-2 shows the MTKMain.bat file before and after the necessary adjustments have been made.

Example 3-2   Editing the MTKMain.bat file to include the path to the sqljdbc.jar file

```
This is the unedited MTKMain.bat file:

@echo off
java -classpath
".;antlr.jar;common.jar;mtk.jar;ifxjdbc.jar;ifxtools.jar;ifxlang.jar;if
xlsupp.jar;jt400.jar;jlpex11.jar;help.jar;jhall.jar;xmistore.jar;cwm.ja
r;orainfxtUDFs.jar;%CLASSPATH%" com.ibm.mtk.MTKMain %1 %2 %3 %4

This is the edited MTKMain.bat file with the path to the sqljdbc.jar
file included:

@echo off
java -classpath
".;antlr.jar;common.jar;mtk.jar;ifxjdbc.jar;ifxtools.jar;ifxlang.jar;if
xlsupp.jar;jt400.jar;jlpex11.jar;help.jar;jhall.jar;xmistore.jar;cwm.ja
r;orainfxtUDFs.jar;<PATH_TO_sqljdbc_JAR_FILE>\sqljdbc.jar;%CLASSPATH%"
com.ibm.mtk.MTKMain %1 %2 %3 %4
```

Sybase and Sybase SQL Anywhere data extraction requirements

To extract data from a database in Sybase or Sybase SQL Anywhere, the following are required:

► JConnect 5_2 *or later* must be installed and accessible through the system $PATH and $CLASSPATH.

► Edit the MTKMain.bat (Windows) or MTKMain.sh (UNIX and Linux) file(s) to add the path to *either* the jconn2.jar or the jconn3.jar files. Example 3-3 shows the MTKMain.bat file before, and after, the necessary adjustments have been made.

Example 3-3   Editing the MTKMain.bat file to include the required path

```
This is the unedited MTKMain.bat file:

@echo off
java -classpath
".;antlr.jar;common.jar;mtk.jar;ifxjdbc.jar;ifxtools.jar;ifxlang.jar;if
xlsupp.jar;jt400.jar;jlpex11.jar;help.jar;jhall.jar;xmistore.jar;cwm.ja
r;orainfxtUDFs.jar;%CLASSPATH%" com.ibm.mtk.MTKMain %1 %2 %3 %4
```

This is the *edited* **MTKMain.bat** file with the path to the sqljdbc.jar
file included:

```
@echo off
java -classpath
".;antlr.jar;common.jar;mtk.jar;ifxjdbc.jar;ifxtools.jar;ifxlang.jar;if
xlsupp.jar;jt400.jar;jlpex11.jar;help.jar;jhall.jar;xmistore.jar;cwm.ja
r;orainfxtUDFs.jar;<PATH_TO_jconn2_or_jconn3_JAR_FILE>\jconnX.jar;%CLAS
SPATH%" com.ibm.mtk.MTKMain %1 %2 %3 %4
```

> **Important:** The client code does not support the Sybase LC_ALL system
> variable. If the variable has been defined, *remove it* before running the Sybase
> client.

3.2.5 Where to install MTK

MTK can be installed on the source database server, target database server, or
at a client on which connectivity to both source and target database server has
been established. Deciding where to install MTK depends on the data to be
migrated.

MTK places the extracted data on the server where the MTK is installed. Hence,
there is a performance advantage to installing MTK where the target database
resides, since it is faster to load data locally than across a network. Also, DB2
requires that LOB data files reside on the local machine. MTK, therefore, cannot
load data into LOB columns unless MTK is running on the target database
server.

For example, if the Oracle database is located on an AIX system and the DB2
target will be deployed onto a Linux system, and there is no data to be migrated
to LOB columns, MTK should be installed on the local AIX system.

3.3 MTK installation

MTK installation is simple and requires minimum preparation. This section
provides MTK installation procedures and the preparation tasks for Windows and
UNIX and Linux platforms.

3.3.1 Windows installation

To install MTK on Windows:

1. Ensure that you have the necessary hardware and software requirements, as outlined in the sections "MTK hardware requirements" and "MTK software requirements".

2. Download MTK from the MTK Download page into any directory.

3. Unzip and extract the package contents.

4. Run the InstallShield wizard and follow the instructions.

   – The installation will default to the C:\MTK directory.

5. Add the path for the JRE to the MTKMain.bat file, as follows:

   a. Go to the directory where MTK is installed.

   b. Open the MTKMain.bat file.

   c. Add PATH=\bin;%PATH.

6. To launch the MTK, select:

   Start → Programs → IBM Migration Toolkit 1.4.0 → Toolkit

3.3.2 UNIX and Linux Installation

To install MTK on UNIX and Linux platforms:

1. Ensure that you have the necessary hardware and software requirements, as outlined in the sections "MTK hardware requirements" and "MTK software requirements".

2. Log in with the user ID under which MTK will be installed.

> **Important:** Do *not* install MTK as root. Install MTK with a user ID that has authority in the db2admin group.

3. Download or copy MTK into a newly created directory.

4. Use a standard tar utility to untar and extract Mtk.tar.gz files. Execute the following command to unpack this file into the current MTK directory or directories specified:

```
tar -xzf mtk.tar.gz
```

5. Verify Java access and that Java is, at least, at level 1.4.2 or greater. Execute the following from the shell to verify the installed version:

```
java -version
```

6. Verify that the DB2 $INSTHOME environment variable is set to your DB2 instance directory and that it is properly exported when you start a new environment. For example, in the Korn shell type this command:

```
echo $INSTHOME
```

The result should be equivalent to:

```
/home/db2inst1
```

7. Launch the MTK from the directory in which it was installed by typing:

```
MTKMain.sh
```

or

```
./MTKMain
```

> **Note:** MTK uses the Korn shell for deployment. If your environment has the Korn shell installed in an unusual place, make a symbolic link to it from /usr/bin/ksh, which is where MTK expects it to be located as follows:
>
> ```
>    ln -s `which ksh` /usr/bin/ksh
> ```
>
> **Important:** Do *not* attempt to install and run MTK in a shared environment (for example, /usr/local/mtk). If multiple users will run MTK on the same system, they should install and run their own copies, using projects and files local to their home directory. Sharing projects and logs can result in conflicts and overwritten files.

3.3.3 Verifying the environment for creating MTK Java UDFs

During the *Deploy to Target* stage of a conversion, MTK creates Java and SQL user-defined functions (UDFs). In order to ensure a successful conversion, it is highly recommended that the ability to create and execute a Java UDF—*outside of MTK*—be tested in the target environment. Before attempting to build and execute a Java UDF, refer to the following IBM Redbooks publications for instructions on setting up the application development environment and building Java routines:

► *Getting Started with Database Application Development*, SC10-4252
► *Developing Java Applications*, SC10-4233

The process of building and executing DB2 Java UDFs, outside of MTK, is demonstrated in the following steps. Example 3-4 contains the source code for the process.

1. Compile the Java modules TestUdf.java (Example 3-4) and TestUdfCli.java **(**Example 3-5**)** by executing the following commands:

```
> javac TestUdf.java
```

```
> javac TestUdfCli.java
```

Example 3-4   Source code for TestUdf.java

```
// Source code for TestUdf.java
import COM.ibm.db2.app.*;
import COM.ibm.db2.jdbc.app.*;

///////
// Java user-defined functions are in this class
///////
class TestUdf extends UDF
{
  // Find-the-vowel vowel example.
  // Return position of first vowel, or signal SQL error.
  public void findvwl (String a, int result) throws Exception
    {
      for (int i = 0; i < a.length (); i++)
        {
  char x = a.charAt (i);
   char y = Character.toUpperCase (x);
    if (y == 'A' || y == 'E' || y == 'I' ||
             y == 'O' || y == 'U' || y == 'Y')
      {
        set (2, i + 1);// SQL indexing begins at 1
        return;
      }
        }

      // return failure message
      setSQLstate ("38700");
      setSQLmessage ("findvwl: No Vowel");
  }

} // end of source for TestUdf.java
```

Example 3-5   Source code for TestUdfCli.java

```
// Source code for TestUdfCli.java

import COM.ibm.db2.jdbc.app.*;
import java.sql.*;

public class TestUdfCli {
  static boolean testDL = false;
```

```java
  public static void main (String argv[]) {
    Connection con = null;
    // URL is jdbc:db2:dbname
    // ** EDIT dbname to YOUR Database Name if you are NOT connecting
    // the SAMPLE database.
    String url = "jdbc:db2:sample";

    try {
      System.out.println ("Java User-defined Function Sample");

      // Load DB2 JDBC application driver
      Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance ();

      // Connect to database
      if (argv.length == 0) {
        // connect with default id/password
        con = DriverManager.getConnection(url);
      }
      else if (argv.length == 2) {
        String userid = argv[0];
        String passwd = argv[1];
        // connect with user-provided username and password
        con = DriverManager.getConnection(url, userid, passwd);
      }
      else {
        System.out.println("\nUsage: java TestUdfCli [username
password]\n");
        System.exit(0);
      }

      System.out.println ("Connected to the database");

      // Execute DECLARE FUNCTION calls to register Java UDFs
      System.out.println ("Declaring the Java UDFs");
      declareUDFs (con, false);// use NOT FENCED mode

    }
    catch (Throwable x)
      {
        try { con.close (); } catch (Throwable y) {}
        System.err.println ("Aborted due to exception.");
        x.printStackTrace ();
      }
  }
```

```java
    // Run DECLARE FUNCTION to register JAVA UDFs
    static void
    declareUDFs (Connection con,
                 boolean fenced) throws Exception
    {
      declareUDF (con, "findvwl", "varchar(500)", "int",
        fenced, false, "TestUdf!findvwl");



    }



    // Build and run just one CREATE FUNCTION statement.
    static void
    declareUDF (Connection con,
                String name,
                String argumentTypes,
                String returnType,
                boolean fenced,
                boolean scratchpad,
                String externalName) throws Exception
    {
      Statement s = con.createStatement ();
      String sqlClean = "drop function " + name;
      try { s.executeUpdate (sqlClean); } catch (SQLException x) { }
      String sql = "create function " + name +
        " (" + argumentTypes + ") " +
        " returns " + returnType +
        (fenced ? " fenced" : " not fenced") +
        (scratchpad ? " scratchpad" : "") +
        " variant no sql no external action " +
        " language java parameter style db2general " +
        " final call disallow parallel dbinfo " +
        " external name '" + externalName + "'";
      s.executeUpdate (sql);
      System.out.println ("Registered Java UDF " + name);
      System.out.println(sql);
      s.close ();
    }

    } //End of source code for TestUdfCli.java
```

2. Execute the following command:

```
> java TestUdfCli
```

This will connect to the SAMPLE database and register the UDF. If the connection string designates a database other than SAMPLE, then the UDF will be registered to that database.

3. Move the TestUdf.class that is created by a successful compile to the SQLLIB/function directory.

4. Execute the following to connect to the database:

```
> db2 connect to Sample
```

(or *your* database)

5. When the UDF is executed with the following command, the results following it should be seen:

```
> db2 "values findvwl('qwerty')"
   1
  -----------
            3
  1 record(s) selected.
```

Porting with MTK

In this chapter we discuss and demonstrate the conversion of database objects and the extraction and loading of data into a DB2 database using the IBM Migration Toolkit (MTK).

The following items are discussed:

► Preparation for porting
► Running MTK
► Extracting or importing metadata into MTK
► The Convert task
► The Refine task
► The Generate Data Transfer Scripts task
► Deploy to Target considerations
► Next steps
► Converting the remaining objects
► Deployment of stored procedures, functions, packages, and triggers
► MTK conversion conclusion

4.1 Preparation for porting

Before beginning a conversion using MTK, the DB2 target environment must be prepared. This section outlines some documentation that is useful in determining prerequisites and requirements that should be in place for the operating system and the database—before installing MTK.

4.2 Overview of available documentation

Regardless of the platform on which DB2 9 will be installed, it is imperative that hardware and software requirements be considered, and satisfied, before beginning the installation. The following list is an *overview* of topics that should be investigated and prepared before the installation of DB2:

► Database installation
► Instance and database creation
► Table space planning
► Security consideration
► Creating DB2 database users

For detailed information about these and other relevant topics, consult the following documents:

► *Getting started with DB2 installation and administration on Linux and Windows*, GC10-4247.

► *Quick Beginnings for DB2 Servers*; GC10-4246

► *Quick Beginnings for DB2 Clients*; GC10-4242

> **Important:** For the most recent information on software requirements, refer to:
>
>   http://www.ibm.com/software/data/db2/udb/sysreqs.html

Multiple partition installation

The DB2 9 manual *Quick Beginnings for DB2 Servers,* GC10-4246 contains the procedures for setting up multi-partitioned databases. The following IBM Redbooks also provide detailed information on installing and configuring DB2 multi-partitioned databases:

► *Scaling DB2 UDB on Windows Server 2003,* SG24-7019
► *Up and Running with DB2 for Linux,* SG24-6899

4.3 Running MTK

MTK is an efficient and time-saving tool for porting database objects and loading data from Oracle to DB2. In the following sections we demonstrate the step-by-step use of MTK in such a conversion.

4.3.1 Migration details

This section describes the lab environment and database structure used for the MTK conversion example.

Environment

Prior to elaborating on the details of the sample conversion from Oracle 10g to DB2 9 using MTK V1.4.9, here is some information about the source and target environments in our laboratory:

► Oracle source operating system:
  AIX 5L for POWER™, Version 5.3

► MTK source O/S:
  AIX 5L for POWER, Version 5.3

► MTK version:
  1.4.9.0

► DB2 target O/S:
  Linux - Red Hat Enterprise Linux AS release 4 (Nahant Update 4); Kernel 2.6.9-42ELsmp

► Oracle Version:
  10.2.0.1.0 64 bit

► DB2:
  Version 9 Fixpack 1; 32 bit

MTK is installed on the *Linux* platform because that is where the DB2 target database will reside. Based on our experience it is most effective to install the tool on the operating system *where the target system resides*. The primary reason for this is due to concerns about loading data. If the data to be loaded already resides on the local file system, the concerns of loading data across a network (bandwidth, network traffic, etc.) go away.

Oracle source database

The Oracle source database consists of the following types and numbers of objects:

► Eleven tables
► Two views

- ► Five indexes
- ► Six foreign keys
- ► Four functions
- ► Five stored procedures
- ► Two packages
- ► One package body
- ► Seven triggers
- ► Two sequences

Appendix F, "Example Oracle database" on page 683 lists the definition of these objects. You can also download the code from the IBM Redbook Web site documented there.

4.3.2 Creating and opening an MTK project

When starting MTK, the Project management panel opens asking you to enter required and optional information for a new project, or to open a previously created object. When creating a *new* project the following information can be entered:

- ► **Project Name** - *Required.* If a project name is not entered, it will default to *Unknown.* When a project is created with the Project Name designated here, a subdirectory with the same name is created on the operating system under the install directory of MTK. For this reason, the Project Name has to conform to the operating system naming standards that MTK is installed on.

- ► **Source database** - *Required.* Choices include Sybase Enterprise, Microsoft SQL Server, Oracle, Informix Dynamic Server, and Sybase SQL Anywhere.

- ► **IBM target database** - *Required.* Choices are DB2 9 for Linux, UNIX, and Windows; DB2 UDB V8.2 for Linux, UNIX, and Windows; DB2 UDB V8.1 for Linux, UNIX, and Windows; DB2 UDB i5/OS V5R4; DB2 UDB i5/OS V5R3; DB2 UDB iSeries® v5R2; DB2 V8 for z-series; and Informix Dynamic Server 10.

- ► **Project description** - *Optional.*

Figure 4-1 shows the Project management screen after the information for the sample Oracle migration project has been entered.
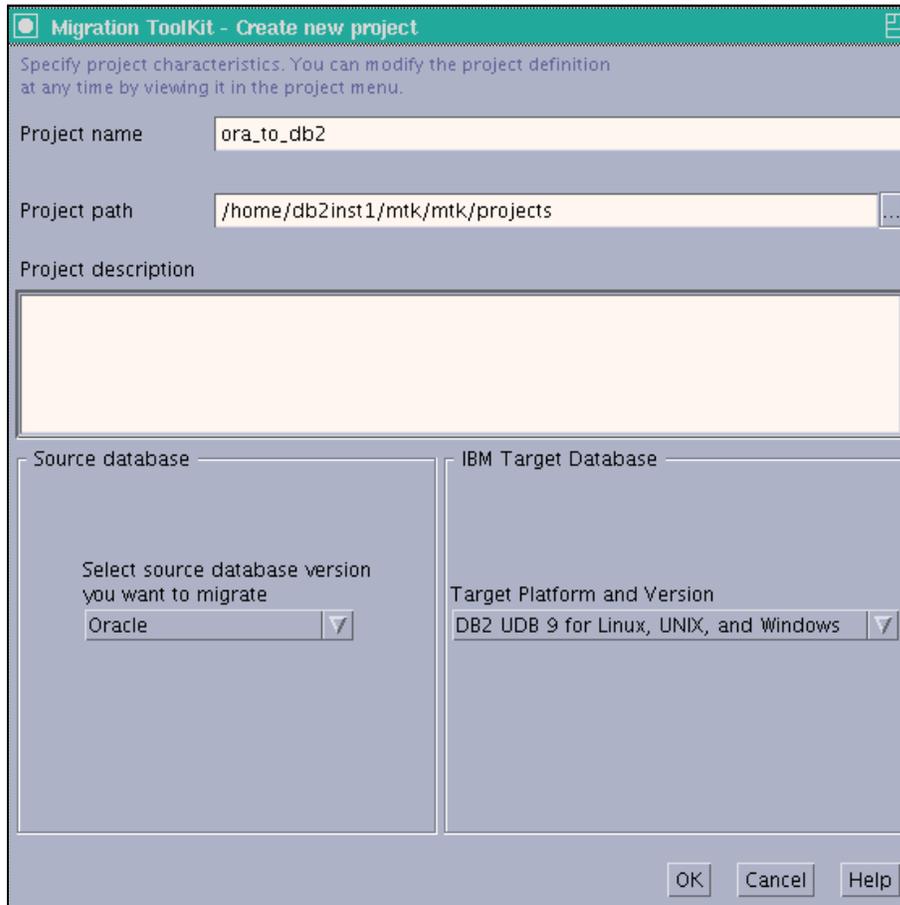
Figure 4-1   MTK Project management screen

4.4 Extracting or importing metadata into MTK

Once the project is created, the Specify Source window opens. During this task the following is accomplished:

► The method of obtaining the metadata (DDL) is designated.
► The DDL is obtained.

Figure 4-2 on page 94 shows the Specify Source tab. Take note of the **Import** and **Export** buttons on the left side of the panel.

► **Import**
   If a metadata file already exists—for example, a script from SQL*Plus, or a

third-party tool—no data movement by MTK is required. In this case **Import** should be selected. Selecting this option permits single or multiple DDL files from your file system to be selected and imported into MTK as the conversion source.

> **Note:** When importing objects for conversion, be aware that some objects will not convert unless *all* of the objects on which it depends are also available within the project. For example, when importing the source for an individual procedure, you must take care that *all* of the underlying definitions for tables, views, and so on, that are referenced within that procedure are also included.
>
> Also, make sure that the definitions for all dependent objects *precede* the objects that rely on them. For example, definitions for tables that are referenced in stored procedures must *precede* the definition of the stored procedure itself.
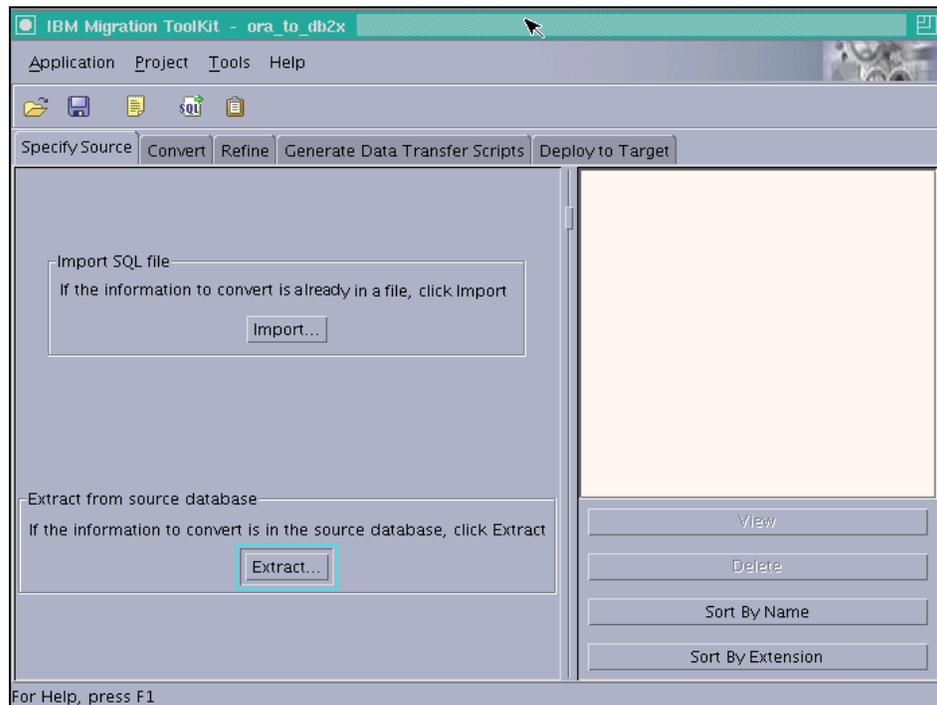
Figure 4-2   The Specify Source tab

► **Extract**

For our sample conversion, we choose **Extract.** Choosing this option signifies:

– That a client connection to the Oracle source system will be made.
– The Oracle metadata (DDL) will be *extracted* using this connection.

The extraction is accomplished by "reading" the system catalogs and then creating a file that will be imported into the tool.

Note1: The Oracle source system to which MTK connects can either be on a local or a remote server. For a valid connection, it is required that the *Oracle Client* be installed and configured in the CLASSPATH on the local machine. On Windows systems ODBC drivers may be used.

Note2: MTK will store the files from the extracted DDL, converted DDL, in a subdirectory of the MTK installation directory. The structure is:

MTK Installation directory

Project directory

YOUR_PROJECT_NAME directory
DDL files

After **Extract** is chosen, the Connect to Database screen opens. The following information must be entered:

► **Service Name** - The service name for the local/remote Oracle database
► **User ID** - The user ID for the schema that owns the Oracle source
► **Password** - The password for the schema that owns the Oracle source

Figure 4-3 shows the Connect to Database screen completed for the service name ora10g, the user ID ora_usr, and the corresponding password for the ora_usr schema.
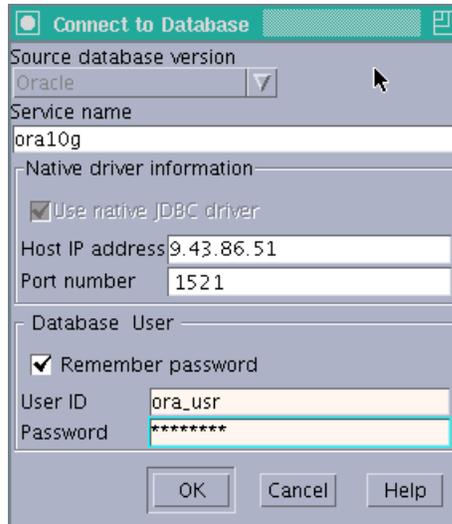
Figure 4-3   Detail of the Connect to Database screen

4.4.1 Choosing objects to extract

After a successful connection to the database, the Extract screen is displayed. This screen shows schemas and objects that are available to be extracted from that schema. Once a schema is chosen, the available objects expand to show six categories of objects (Figure 4-4). The categories are displayed even if the current database does not contain any objects of that category. The categories are:

► Sequences
► Tables
► Views
► Procedures/functions
► Triggers
► Packages
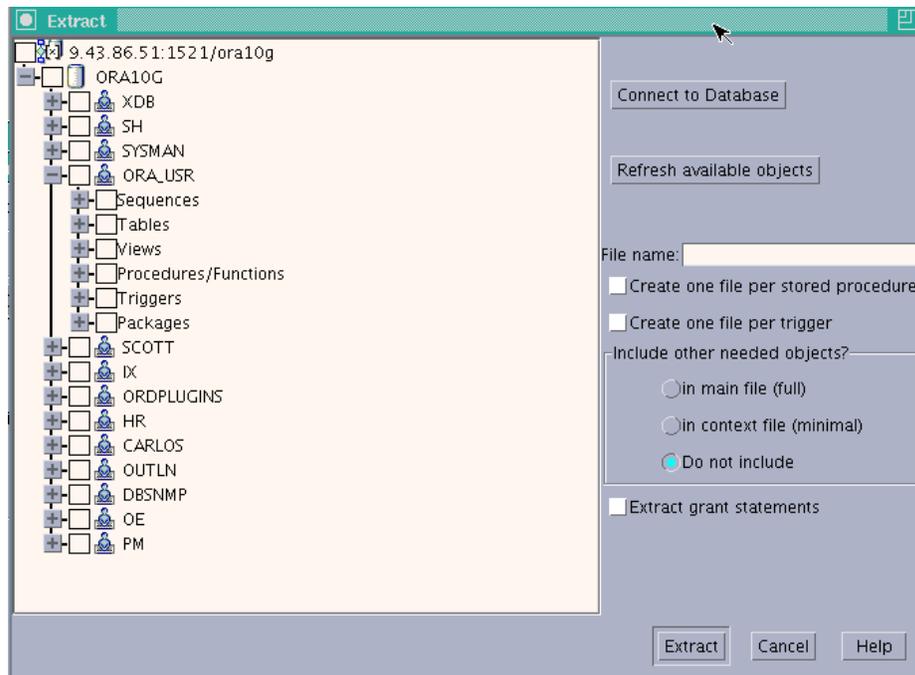
Figure 4-4   The Extract screen showing the objects that are available from the ORA_USR schema.

Once a category is selected, choosing the plus sign (+) expands the category to show the individual objects that exist for that category. It is now possible to make selections of various objects to be extracted from each category. For example, it is possible to choose any of the following combinations for extraction:

► Individual objects from a *single* category (a single table, view, sequence, etc.)

► Individual objects from *multiple* categories (a table, a view, a sequence, etc.)

► All objects from *single* categories (all tables, all views, all sequences, etc.)

► All objects from *multiple* categories (all tables and all views, all sequences and all triggers, etc.)

► All objects from *all* categories (the entire database schema)

Figure 4-5 shows the Extract screen after multiple categories (tables, views, and sequences) have been selected.
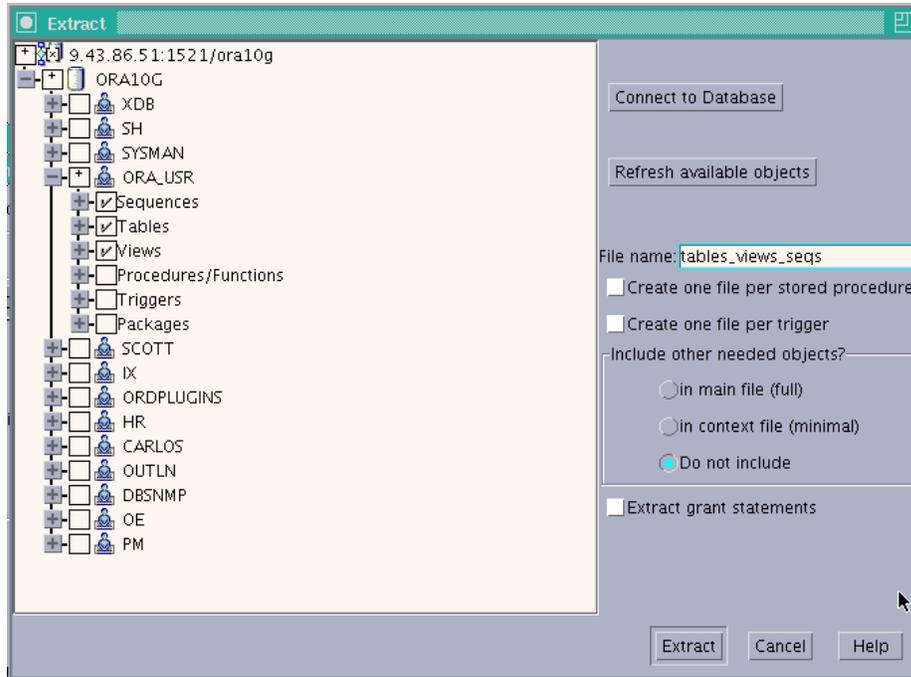
Figure 4-5   The Extract screen with tables, views, and sequences selected

Figure 4-6 on page 99 shows the Extract screen with the **ORA_USR schema** selected. In this example the categories have been expanded to show individual objects that have been selected.
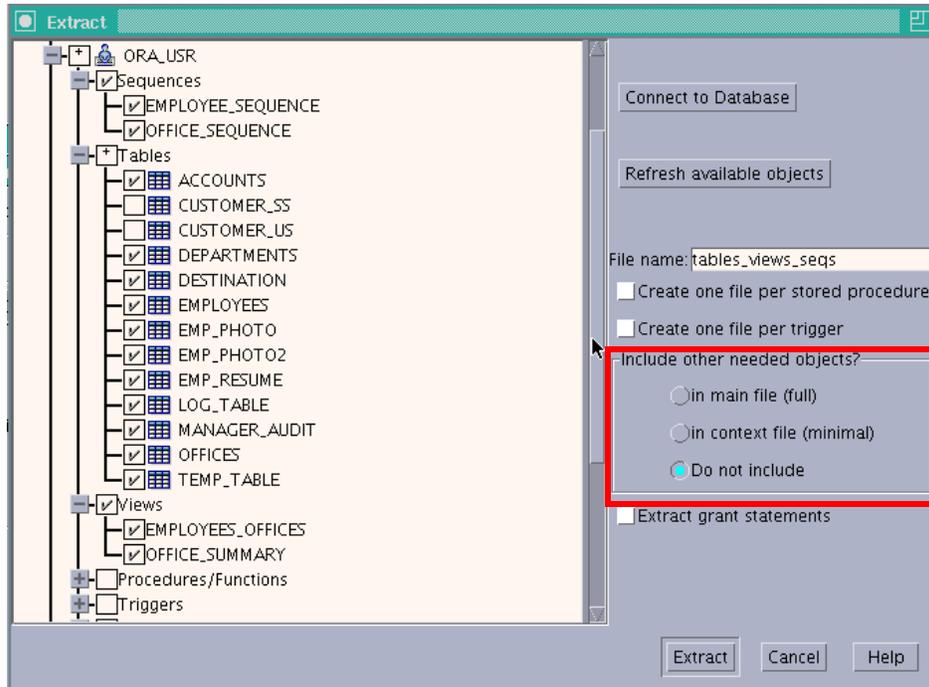
Figure 4-6   The Extract screen with tables, views, and sequences selected

> **Note:** When *extracting* objects for conversion, be aware that some objects will *not* convert correctly (or at all) unless all depending objects are also extracted. For example, when selecting an individual procedure for conversion, *all* of the underlying definitions for tables that are referenced by that procedure must also be extracted. This process will be done automatically if, under the *Include other needed objects?* section of the Extract screen, either **in main file (full)** or **in context file (minimum)** is selected (refer to Figure 4-6 highlighted in the red box). When either of these options is chosen, MTK will *automatically* include all the relevant definitions to correctly convert the selected objects. Also, MTK will extract the objects in correct dependent order.

4.4.2 Import or extract strategies

As previously stated, it is possible to choose one single object or the entire database for extraction. Although it may seem initially attractive to choose the entire database, we recommend that the extraction strategy be dictated by the size of the source database.

Some aspects of database sizing in regard to MTK should be briefly mentioned here. The accurate sizing of a database, from a migration point of view, should entail a thorough analysis on several different levels. This investigation should include information such as: the number of lines of code; the complexity of the code; the conformance or non-conformance of the code to ANSI standards; the number of objects; and the types of objects—just to name a few.

One of the best uses of MTK is as an aid in doing this type of analysis. MTK can be used to find, in detail, much of the information that is required to successfully analyze the database.

With that said, a general and simplistic approach to sizing the database may be used in the *initial* stages of analysis. In this early stage, for example, there might be little, or no, information about complexity or conformance to ANSI standards. In order to gather that information we would have to develop some guidelines for beginning the analysis. The guidelines at this stage are usually along these lines:

► Large database

  More than 200 stored procedures and functions (stand-alone or in Oracle Packages), and triggers.

► Small database

  Less than 200 stored procedures and functions (stand-alone or in Oracle Packages), and triggers.

Large databases

For large databases, the extraction strategy should focus on creating *separate* files for each individual category. For example, a separate extraction file should be created for all tables, sequences, views, triggers, procedures, packages, and functions. This strategy facilitates "tracking down," analyzing, and perhaps "fixing" possible issues that may arise in the conversion of a particular object category. This is usually easier than trying to understand several complex issues that may arise across a spectrum of interrelated categories.

Small databases

For smaller databases, it is recommended that the extraction files be grouped according to *dependencies* and *dependents*. For example, one file may consist of all tables, views, and sequences; and another file of procedures, functions, packages, and triggers. In this strategy, the first file will allow the objects on which the second file depends to be created and analyzed before converting the second file. In this way, we are also able to contain the messages and the interrelated issues that may occur.

Sample conversion

Considering the size of our sample database, we adhere to the recommendation for small databases. For our extraction, the first file contains all tables, views, and sequences; a second file contains all procedures, functions, packages, and triggers.

For the first file, all objects in the categories Tables, Views, and Sequences are selected. For File name, we chose and entered `tables_views_seqs` (Figure 4-6 on page 99).

> **Note:** MTK will add the suffix .src to the File name entered on the Extract screen. In our example, for example, MTK generates tables_views_seqs.*src* for the file name tables_views_seqs that was entered.

Once this is completed, **Extract** is clicked, and the metadata extraction begins.

After the first file is created, we repeat the process, this time selecting all procedures, functions, packages, and triggers. For this file we choose and enter the name `procs_pkgs_trigs` (Figure 4-7).
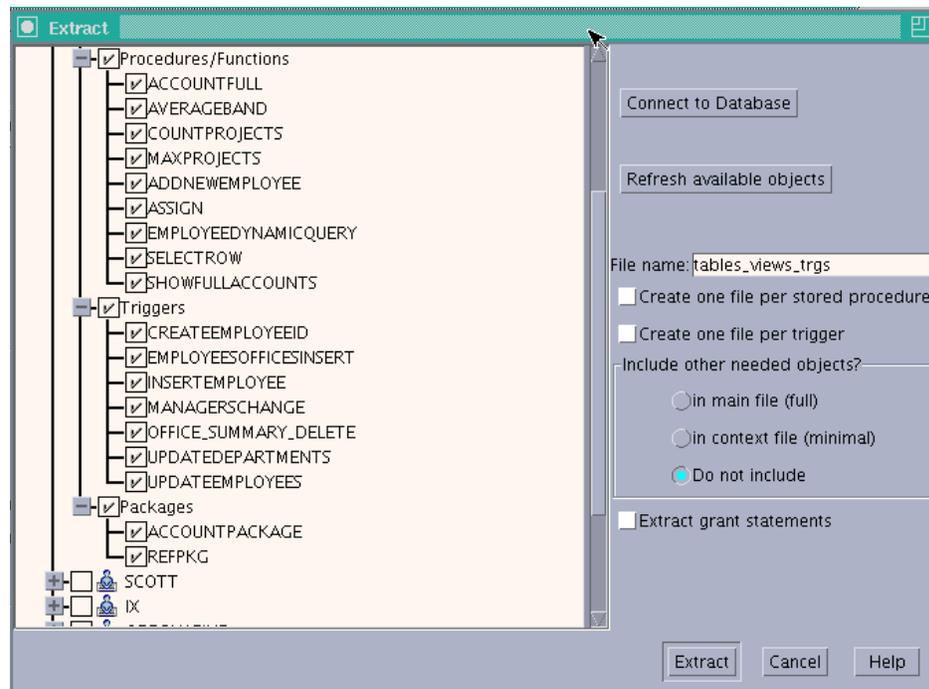
Figure 4-7   Selecting the Procedures, Packages, and Trigger objects for extraction

4.4.3 Viewing extracted files

Once the extraction has completed, the Extract screen closes and Specify Source tab remains, as shown in Figure 4-8. Note that on the right side of the panel the files that were created during the extraction of tables_views_seqs.src and procs_pkgs_trgs.src are now visible.
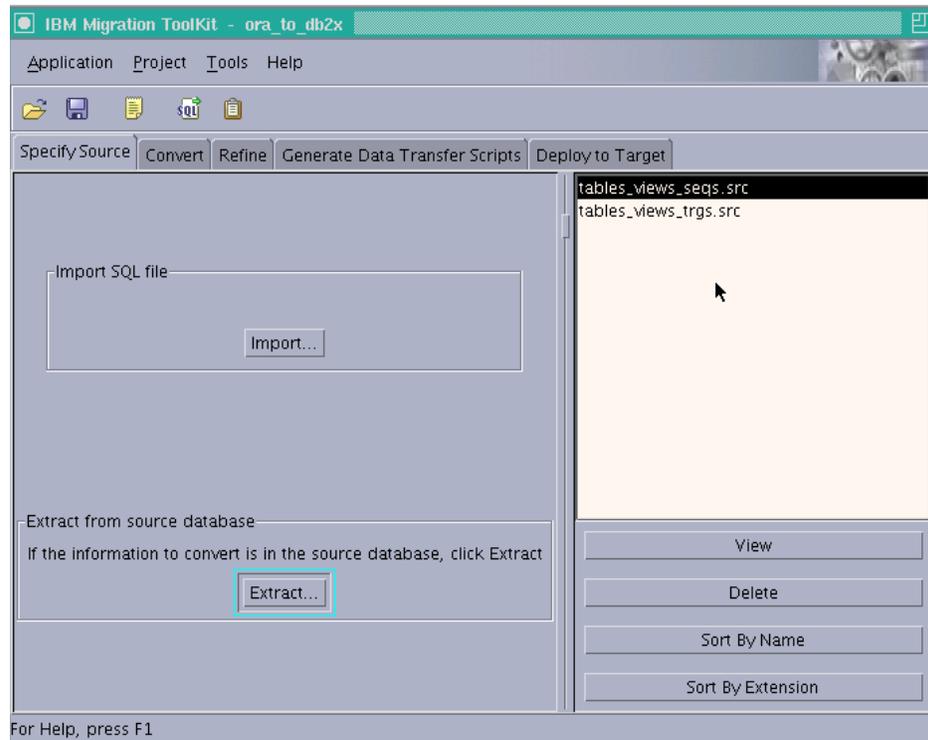
Figure 4-8   The Specify Source tab after all the metadata files have been extracted.

The extracted files can be examined by highlighting the file name and then clicking **View** (located on the lower right side of the panel). In our example, when tabs_views_seqs.src is selected and then viewed, we see the screen displayed in Figure 4-9 on page 103.

Figure 4-9   Viewing extracted files in the Text Editor

You can edit and save this file when it is viewed from this panel. This is sometimes useful for making changes to source code *before* it is to be converted.

The file was imported. The edited file is a copy in the project directory, and the original file will neither be changed nor damaged, and it can be reimported in case of user errors during editing.

You can specify a preset or external editor by selecting **Applications** → **User Preferences** on the MTK menu bar, as shown in Figure 4-10.

Figure 4-10   Selecting User Preferences on the MTK menu bar

4.5 The Convert task

Once the extraction files have been created we proceed to the next task, *Convert*. On the Convert tab there are a few requirements that need to be completed:

1. Select the files in the left-hand pane to be converted.

2. Enter a prefix for the generated files (that is, a name for the file that will be generated from the conversion).

3. Select **Convert** to begin the conversion.

For our example we have retained the name of the source file (tables_views_seqs) as the prefix for generated files; as a result, the full name of the conversion file will be tables_views_seqs.db2 as shown in Figure 4-11 on page 104.
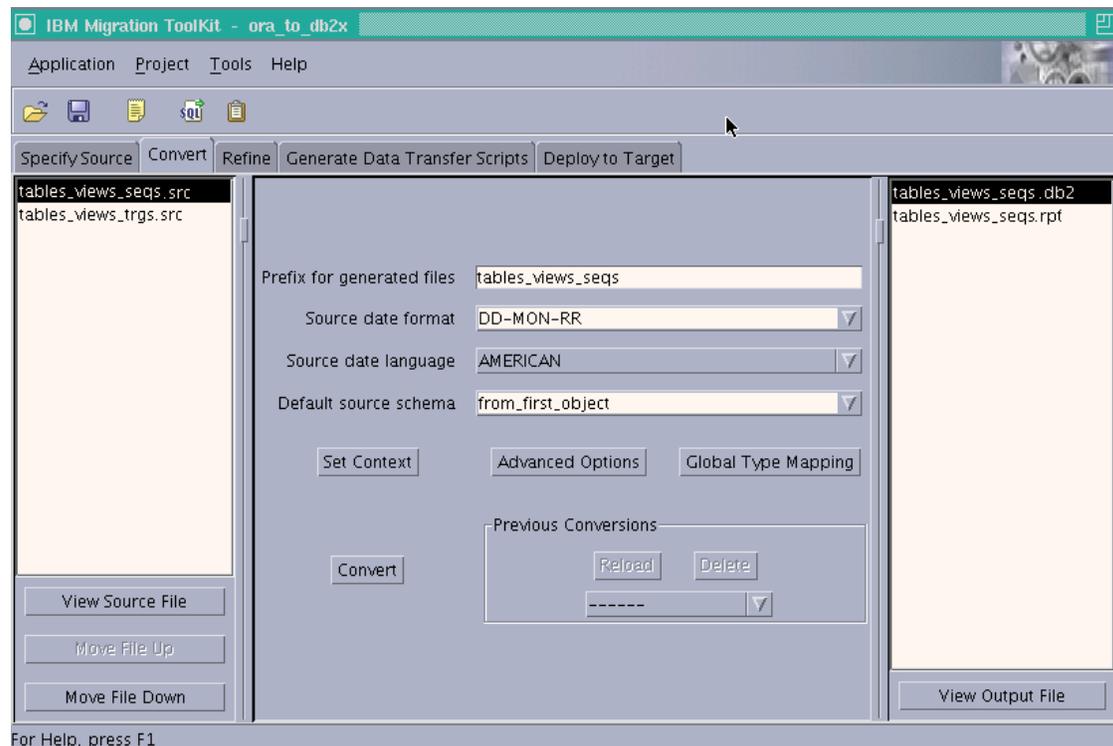
Figure 4-11   The Convert tab after the tables, views, and sequences have been converted

In addition to the required fields, there are several options that may be chosen that will affect the content of the generated conversion file:

- ► Global Type Mapping
- ► Advanced Options
- ► Set Context

Global Type Mapping

If **Global Type Mapping** is selected, a screen opens (Figure 4-12) that allows *some* of the default MTK data type mappings to be changed. Only fields that have the "pad and pencil" icon are available to be edited. For example, the Date field in Oracle is typically mapped to Timestamp in DB2; in this screen it is possible to replace TIMESTAMP with DATE or TIME as the default mapping. It is important to remember that whatever data type mappings are altered will be applied to *all objects in the entire project.* It is *not* possible, from this screen, to set a different data type for each individual object.
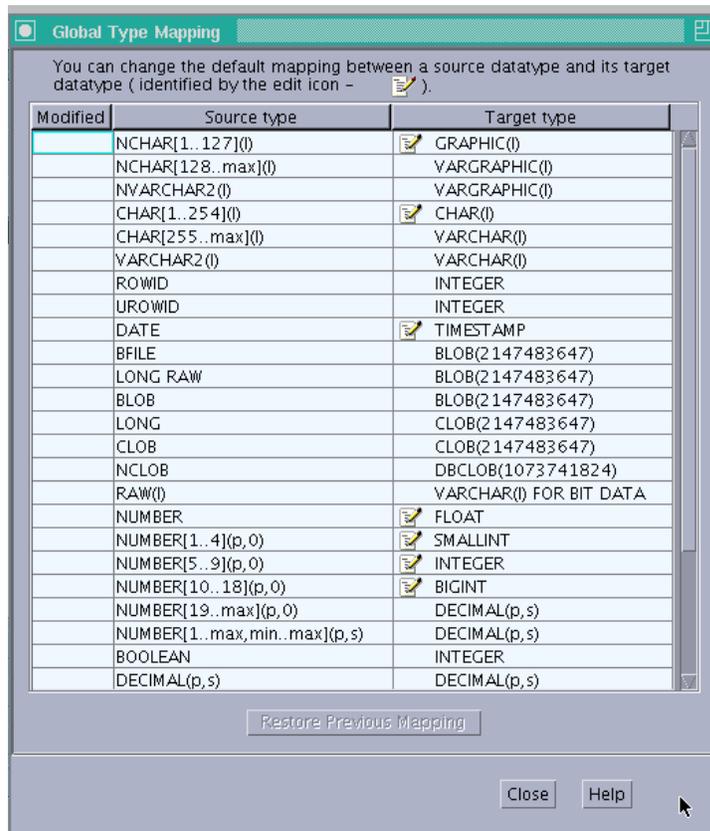
Figure 4-12   The Global Type Mapping screen

Advanced options

If **Advanced Options** is selected, a screen opens (Figure 4-13) that shows several features that will affect the content of the generated conversion file. The features are grouped into three categories:

- ▶ General converter options
- ▶ Converter options for tables, views, indexes
- ▶ Converter options for procedures, functions, triggers
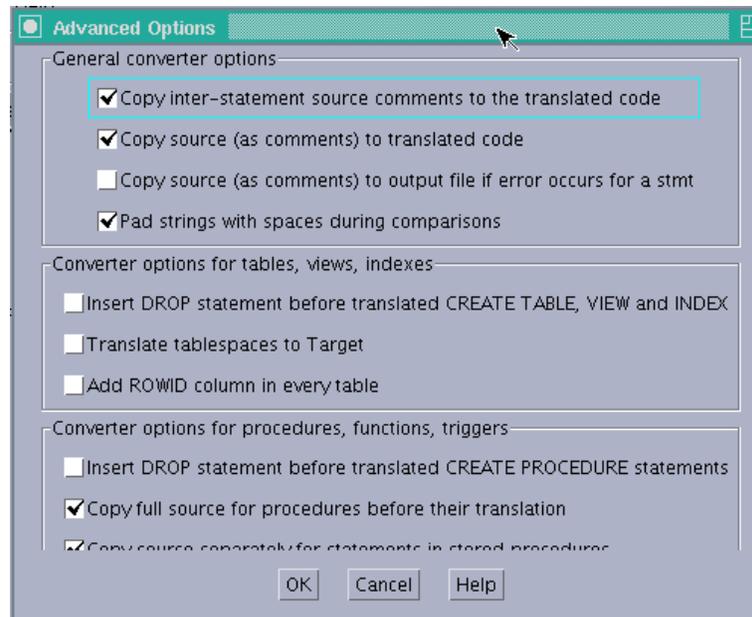
Figure 4-13   The Advanced Options screen

Most of the options pertain to whether the converted code will contain the Oracle source code, as comments, in the generated file. On this screen, the following options are checked by default:

- ▶ Copy inter-statement source comments to the translated code.
- ▶ Copy source (as comments) to the translated code.
- ▶ Pad strings with spaces during comparisons.
- ▶ Copy full source for procedures before their translation.
- ▶ Copy source separately for statements in procedures.

> **Note:** Leaving the options as-is generates a conversion file that contains *all* of the Oracle source code (as comments) along with the converted DB2 code. When first using the tool, it is recommended to *leave the defaults as-is*. This will allow examination and comparison between the Oracle source and the DB2 target *within the same file*. Also, in many cases, the comparison serves as an excellent teaching tool for those who may not be familiar with the syntax of the DB2 SQL Procedure Language.

For our example, we retain all the defaults, but we also add the **Translate tablespaces to DB2** option. Choosing this option causes the converter to include table space names that retain the names from the Oracle source code. The create table space statements are also added at the beginning of the script file. This is important for our sample porting project because we intend to deploy the converted objects into a DB2 database that will contain tables, indexes, and blob/clob data in table spaces that have been created using these names. In the case where the table space mapping from Oracle to DB2 is different, the table space reference can easily be modified.

Once the options have been entered, we do the following required steps:

1. Choose the file to be converted. For our example, we choose tables_views_seqs.src.

2. Choose and enter a `Prefix for generated files`. For our example, we allow it to default to the name of the currently selected file.

3. Click **Convert**.

The conversion process now begins. During the conversion, the message `Please wait…Converting files…` is displayed along with the elapsed time. Eventually the message will change to `Please wait…Initializing Refine` as the conversion completes.

Set Context
The context of a conversion indicates the source files that MTK uses when it executes a conversion. If **Set Context** is chosen, source DDL files that contain other objects MTK needs for conversion are made available to MTK. Details of this option are discussed in 4.10, "Converting the remaining objects" on page 138.

4.6 The Refine task

Once the Conversion task has completed, the next task is executed using the Refine tab (Figure 4-14 on page 109). The Refine tab is subdivided, on the lower

left side of the pane, into four separate subtabs titled Oracle, DB2, Report, and Messages.

► Oracle
  This tab displays all of the objects from Oracle source script.

► DB2
  This tab displays all of the corresponding objects that have converted in DB2.

► Report
  The Report view displays the message, sorted by database object. When you click the **Report** tab, the right panel displays the messages, grouped by message number, in decreasing order of importance. You can expand the source file in a tree view to display objects that contain messages. Often the same message will occur in many places. You can filter the messages that appear in the tree by clicking **Hide message** in the tree for each message you do not want to see. Click the button again to have the message reappear.

► Messages
  The Messages view displays the messages, sorted by message category and number. When you click the **Messages** tab, the right panel displays the messages, grouped by message number in decreasing order of importance. You can expand a message category in a tree view to display the list of messages that occurred in the file.
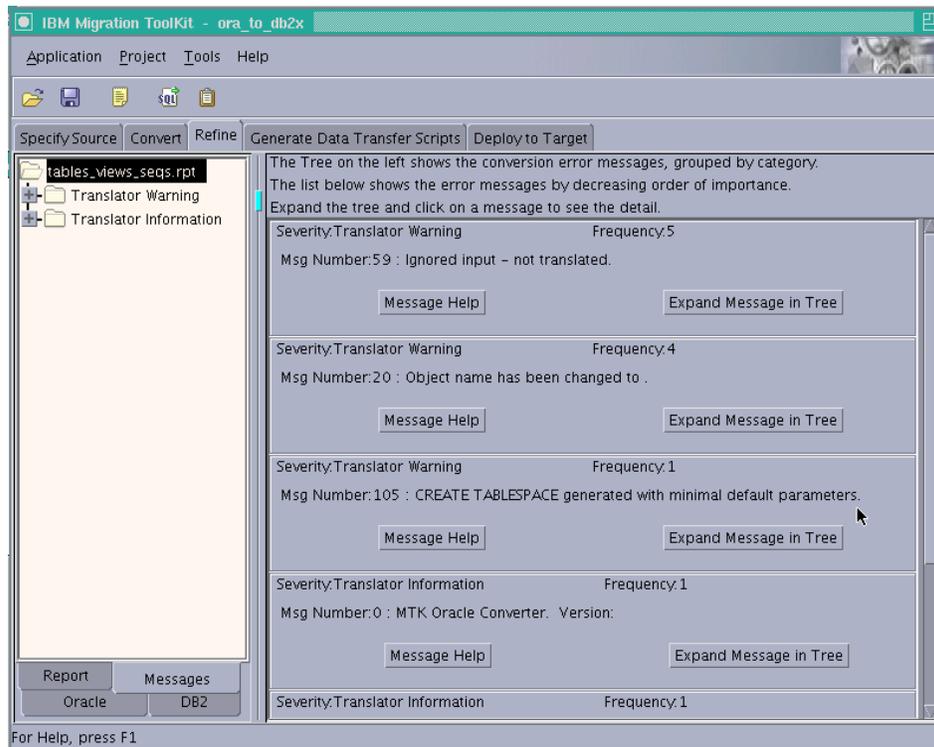
Figure 4-14   The Refine tab

4.6.1 Message categories and migration impact

As previously stated, the Messages view displays the messages sorted by message category. The message categories are:

► Input Script Error
► Translator Information
► Translator Warning
► Translator Error

Input Script Error

This message category occurs when the input script or set of objects to be converted is incomplete. Most frequently, messages in this category occur when an object is missing the definition for an object on which it depends, for example a stored procedure, which refers to a table for which the definition is missing. Sometimes an object definition does exist, but its use may require qualification with a database or owner name. Other errors in this category include PL/SQL syntax errors.

Migration impact - low

Since this type of message is easily understood and corrected (usually by including the missing definition in the source file), the migration impact is low and has little bearing on the final product or on the level-of-effort to achieve that product.

This type of message occurs most frequently when files are taken into the tool through IMPORT. As can be expected, it is more likely that due to human error, an incomplete DDL may be gathered as the conversion source.

Translator information

This category occurs when a correct DB2 translation exists, but when more information is necessary to describe some unusual or exceptional property of the translation. For example, messages in this category are used to highlight the fact that the name of a PL/SQL object or identifier has been changed to satisfy the DB2 restrictions on identifier formation (such a change might be relevant to client programs that refer to the object by name).

Migration impact - low/medium

This type of message should be examined to understand the scope of the message. If the message indicates that an object name has been changed to conform to a DB2 specification, this may have little or no impact on the migration effort. If, however, the changes generated by the converter require alterations to the client code, the effort may be more extensive.

Translator Warning

This category occurs when the translation of the PL/SQL code to which the message refers might be incomplete or incorrect in certain unusual or exceptional cases. The message typically describes the circumstances in which the translation will not be correct.

Migration impact - medium

This type of message needs to be examined to determine if the circumstances described are relevant to your application. If so, manual intervention *may be required* to successfully convert and deploy the object.

Translator Error

This category is used for PL/SQL statements for which no translation is possible. Most frequently, this message category is used when no equivalent DB2 functionality exists. It is also used in cases where a correct translation requires application-specific information. It also occurs for certain complex or rarely used constructs.

Migration impact - high

This type of error usually indicates that *some degree* of manual intervention will most likely be required. It is important that the analyst review the code to understand for which objects, and to what degree, the manual intervention will be necessary.

4.6.2 The Messages sub-tab

The Messages sub-tab, Figure 4-15 on page 112, is divided into left- and right-hand panes. From these panes it is possible to perform the following actions:

- ▶ View Translator Messages by message number
- ▶ View the corresponding location in the Oracle source code to which a message pertains
- ▶ View the corresponding location in the DB2 converted code to which a message pertains
- ▶ Obtain additional information regarding a particular message
- ▶ Search the source or converted code for a word or phrase
- ▶ Go to a specified line number in the source or converted code
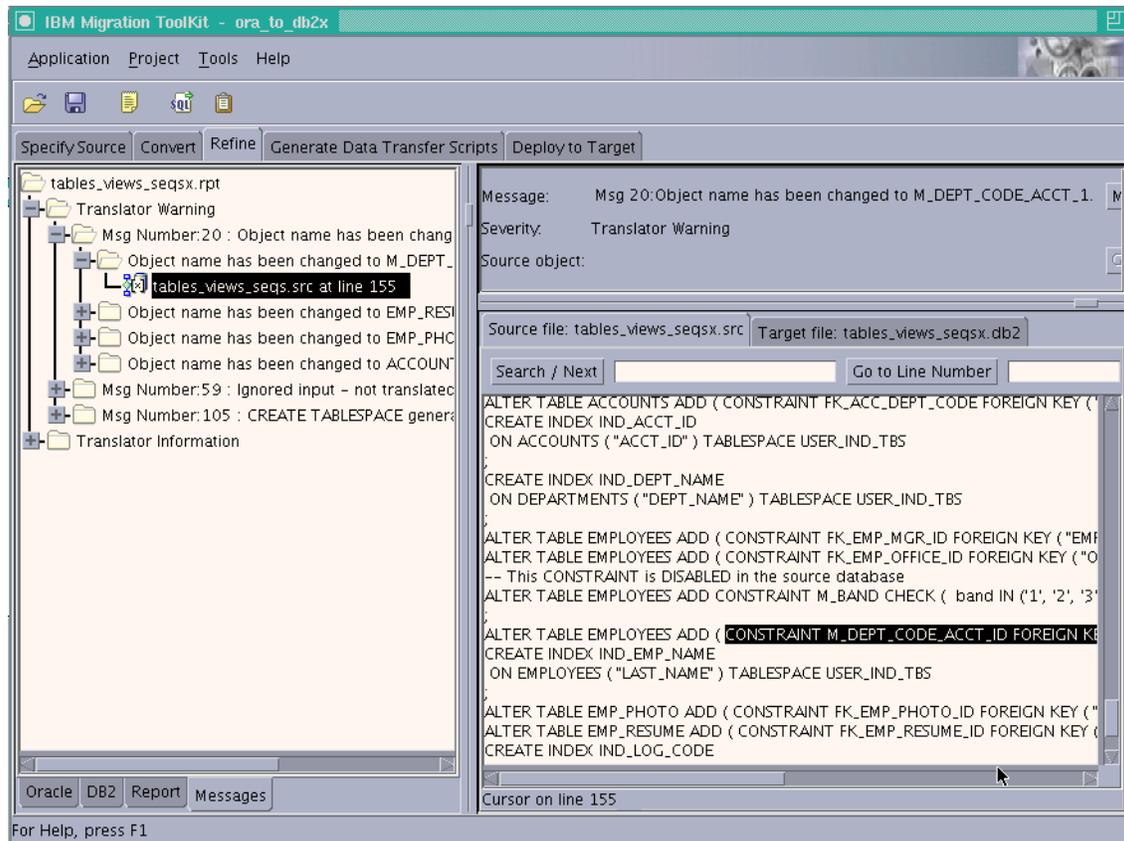
Figure 4-15   The Messages sub-tab

View Translator Messages by message number

Figure 4-15 shows the Messages tab for our example. In this example, the messages are grouped into two categories:

► Translator Warning
► Translator Information

For each category we can drill down to a specific instance of a particular message. To accomplish this, follow these steps:

1. Expand the *category*.
2. Expand the *message number*.
3. Expand the *message description*.

View the source or converted code

Once the message description is expanded, highlighting a specific instance of a particular message will open the right-hand pane to the corresponding line in the *Oracle* source code to which the message refers. In our example (Figure 4-15 on page 112), when highlighting the object pertaining to message number 20 in the left-hand pane, the right-hand pane opens to the relevant line in the Oracle source code (line #155) to which the message pertains.

It is possible to toggle from the Oracle source code to the converted DB2 code by choosing the tabs:

▶ Source file: *<your_source_file_name.src>* tab to examine the Oracle source code

▶ Target file**: *<your_conversion_file_name.db2>* tab to examine the DB2 converted code

The following example (Figure 4-16) shows the view of the converted DB2 file when the *Target file: tables_views_seqs.db2* tab is selected.
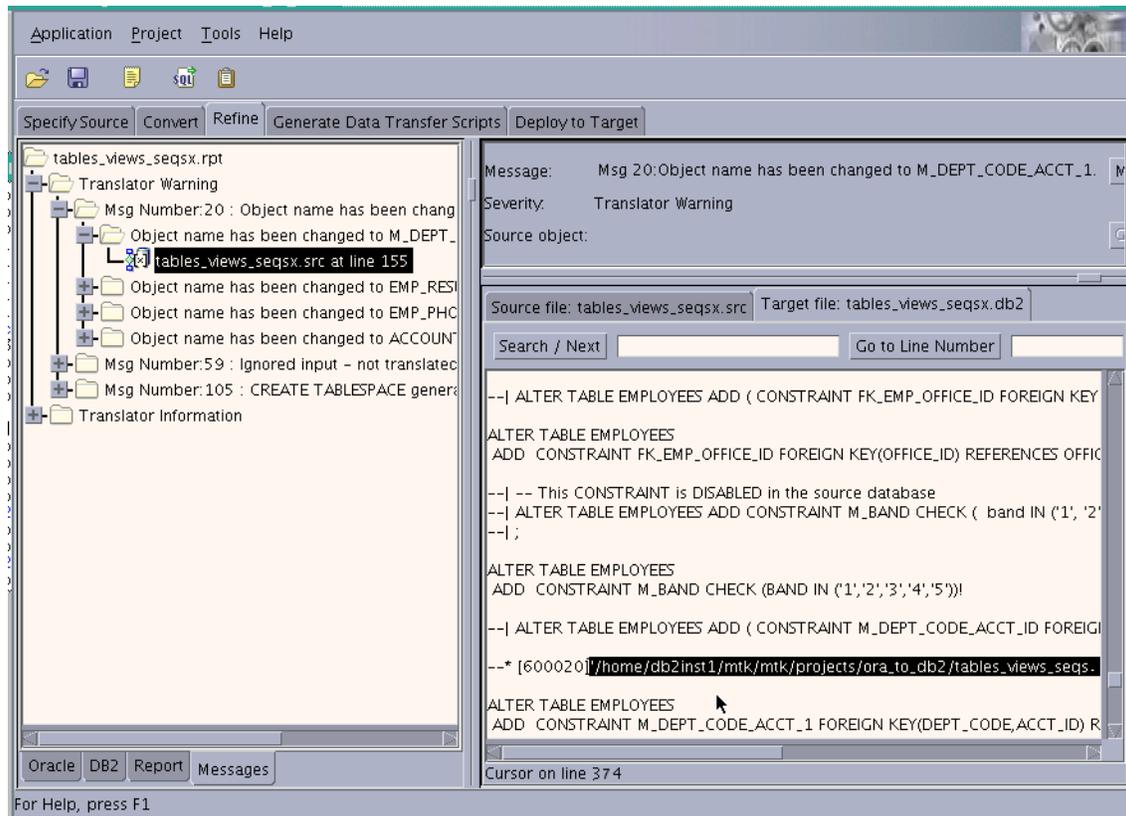
Figure 4-16   Viewing the converted DB2 code that pertains to message 20

Additional message information

If **Message Help** in the upper right-hand corner of the Message sub-tab is clicked, a screen containing additional context-sensitive information about the message number in question will open. In our example, the Message Help screen shows additional information regarding message 20. See Figure 4-17.
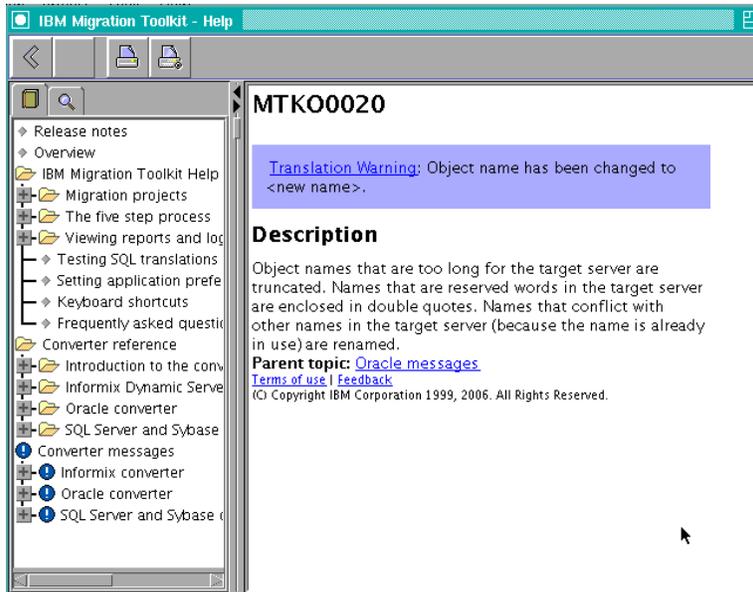
Figure 4-17   Message Help screen displayed for message 20

4.6.3 Translator Messages

The following section contains information regarding the messages that were generated for each category during our example migration. In our example, we received messages in the following categories:

► Translator Warning
► Translator Information

Translator Warning messages

The following Translator Warning messages occurred:

```
Msg number 20: object name has been changed to <new name>
Msg number 59: Input ignored, not translated.
Msg number 105: CREATE TABLESPACE generated with minimal default
parameters
```

► Message number 20

There are four occurrences of message 20 in this category. The message descriptions state:

```
Object name has been changed to M_DEPT_CODE_ACCT_1
Object name has been changed to EMP_RESUME_UK11051
Object name has been changed to EMP_PHOTO_PK110581
Object name has been changed to ACCOUNTS_DEPT_COD1
```

Message Help indicates:

```
Object names that are too long for DB2 are truncated. Names that are
reserved words in the target server are enclosed in double quotes. Names
that conflict with other names in the target server (because the name is
already in use) are renamed.
```

After examining the source code, it is found that the names for the corresponding objects in Oracle, all of which are constraints, were:

```
M_DEPT_CODE_ACCT_ID
EMP_RESUME_UK11058551798461
EMP_PHOTO_PK11058611148823
ACCOUNTS_DEPT_CODE_ACCT_ID
```

For these objects, MTK truncated the constraint names *to conform to the DB2 limit of 18 characters for constraints*.

► Message number 59

There are five occurrences of message 34 in this category. The message description states:

```
Msg number 59: Input ignored - not translated
```

Message Help indicates:

```
This input is ignored, since it is not supported in the target server. This
omission should not cause the target server code to produce different
results from the corresponding Oracle code.
```

The Oracle source indicates that the following statements contain references for creation in particular table spaces:

```
ALTER TABLE ACCOUNTS ADD CONSTRAINT
CREATE INDEX IND_DEPT_NAME
ALTER TABLE EMPLOYEES ADD CONSTRAINT
CREATE INDEX IND_LOG_CODE
CREATE INDEX IND_OFFICE_BLD
```

Specifying table spaces for index and constraint creation, outside of the create table statement is not allowed in DB2. As indicated by the converter message, however:

```
This input is ignored, since it is not supported in the target server. This
omission should not cause the target server to produce different results
from the corresponding Oracle code
```

► Message number 105

There is one occurrence of message 105 in this category. The message description states:

```
Msg number 105: CREATE TABLESPACE generated with a minimal default
parameters.
```

Message Help indicates:

```
The "TABLESPACE" option of the translator is turned on. For each TABLESPACE
clause found, a "CREATE TABLESPACE" statement is generated at the beginning
of the output file. This warning indicates that minimal default parameters
have been used and, therefore, you must change or enhance these, depending
on the physical properties that are required.
```

The converted DB2 source code shows the handling of the CREATE TABLESPACE command:

```
CREATE REGULAR TABLESPACE USER_DATA_TBS
MANAGED BY SYSTEM
USING ('USER_DATA_TBS')!
```

Translator Information messages

In the Translator Information category the following messages occurred:

```
Message number 0
Message number 34
Message number 108
```

▶ Message number 0:

There is one occurrence of this message in this category. The message description states:

```
Msg 0: MTK Oracle Converter. Version: <mtk version>
```

Message Help indicates:

```
Specifies the version of the Oracle converter.
```

This is a General Message displaying the MTK Oracle converter version.

▶ Message number 34

There is one occurrence of message 34 in this category. The message description states:

```
No translation available, but statement has been taken into account
```

Message Help indicates:

```
There is no DB2 translation available, but the converter will use the
information in the statement in translating the statements that follow.
```

The Oracle source indicates that the message refers to the connection statement:

```
CONNECT ORA_USR;
```

The conversion indicates that although the connection statement is not *expressly* converted, the implications of the connection statement will be handled by DB2.

▶ Message number 108:

There is one occurrence of this message in this category. The message description states:

```
Translation Ratio: % (statements were translated successfully)
```

Message Help indicates:

```
This provides an assessment of the provided translation by giving the ratio
of Oracle statements translated without producing any error message out of
the total number of statements. This number is provides a general
indication regarding the success of the automated translation and does not
intend to give an exact and accurate measure.

Statement here designates Oracle SQL and PL/SQL statements. For instance,
in a CREATE PROCEDURE, the whole SQL statement is counted as 1 (one) and
each PL/SQL statement inside the body of the procedure is also counted as
one.
```

This is a General Message displaying a ratio of Oracle statements translated without producing an error message. In the example the ratio is 100%.

Sub-tabs on the Refine tab

When examining messages on the Refine tab it is possible to switch to any of four sub-tabs (located in the lower left of the Refine screen (Figure 4-18 on page 118). In our example the sub-tabs are:

▶ Messages
▶ Report
▶ DB2
▶ Oracle

Figure 4-18 Sub-tabs on the Refine tab

The sub-tabs enable the information to be viewed and *grouped* in several different ways. For example, when viewing the information in the Messages sub-tab, the information is grouped by *message type*; changing to the Report sub-tab enables examination of the same information grouped by *object type*. Selecting the Oracle sub-tab permits the source code for a particular object to be viewed or *edited*. Choosing the DB2 sub-tab allows the converted DB2 source to be viewed. A particular object can be viewed through all these perspectives by selecting it on *any* sub-tab and then changing to another sub-tab.

4.6.4 Refining the metadata conversion

In addition to viewing the results of the conversion, the Refine step affords the opportunity to make changes in the source code. It is possible, for example, to change table, column, stored procedure, or function names or make changes to any DDL. To apply any changes made during the refine process, however, you *must return to the Convert step* to apply these changes. After reconverting, the converter merges the refinement changes with the original extracted source to produce updated target DB2 code. The original source code is *not* changed. This convert-refine process may be repeated until the results are satisfactory.

If issues still remain after the refine-convert process, consider the following: First, see if any further changes can be made to the source metadata. If this approach does not achieve the desired results, it is also possible to alter the converted DB2 code.

Before making any DB2 changes, prepare a backup copy of the .db2 file you intend to change, and rename the backup. We recommend that you make your changes to DB2 *after* leaving the Refine step. *Do not return to the Convert step after making any manual DB2 SQL changes.* Conversion of the source metadata replaces the existing DB2 file, destroying any manual changes.

Once the DB2 source has been successfully generated, it is possible to proceed to either the Generate Data Transfer Scripts tab to prepare the scripts for data transfer, or to the Deploy to DB2 tab to deploy the DB2 metadata to the target server.

> **Important:** If you choose to go directly from the Refine tab to the Deploy tab, skipping the Generate Data Transfer Scripts, then moving data—online or offline—will *not* be an option.

Making changes to the DB2 source

In our example, we made a change to the definition of the Employees table by altering the DB2 target code. This alteration involves changing the EMP_ID column to an IDENTITY column. Here is our reasoning:

In the Oracle source the EMP_ID column is defined as INTEGER NOT NULL; the values for this column are automatically generated by a sequence (Employee_Sequence) that is activated by a trigger, CreateEmployeeID. We intend to replace this functionality by creating the EMP_ID column as an IDENTITY column; this will allow the table to automatically generate values for the EMP_ID *without* the need for a sequence or a trigger.

Identity columns

Here is some information on IDENTITY columns from the *Application Development Guide: Building and Running Applications V8*, SC09-4825:

► Rather than using cumbersome insert and update triggers, DB2 enables you to include generated columns in your tables using the GENERATED ALWAYS AS clause. Generated columns provide automatically updated values derived from an SQL expression.

► DB2 application developers often need to create a primary key for every row in a table. If you create a table that uses an identity column for the primary key, DB2 automatically inserts a unique value. When you use identity columns, your applications can benefit from increased performance due to a reduction in lock contention.

Identity column considerations

Since the Employees table will be loaded with data that includes EMP_ID values that were already generated by an Oracle sequence, we need to take care to:

► Preserve the original values of EMP_ID currently in the Employees table.

► Preserve the increment of the original sequence.

► Generate new values that *will not conflict* with the current values in the column.

Identity Column syntax

Here is a brief explanation of the syntax for creating an IDENTITY column. Syntax:

```
GENERATED BY DEFAULT START WITH numeric-constant, INCREMENT BY
numeric-constant
```

Where:

► GENERATED BY DEFAULT
Indicates that DB2 will generate a value for the column when a row is inserted into the table, or updated, specifying DEFAULT for the column, unless an explicit value is specified. BY DEFAULT is the recommended value when using data propagation or doing unload or reload.

► START WITH `numeric-constant`
Specifies the *first* value for the identity column.

► INCREMENT BY `numeric-constant`
Specifies the interval between *consecutive* values of the identity column.

The Oracle definition of the EMP_ID column in the Employees table is as follows:

```
CREATE TABLE EMPLOYEES (EMP_ID INTEGER NOT NULL, …
```

Here is the column definition we will create for the DB2 target:

```
CREATE TABLE EMPLOYEES (EMP_ID INTEGER NOT NULL GENERATED BY DEFAULT AS
IDENTITY (START WITH ???, INCREMENT BY ???), …
```

To complete the column definition we need only:

► Supply a starting value that will be greater than the current maximum EMP_ID value that is currently in the table. The best and easiest way of doing this is to:

– Execute **Select MAX(EMP_ID) from Employees**

– Retrieve the result and set the START WITH VALUE equal to that value plus one. In our example the result of that process is 10011.

► Duplicate the increment value (increment by 1) from the Oracle Employee_Sequence.

The completed definition for the EMP_ID column in the Employees table is:

```
CREATE TABLE EMPLOYEES (EMP_ID INTEGER NOT NULL GENERATED BY DEFAULT AS
IDENTITY (START WITH 10011, INCREMENT BY 1), …
```

Editing the DB2 target code

To edit the DB2 code, do the following:

1. Open the Convert tab

2. Select **tabs_views_seqs.db2** in the right-hand pane

3. Click **View Output File**.

4. When the Text Editor opens (Figure 4-19 on page 122), enter the changes as shown in Example 4-1.

Example 4-1   Editing the DB2 code to create an Identity column

```
CREATE TABLE EMPLOYEES(
EMP_ID INTEGER NOT NULL GENERATED BY DEFAULT AS IDENTITY (START WITH 10011,
INCREMENT BY 1,
    FIRST_NAME VARCHAR(20),
    LAST_NAME VARCHAR(20),
    DEPARTMENT VARCHAR(30),
    CURRENT_PROJECTS SMALLINT,
    EMP_MGR_ID INTEGER,
    DEPT_CODE CHAR(3) NOT NULL,
    ACCT_ID SMALLINT NOT NULL,
    OFFICE_ID INTEGER,
    BAND CHAR(1)
)
IN USER_DATA_TBS!
```

5. Save the changes.

The edited definition will be used when the table is created in DB2 during the Deployment phase of the conversion.
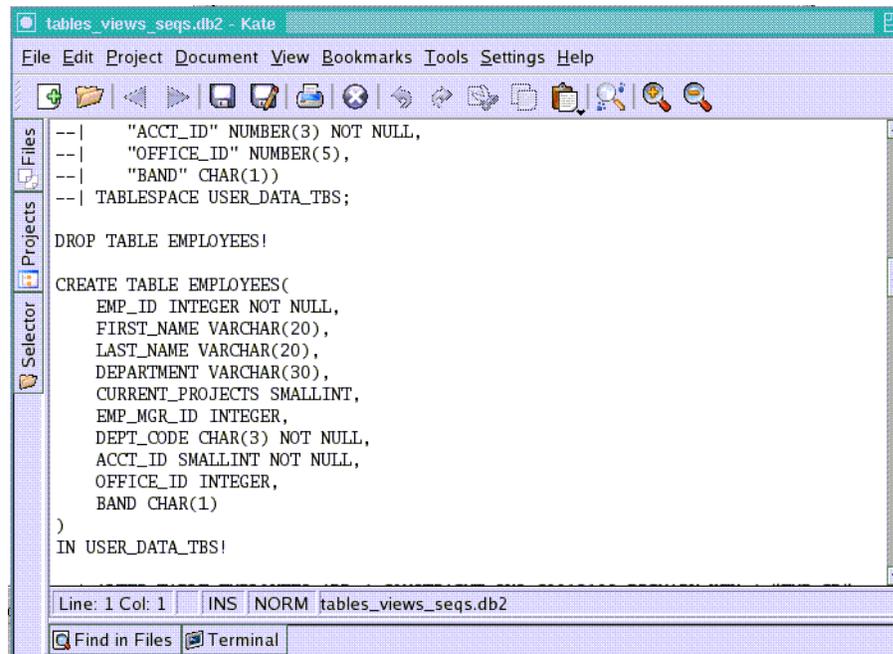

```
--|     "ACCT_ID" NUMBER(3) NOT NULL,
--|     "OFFICE_ID" NUMBER(5),
--|     "BAND" CHAR(1))
--| TABLESPACE USER_DATA_TBS;

DROP TABLE EMPLOYEES!

CREATE TABLE EMPLOYEES(
    EMP_ID INTEGER NOT NULL,
    FIRST_NAME VARCHAR(20),
    LAST_NAME VARCHAR(20),
    DEPARTMENT VARCHAR(30),
    CURRENT_PROJECTS SMALLINT,
    EMP_MGR_ID INTEGER,
    DEPT_CODE CHAR(3) NOT NULL,
    ACCT_ID SMALLINT NOT NULL,
    OFFICE_ID INTEGER,
    BAND CHAR(1)
)
IN USER_DATA_TBS!
```

Figure 4-19   Viewing the EMPLOYEES table in the Text Editor

4.7 The Generate Data Transfer Scripts task

The next task in our conversion is the Generate Data Transfer Scripts tab. From this tab it is possible to:

▶ Generate scripts to unload data from the Oracle database

▶ Generate scripts to Import or Load data into DB2

This tab is divided into three panes. When initially opened, the left pane shows all the tables in the Oracle schema that are being converted; the right pane shows the file that contains all of the converted DB2 code; the middle pane contains options that may be specified when IMPORT or LOAD is selected (Figure 4-20 on page 123).
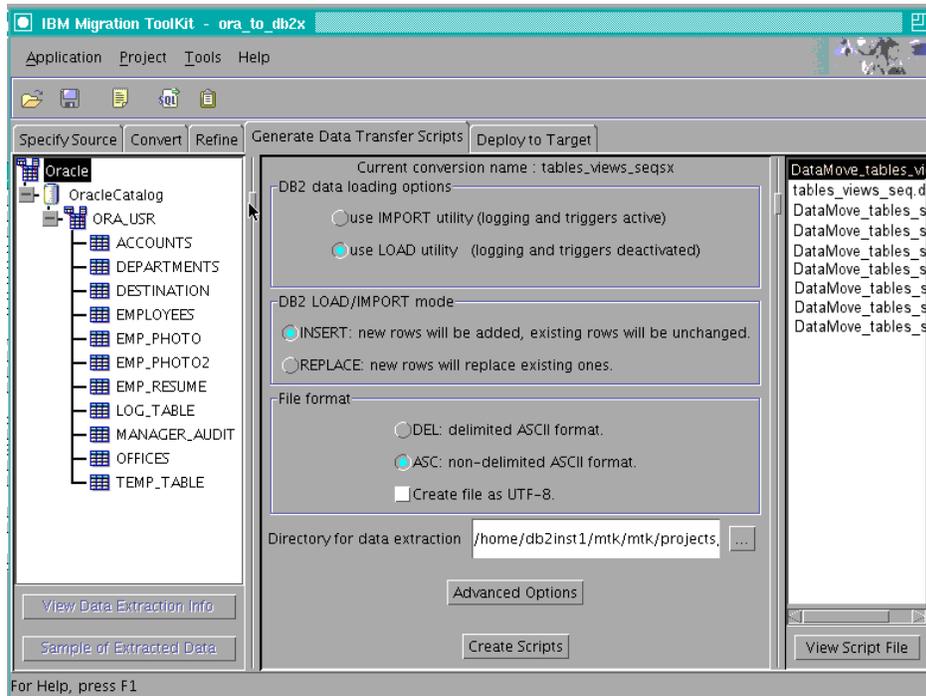
Figure 4-20   The Generate Data Transfer Scripts task

Choosing Data Import or Load

The choice of whether to IMPORT or LOAD data into DB2 is usually influenced by the amount of data.

For small amounts of data, IMPORT is usually sufficient. This method inserts data into a database table from an external file—*one row at a time*. During the import of data, the table remains accessible to other applications. All applicable constraints and triggers remain in effect, and are activated in the usual way as rows are inserted.

For large amounts of data, LOAD is usually preferable. The LOAD utility constructs page images containing many rows and inserts them into the database one page at a time. It requires exclusive access to the table space being loaded. During the loading of data, the table space that contains the table is not accessible to other applications. Applicable constraints and triggers are deactivated for the duration of the load. Applicable check constraints and foreign key constraints are enforced at the end of the load process. The enforcement of business rules that are implemented by triggers are not guaranteed after the LOAD completes.

During LOAD and IMPORT, MTK disables the referential integrity. MTK deletes the foreign keys before load and import, and recreates them after the process is complete.

IMPORT or LOAD options

The following options are available for IMPORT or LOAD:

► MODE
  – INSERT: IMPORT inserts new rows without affecting the existing content.
  – REPLACE: IMPORT deletes all existing data and replaces it with the imported data.

► File formats
  – DEL: delimited ASCII format

    DEL specifies that the data is represented in delimited text files. These files contain streams of data values, ordered by row, and then by column. Column delimiters separate the values (a comma is the default), and new line characters separate the rows. Character strings are enclosed in string delimiters (a double-quote is the default). NULL values are represented by nothing between the column delimiters for a particular column.

    If DEL is selected, some advanced options are automatically set by the MTK:

    • Character delimiter first: The character delimiter is given the highest priority, so that a special character between two character delimiters will be read as just another character.
    • Column delimiter: Set to a comma (**,**).
    • Character string delimiter: Set to a double quote (").

    > **Note:** During extraction, MTK doubles any string delimiter found within a string. This allows the use of any string delimiter without a problem. There is no need to search the database for a character never used in the data.

► ASC: non-delimited ASCII format

  This format specifies that the data is represented in non-delimited text files, which are characterized by columns of data in fixed positions. No delimiters are needed. Nulls are identified by a table of null value indicators at the end of a row.

  If **ASC** is chosen, MTK sets the record length advanced option. Instead of new line characters marking the end of each record, the length of the data is used to set the number of characters read for each row.

Note: For complete information about IMPORT or LOAD options, refer to *DB2 version 9 Data Movement Utilities Guide and Reference*, SC10-4227.

Advanced IMPORT or LOAD options

Whether IMPORT or LOAD is chosen, clicking **Advanced Options** opens a panel that displays additional options for data movement. Figure 4-21 on page 125 shows the panel opened to the options available for LOAD. Take note of the *messages file* parameter. It is recommended that this parameter be completed, since the generated file may prove useful if it is necessary to "debug" the execution of an IMPORT or LOAD. When this parameter is specified, MTK writes the output of IMPORT and LOAD to the message file specified here.
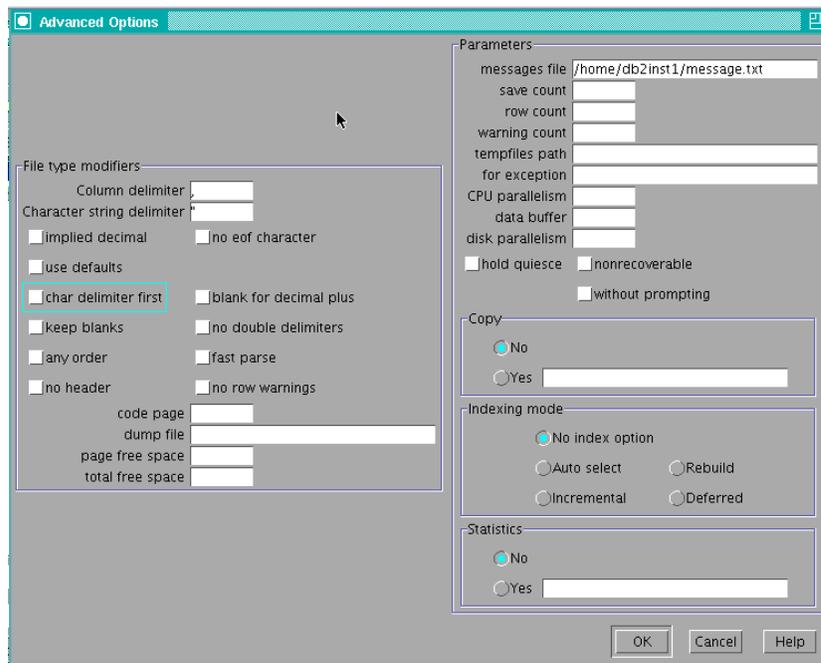
Figure 4-21   Advanced Options screen for LOAD

4.7.1 Creating unload and load scripts

For our example we LOAD the data into DB2; all the defaults will be taken:

► MODE: insert
► File format: ASC non-delimited ASCII format
► Directory for data extraction: MTK default

On the Advanced Options screen the following is chosen:

► Advanced Options: the *messages file* parameter indicates that messages will be written to /home/db2inst1/messages.txt.

Figure 4-22 shows a screen clip after **Create Scripts** is selected, and scripts have been generated.
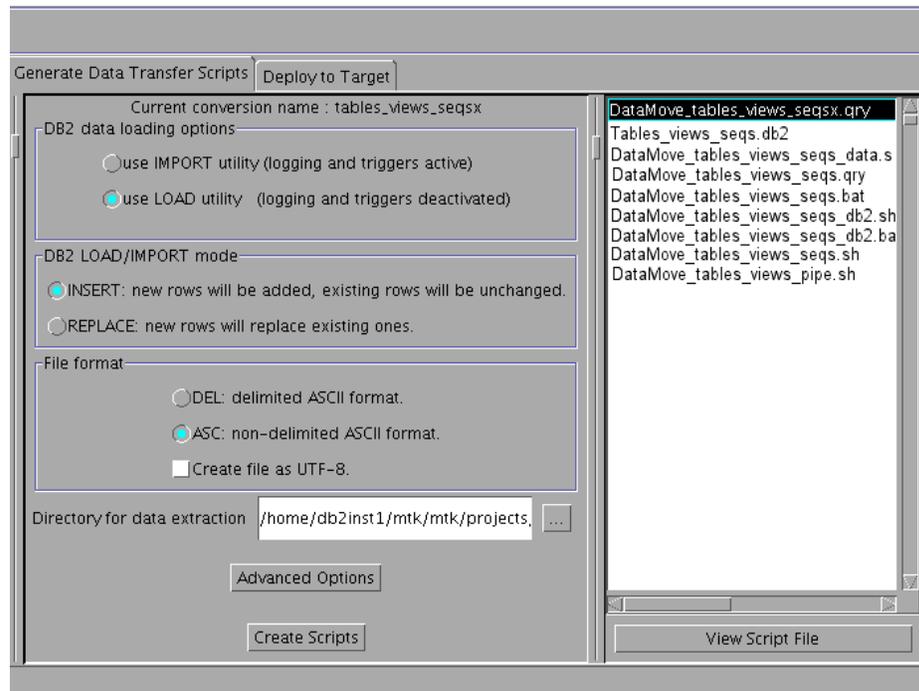
Figure 4-22   The Generate Data Transfer Scripts screen after script generated

4.7.2 Files generated by the Generate Data Transfer Script task

After **Create Scripts** is clicked on the Generate Data Transfer Scripts task, the following files are created:

► *DataMove_your_file_name_data.bat*

► *DataMove_your_file_name_data.sh*

   The files ending in _data.bat, or _data.sh contain statements to *extract* data from the source database.

► *DataMove_your_file_name.qry*

   The file ending in .qry contains examples of statements generated for selecting and converting the data from the source database.

- *DataMove_*`your_file_name`*_db2.bat*

- *DataMove_*`your_file_name`*_db2.sh*

  The files ending in _db2.bat, or _db2.sh contain statements to *execute the load* data into DB2.

- *DataMove_*`your_file_name`*.bat*

- *DataMove_*`your_file_name`*.sh*

  The files ending in .bat, or .sh contain DB2 *load* scripts.

- *DataMove_*`your_file_name`*_pipe.sh*

  This file may be used to load data through a pipe.

- `your_file_name.`*db2*

  This file contains the script for the converted DB2 source code that will be executed during the deployment.

For our example, the following scripts were created:

- DataMove_tabs_views_seqs_data.bat
- DataMove_ tabs_views_seqs _data.sh
- DataMove_ tabs_views_seqs.qry
- DataMove_ tabs_views_seqs _db2.bat
- DataMove_ tabs_views_seqs _db2.sh
- DataMove_ tabs_views_seqs.bat
- DataMove_ tabs_views_seqs.sh
- tabs_views_seqs.db2
- DataMove_ tabs_views_pipe.sh

> **Note:** In cases where there may not be enough resources available on one machine, it is possible to convert and refine the metadata in one location, such as your desktop workstation, and later go to the server system to migrate the large amount of data. For these reasons, it is also possible to *manually* extract the data from the source, and load/import the data into DB2 using the scripts created in this task. Read the MTK documentation for more information on this topic.

4.8 Deploy to Target

From the Deploy to Target tab, it is possible for MTK to perform the following actions:

- Create (or recreate) a local DB2 database
- Deploy the converted objects into a local or remote DB2 database
- Extract and store data from the source database

Figure 4-23 shows the Deploy to Target panel.
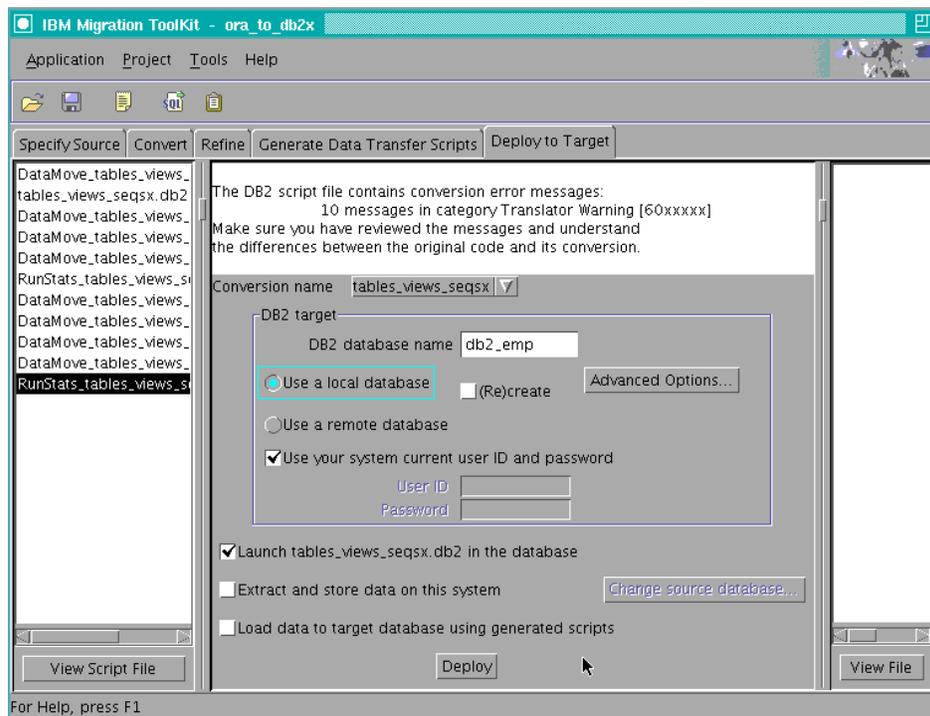
Figure 4-23   The MTK Deploy to Target task screen

Before executing a deployment, it is recommended that the subsequent sections **"Considerations"** and **"Deployment strategy"** be read and understood.

4.8.1 Considerations

Before you start the DB2 deployment, consider the following:

► Database creation: Manual or MTK?
► Object deployment strategy
► Database access

MTK database creation

When deploying objects to a database that is *remote* to the system on which MTK is running, be aware that creating a new database during deployment is not an option. In this case, the database must first be created on the remote system and then registered in the local catalog. Refer to the CATALOG DATABASE command in the DB2 *Command Reference,* SC10-4226 for more information.

When deploying to a database that will be created by MTK on the server where the toolkit is running (local), be aware of the following criteria:

► When MTK creates the database, a buffer pool with a page of size 32 KB and three table spaces of the same size are created.

These provide enough space for the deployment of tables with any row length. Before launching the database into production, do some performance tuning and adjust the size of the bufferpool as necessary.

► When choosing MTK to create your database, only System Managed Tablespaces (SMS) can be created. If your database design requires that tables, indexes, or BLOB data be deployed into their own table spaces, then you *must* use Database Managed Tablespaces (DMS). In this case, the database should be created (either locally or remotely) *before* the objects are deployed into it.

Deploying objects - extracting and deploying data

There are a few considerations for deploying objects and data depending on your environment and requirements:

► When deploying to a local system, **(Re)create** must be selected unless you want to add the objects and data to an already existing database.

► Converted objects and data can be deployed to DB2 *at the same time or separately*. For example, you might want to deploy the DDL and load some sample data to test your procedures during the day. Then, at night, when network usage is low and the database has been tested, the remainder of the data can be loaded.

► The database can be deployed to a local or a remote system. No prerequisites exist for deploying the database locally. If you choose to deploy to a remote system, you can:

  – Specify that the IBM Migration Toolkit access the remote database directly. See "Considerations for remote database access" on page 130.

  – Copy the IBM Migration Toolkit and the project directory to the system running DB2 and then deploy the database locally.

► If you are familiar with DB2 on operating systems other than Windows, the batch files generated by MTK can be modified to accommodate deployments on those systems. Ensure that the correct version of Java is installed on the target system, copy the mtk.jar file and the batch files, and run the batch files.

► If there are many large objects (BLOBS, CLOBs), special consideration should be taken. When LOBs are extracted, each individual LOB data type is put into its own file. Tables with many LOBs can create thousands of external files and supporting subdirectories for them. That means a file system can run out of directory space long before running out of actual mount point space.

So, space planning for temporary data files during data movement of tables with these large objects should be carefully considered.

► The Import and Load utilities use the memory area controlled by database parameter *util_heap_sz*. Medium to large table processing could benefit from an increase in memory space if it is available on your server.

Considerations for remote database access

If you need to access the database remotely during migration, consider the following:

► Make sure that the data types of the data extracted from the source database are consistent with the target DB2 database that is deployed in this step. *Do not attempt to load the data into a database created by other means.*

► Ensure that you have not modified the conversion since the last time the data was extracted. Data should be extracted and deployed only *after* all desired changes have been made to the final conversion.

► Make certain that any data that is being deployed is first extracted to the file system where the IBM Migration Toolkit is installed before it is loaded into DB2. If the table contains millions of rows, ensure the filesystem can accommodate the size of the largest object in the source database. On UNIX file systems, you can modify the file size limit with the `ulimit` command.

► If you choose to deploy to a system that is remote to the system on which the toolkit is running, then you cannot choose to create a new database during deployment. The database must first be created on the remote system and registered in the local catalog. See CATALOG DATABASE in the DB2 *Command Reference,* SC10-4226 for more information.

► Data cannot be loaded into LOB columns during remote deployment. The LOBPATH parameter in the LOAD or IMPORT command must refer to a directory on the DB2 server. You can load data into LOB columns by moving the generated scripts to the target machine and running them on the desired DB2 server (see Chapter 6, "Data conversion" on page 265).

► During deployment, the connection to DB2 uses a Java native driver, not ODBC. If you encounter problems when connecting remotely (such as a "no suitable SQL driver" error), ensure the following directory and files are in the Java CLASSPATH:

  – %DB2PATH%/java/db2java.zip;
  – %DB2PATH%/java/runtime.zip;
  – %DB2PATH%/java/sqlj.zip;
  – %DB2PATH%/bin

4.8.2 Deployment strategy

The deployment information for the objects that were just converted is as follows:

► The target database has been created and resides on a *local* Linux server.

  The database is designed with SMS table spaces. The tables will be stored in a specified table space. The indexes and BLOB data will be stored in the table space where the associated table is placed.

► The Oracle data will be extracted onto the LINUX machine where DB2 and MTK are installed.

► All of the *currently converted* objects will be created on the target DB2 database.

► The Oracle data will be *extracted* from the Oracle source and stored on the target DB2 file system

► The Oracle data, stored on the DB2 file system, will be *loaded* into DB2.

To execute the plan outlined above, the appropriate information must be entered on the Deployment screen:

► **DB2 database name**
  db2_emp

► **Use local database**
  Selected

► **Use your system current user ID and password**
  Unselected

► **User ID**
  db2inst1

► **Password**
  Enter your password

► **Launch tabs_views_seq.db2 in the database**
  Checked

► **Extract and store data on this system**
  Checked

► **Load data to target database using generated scripts**
  Checked

Figure 4-24 on page 132 shows the Deploy to Target screen after all the pertinent information has been entered.
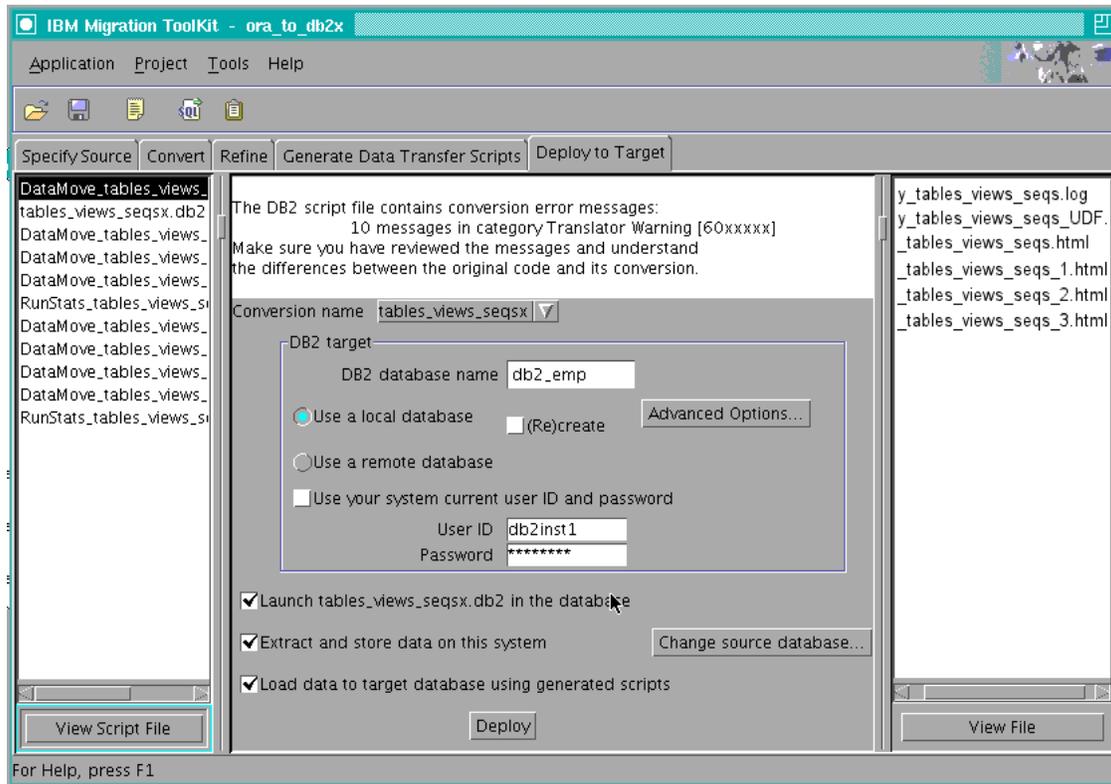
Figure 4-24   The Deploy to Target screen with relevant information entered

Once the deployment options are completed, clicking **Deploy** will begin the task.

Recommendations for Deploy to Target Advanced Options

When executing a deployment it is possible to specify advanced options that will influence actions that are taken during the deployment. When selecting **Advanced Options** on the Deploy to Target panel, another panel opens that lets you specify the following:

► Territory
► Code Set
► Collating Sequence
► Deploy MTK UDFs
► Execute Runstats on the DB2 catalog
► Execute Runstats on the DB2 data after load/import

Of these options, Territory, Code Set, and Collating Sequence will directly affect database creation. As a result, when creating databases using MTK, it may be

important to alter these parameters to create the appropriate repository for your data. When creating a UNICODE database, for example, it would be necessary to specify TERRITORY US and Code Set UTF-8. For our example, since the database will be created outside of MTK, these options are irrelevant.

The Deploy MTK UDFs option need only be specified *once*, when the initial objects are created in the target server. Subsequent executions of this option are *usually* unnecessary unless the DDL for these objects has changed. For this reason, we recommend that this option be unchecked during subsequent object or data deployment.

The Execute Runstats options are recommended when LOADing or IMPORTing data; they are unnecessary when the deployment only involves objects.

Figure 4-25 shows the Advanced Options panel of the Deploy to Target task.
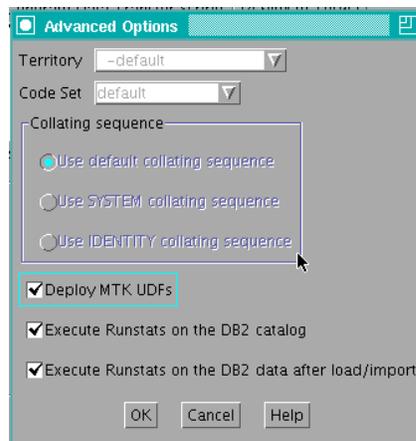
Figure 4-25   The Advance Options panel of the Deploy to Target task

Deployment of Java and SQL User-Defined Functions

During deployment, MTK will automatically install Java and SQL User-Defined Functions (UDFs). During installment, a schema named ORA is created to contain these functions.

> **Note:** The Java and SQL UDFs are provided to simulate Oracle functions that do not have exact DB2 equivalents. This implementation strives to guarantee that the results of executing the code in the source environment will yield the same results as when executed in the target environment.
>
> The UDFs are programmed in SQL where possible but, when necessary, they are implemented in Java.

The installation of these functions in a database involves one to two files for every Oracle MTK target:

► A script file (mtkora.udf): containing a template for SQL UDF's source code
► A .jar file (oraUDFs.jar): containing the source code for Java UDFs

These files are found in the directory where MTK was installed. Should it be necessary to drop the functions created by the mtkora.udf script, this directory will also contain a script (mtkoradrop.udf) to drop them.

During deployment, the following takes place with these .jar and script files:

1. In the mtkora.udf file:

   The Java CREATE FUNCTION statements are altered to specify the database under which the jar will be installed. See Example 4-2.

Example 4-2   Examples of changes made to the mtkora.udf file

```
...
CREATE FUNCTION ORA.TO_DATE(dateStr VARCHAR(60))
 RETURNS TIMESTAMP
 EXTERNAL NAME 'ora.udfjar:com.ibm.mtk.udf.oracle.OraDB2UDFsv2.to_date'
 LANGUAGE java
 PARAMETER STYLE JAVA
 DETERMINISTIC
 FENCED
 NO EXTERNAL ACTION
 NO SCRATCHPAD
 NO FINAL CALL
 NO SQL
 NO DBINFO
...
After alteration:
...
CREATE FUNCTION ORA.TO_DATE(dateStr VARCHAR(60))
 RETURNS TIMESTAMP
 EXTERNAL NAME 'ora.db2_emp:com.ibm.mtk.udf.oracle.OraDB2UDFsv2.to_date'
 LANGUAGE java
 PARAMETER STYLE JAVA
 DETERMINISTIC
 FENCED
 NO EXTERNAL ACTION
 NO SCRATCHPAD
 NO FINAL CALL
 NO SQL
 NO DBINFO
...
```

Note that the line 'ora.udfjar:com... was changed to specify the name of the database under which the UDFs will be installed: ora.db2_emp:com.

MTK saves the altered version of this file in the PROJECTS directory under a subdirectory corresponding to your project name, for example:

```
PROJECTS/<MY_MTK_PROJECT_NAME>/mtkora.udf
```

2. During deployment, if **Deploy MTK UDFs** is selected, MTK installs the oraUDFs.jar file in the database. This is achieved with the following command executed by an MTK script:

```
CALL
SQLJ.INSTALL_JAR('file:/home/db2inst1/mtk/oraUDFs.jar','ora.db2_emp')
```

Once the jar file is installed on UNIX, it can be found in the following directory:

```
$DB2INSTHOME/sqllib/function/jar/ORA/
```

The DEPLOY_YourProjectName_Udf.log file contains all the information regarding the success or failure of UDF deployment in your environment. It is recommended that this be examined to determine whether the UDFs have been successfully deployed. In our example, a successful deployment returned the following message:

```
Creation of MTK UDFs...
CALL SQLJ.INSTALL_JAR('file:/home/db2inst1/mtk/oraUDFs.jar','ora.db2_emp')
DB20000I    The CALL command completed successfully.
```

If any other message is returned, your DB2 Java environment may be installed incorrectly. In these cases refer to either *IBM Migration Toolkit Users Guide and Reference,* which is included with the product download, or *Developing Java Applications*, SC10-4233 for Java setup assistance.

4.8.3 Deployment results

When the deployment process has completed, the Verification report opens. This report will give a clear indication of the status of your conversion project. It shows:

► The number of objects in the source file

► The number of objects successfully deployed into DB2

► The number of objects missing from DB2

  – Objects missing from DB2 appear in *red*. This usually indicates some problem during the deployment stage.

  – Warning messages are presented in *purple*.

  – Successfully deployed item are shown in *black*.

► The number of foreign keys found in DB2

- ▶ Information about Unique, Primary, and Check constraints
- ▶ The name of an object in the source database, and the name as it appears in the DB2 database
- ▶ The DB2 schema into which the object or data was deployed
- ▶ The type of object
- ▶ Whether a specific object was successfully deployed into DB2
- ▶ Whether data was successfully extracted from the source
- ▶ Whether data was successfully loaded into DB2

Figure 4-26 shows a portion of the Verification report for our conversion example.

Verification report from tables_views_seqs

To see why objects or rows are missing, review target database log here.

32 objects checked

32 objects found in target database

0 objects missing from target database

7 foreign keys found in target database

Unique, Primary and Check Constraints

32 objects checked

Source NAME	DB2 NAME	DB2 SCHEMA	TYPE	In DB2	Data extracted	Dat Dl
EMPLOYEE_SEQUENCE	EMPLOYEE_SEQUENCE	DB2INST1	SEQUENCE	true		
OFFICE_SEQUENCE	OFFICE_SEQUENCE	DB2INST1	SEQUENCE	true		
ACCOUNTS	ACCOUNTS	DB2INST1	TABLE	true	9 rows	9 rows
ACCOUNTS_DEPT_CODE_ACCT_ID	ACCOUNTS_DEPT_COD1	DB2INST1	PRIMARYKEY	true		
FK_ACC_DEPT_CODE	FK_ACC_DEPT_CODE	DB2INST1	FOREIGNKEY	true		
IND_ACCT_ID	IND_ACCT_ID	DB2INST1	INDEX	true		
DEPARTMENTS	DEPARTMENTS	DB2INST1	TABLE	true	5 rows	5 rot

Figure 4-26   The Verification report

Additional information about the conversion may be found by viewing:

- ▶ The Deploy_your_file_name.log from the left panel of the Deploy to Target panel.
- ▶ Choosing **Other reports for this conversion** from the Verification report panel.

► Selecting **Tools** → **Migration reports** on the MTK menu bar. You can navigate to the reports from current or previous migrations using the contents listing in the left frame of the browser window.

The name and information for each report are as follows:

► Conversion summary report

The conversion summary report presents details on a particular conversion. You will most likely have many conversions during your migration of each database. You can view each of the conversion reports and compare the objects that were successfully converted, as well as various errors reported each time.

► Translation error message reports

The Translation error message report details specific error messages found during each translation. These reports present the information in two ways: 1) error messages grouped by object type and 2) error messages grouped by message type. These reports can be useful when determining how to prioritize your work, for example, if errors with the same message number exist for many objects, it might be wise to solve the problem associated with that error before continuing. Or, when viewing the messages by category, you might choose delay solving issues related to the translator information category until errors related to more significant categories have been resolved.

► Estimate of table size report

This report presents an estimate of table sizes. This is useful in determining correct table space size during database creation or adjustments that might need to be made to a table space that will ensure a correct deployment.

► Large statement warnings report

During conversion some SQL statements are too long to be converted. The large statement warnings report lists each of the statements that cannot be converted for this reason.

► Verification report and deployment log

The verification report and deployment log generate messages relating to the overall status of the deployment. These messages may regard either information or problems that might have occurred during deployment.

If *in-context* files have been specified, which contain objects that have not yet been deployed into DB2, both the report and log will show that the in-context files failed to deploy. By design, objects contained within in-context files are not deployed.

4.9 Next steps

The first part of our migration example, that is, converting all of the objects upon which the other objects (stored procedures, functions, packages, and triggers) depend, is now completed. Based on information produced in the Verification report, it is determined that all tables, views, and sequence objects from the original Oracle database were converted correctly, and more importantly, deployed into DB2. Additionally, the report verifies that data was extracted from Oracle, and that the same number of rows were loaded into DB2. At this juncture, the process of converting the stored procedures, functions, packages, and triggers may begin.

Strategy

The strategy for the remainder of the project is:

- ▶ Convert the remaining objects.
- ▶ Examine and evaluate the messages from the translator.
- ▶ Deploy the remaining objects into DB2.
- ▶ Manually convert objects that are not automatically converted by MTK.

4.10 Converting the remaining objects

To convert the remaining objects, return to the Convert tab. It is significant to remember that the converter needs to "know" the definitions for the previously converted tables, sequences, and views to correctly convert the remaining objects. To accomplish this, these files will be designated as *context files* for the next part of the conversion. To specify objects as context files, complete the following:

- ▶ Click **Set Context** on the Convert tab.

- ▶ The Set Context screen opens, displaying two panels. The left panel is labeled Source Files. Under Source Files all of the .src files that have been extracted from the source are shown. The right pane is labeled "Selected context files." When first opened, there will be no files indicated under this heading.

- ▶ To designate context files, choose one (or several) files in the right-hand panel and then click **>**. This will bring the selected file(s) into the Selected context files panel. Figure 4-27 on page 139 shows the Set Context screen after **tabs_views_seqs.src** has been selected as a context file.
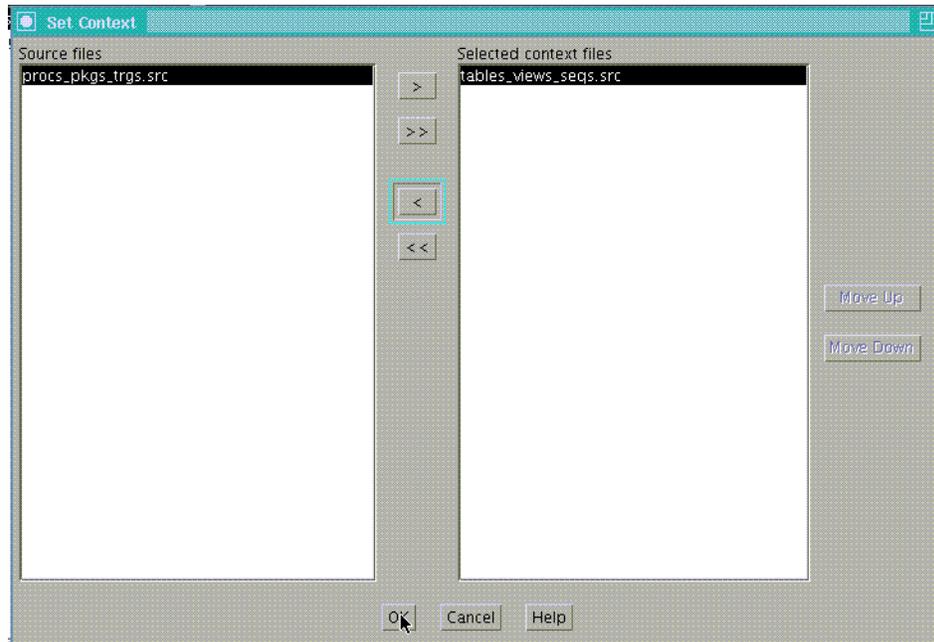
Figure 4-27   The Set Context screen after a file has been selected as Context

Once the context selections have been made, clicking **OK** returns the Convert tab.

On the Convert tab, the tabs_views_seqs.src file has the indication (context) beside it. This specifies that the file is now a context file; it will be used as a dependent file for the conversion, but it will not be reconverted.

To begin conversion, complete the required steps:

1. Select the files in the left-hand panel that will be converted. We choose the **procs_pkgs_trgs.src** file.

2. Enter a prefix for the generated files (that is, enter a name for the file that will be generated from the conversion). We accept the default.

3. Click **Convert** to begin the conversion.

Figure 4-28 shows the Convert tab, before conversion, with the context file designated and all required steps completed.
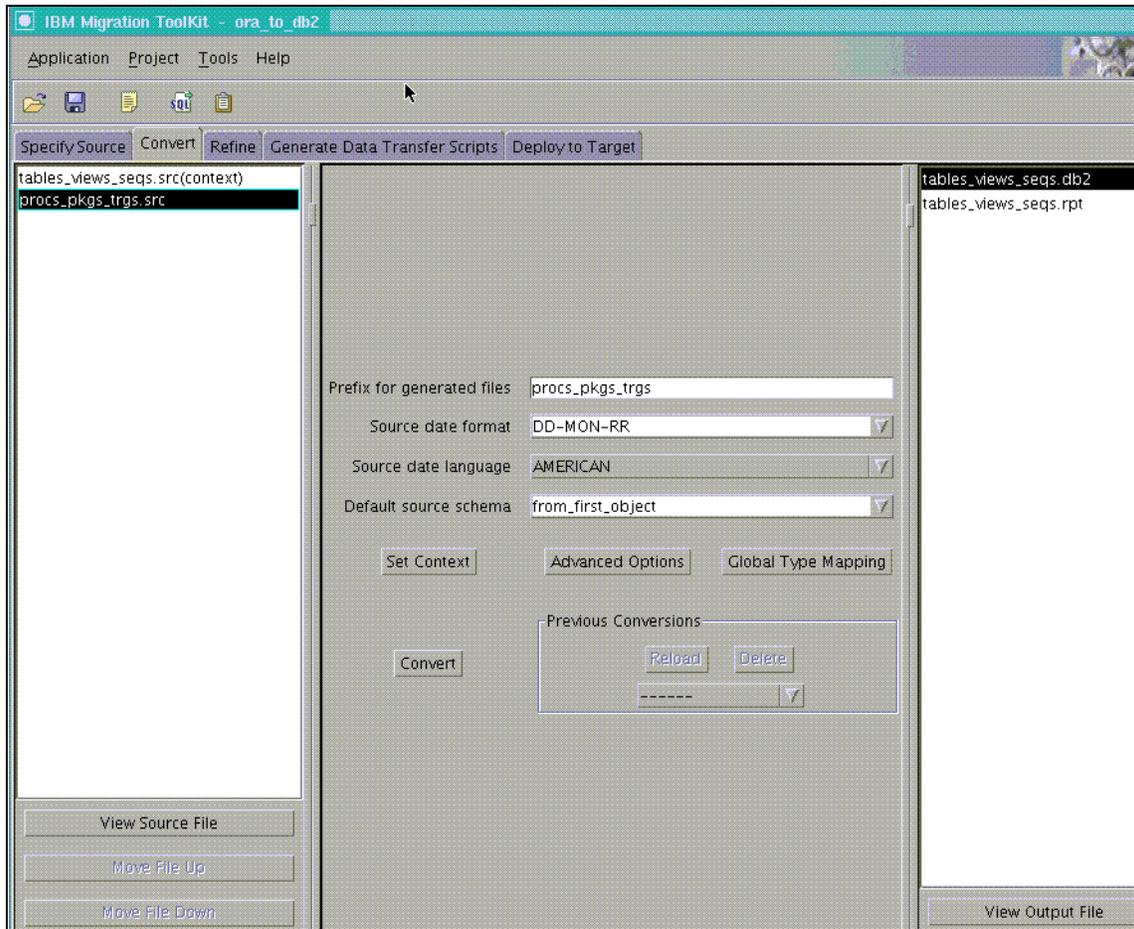
Figure 4-28   The Convert tab - with context files and the source file procs_pkgs_trgs selected

4.10.1 Translator Messages

In this section, we discuss the messages for the conversion of stored procedures, functions, packages, and triggers. Although messages from the previous conversion file will still be available for viewing, our discussion will *not* include them. When examining the messages, items that need to be converted manually will be noted as such. These items will then be discussed at a later point in this chapter.

Figure 4-29 on page 141 shows the Refine tab after the conversion of the procs_pkgs_trigs.src file. Messages in the following categories were received after this portion of the conversion:

► Input script error
► Translator error
► Translator Warning
► Translator information
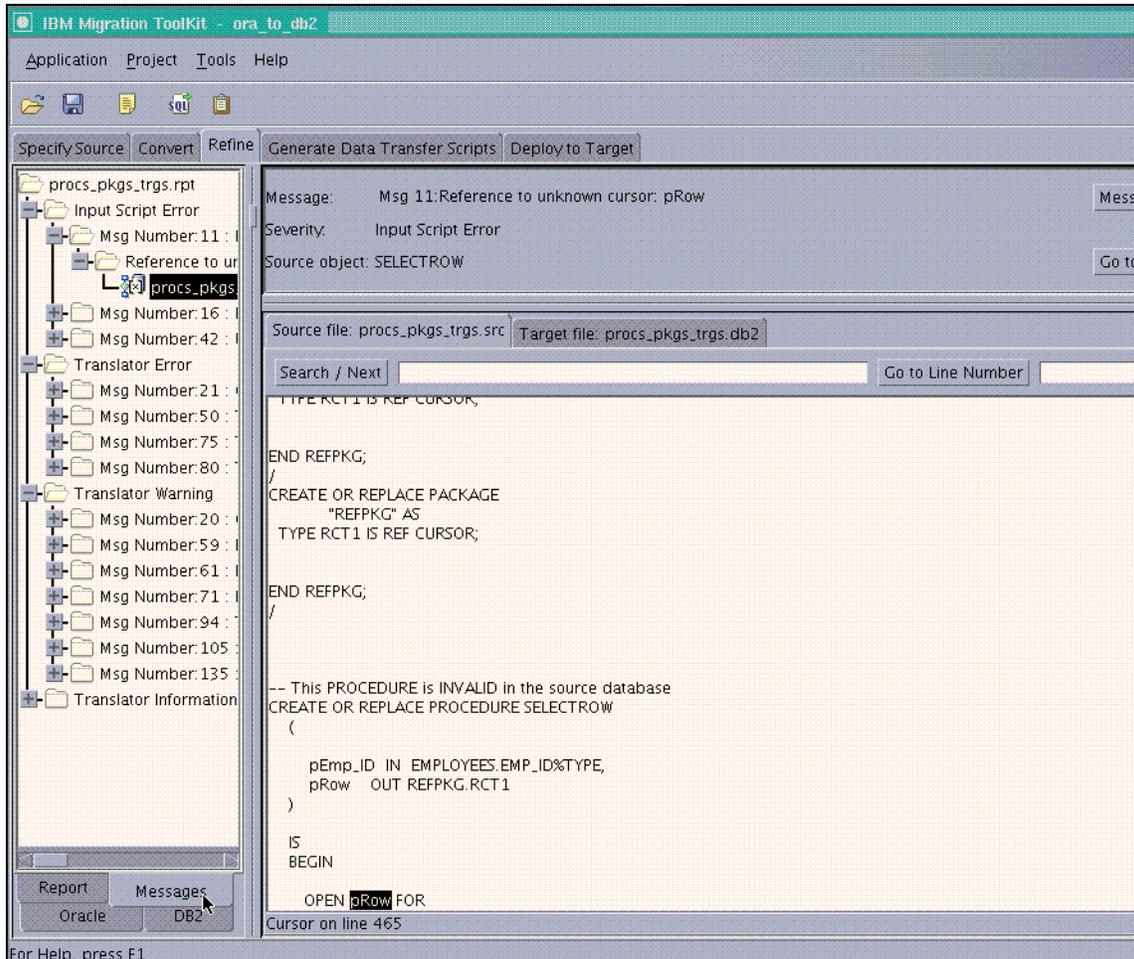
Figure 4-29   The Refine tab after the conversion of procs_pkgs_trigs.src

Input script error messages

There are two input script error messages

► Message 11

This message occurs once, and it refers to the stored procedure Selectrow.

```
Msg 11: Reference to unknown cursor: pRow
```

Message Help indicates:

```
The translator is not aware of the definition of this object.
Ensure that the definition exists in the file being translated or in one of
the files used as context for this translation. Also, ensure that the
schema name is specified as indicated.
```

When the statement is investigated, we recognize that pRow is an Oracle Reference Cursor. Oracle Reference Cursors have to be converted *manually* to the corresponding DB2 functionality. This is marked as manually converted, and will be discussed later in this chapter.

► Message 42

This message occurs once; it refers to the stored procedure Selectrow.

```
Msg 42: unrecognized data type: REFPKG.RCT1
```

Message Help indicates:

```
This data type is not recognized by the translator. The reason might
be that:
```
– It is not a valid Oracle data type.
– It is a user-defined type, not yet handled by the translator (for example, collections and object types).
```
If the data type is encountered as a parameter to a procedure, a
suggested translation is provided, but it is commented out because
it will probably require manual editing.
```

Upon investigation it is recognized that the data type REFPKG.RCT1 is an Oracle Reference Cursor. This is marked to be manually converted, and will be discussed later in this chapter.

Translator error messages

We have three translator error messages:

► Message 21

Fourteen instances of Message 21 occurred in the translator error category:

```
Msg 21: Call to Procedure DBMS_SQL.PARSE is not supported
Msg 21: Call to Procedure DBMS_SQL.DEFINE_COLUMN is not supported
Msg 21: Call to Procedure DBMS_SQL.COLUMN_VALUE is not supported
Msg 21: Call to Procedure DBMS_SQL.CLOSE_CURSOR is not supported
Msg 21: Call to Procedure DBMS_SQL.BIND_VARIABLE is not supported
Msg 21: Call to Procedure DBMS_SQL.OPEN_CURSOR is not supported
Msg 21: Call to Procedure DBMS_SQL.FETCH_ROWS is not supported
Msg 21: Call to Procedure DBMS_SQL.EXECUTE is not supported
```

The fourteen instances of this message all refer to the same object, the stored procedure EmployeeDynamicQuery.

Message Help indicates:

```
This Oracle function or procedure call is not translated to the target
server.
```

When the procedure is analyzed, the syntax is recognized as Dynamic SQL, which is implemented through the Oracle DBMS_SQL package. Dynamic SQL can be easily converted into DB2 functionality, but the conversion will have to be done *manually*. We mark this procedure for manual conversion and continue.

► Message 50

Two occurrences of Message 50 are found in this category:

Msg 50: This statement is not supported in the target server dynamic compound statement

The two occurrences of this message are both in regard to a cursor created in the trigger UpdateDepartments.

Message Help indicates:

```
This statement is not supported in the target server dynamic
compound statement. Dynamic compound statements are used as
bodies for top-level anonymous blocks, user-defined functions,
and triggers. Procedures use the target server compound
statements as bodies that are less restrictive. Some statements
that are not allowed inside target server dynamic compound
statement are:
— Nested blocks
— Statements containing CASE expressions
— GOTO
— Procedure calls
— Cursors
— COMMIT
— Exception handlers
If the compound statement is found in an Oracle function,
depending on the use of the function, changing it to a procedure
might be a better approach.
```

Since this type of cursor statement is *not allowed* inside a DB2 trigger, the cursor logic will need to be *manually* converted. The object is marked for manual conversion and the conversion continues.

► Message 75

One instance of Message 75 occurs in the translator error category:

Msg 75: This statement is not supported in the target server
Before Trigger
The message occurs in relation to the trigger InsertEmployee.

Message Help indicates:

This statement is not supported in a target server Before
Trigger.
The following statements are not supported in this context:
- INSERT
- DELETE
- UPDATE

After examining the trigger, it is seen that it includes DML statements (INSERT, UPDATE, DELETE). DML statements are restricted in Before triggers in DB2. The conversion, although not complicated, will also have to be done *manually.*

► Message 80

This message occurs once in the translator error category:

Msg 80: This package item is not translated.

This message occurs in relation to the Package object REFPKG.

Message Help indicates:

Only the following items are supported inside a package: function
specifications, functions, procedure specifications, procedures, and
constants. Variable declarations, cursor declarations and type
definitions are not translated.

After examining the source code it is determined that, although this package item will not be converted, it will not be necessary in our conversion. This occurs for two reasons:

– MTK converts Oracle schema in the recommended manner, that is, by converting the Oracle Package name to a DB2 Schema name.

– The package item, a reference cursor, will be handled when the procedure SelectRow is converted manually.

Translator Warning messages

The following Translator Warning messages were generated:

► Message 20

15 instances of Message 20 occurred in the Translator Warning category:

Message 20: Object name has been changed to <new name>.

This message pertains to the objects shown in Table 4-1.

Table 4-1   Message 20 objects

Source object name	Conversion name	Object type
ACCT_ID	ACCT_ID1	Correlation name
BAND	BAND1	Correlation name
c_RegisteredEmployees	c_RegisteredEmplo1	Cursor name
DEPT_CODE	DEPT_CODE1	Correlation name
EMP_MGR_ID	EMP_MGR_ID1	Correlation name
EmployeesOfficesInsert	EmployeesOfficesI1	Trigger (insert)
ManagersChange	ManagersChange_FO1	Trigger (insert)
	ManagersChange_FO2	Trigger (delete)
	ManagersChange_FO3	Trigger (update)
office_summary_delete	office_summary_de1	Trigger
UpdateDepartments	UpdateDepartments1	Trigger (insert)
	UpdateDepartments2	Trigger (delete)
	UpdateDepartments3	Trigger (update)
UpdateEmployees	UpdateEmployees_F1	Trigger (insert)
	UpdateEmployees_F2	Trigger (update)

Message Help indicates:

```
Object names that are too long for the target server are
truncated. Names that are reserved words in the target server are
enclosed in double quotes. Names that conflict with other names
in the target server (because the name is already in use) are
renamed.
```

Upon examination of the source code, we observed that names were altered for three types of objects:

– Triggers

Reason:
Trigger names were changed in two cases. First, because the trigger name exceeded the maximum number of characters (18) allowed, as in the case of EmployeesOfficesInsert and office_summary_delete. In this case, the names were truncated to conform to DB2 standards. Secondly, because *additional* triggers needed to be created. This occurs when an Oracle trigger specifies more than one operation (INSERT, UPDATE,

DELETE) for the source trigger. In these circumstances, DB2 requires
(and MTK creates) an individual trigger for each operation.

– Cursors

Reason:
DB2 specifies that cursor names cannot exceed 18 characters.

– Correlation names

Reason:
Renaming these variables is related to a message that is best explained
by the Message Help from message 71:

> The target server does not accept references to OLD from an
> inserting trigger or references to NEW from a deleting
> trigger. In the WHEN clause of the trigger these references
> are translated to NULL. In the body of the trigger they are
> translated to a variable generated for this purpose.

► Message 61

Two instances of Message 61 were generated in the Translator Warning
category:

> **Msg 61:** Parameter defaults are not supported in the target server
> procedure definitions. Calls to the procedure are adjusted
> accordingly.

The two instances occur in the procedure EmployeeDynamicQuery.

Message Help indicates:

> In procedure and function declarations, the optional DEFAULT
> value of a parameter is not translated, but the translator will
> use the value as necessary through the remainder of the
> translation.

After examining the source code we understand that the Oracle procedure
employs default parameters that may be used when the procedure is invoked.
Although this behavior is *not supported* in DB2, MTK accounts for it when the
procedure is called from *other* stored procedures.

Additional information adds the following explanation:

> ...Default values for parameters are not translated in the
> parameter list, but the converter remembers them and adds them to
> each procedure call when the corresponding argument is missing.

► Message 71

Eight instances of Message 71 occur in the Translator Warning category:

> **Msg 71:** Reference to OLD or NEW column translated to <NULL or Variable>

The eight instances occur in the ManagersChange trigger.

Message Help indicates:

> The target server does not accept references to OLD from an
> inserting trigger or references to NEW from a deleting trigger.
> In the WHEN clause of the trigger, these references are
> translated to NULL. In the body of the trigger, they are
> translated to a variable generated for this purpose.

► Message 94

There are two occurrences of Message 94 in this category:

> **Msg 94:** The function <function_name> is translated to the target
> server as a Procedure.

This message refers to the functions MaxProjects, CountProjects, and AverageBand.

Message Help indicates:

> This Oracle user-defined function is translated to the target
> server as a procedure. This happens with functions with
> parameters in OUT mode. Since this feature is not available in
> the target server, the translator uses a target server procedure
> instead. The calls to the Oracle function will be translated into
> procedure calls.

► Message 135

There is one occurrence of Message 135 in the Translator Warning category. This arises in the trigger ManagersChange.

> **Msg 135:** BEFORE trigger was translated to AFTER trigger.

Message Help indicates:

> BEFORE triggers are usually translated to NO CASCADE BEFORE
> triggers in DB2. Unfortunately this kind of trigger does not
> allow DML statements (INSERT, UPDATE and DELETE) and the FOR EACH
> STATEMENT clause.
> If the source trigger does use one of the features, the
> translator will try to translate the BEFORE trigger to an AFTER
> trigger, which does not have those restrictions.
> This special translation is not possible if the body of the
> trigger refers to the table as the object of the trigger, or if a
> column of the NEW table is assigned.

Translator Information

The Translator Information messages received in our conversion example are:

► Message 0

There is one occurrence of this message in this category:

   Msg 0: MTK Oracle Converter. Version: <mtk version>

Message Help indicates:

   Specifies the version of the Oracle converter.

The version number of the converter is specified.

► Message 34

Four instances of Message 34 occurred in the Translator Information category:

   Msg 34: No DB2 translation available, but statement has been taken into account

The four instances occurred in relation to the following statements:

– CONNECT ORA_USR

– In the Create Package AccountPackage statement for:
   • Create Procedure AddEmployee
   • Create Procedure RemoveEmployee
   • Create Procedure AccountList

Message Help indicates:

   There is no DB2 translation available, but the information in the statement will be used by the converter in translating the statements that follow.

After examining the source we understand that:

– Regarding CONNECT ORA_USR, the message indicates that although the connection statement is not expressly converted, the implications of the connection statement will be handled by DB2. ORA_USR will be used as the default schema for unqualified database object names.

– Regarding the Create Procedure statements, we recognize that Oracle packages will be converted to objects *within a specified schema.* This schema name will be the same as the original Oracle Package name.

► Message 108

There is one occurrence of this message in this category:

   Msg 108: Translation Ratio: <percentage>% (<absolute ratio> statements were translated successfully)

Message Help indicates:

```
This provides an assessment of the provided translation by giving
the ratio of Oracle statements translated without producing any
error message out of the total number of statements. This number
is provides a general indication regarding the success of the
automated translation and does not intend to give an exact and
accurate measure.
Statement here designates Oracle SQL and PL/SQL statements. For
instance, in a CREATE PROCEDURE, the whole SQL statement is
counted as 1 and each PL/SQL statement inside the body of the
procedure are also counted as one.
```

The translation ratio is reported as 84.67%—116 of 137 statements were translated successfully.

4.10.2 Status

We recommend that the collected information be evaluated at this juncture in the conversion. First, it must be established that for each Translator Message category all of the messages and implications of the messages are understood and accounted for. This evaluation will assist in obtaining a reasonable expectation as to the number and type of objects that will convert automatically. Likewise, it will assist in ascertaining the number and type of, and reason why, objects will need to be converted manually.

Given the information already gathered about the objects from our conversion, it is expected that *most* of the objects will deploy successfully. It is also expected, however, that after the completion of deployment several objects *will not* deploy successfully. We anticipate that the following objects will be in that category, and that manual intervention will be required before they can be successfully deployed:

- ▶ Stored procedures
  - – SelectRow
  - – EmployeeDynamicQuery

- ▶ Triggers
  - – InsertEmployee
  - – UpdateDepartments

4.11 Deployment of the remaining objects

The Generate Data Transfer Scripts task will be bypassed because scripts for extracting and loading data into DB2 were already created in an earlier part of

our conversion (refer to 4.7.1, "Creating unload and load scripts" on page 125").
The focus of the next set of tasks will center on deploying stored procedures,
functions, packages, and triggers into the DB2 database.

During this phase, the procs_pkgs_trgs.db2 file will be deployed. To begin this,
the procs_pkgs_trgs.db2 file is selected on the Generate Data Transfer screen
and the following required information is completed:

▶ DB2 database name:
  db2_emp

▶ Use a local database:
  Selected

▶ User ID
  db2inst1

▶ Password
  Enter db2inst1 password

▶ Launch *procs_pkgs_trgs.db2* in the database
  Selected
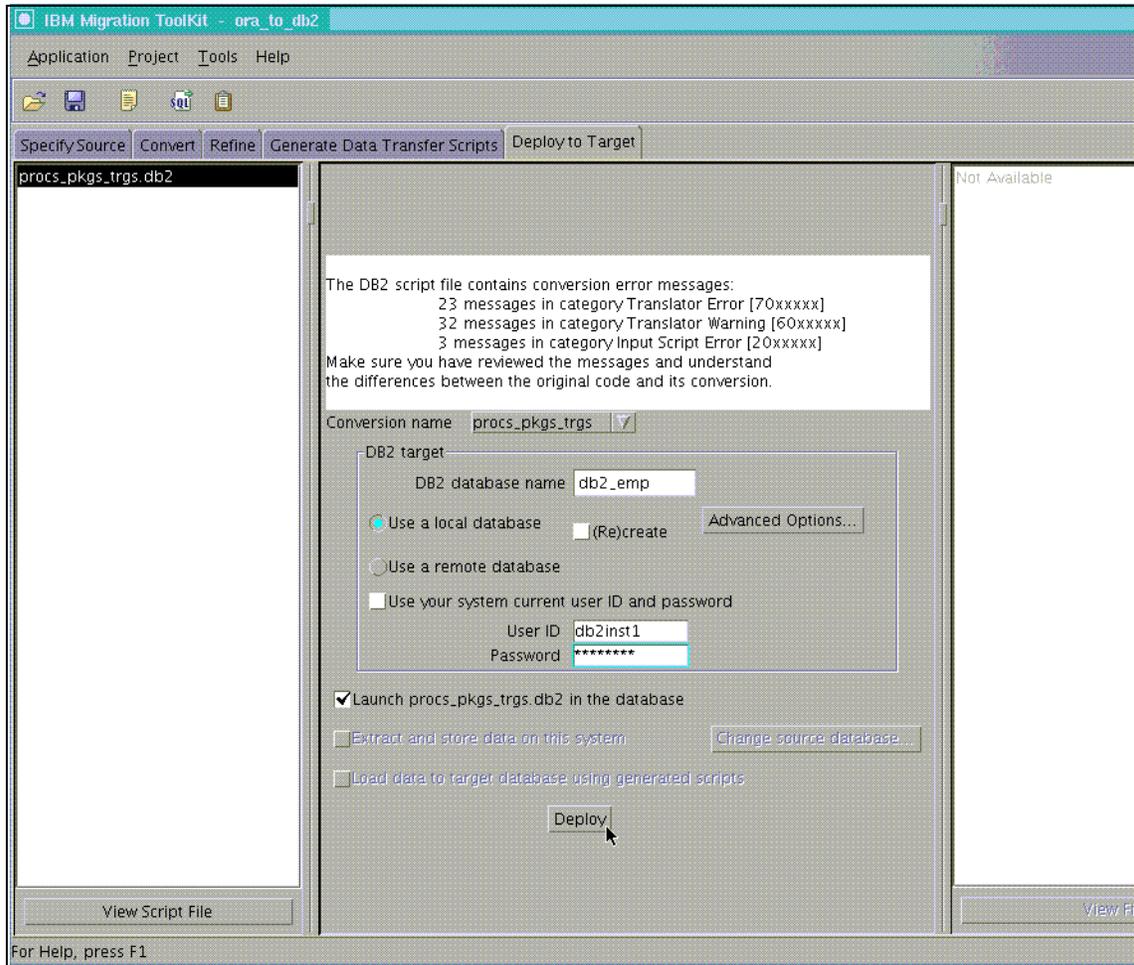
Figure 4-30   The deployment screen viewed before deploying procedures, packages, and triggers

4.11.1 Verification report

After deployment has completed, MTK returns to the Verification Report. It is important to remember that the information shown by the Verification Report is cumulative, that is, the total number of items includes objects from *all* parts of a conversion. In our example this includes objects from the conversion of tables, views, and sequences *as well as* procedures, packages, and triggers.

The information at the top of the Verification Report indicates that of the 57 total objects in the conversion, 50 were successfully deployed (see Figure 4-31).

Figure 4-31   The Verification Report showing the tally of deployed objects

The report also indicates that *7* objects are *missing from DB2*, that is, were *not* successfully deployed. If the hyperlink **7 objects missing from DB2** is clicked, another report opens that shows that the seven missing objects are:

► Procedures:

– EmployeeDynamicQuery
– SelectRow

► Triggers:

– CreateEmployeeID
– InsertEmployee
– UpdateDepartments:

   Three triggers were created, one each for INSERT, UPDATE, and DELETE.

It was expected that most of the items on this list would fail deployment for reasons that were previously described. The only unforeseen event is the failure of the creation of the trigger CreateEmployeeID, see Figure 4-32.

Figure 4-32   The Verification Report showing objects missing from DB2

Because the generation of the Employee IDs (the EMP_ID) column in the
Employees table is now being implemented through an IDENTITY column, the
failure of the deployment of CreateEmployeeID is actually *irrelevant*; see
"Identity columns" on page 120. The creation of an IDENTITY column replaces
the necessity for the creation of the CreateEmployeeID trigger.

4.12 Manual conversion for ORA_EMP database objects

At this point, the conversion of all objects that can be successfully converted and
deployed *automatically* into DB2 by the MTK is completed. Indeed, the majority
of the object and data conversions have been accomplished. Yet, there are still
some "loose ends" such as objects that we noted for manual conversion that
must be dealt with.

The triggers and procedures from the sample ORA_EMP database that will be
converted manually are:

► Triggers:
    – InsertEmployee

- UpdateDepartments
- ▶ Procedures:
  - SelectRow
  - EmployeeDynamicQuery

The objects in the preceding list of triggers and procedures use Oracle features that require a different implementation in DB2, or are not supported by DB2. Some of these objects can be converted by using equivalent DB2 features, others by changing the source code logic. In this section feature differences, as well as potential ways of converting and deploying triggers and procedures that implement these features from Oracle to DB2, are shown.

Both Oracle and DB2 have triggers. There are, however, some differences in the implementation of triggers in these databases. Because of these differences manual conversion is required. In this section, conversion and deployment of triggers are shown using two trigger examples from our database:

- ▶ InsertEmployees
- ▶ UpdateDepartments

Example 1: InsertEmployee

When the deployment log created by MTK (Deploy_procs_pkgs_trgs.log) is opened, the following error message, generated by DB2 during the object deployment, is seen:

```
DB21034E  The command was processed as an SQL statement because it
was not a valid Command Line Processor command.  During SQL
processing it returned:
SQL0797N  The trigger "DB2INST1.INSERTEMPLOYEE" is defined with an
unsupported triggered SQL statement.  LINE NUMBER=38.  SQLSTATE=42987
```

In this example, the unsupported triggered SQL statement refers to an UPDATE that DB2 does not permit in a BEFORE trigger. The correct conversion of the InsertEmployee trigger requires that the BEFORE trigger be converted to an AFTER trigger.

Example 4-3 shows the Oracle source code for trigger InsertEmployee.

Example 4-3 Trigger InsertEmployee Oracle source code

```
CREATE TRIGGER InsertEmployee
  BEFORE INSERT ON employees
  FOR EACH ROW
DECLARE
  v_num_projects accounts.num_projects%TYPE;
BEGIN
  SELECT num_projects
```

```
    INTO v_num_projects
    FROM accounts
    WHERE dept_code = :new.dept_code
    AND acct_id = :new.acct_id;
  UPDATE accounts
    SET current_employees = current_employees + 1
    WHERE dept_code = :new.dept_code
    AND acct_id = :new.acct_id;
END InsertEmployees;
```

Example 4-4 shows the converted DB2 code of trigger InsertEmployee.

Example 4-4   Trigger InsertEmployee DB2 conversion code

```
CREATE TRIGGER InsertEmployee
AFTER INSERT ON employees
REFERENCING NEW AS new
FOR EACH ROW
MODE DB2SQL
BEGIN ATOMIC
    DECLARE v_num_projects SMALLINT;
    SET (v_num_projects) = (SELECT "NUM_PROJECTS"
            FROM ACCOUNTS
            WHERE "DEPT_CODE" = NEW."DEPT_CODE"
                AND "ACCT_ID" = NEW."ACCT_ID");
    UPDATE ACCOUNTS
            SET "CURRENT_EMPLOYEES" = "CURRENT_EMPLOYEES" + 1
            WHERE "DEPT_CODE" = NEW."DEPT_CODE"
                AND "ACCT_ID" = NEW."ACCT_ID";
END
```

> **Important:** Note that in some cases you *cannot* change a BEFORE trigger to an AFTER trigger. For example, if an application inserts a row in table A that needs a parent row in table B and the parent row in B has to be created/generated by the trigger, then a BEFORE trigger is needed. In this case the solution will need to be implemented through application logic.

Example 2: UpdateDepartments

This trigger employs an explicit CURSOR declaration, which is not permitted in a DB2 trigger. This example demonstrates how to change the *explicit* cursor (c_projects) to an *implicit* cursor using a FOR loop. This is accomplished by putting the cursor's SELECT statement directly into the FOR loop statement; this is shown in Example 4-6.

Example 4-5 shows the Oracle source code of trigger UpdateDepartments.

Example 4-5   Trigger UpdateDepartments Oracle source code

```
CREATE TRIGGER UpdateDepartments
AFTER INSERT OR DELETE OR UPDATE ON employees
DECLARE
  CURSOR c_projects IS
    SELECT dept_code
          ,COUNT(*) AS total_employees
          ,SUM(current_projects) AS total_projects
      FROM employees
      GROUP BY dept_code;
BEGIN
  FOR v_project_rec in c_projects LOOP
    UPDATE departments
    SET  total_projects  = v_project_rec.total_projects
        ,total_employees = v_project_rec.total_employees
    WHERE dept_code = v_project_rec.dept_code;
  END LOOP;
END UpdateDepartments;
```

The DB2 conversion of trigger UpdateDepartments is shown in Example 4-6.

Example 4-6   DB2 conversion of trigger UpdateDepartments

```
CREATE TRIGGER UpdateDepartments1
AFTER INSERT ON EMPLOYEES
REFERENCING NEW_TABLE AS new
FOR EACH STATEMENT
MODE DB2SQL
BEGIN ATOMIC
  X:                                                        -- [1]
  for ROW as                                                -- [2]
    SELECT dept_code                                        -- [3]
          ,COUNT(*) AS total_employees
          ,SUM(current_projects) AS total_projects
    FROM employees group by dept_code
  DO
    UPDATE departments                                      -- [4]
    SET    total_projects  = row.total_projects
          ,total_employees = row.total_employees
    WHERE dept_code = row.dept_code;
  END FOR X;
END
```

Notes

► [1]: **"X"** is the specified label for the FOR statement.

► [2]: The for-loop-name (ROW) is used to qualify the column names returned by the specified SELECT statement.

► [3]: In a trigger, function, method, or dynamic compound statement, the SELECT statement must consist of only a FULL SELECT with optional common table expressions.

► [4]: This section specifies a statement (or statements) that will be invoked for each row of the table.

> **Note**: For complete information regarding FOR loop syntax and usage consult the DB2 manual *SQL Reference Volume 2,* SC10-4250.

Manual deployment of triggers

The triggers InsertEmployee, and UpdateDepartments are deployed into DB2 from a command window. Here is the process:

1. After creating the trigger, save the script file on your file system.

   In order to execute multiple commands, your script file *must* end with a terminating character other than the default semi-colon (`;`). Some typical termination character choices are `!`, or `@`. MTK uses `!`.

2. Open a DB2 Command Window in the directory where the converted procedure source resides:

   – To open a DB2 Command Window on Windows:

     Click **Start** and select **Programs** → **IBM DB2** → **Command Window.**

   – To open a DB2 Command Window on UNIX:

     Open any operating system Command Window.

3. Connect to the database:

   ```
   db2 connect to your_database_name
   ```

   or

   ```
   db2 connect to your_database_name USER your_userid USING
   your_password
   ```

4. Execute the following from the Command Window. The termination character must be specified:

   ```
   db2 –td! –vf  your_script_file_name
   ```

5. We recommend to pipe the output to another file so that any messages generated during the creation may be viewed at a later time. This may be done as follows:

   ```
   db2 –td! –vf  your_script file_name >  your_output_file_name
   ```

Here are the steps as they would be executed for the trigger InsertEmployee. Although not shown here, the steps will be the same (except for file names) for the remaining triggers UpdateDepartments1/2/3.

1. The source code is saved as InsertEmployee.db2; see Example 4-7.

Example 4-7   InsertEmployee.db2

```
CREATE TRIGGER InsertEmployee
  AFTER INSERT ON employees
  REFERENCING NEW AS new
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    DECLARE v_num_projects SMALLINT;
    SET (v_num_projects) = (SELECT "NUM_PROJECTS"
                             FROM ACCOUNTS
                             WHERE "DEPT_CODE" = NEW."DEPT_CODE"
                             AND "ACCT_ID" = NEW."ACCT_ID");
    UPDATE ACCOUNTS
          SET "CURRENT_EMPLOYEES" = "CURRENT_EMPLOYEES" + 1
          WHERE "DEPT_CODE" = NEW."DEPT_CODE"
          AND "ACCT_ID" = NEW."ACCT_ID";
  END !  --** the exclamation point (!) is the termination character
```

2. Open a DB2 Command Window.

3. Connect to the database:

   db2 connect to db2_emp

   or

   db2 connect to *your_database* USER *your_userid* USING *your_password*

4. Once the following command is executed, the results are piped to the message.out file:

   db2 –td! –vf InsertEmployee.db2 **>** message.out

5. DB2 responds with the message:

   DB20000I *The SQL command completed successfully.*

The message.out file should be viewed for messages, especially if any other message than The SQL command completed successfully is returned*.*

4.12.1 Stored procedures

In our example Oracle database there are five stored procedures (not including stored procedures that reside in packages). MTK has converted three of these stored procedures *automatically* and encountered conversion issues that prohibited the automatic conversion of the procedures of the remaining two,

EmployeeDynamicQuery and SelectRow. When the Migration Reports and the log file were examined, it was determined that these two procedures use Oracle features that require manual intervention to convert them to the format that can be accepted by DB2.

Example 1: EmployeeDynamicQuery

This example shows how to convert Oracle Dynamic SQL, implemented through the DBMS_SQL package to equivalent DB2 code. In the example the Oracle source is shown first, and the corresponding DB2 conversion follows. The converted source code is accompanied by some explanatory notes.

Example 4-8 shows the Oracle source code for the procedure EmployeeDynamicQuery.

Example 4-8   EmployeeDynamicQuery Oracle source code

```
CREATE PROCEDURE EmployeeDynamicQuery (
  p_department1 IN employees.department%TYPE DEFAULT NULL,
  p_department2 IN employees.department%TYPE DEFAULT NULL) AS
  v_CursorID   INTEGER;
  v_SelectStmt VARCHAR2(500);
  v_FirstName  employees.first_name%TYPE;
  v_LastName   employees.last_name%TYPE;
  v_Department employees.department%TYPE;
  v_Dummy      INTEGER;
BEGIN
   v_CursorID := DBMS_SQL.OPEN_CURSOR;
   v_SelectStmt := 'SELECT first_name, last_name, department
                    FROM employees
                    WHERE department IN (:d1, :d2)
                    ORDER BY department, last_name';
  -- Parse the query.
  DBMS_SQL.PARSE(v_CursorID, v_SelectStmt, DBMS_SQL.NATIVE);
  -- Bind the input variables.
  DBMS_SQL.BIND_VARIABLE(v_CursorID, ':d1', p_department1);
  DBMS_SQL.BIND_VARIABLE(v_CursorID, ':d2', p_department2);
  DBMS_SQL.DEFINE_COLUMN(v_CursorID, 1, v_FirstName, 20);
  DBMS_SQL.DEFINE_COLUMN(v_CursorID, 2, v_LastName, 20);
  DBMS_SQL.DEFINE_COLUMN(v_CursorID, 3, v_Department, 30);
  v_Dummy := DBMS_SQL.EXECUTE(v_CursorID);
  LOOP
    IF DBMS_SQL.FETCH_ROWS(v_CursorID) = O THEN
      EXIT;
    END IF;
    DBMS_SQL.COLUMN_VALUE(v_CursorID, 1, v_FirstName);
    DBMS_SQL.COLUMN_VALUE(v_CursorID, 2, v_LastName);
    DBMS_SQL.COLUMN_VALUE(v_CursorID, 3, v_Department);
    INSERT INTO temp_table (char_col)
```

```
        VALUES (v_FirstName || ' ' || v_LastName || ' is a ' ||
                v_Department || ' department.');
    END LOOP;
    DBMS_SQL.CLOSE_CURSOR(v_CursorID);
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_SQL.CLOSE_CURSOR(v_CursorID);
      RAISE;
END EmployeeDynamicQuery;
```

Example 4-9 shows the procedure EmployeeDynamicQuery after it has been converted into a format acceptable to DB2. Some of the statements have been numbered in order to provide a more complete explanation of the solution.

Example 4-9   Converted DB2 code of Procedure EmployeeDynamicQuery

```
CREATE PROCEDURE EmployeeDynamicQuery (IN p_department1 VARCHAR(30),
                    IN p_department2 VARCHAR(30) )                  --[1]
    LANGUAGE SQL
    BEGIN
      DECLARE v_FirstName VARCHAR(20);
      DECLARE v_LastName VARCHAR(20);
      DECLARE v_Department VARCHAR(30);
      DECLARE at_end SMALLINT DEFAULT 0;                            --[2]
      DECLARE v_SelectStmt VARCHAR(500);                           --[3]
      DECLARE v_Cursor_stmt STATEMENT;                              --[4]
      DECLARE v_Cursor cursor for v_Cursor_stmt;
      DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING, NOT FOUND  --[5]
      BEGIN
        set at_end =1;                                             --[6]
        RESIGNAL;
      END;
      SET v_SelectStmt = 'SELECT first_name, last_name, department
                          FROM employees
                          WHERE department IN (?, ?)
                          ORDER BY department, last_name';         --[7]
      PREPARE v_Cursor_stmt FROM v_SelectStmt;                     --[8]
      OPEN v_Cursor USING p_department1, p_department2;            --[9]
      WHILE (at_end = 0) DO                                        --[10]
        FETCH v_cursor into v_FirstName,v_lastName, v_Department;  --[11]
        INSERT INTO TEMP_TABLE ("CHAR_COL")
        VALUES (COALESCE(v_FirstName, '') ||' ' ||
                COALESCE(v_LastName, '') || ' is in the ' ||
                COALESCE(v_Department, '') || ' department.');
      END WHILE;
      COMMIT;
END
```

Notes

- ► [1]: The Oracle %TYPE variables are converted to the base data types from the corresponding DB2 tables.

- ► [2]: The variable at_end is declared to hold a value that will be set when the EXIT Handler is executed at [5].

- ► [3]: The v_SelectStmt variable is declared as varchar(500) so that it will be large enough   to hold the SQL statement for the declared cursor at [7].

- ► [4]: A statement object is declared to hold the prepared form of v_SelectStmt at   [8].

- ► [5]: An Exit Handler is declared. This will execute when NO DATA FOUND during the Fetch statement at [11].

- ► [6]: When it executes, the Exit Handler sets the value of at_end to 1. This value will be checked to determine if the WHILE loop at [10] should continue.

- ► [7]: Before opening the cursor at [9] the v_CursorStmt object is prepared.

Example 2: SelectRow

The conversion of SelectRow presents the issue of converting Oracle cursor variables. When the cursor variable is an OUT parameter, it can usually be converted to DB2 using a *Dynamic Result Set*. In general, although manual, this is a very simple conversion.

Example 4-10 shows the Oracle source code of procedure SelectRow.

Example 4-10   SelectRow Oracle source

```
CREATE PROCEDURE SELECTROW
    (pEmp_ID  IN  EMPLOYEES.EMP_ID%TYPE,
        pRow    OUT REFPKG.RCT1)
    IS
    BEGIN
        OPEN pRow FOR
        SELECT FIRST_NAME, LAST_NAME, DEPARTMENT, BAND
        FROM EMPLOYEES
        WHERE Emp_ID = pEmp_ID;
    END;
```

Example 4-11 shows the DB2 conversion of the procedure SelectRow. Following the example is an explanatory note.

Example 4-11   SelectRow DB2 conversion

```
CREATE PROCEDURE SELECTROW (IN pEmp_ID INTEGER)
```

```
LANGUAGE SQL
DYNAMIC RESULT SETS 1                                        -- [1]
BEGIN
  DECLARE v_Cursor cursor WITH RETURN                        -- [2]
    TO CALLER for                                            -- [3]
    SELECT first_anme,
           last_name,
           department,
           band
    FROM employees
    WHERE emp_id = pEmpID;

  OPEN v_Cursor ;                                            --[4]
END
```

Notes

► [1] DYNAMIC RESULT SETS: This clause specifies the maximum number of result to be returned.

► [2] WITH RETURN: This clause indicates that the cursor is intended for use as a result set from a stored procedure.

► [3] TO CALLER: Specifies that the cursor can return a result set to the caller. For example, if the caller is another stored procedure, the result set is returned to that stored procedure. If the caller is a client application, the result set is returned to the client application.

► [4] The cursor v_Cursor is opened, and stays open, to return the Result set.

4.12.2 Manual deployment of stored procedures

The stored procedures EmployeeDynamicQuery and SelectRow are deployed into DB2 from a Command Window. Here is the process:

► After creating the procedure, save the script file on your file system:

In order to execute multiple commands, as in a stored procedure, your script file must end with a terminating character other than the default semi-colon (;). Some typical termination character choices are **!**, or **@**.

► Open a DB2 Command Window in the directory where the converted procedure source resides:

– To open the DB2 Command Window on Windows:

Click **Start** and select **Programs** → **IBM DB2** → **Command Window.**

– To open the DB2 Command Window on UNIX, open any operating system Command Window.

► Connect to the database using the command:

```
db2 connect to your_database_name
```

or

```
db2 connect to your_database_name USER your_userid USING
your_password
```

► Execute the following from the Command Window (termination character must be specified):

```
db2 –td! –vf  your_script_file_name
```

► We recommend to pipe the output to another file so that any messages generated during the creation may be viewed at a later time. This may be done as follows:

```
db2 –td! –vf  your_script file_name > your_output_file_name
```

Here are the steps implemented for the procedure SelectRow. Although not shown here, the steps are the same (except for the file names) for the procedure EmployeeDynamicQuery:

► The script shown in Example 4-12 is created and saved as SelectRow.db2.

Example 4-12   DB2 stored procedure SelectRow

```
CREATE PROCEDURE SELECTROW (IN pEmp_ID INTEGER)
  LANGUAGE SQL
  DYNAMIC RESULT SETS 1
  BEGIN
    DECLARE v_sqlStmt VARCHAR (200);
    DECLARE v_stmt STATEMENT;
    DECLARE v_Cursor CURSOR WITH RETURN for v_stmt;
    SET v_sqlStmt = 'SELECT FIRST_NAME, LAST_NAME, DEPARTMENT, BAND
                     FROM EMPLOYEES
                     WHERE Emp_ID = ?';
    PREPARE v_stmt FROM v_sqlStmt;
    OPEN v_Cursor USING pEmp_ID;
  END !   ** the exclamation point (!) is the termination character.
```

► Open a DB2 Command Window.

► Connect to the database:

```
db2 connect to db2_emp
```

Execute the following command to pipe the output to the message.out file:

```
db2 –td! –vf Selectrow.db2 > message.out
```

► DB2 responds with the message:

```
DB20000I The SQL command completed successfully.
```

The message.out file should be viewed for messages, especially if any message other than `The SQL command completed successfully` is returned.

All the objects in our example Oracle database ORA_EMP have now been successfully converted and deployed into our DB2 database.

5

Conversion reference

In the previous chapter, we demonstrate how the IBM Migration Toolkit (MTK) is able to automate the conversion of most Oracle database objects and proprietary SQL to DB2. However, if your database uses many Oracle proprietary features, there may be some SQL that the MTK cannot automate. In this chapter, we show examples of how to perform conversions manually, with a focus on converting Oracle objects and features that are not automated by the MTK.

We start by exploring tools and methods for building DB2 triggers, functions, and procedures. Through examples, we look at techniques for converting frequently used PL/SQL features to DB2 SQL PL code. We also discuss how to build external procedures and functions using C/C++ and Java.

Although we start this chapter by demonstrating basic syntax for creating procedures, triggers, and functions in both Oracle and DB2, the intent is to demonstrate how to convert these objects manually when needed - not to be a complete reference for developing DB2 SQL PL (other books are available for that purpose). The reader is expected to have some prior application development experience in both Oracle PL/SQL and DB2 SQL PL in order to fully understand the examples used in this chapter.

5.1 Tools

There are two tools commonly used for manually creating or converting stored procedures, triggers, and functions:

► The Developer Workbench (DWB)
► The DB2 Command Window.

This section provides a brief explanation of each tool.

5.1.1 Developer Workbench

The Developer Workbench (DWB) is available as a free download in DB2 9. It is a visual tool that aids in the rapid development of DB2 business objects. This newly designed tool is based on the Eclipse framework and replaces the Development Center from previous versions of DB2.

The following tasks can be executed with DWB:

► Create, view, and edit database objects (such as tables and schemas).

► Explore and edit data in tables and rows.

► Visually build SQL and XQuery statements.

► Develop and deploy stored procedures, user defined functions (UDFs), routines, and scripts.

► Debug SQL and Java stored procedures.

► Develop SQLJ applications.

► Develop queries and routines for XML data.

► Perform data movement (such as load and extract).

► Collaborate and share projects with team members.

► Migrate projects from the DB2 version 8 DB2 Development Center.

Thorough examination of the capabilities of DWB is beyond the scope of this IBM Redbooks publication. Detailed information about this tool and its capabilities can be found in the tutorials:

► *DB2 Developer Workbench, Part 1: DW Concepts and Basic Tasks*.

► *DB2 Developer Workbench, Part 2: Developer Workbench and stored procedures*

► *DB2 Developer Workbench, Part 3: Developer Workbench and XML*

These tutorials may be downloaded from DeveloperWorks at the following Web site:

```
http://www.ibm.com/developerworks/views/db2/libraryview.jsp?search_by=DB2+developer+workbench+(DWB)
```

Figure 5-1 shows the Developer Workbench interface.
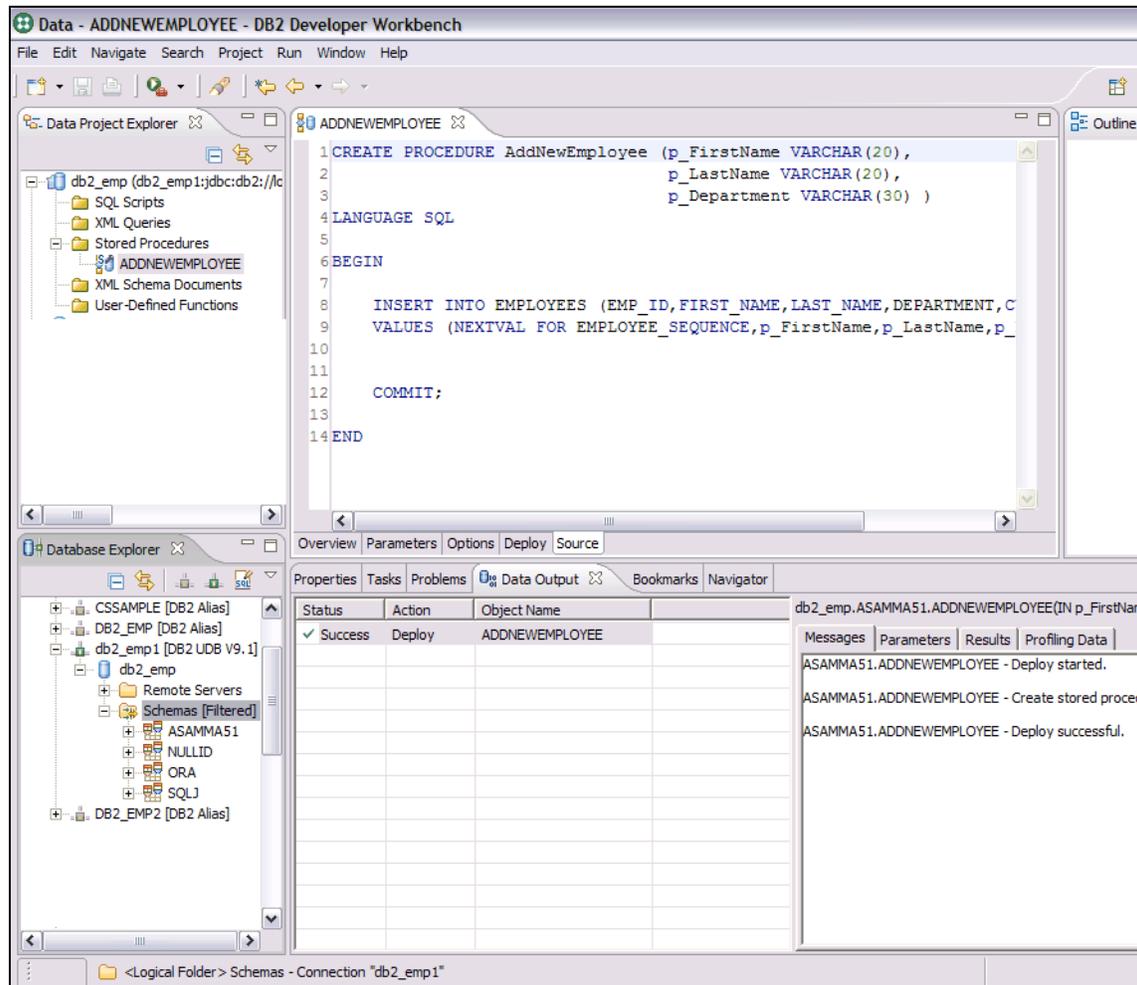
Figure 5-1   The Developer Workbench

5.1.2 The DB2 Command Window

The DB2 Command Window behaves like a command window from your operating system. From the DB2 Command Window. you can execute operating

system commands, DB2 commands, or SQL statements, and then view the output.

In this book, the DB2 Command Window is used for the deployment of manually converted procedures and triggers.

Figure 5-2 shows the DB2 Command Window on AIX.
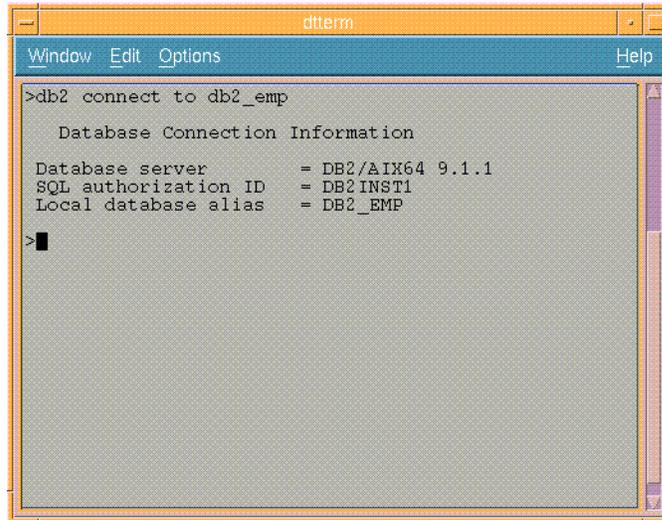
Figure 5-2   The DB2 Command Window on AIX

Figure 5-3 shows the DB2 Command Window on Windows.
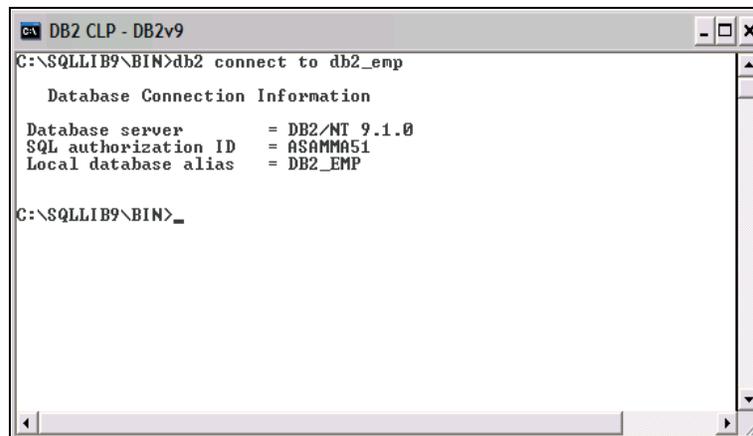
Figure 5-3   The DB2 Command Window on Windows

A complete discussion about the DB2 Command Window can be found in the following IBM Redbooks publications:

► *The Exploitation of DB2 on the Windows Environment,* SG24-6893
► *DB2 Evaluation Guide for Linux and Windows,* SG24-6934

5.1.3 Control Center

The *Control Center* is used to manage and administer systems, DB2 Database instances, DB2 Database for OS/390 and z/OS subsystems, databases, and database objects such as tables and views. From the Control Center, you can access other centers and tools which may assist in optimizing queries; creating and scheduling jobs and scripts; performing data warehousing tasks; creating database objects; and working with DB2 and IMS™ commands.

To open the Control Center:

► In Windows, click **Start** → **Programs** → **IBM DB2** → **General Administration Tools** → **Control Center**.

► In Linux, open the IBM DB2 folder on the desktop and click **Control Center**.

► In UNIX, enter the **db2cc** command from a console window.

Tasks from the Control Center

Some of the key tasks that can be performed using the Control Center are:

► Add DB2 systems, federated systems, DB2 for z/OS and OS/390 systems, IMSysplexes, instances, databases, and database objects to the object tree.

► Manage database objects. You can create, alter, and drop databases, table spaces, tables, views, indexes, triggers, and schemas. You can also manage users.

► Manage data. You can load, import, export, and reorganize data. You can also gather statistics.

► Perform preventive maintenance by backing up and restoring databases or table spaces.

► Configure and tune instances and databases.

► Manage database connections, such as DB2 Connect servers and subsystems.

► Manage IMS systems.

► Manage DB2 for z/OS and OS/390 subsystems.

► Manage applications.

► Analyze queries using Visual Explain to look at access plans.

► Launch other tools such as the Command Editor and the Health Center.

In many cases, advisors, launchpads, and wizards are available to help you perform these tasks quickly and easily.

The Control Center interface

The Control Center interface (Figure 5-4) is available in three different views:

► Basic
This view provides core DB2 functionality, which includes the essential objects, such as databases, tables, and stored procedures.

► Advanced
This view displays all objects and actions available in the Control Center. Select this view if you are working in an enterprise environment and want to connect to DB2 for z/OS or IMS.

► Custom
This view gives you the ability to tailor the object tree and the object actions to your specific needs.
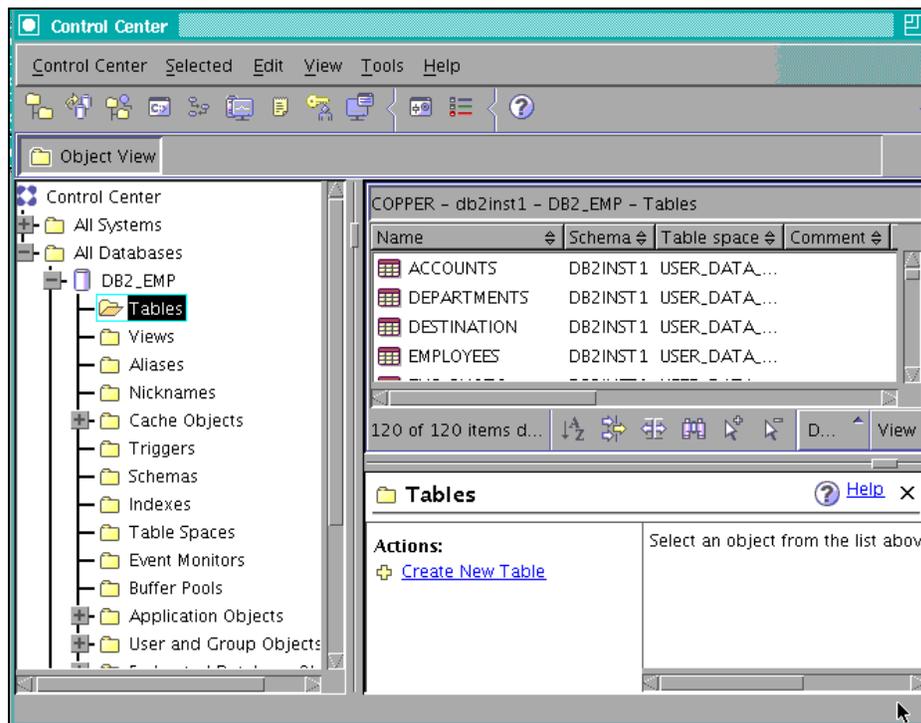
Figure 5-4   The DB2 Control Center Interface

5.1.4 Recommended reading materials

Reading *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS,* ISBN 0131477005, is recommended for learning DB2 SQL PL. This book provides full coverage of SQL procedures, functions, and triggers. Paul Yip, a co-author of the book, provided technical consultation and some of the material for this chapter.

The following DB2 9 manuals are also good sources of information:

- ▶ *SQL Reference Volume 1*, SC10-4249
- ▶ *SQL Reference Volume 2*, SC10-4250
- ▶ *Getting Started with Database Application Development,* SC10-4252
- ▶ *Quick Beginnings for DB2 Servers,* GC10-4246
- ▶ *Quick Beginnings for DB2 Clients,* GC10-4242

5.2 Comparing SQL PL and inline SQL PL

DB2 SQL procedures are created using a high level language known as DB2 SQL Procedural Language (often called SQL PL), which has many direct equivalents and mappings to Oracle's PL/SQL language. Therefore, converting Oracle stored procedures to DB2 stored procedures is usually straightforward. For triggers, functions, and dynamic compound statements, DB2 uses a subset language called inline SQL PL. Before looking at code samples here, it is important to clarify the difference between SQL PL and inline SQL PL.

SQL procedures are processed natively but compiled into a package. SQL functions and triggers are inlined into the query/SQL statement. This implementation difference results in some SQL PL elements that are supported in SQL procedures, but not in triggers and UDFs. SQL PL support in triggers and UDFs is a subset of that in stored procedures and includes support for the following SQL PL elements:

- ▶ DECLARE <variable>
- ▶ FOR
- ▶ GET DIAGNOSTICS
- ▶ IF
- ▶ ITERATE
- ▶ LEAVE
- ▶ SIGNAL
- ▶ WHILE
- ▶ SET

The following subsections discuss the basics of creating DB2 stored procedures, triggers, and user-defined functions. We also compare variable declaration,

conditional statements, and flow of control statements between Oracle PL/SQL and DB2 SQL PL. This chapter (from 5.2, "Comparing SQL PL and inline SQL PL" onward) is organized to serve as a convenient quick reference for you when converting specific features of Oracle PL/SQL to DB2 SQL PL.

For the full documentation of SQL PL features, refer to *IBM DB2 SQL Reference, Volume 1, SC09-4844*; Volume 2, SC09-4845; or *DB2 SQL Procedural Language for Linux, UNIX, and Windows.*

Delimiter

In DB2, statements in trigger, function, and procedure are separated by a semi-colon (;) which also is the default statement terminator when running a group of SQL statements in DB2 CLP. Therefore, in trigger, procedure, or function, a semi-colon (;) cannot be used as the terminator. You can choose other characters such as @ or ! as the termination character. All the DB2 examples in this chapter use ! as the termination character. To execute the file which contains the function, trigger, or procedure to create the object in DB2, use the following command:

db2 -td<termination_character> **-f** file-name

Where termination_character = @ or ! etc. For example, if the termination character is ! and file-name is testproc.db2, the command will be:

db2 -td! -f testproc.db2

5.2.1 Create procedure

Oracle uses the following syntax to create a stored procedure:

```
CREATE OR REPLACE PROCEDURE process_withdrawal( Account_Id VARCHAR2
                                               ,Cheque_No VARCHAR2
                                               ,Amount NUMBER
, RetValue OUT NUMBER)
IS
...
```

DB2 uses slightly different syntax:

```
CREATE PROCEDURE process_withdrawal ( IN Account_Id VARCHAR(10)
                                     ,IN Cheque_No VARCHAR(10)
                                     ,IN Amount DECIMAL(10,2),
,OUT RetValue INTEGER)
...
```

Notes:

► DB2 does not support the REPLACE clause in the create procedure statement, so you may want to drop the procedure before creating one. When developing procedures using the Developer Workbench, the tool may be configured so that procedures are automatically dropped if you choose to rebuild a procedure.

► DB2 does not allow a character data type to be specified without length. As a result, the declaration for a %TYPE column (to which a parameter corresponds) must be checked for proper declaration. If converting with the MTK, this is done for you automatically.

► Output parameters of mode OUT or INOUT must be identified as such because mode IN is the default

5.2.2 Create trigger

This section presents a high-level overview of the differences in trigger definitions between Oracle and DB2. For a detailed description of triggers, refer to the following IBM Redbooks publications:

► *Developing SQL and External Routines*, SC10-4373
► *SQL Reference Volume 2*, SC10-4250

Example 5-1 shows a simple Oracle trigger, which has set a value to a table column before inserting a row.

Example 5-1   Simple Oracle trigger

```
CREATE OR REPLACE TRIGGER connect_audit_trg
   BEFORE INSERT ON connect_audit
   FOR EACH ROW
BEGIN
   :new.timestamp := SYSDATE;
END;
```

Example 5-2 shows how to define the corresponding trigger in DB2.

Example 5-2   Simple DB2 trigger

```
CREATE TRIGGER connect_audit_trg
   NO CASCADE BEFORE INSERT ON connect_audit
   REFERENCING NEW AS n
   FOR EACH ROW
   MODE DB2SQL
BEGIN ATOMIC
   SET n.timestamp = CURRENT TIMESTAMP;
END!
```

> **Notes:**
>
> - ► DB2 does not support the REPLACE clause in the create trigger statement, so you may want to drop the trigger before creating one.
> - ► The NO CASCADE statement is an option for BEFORE triggers. It specifies that the triggered action of the trigger will not cause other triggers to be activated.
> - ► With the REFERENCING NEW AS n clause, you can associate a qualifier n to the new values provided by the initiated insert statement.
> - ► DB2 supports BEFORE and AFTER triggers for UPDATE and DELETE for each row changed. For each statement changed, only AFTER triggers are supported.

Example 5-3 is an Oracle trigger that inserts some values of a deleted row into a history table.

Example 5-3   Oracle trigger with DML command

```
CREATE TRIGGER emp_history_trg
   AFTER DELETE ON employees
   FOR EACH ROW
BEGIN
   INSERT INTO emp_history( emp_id
                           ,first_name
                           ,last_name)
   VALUES ( :old.emp_id
          ,:old.first_name
          ,:old.last_name );
END;
```

Example 5-4 shows how to define the corresponding trigger in DB2.

Example 5-4   DB2 trigger with DML command

```
CREATE TRIGGER emp_history_trg
   AFTER DELETE ON EMPLOYEES
   REFERENCING OLD AS d
   FOR EACH ROW
BEGIN ATOMIC
   INSERT INTO emp_history ( emp_id
                            ,first_name
                            ,last_name)
   VALUES ( d.emp_id
          ,d.first_name
          ,d.last_name);
END!
```

Note that DB2 does not allow a DML statement in BEFORE triggers. This means that the Oracle BEFORE trigger that contains UPDATE, INSERT or DELETE statements will be converted to a DB2 AFTER trigger—and thus, this trigger will be activated *after* the statement is executed.

MERGE statement and triggers

The MERGE statement can execute update, delete, and insert operations. The applicable UPDATE, DELETE, or INSERT triggers are activated for the MERGE statement when an update, delete, or insert operation is executed.

5.2.3 Create function

A function that is created by a user that is not one of the built-in functions in DB2 is called a *user-defined function* (UDF). DB2 supports five different kinds of functions:

- ► SQL scalar, table, or row function
- ► Sourced or template function
- ► OLE DB function
- ► External table function
- ► External scalar function

In this chapter, we discuss only SQL functions. For a description of the other function types, refer to *SQL Reference, Volume 2,* SC10-4250.

The most common UDF definition for SQL functions looks like the one shown in Example 5-5.

Example 5-5   Common UDF definition

```
CREATE FUNCTION function_name ( parameters )
   RETURNS return_type
   LANGUAGE SQL
   READS SQL DATA
   RETURN statement
```

Example 5-6 illustrates a DB2 UDF that duplicates an Oracle built-in function WIDTH_BUCKET:

Example 5-6   DB2 UDF of Oracle function WIDTH_BUCKET

```
CREATE FUNCTION DB2ADMIN.width_bucket( col_val double, low_val double,
high_val double, num_of_buck int  )
    RETURNS integer
```

```
------------------------------------------------------------------------
-
-- SQL UDF (Scalar)
------------------------------------------------------------------------
-
F1: BEGIN ATOMIC

    declare v_each_buck double default 0.00;
    declare result int default 0;

    if col_val < low_val then return 0;
    end if;
    if col_val > high_val then return num_of_buck + 1;
    end if;
    set v_each_buck = (high_val - low_val)/num_of_buck;
    set result = ((col_val - low_val)/v_each_buck)+ 1;
    return result;

END
```

Notes

► Input parameters are optional. However, the parentheses ( ) are mandatory.

► The mandatory return type is one of the following kinds:

– **Scalar**
   A *scalar function* returns a single value each time it is invoked, and is
   generally valid wherever an SQL expression is valid. A scalar function
   cannot contain a DML statement (UPDATE, INSERT, and DELETE).

   When converting an Oracle function that contains those statements, you
   can convert it to DB2 stored procedure or user-defined table function,
   because those functions allow the MODIFY SQL clause, and can contain
   a DML statement.

– **Table**
   A *table function* may be used in a FROM clause and returns a table.
   Example 5-7 shows a table function that contains a DML statement and
   allows you to MODIFY SQL DATA.

Example 5-7

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
      RETURNS TABLE (EMPNO CHAR(6),
                     LASTNAME VARCHAR(15),
                     FIRSTNAME VARCHAR(12))
      LANGUAGE SQL
      MODIFIES SQL DATA
```

```
                    NO EXTERNAL ACTION
                    DETERMINISTIC
                    BEGIN ATOMIC
                      INSERT INTO AUDIT
                      VALUES (USER,
                              'Table: EMPLOYEE Prd: DEPTNO = ' CONCAT DEPTNO);
                      RETURN
                        SELECT EMPNO, LASTNAME, FIRSTNME
                          FROM EMPLOYEE
                          WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
                    END
```

- – **Row**
  A *row function* may be used as a transform function and returns a row.
- ► LANGUAGE SQL is an optional statement.
- ► READS SQL DATA is optional and is implied by default for the SQL function.
- ► Following to the RETURN keyword, you have to specify the SQL function body.

5.2.4 Variables declaration and assignment

Oracle PL/SQL permits declaring variables in four different places:

- ► In a parameter list of a stored procedure or function
- ► Within the body of a stored procedure, function, or trigger
- ► Within a package declaration
- ► Within a package body declaration

In DB2, the notion of package is unrelated to Oracle's concept of a package as a logical grouping of functions and procedures. In DB2, *schemas* are used as the means of logically grouping procedures and functions. For this reason, variables can be declared:

- ► In a parameter list of a stored procedure or function
- ► In the body of a stored procedure, function, or trigger

DB2 SQL PL supports native data types and user-defined distinct types for the variable declaration. It requires the DECLARE clause, and uses the optional DEFAULT clause to initialize variables. For example, here is a PL/SQL declaration:

```
l_balance  NUMBER(10,2) :=0.0;
```

This is converted to SQL PL as:

```
DECLARE l_balance NUMERIC(10,2) DEFAULT 0.0;
```

> **Note:** Variable declarations in a DB2 SQL procedure have to be placed in the BEGIN ... END block.

SQL PL uses a SET statement to assign values to variables. Here, for example, is a PL/SQL assignment:

```
l_balance := 19.99;
```

This will be converted as:

```
SET l_balance = 19.99;
```

> **Note:** The SET statement also can be used to assign a local variable with a table column value:
>
> ```
> SET l_balance =(SELECT balance from account_info where account_no = actNo);
> ```
>
> The SELECT statement should return one row only. An error will be returned if more then one row is selected. You can use a FETCH FIRST n ROWS in the SELECT statement to control the number of rows returned.

Oracle supports defaults for parameters in the parameters list of a procedure or function. DB2 does not support such defaults; that requires the addition of the logic to duplicate the same behavior. The following is an Oracle parameters assignment example:

```
CRETAE procedure create_dept (
new_dept IN CHAR DEFAULT 'TEMP'
dept_numb IN NUMBER DEFAULT 0)
```

This can be converted to DB2 as follows:

```
CREATE PROCEDURE create_dept ( new_dept char(5), dept_numb INT)
   BEGIN
      IF new_dept = NULL THEN
         SET new_dept = 'TEMP';
      END IF;
      IF dept_numb = NULL THEN
         SET dept_numb = 0;
      END IF;
```

5.2.5 Conditional statements and flow control

SQL PL and PL/SQL provide similar functionality and syntax for conditional statements and a variety of LOOP statements to provide flow control. Table 5-1 maps Oracle PL/SQL statements to DB2 SQL PL.

Table 5-1   Mapping of conditional statements

Oracle PL/SQL	DB2 SQL PL
| ```
IF - THEN - END IF;
IF - THEN - ELSE - END IF;
IF - THEN - ELSIF - END IF;
``` | ```
IF - THEN - END IF;
IF - THEN - ELSE - END IF;
IF - THEN - ELSEIF - END IF;
``` |
| ```
LOOP
    statements;
END LOOP;
``` | ```
[L1:] LOOP
     statements;
     LEAVE L1;
END LOOP [L1];
``` |
| ```
WHILE condition LOOP
    statements;
END LOOP;
``` | ```
WHILE condition DO
     statements;
END WHILE;
``` |
| ```
LOOP
    statements;
EXIT WHEN condition;
END LOOP;
``` | ```
REPEAT
     statements;
UNTIL condition;
END REPEAT;
``` |
| ```
OPEN cursor_variable FOR
select_statement;
``` | ```
FOR variable AS cursor_name
CURSOR  FOR select_statement DO
     statements;
END FOR;
``` <br> Note: If cursor_variable is a REF cursor, DB2 would define the cursor and open it with RETURN TO CLIENT/CALLER. |
| ```
FOR l_count IN
lower_bound ..upper_bound
LOOP
  statements;
END LOOP;
``` | No corresponding statements, but the MTK will convert it as: <br><br> ```
SET l_count = lower_bound;
WHILE l_count <= upper_bound DO
statements;
  SET l_count = l_count + 1;
END WHILE ;
``` |

Oracle PL/SQL	DB2 SQL PL
| ```
CASE expression
WHEN condition_1 THEN result_1
WHEN condition_2 THEN result_2
... WHEN condition_n THEN
result_n ELSE result END
``` | ```
CASE expression
WHEN cond_1 THEN statement1
WHEN cond_2 THEN statement2 ...
WHEN cond_n THEN statementN
ELSE statement END
``` |

5.3 Dynamic SQL

In Chapter 4, "Porting with MTK" on page 89, we provide a conversion example of the Oracle DBMS_SQL package, which the MTK does not support. DBMS_SQL can be converted using dynamic SQL in DB2.

In this section, we provide additional examples that show how to convert dynamic SQL

Example 1 - get_emp_name

Example 5-8 and Example 5-9 are two Oracle stored procedures that use dynamic SQL. Example 5-8 uses the DBMS_SQL package.

Example 5-8   PL/SQL procedure with usage of DBMS_SQL

```
CREATE PROCEDURE get_emp_name_v1(emp_id NUMBER) AS
    cursor_name      INTEGER;
    rows_processed   INTEGER;
    sql_stmt         VARCHAR2(1000);
BEGIN
    cursor_name := dbms_sql.open_cursor;
    sql_stmt := 'SELECT last_name FROM employees WHERE emp_id = :x';
    DBMS_SQL.PARSE(cursor_name, sql_stmt, dbms_sql.native);
    DBMS_SQL.BIND_VARIABLE(cursor_name, ':x', emp_id);
    rows_processed := dbms_sql.execute(cursor_name);
    DBMS_SQL.close_cursor(cursor_name);
EXCEPTION
WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Example 5-9 shows a procedure with the same behavior using dynamic SQL. (Dynamic SQL is available in Oracle since version 8).

Example 5-9   PL/SQL procedure with usage of native dynamic SQL

```
CREATE OR REPLACE PROCEDURE get_emp_name_v2(emp_id IN NUMBER) AS
   sql_stmt   VARCHAR2(1000);
   v_result   VARCHAR2(20);
BEGIN
   sql_stmt := 'SELECT last_name FROM employees WHERE emp_id = :x';
   EXECUTE IMMEDIATE sql_stmt
      INTO v_result
      USING emp_id;
   dbms_output.put_line(v_result.last_name);
END;
```

When using the MTK to convert the procedure get_emp_name_v2(), it will report that the **EXECUTE IMMEDIATE** command cannot be converted because the MTK cannot guarantee the correctness of converted dynamic SQL (only a true runtime test can determine correctness). DB2 supports dynamic SQL in procedures using the same type of syntax supported in Oracle (see Example 5-9). If the dynamic SQL statement is an INSERT, UPDATE, or DELETE statement, conversion to DB2 is usually straightforward. If the dynamic statement is a SELECT, however, it needs to be converted using a dynamic cursor in DB2 (as shown in Example 5-10).

Example 5-10   SQL PL procedure with native dynamic SQL

```
CREATE PROCEDURE get_emp_name_v2 ( IN emp_id FLOAT)
LANGUAGE SQL
BEGIN
    DECLARE v_dyn_sql    VARCHAR(1000);
    DECLARE v_sql_stmt   STATEMENT;
    DECLARE c_employees  CURSOR FOR v_sql_stmt;

    SET v_dyn_sql = 'SELECT last_name FROM employees WHERE emp_id = '
                    || CHAR(emp_id);
    PREPARE v_sql_stmt FROM v_dyn_sql;
    OPEN c_employees;
    -- FETCH ...
    CLOSE c_employees;
END!
```

Example 2: update_emp_office

Example 5-11 shows a DB2 stored procedure with a dynamic UPDATE statement.

Example 5-11   Dynamic UPDATE with EXECUTE IMMEDIATE

```
CREATE PROCEDURE update_emp_office_v1 ( IN  v_emp_id FLOAT
                                       ,IN  v_office_id FLOAT
                                       ,OUT v_num_changes INTEGER)
LANGUAGE SQL
BEGIN
    DECLARE v_dyn_sql    VARCHAR(1000);

    SET v_dyn_sql = 'UPDATE employees' ||
                    ' SET office_id = ' || CHAR(v_office_id) ||
                    ' WHERE emp_id = ' || CHAR(v_emp_id);
    EXECUTE IMMEDIATE v_dyn_sql;

    GET DIAGNOSTICS v_num_changes = row_count;
END!
```

In Example 5-8, the variable `rows_processed` contained the number of rows affected by the dynamic SQL statement. In DB2, the same result can be achieved using GET DIAGNOSTICS. With the GET DIAGNOSTICS statement, the number of row changed due to the last INSERT, UPDATE, or DELETE may be returned.

Use EXECUTE IMMEDIATE if the SQL statement only needs to be executed once or infrequently. If the SQL statement needs to be executed repeatedly, use the PREPARE and EXECUTE statements.

When using the EXECUTE statement, parameter markers can be employed. Parameter markers are designated by the question mark (?), as shown in Example 5-12. Be aware that the EXECUTE statement *cannot* be used with a SELECT or VALUES statement.

Example 5-12 demonstrates the use of a dynamic SQL statement using PREPARE and EXECUTE instead of EXECUTE IMMEDIATE.

Example 5-12   Dynamic UPDATE with EXECUTE and PREPARE

```
CREATE PROCEDURE update_emp_office_v2 ( IN  v_emp_id FLOAT
                                       ,IN  v_office_id FLOAT
                                       ,OUT v_num_changes INTEGER)
LANGUAGE SQL
BEGIN
    DECLARE v_dyn_sql    VARCHAR(1000);
    DECLARE v_stmt       STATEMENT;
```

```
        SET v_dyn_sql = 'UPDATE employees' ||
                        ' SET office_id = ?' ||
                        ' WHERE emp_id = ?';

        PREPARE v_stmt FROM v_dyn_sql;
        EXECUTE v_stmt USING v_office_id, v_emp_id;

        GET DIAGNOSTICS v_num_changes = row_count;
END!
```

Example 3: get_max_band

Example 5-13 demonstrates a Java user-defined function (UDF) with dynamic
SQL. The function uses the invoker's database connection with all its
authentications. The prepare is followed by the assignment of an input variable
called inOfficeID, to the value of the parameter marker. The SQL statement
may contain a full-select.

Example 5-13   Java UDF with dynamic SQL

```
import COM.ibm.db2.app.*;
import java.sql.*;

public class UDFemp extends UDF
{
  public void maxBand(int inOfficeID, String outBand)
    throws Exception
    {
      try
      {
        // Get caller's connection to the database
        Connection con =
          DriverManager.getConnection("jdbc:default:connection");

        String query = "SELECT max(band) " +
                       "FROM employees " +
                       "WHERE office_id = ?";

        PreparedStatement stmt = con.prepareStatement(query);
        stmt.setInt(1, inOfficeID);

        ResultSet rs = stmt.executeQuery();

        while(rs.next())
        {
```

```
          outBand = rs.getString(1);
        }

        set(2, outBand);

        rs.close();
        stmt.close();
        con.close();
      }

      catch (SQLException sqle)
      {
        setSQLstate("38999");
        setSQLmessage("SQLCODE = " + sqle.getSQLState());
        return;
      }
    }
}
```

The corresponding CREATE FUNCTION statement for this Java UDF is shown in Example 5-14:

Example 5-14   The CREATE FUNCTION statement for a Java UDF

```
CREATE FUNCTION get_max_band(INTEGER)
RETURNS CHAR
EXTERNAL NAME 'UDFemp!maxBand'
FENCED
CALLED ON NULL INPUT
VARIANT
READS SQL DATA
PARAMETER STYLE DB2GENERAL
LANGUAGE JAVA
NO EXTERNAL ACTION!
```

5.4 Cursor conversion

Because DECLARE CURSOR syntax is *not* supported in triggers and functions, it is usually the case that some manual conversion of cursors may required.

This section covers the conversion of cursors in stored procedures. The examples focus on methods for converting Oracle *explicit* cursors in function, as well as how to handle converting cursor attributes.

In general, Oracle PL/SQL and DB2 SQL PL are similar in their syntax and support for cursor operations; Table 5-2 lists some of the similarities and differences.

Table 5-2   Mapping Oracle and DB2 Cursor operation

Operation	Oracle	DB2
Declaring a cursor	`CURSOR` *cursor_name* `[(`*cursor_parameter(s)*`)]` `IS` *select_statement*;	`DECLARE` *cursor_name* `CURSOR [WITH HOLD] [WITH RETURN] [TO CALLER` | `TO CLIENT] FOR` *select-statement*
Opening a cursor	`OPEN` *cursor_name* `[(`*cursor_parameter(s)*`)];`	`OPEN` *cursor_name* `[USING` *host-variable*`]`
Fetching from cursor	`FETCH` *cursor_name* `INTO` *variable(s)*	`FETCH [`*from*`]` *cursor_name* `INTO` *variable(s)*
Update fetched row	`UPDATE` *table_name* `SET` *statement(s)...* `WHERE CURRENT OF` *cursor_name*;	`UPDATE` *table_name* `SET` *statement(s)...* `WHERE CURRENT OF` *cursor_name*
Delete fetched row	`DELETE FROM` *table_name* `WHERE CURRENT OF` *cursor_name;*	`DELETE FROM` *table_name* `WHERE CURRENT OF` *cursor_name*
Closing cursor	`CLOSE` *cursor_name*;	`CLOSE` *cursor_name*

5.4.1 Converting an explicit cursor in a procedure

An Oracle SQL cursor is defined similar to a variable in a procedure. The definition is in the stored procedure specification. Example 5-15 is a sample Oracle procedure with an explicit cursor defined in the specification.

Example 5-15   Oracle procedure with explicit cursor

```
PROCEDURE get_sum_projects( v_office_id   IN  NUMBER
                           ,sum_projects  OUT NUMBER)
AS
   v_prj   NUMBER(3);
   CURSOR c1 IS
      SELECT current_projects
      FROM employees
      WHERE office_id = v_office_id;
BEGIN
   sum_projects := 0;
```

```
      OPEN c1;
      LOOP
         FETCH c1 INTO v_prj;
         EXIT WHEN c1%NOTFOUND;
         sum_projects := sum_projects + v_prj;
      END LOOP;
END;
```

In DB2, the SQL cursor must be defined in the procedure body. To use
procedure parameters within the cursor, the SQL statement for the cursor must
be defined and prepared. Example 5-16, which is the conversion of the source in
Example 5-15, shows a cursor definition and the PREPARE statement.

Example 5-16   DB2 Stored procedure showing conversion of an explicit cursor

```
CREATE PROCEDURE get_sum_projects( IN v_office_id    INTEGER
                                  ,OUT sum_projects  INTEGER)
BEGIN
  DECLARE SQLCODE INT DEFAULT 0;
  DECLARE  v_prj   SMALLINT default 0;
  DECLARE v_no_data   SMALLINT DEFAULT 0;

DECLARE c1 CURSOR FOR
      SELECT current_projects
      FROM employees
      WHERE office_id = v_office_id;

  DECLARE CONTINUE HANDLER FOR NOT FOUND
       SET v_no_data = 1;

   SET sum_projects  = 0;
   OPEN c1;
   FETCH c1 INTO v_prj;
   WHILE (v_no_data =0)
DO
   SET sum_projects = sum_projects + v_prj;
   FETCH c1 INTO v_prj;

   END while;
   CLOSE c1;

END!
```

5.4.2 Converting an explicit cursor in functions and triggers

It is important to understand that DB2 triggers and functions must use *inline* SQL PL. Oracle cursor operations in triggers and functions, such as explicit cursors, must be converted to the corresponding inline SQL PL syntax.

In 4.12, "Manual conversion for ORA_EMP database objects" on page 153, we provide an example of converting an explicit cursor in a trigger. Here, Example 5-17 shows how to convert an Oracle explicit cursor in a function using a FOR LOOP. The same method may be used in procedures; in fact, using a FOR cursor can yield better performance results.

Example 5-17 shows the Oracle function source code.

Example 5-17   Function with an explicit cursor in Oracle

```
CREATE OR REPLACE FUNCTION CountProjects (
  /* Returns the number of projects in which the employee
     identified by p_emp_ID is currently engaged */
  p_empID IN employees.emp_ID%TYPE)
RETURN NUMBER AS

  v_TotalProjects NUMBER;

  -- Total number of projects
  v_AccountProjects NUMBER;

  -- projects for one account
  CURSOR c_DeptAccts IS
    SELECT dept_code, acct_id FROM employees
      WHERE emp_id = p_empID;
BEGIN

  FOR v_AccountRec IN c_DeptAccts LOOP
    -- Determine the projects for this account.

    SELECT num_projects INTO v_AccountProjects
      FROM accounts WHERE dept_code = v_AccountRec.dept_code
      AND acct_id = v_AccountRec.acct_id;

    -- Add it to the total so far.
    v_Totalprojects := v_Totalprojects + v_AccountProjects;

  END LOOP;

  RETURN v_Totalprojects;
```

```
END CountProjects;
```

The converted DB2 code is shown in Example 5-18.

Example 5-18   Conversion using a FOR LOOP in DB2

```
CREATE FUNCTION CountProjects( p_empID INTEGER)
RETURNS INTEGER
LANGUAGE SQL
BEGIN ATOMIC
   DECLARE v_TotalProjects INT DEFAULT 0;
   DECLARE v_AccountProjects INT;
   X: FOR v_DeptAccts as                                        -[1]
      Select dept_code, acct_id
      FROM employees
      WHERE emp_id = p_empID
      DO
         SET v_AccountProjects = (
         SELECT num_projects                                    -[2]
         FROM accounts
             WHERE dept_code = v_DeptAccts.dept_code
             AND acct_id = v_DeptAccts.acct_id);
         SET v_Totalprojects = v_Totalprojects + v_AccountProjects;
   END FOR X;
   RETURN v_Totalprojects;
END!
```

The notes in Example 5-18 are explained as follows:

[1] The FOR LOOP X is declared and the values that will be used in the
       WHERE clause in the SET statement are selected.

[2] SELECT INTO is not supported in inline SQL. The equivalent can be
       achieved using the SET statement.

5.4.3 Converting cursor attributes

Oracle supports *cursor attributes* to get information about the current status of a
cursor. In DB2, SQLCODE or SQLSTATE can be used to obtain the same
information. Table 5-3 shows the mapping of Oracle cursor attributes to DB2
SQLCODE/SQSTATE values.

Table 5-3   Mapping of cursor attributes

Oracle	DB2
%ISOPEN	Upon open a cursor, DB2 SQLCODE -502 is returned if the cursor is already open. On FETCH, SQLCODE -501 is returned if the cursor is not open yet. However, In DB2 procedure, you will not be able to test for any negative sqlcodes (exceptions) unless they are contained within an exception handler because control will be returned to the calling application or procedure. If %isopen is used to test if a cursor is open before closing it, it may not be needed in DB2. DB2 will close the cursor for you. The SQLSTATE associated with SQLCODE -501 is 24501. The SQLSTATE associated with SQLCODE -502 is 24502
%NOTFOUND	if (SQLCODE = 100) or if SQLSTATE = '02000'
%FOUND	if (SQLCODE = 0) or if SQLSTATE = '00000'
%ROWCOUNT	Use a counter variable to retrieve number of rows fetched from the cursor

The following examples demonstrate how to convert these Oracle attributes to DB2.

%ISOPEN

Consider the following Oracle code fragment that uses %ISOPEN:

```
IF c1%ISOPEN THEN
     fetch c1 into var1;
ELSE                           -- cursor is closed, so open it
   OPEN c1;
     fetch c1 into var1;
END IF;
```

You can implement the same logic in DB2 using CONDITION HANDLER:

```
DECLARE cursor_notopen CONDITION FOR SQLSTATE 24501;
DECLARE CONTINUE HANDLER FOR cursor_notopen
BEGIN
```

```
        open c1;
        FETCH c1 int var1;
    END;
      ...
    FETCH c1 into var1;
```

For a more detailed discussion of CONDITION HANDLERS, refer to 5.6,
"Condition handling" on page 200.

%NOTFOUND

Here is an Oracle example that uses %NOTFOUND:

```
OPEN cur1;
LOOP
  FETCH cur1  INTO v_var1;
    EXIT WHEN cur1%NOTFOUND;
  ...
END LOOP;
```

In DB2, this can be implemented by using CONDITION HANDLERS or by
checking the SQLCODE value:

```
DECLARE SQLCODE int DEFAULT 0;
……
OPEN c1;
L1: LOOP
      FETCH c1 INTO v_var1;
      IF SQLCODE = 100 THEN
              LEAVE L1;
      END IF;
      ...
END LOOP L1;
```

%ROWCOUNT

SQL %ROWCOUNT yields the number of rows affected by an INSERT,
UPDATE, or DELETE statement, or returned by a SELECT INTO statement.
%ROWCOUNT yields 0 if an INSERT, UPDATE, or DELETE statement affected
no rows, or if a SELECT INTO statement returned no rows.

The use of %ROWCOUNT can be demonstrated by the following Oracle
examples. First, consider the example that uses %ROWCOUNT to determine the
condition for the loop:

```
LOOP
    FETCH c1 INTO my_ename, my_deptno;
    IF c1%ROWCOUNT > 10 THEN
```

```
            EXIT;
        END IF;
        ...

    END LOOP;
```

This Oracle code will process only the first 10 rows for the given cursor. This logic can be implemented in DB2 using the FETCH FIRST N ROWS ONLY clause in the cursor declaration, and processing this cursor till NOT FOUND.

```
DECLARE c1 CURSOR FOR
    SELECT ename, deptno
      FROM  emp_table
     FETCH FIRST 10 ROWS ONLY;

DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
             SET end-of-fetch = 1;
    END;

L1: LOOP
      FETCH c1 INTO my_ename, my_deptno;
      IF end-of-fetch = 1 THEN
          LEAVE L1;
      END IF;
      ............
END LOOP L1;
```

If %ROWCOUNT is used to determine how many rows from the cursor have been processed at any given time, it will look like the following Oracle code:

```
LOOP
    FETCH c1 INTO my_ename, my_deptno;
    IF c1%ROWCOUNT > 10 THEN
        ...
    END IF;
    ...

END LOOP;
```

In DB2, you would need to use a local variable (counter) to store this information after each fetch from the cursor:

```
DECLARE v_CURCOUNT INT DEFAULT 0;
    ........
    L1: LOOP
          FETCH c1 INTO my_ename, my_deptno;
```

```
                        v_CURCOUNT = v_CURCOUNT + 1;
                        IF vCURCOUNT > 10 THEN

                        ……
                        END IF;

                    ……
                END LOOP L1;
```

In the following example, %ROWCOUNT is used to take action if more than ten rows have been deleted:

```
DELETE FROM emp_table WHERE ...
IF SQL%ROWCOUNT > 10 THEN        -- more than 10 rows were deleted
    ...

END IF;
```

DB2 uses the GET DIAGNOSTICS statement to return the number of rows affected by an INSERT, UPDATE, or DELETE statement:

```
DECLARE rc   INT DEFAULT 0;
…….
DELETE FROM emp_table WHERE ...
GET DIAGNOSTICS rc = ROW_COUNT;
IF rc  > 10 THEN       -- more than 10 rows were deleted
    ...

END IF;
```

> **Important:** The GET DIAGNOSTICS statement is *not* supported for the SELECT or SELECT INTO statement. Note the following points:
>
> ► SQLCODE will be 100 if no row is selected.
> ► SQLCODE will be 0 if one row is selected.
> ► SQLCODE will be -811 (SQLSTATE 21000 - SQLERROR) if more than one row is selected.

%FOUND

Note that Oracle treats *any* SQL statement as an implicit cursor. Implicit cursor attributes return information about the execution of an INSERT, UPDATE, DELETE, or SELECT INTO statement.

The values of the implicit cursor attributes always refer to the most recently executed SQL statement. Before Oracle opens the SQL cursor, the implicit cursor attributes yield NULL.

In the following example, %FOUND is used to insert a row if a delete succeeds:

```
    DELETE FROM emp WHERE empno = my_empno;
    IF SQL%FOUND THEN    -- delete succeeded
       INSERT INTO emp_table VALUES (my_empno, my_ename);
```

This code can be converted to DB2 as follows:

```
    DELETE FROM emp WHERE empno = my_empno;
    IF SQLCODE = 0  THEN    -- delete succeeded
       INSERT INTO emp_table VALUES (my_empno, my_ename);
```

5.5 Collections

A collection is Oracle's version of arrays. A *collection* is a single-dimension list of an ordered group of elements, all of the same type (for example, bonuses for employees). Each element has a unique subscript that determines its position in the collection.

Oracle PL/SQL provides two kinds of collections: *nested tables* and *variable-size arrays* (also known as *varrays*). Collections can have only one dimension, and must be indexed by integers. Collections can be passed as parameters—thus, they can be used to move columns of data into and out of database tables, or between stored procedures and client-side applications.

5.5.1 Nested tables and varrays

To understand how nested tables and varrays can be converted to DB2, we first address the difference between Oracle nested tables and varrays.

Nested tables are items of type TABLE within the database. You can view nested tables as one-column database tables or one-dimension arrays. Oracle stores the rows of a nested table in no particular order. But when you retrieve the nested table into a PL/SQL variable, the rows are given consecutive subscripts starting at 1.

Unlike variable arrays, which have a fixed size, nested tables can grow dynamically with no upper bound. Another difference is that variable arrays *must* have consecutive subscripts, which prevents you from deleting individual elements from an array. Initially, nested tables have consecutive subscripts, but they can be *sparse* (that is, have nonconsecutive subscripts).

Because DB2 does not support collections, the most generic way to convert a nested table is by using the DB2 Declared Global Temporary Table (DGTT), where the first column stores the value of the subscript, and the second column stores the value of Oracle nested table.

> **Note:** DB2 temporary tables are *not* similar to Oracle temporary tables. DB2 temporary tables are memory-bound (provided that sufficient memory is available). They are visible only to the connection that declares it, and they exist only for as long as a connection is maintained (or dropped). If you disconnect, the table is automatically cleaned up.

> **Tip:** To use DGTTs, you must create a user temporary table space (none exists by default). In the simplest case, you can use:
>
> **create user temporary tablespace** usertemp1 **managed by system using** ('usertemp1')
>
> The size of the buffer pool associated with this table space will affect how memory-bound DGTTs are at runtime.

To clarify this concept, refer to Example 5-19; it fills the nested table EmpList with names of the employees for a given department from table emp_table.

Example 5-19   Oracle code using nested table

```
DECLARE
  TYPE EmpList IS TABLE OF emp_table.ename%TYPE ;
  CURSOR c1 IS
     SELECT emp_name
     FROM emp_table
     WHERE dept = v_dept;
  EmpName emp_table.ename%TYPE;
  empNum NUMBER;
BEGIN
  LOOP
    FETCH c1 INTO EmpName;
    WHEN c1%NOTFOUND EXIT;
    empNum := empNum + 1;
    EmpList(empNum):= EmpName;
  END LOOP;
  CLOSE c1;
END;
```

The same can be implemented in DB2 using DGTT, as shown in Example 5-20.

Example 5-20   DB2 code using DGTT

```
DECLARE SQLCODE INT DEFAULT 0;
```

```
DECLARE v_empname  varchar(30);
DECLARE v_num INT  DEFAULT 0;
DECLARE c1 CURSOR FOR
        SELECT emp_name
        FROM emp_table
        WHERE dept = v_dept;
DECLARE GLOBAL TEMPORARY TABLE SESSION.temp_emp_list
        (num integer, EmpName varchar(30))
        WITH REPLACE
        ON COMMIT PRESERVE ROWS
        NOT LOGGED;

OPEN c1;
   WHILE (SQLCODE = 0) DO
      FETCH c1 INTO v_empname;
      SET v_num = v_num +1;
      INSERT INTO SESSION.temp_emp_list
         VALUES (v_num,v_empname);
   END WHILE;
CLOSE c1;
```

The code is even more efficient if converted as shown in Example 5-21.

Example 5-21   Efficient DB2 code using DGTT

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.temp_emp_list
        (num integer,EmpName varchar(30))
WITH REPLACE
ON COMMIT PRESERVE ROWS
NOT LOGGED;

INSERT INTO session.temp_emp_list
        SELECT row_number() over(), emp_name
        FROM emp_table
        WHERE dept =  v_dept;
```

To convert Oracle varrays, you can also use DGTT—or sometimes the redesign can help achieve the same functionality.

5.5.2 Bulk collect

In Oracle 8i and higher, you can fetch more than one row at a time into a collection by using the BULK COLLECT clause. This clause is used as part of the

SELECT INTO, FETCH INTO, or RETURNING INTO clause, and will retrieve rows from the query into indicated collections.

Example 5-19 on page 194 can be rewritten using the BULK COLLECT clause as follows:

```
DECLARE
    TYPE EmpList IS TABLE OF emp_table.ename%TYPE ;
    CURSOR c1 IS
        SELECT emp_name
        FROM emp_table
        WHERE dept = v_dept;
BEGIN
    OPEN c1;
    FETCH c1 BULK COLLECT INTO EmpList;
    CLOSE c1;
END;
```

or

```
DECLARE
    TYPE EmpList IS TABLE OF emp_table.ename%TYPE ;
BEGIN
    SELECT emp_name BULK COLLECT INTO EmpList;
END;
```

Oracle will treat SELECT INTO as an implicit cursor. It will fetch the data starting at index 1, and successively overwrite elements in the output collection EmpList until it has retrieved all requested rows.

To convert the BULK COLLECT statement to DB2, a DB2 DGTT can be used as shown in Example 5-20 on page 194. In some cases the INSERT INTO (SELECT * FROM) statement can be used, as shown in Example 5-22.

Example 5-22   DB2 code using INSERT INTO

```
DECLARE v_empname   varchar(30);
DECLARE v_num INT  DEFAULT 0;
DECLARE GLOBAL TEMPORARY TABLE SESSION.temp_emp_list
        (num INTEGER, EmpName VARCHAR(30))
        WITH REPLACE
        ON COMMIT PRESERVE ROWS
        NOT LOGGED;
INSERT INTO SESSION.temp_emp_list (
    SELECT emp_name
    FROM emp_table
    WHERE detp = v_dept);
```

For more information about the GLOBAL TEMPORARY TABLE, refer to the IBM DB2 9 publication *SQL Reference Volume 2*.

5.5.3 Passing result sets between procedures

In 5.5.2, "Bulk collect" on page 195, we discuss general conversion principles for Oracle collection. Here, we demonstrate special cases of passing multiple row result sets from one procedure to another.

It is often convenient to manipulate many variables at once as one unit. Oracle nested tables and varrays are frequently used to implement this kind of application. In Example 5-23, all employees from a specified department who have an account code equal to 307 need to be retrieved, and the result passed to an PL/SQL block (it could as well be a client program). The example illustrates a PL/SQL procedure that returns nested tables as the output parameter.

Example 5-23   PL/SQL procedure returns nested table

```
CREATE PACKAGE BODY AccountPackage AS
  PROCEDURE AccountList(p_dept_code   IN     accounts.dept_code%TYPE,
                        p_acct_id     IN     accounts.acct_id%TYPE,
                        p_IDs         OUT    t_EmployeeIDTable,
                        p_NumEmployees IN OUT NUMBER) IS

      v_EmployeeID  Employees.Emp_id%TYPE;

      -- Local cursor to fetch the registered Employees.
      CURSOR c_RegisteredEmployees IS
        SELECT Emp_id
          FROM Employees
          WHERE dept_code = p_dept_code
          AND acct_id = p_acct_id;
    BEGIN
      /* p_NumEmployees will be the table index. It will start at
         0, and be incremented each time through the fetch loop.
         At the end of the loop, it will have the number of rows
         fetched, and therefore the number of rows returned in
         p_IDs. */
      p_NumEmployees := 0;
      OPEN c_RegisteredEmployees;
      LOOP
        FETCH c_RegisteredEmployees INTO v_EmployeeID;
        EXIT WHEN c_RegisteredEmployees%NOTFOUND;
```

```
          p_NumEmployees := p_NumEmployees + 1;
          p_IDs(p_NumEmployees) := vEmployeeID;
        END LOOP;
     END AccountList;
END AccountPackage;
```

Note that type *t_EmployeeIDTable* should be declared within the AccountPackage specification, as follows:

```
 TYPE t_EmployeeIDTable IS TABLE OF Employees.Emp_id%TYPE;
```

The AccountList procedure can be called from the following PL/SQL block:

```
DECLARE
  v_DeptEmployees   AccountPackage.t_EmployeeIDTable;
  v_NumEmployees    BINARY_INTEGER := 20;

BEGIN
  -- Fill the PL/SQL table with employees from dept 'BA'
  AccountPackage.AccountList('BA', 307,
v_DeptEmployees,v_NumEmployees);

  -- Insert these employee into temp_table

  FOR v_LoopCounter IN 1..v_NumEmployees
  LOOP
    INSERT INTO temp_table (num_col, char_col)
      VALUES (v_DeptEmployees(v_LoopCounter),
              'In Department BA');
  END LOOP;
END;
```

Using nested table t_EmployeeIDTable, the results from cursor c_RegisteredEmployees are passed to the calling block as one unit or as one output variable.

DB2 has a different mechanism for processing multiple rows results. SQL procedure uses the following to return result sets to a caller:

► Specify the DYNAMIC RESULT SETS clause in a CREATE PROCEDURE statement.

► Declare the cursor using a WITH RETURN clause.

► Keep the cursor open for the client application.

► Unlike Oracle, no parameter is required in order for the result set to be passed out of this procedure.

Example 5-24 shows how an SQL procedure can pass results from the same cursor to the calling application. The name of the cursor has been changed to adhere to the DB2 18 character limit.

Example 5-24   SQL procedure returns multiple rows using CURSOR WITH RETURN

```
CREATE PROCEDURE AccountPackage.AccountList ( IN p_dept_code CHAR(3),
                                              IN p_acct_id SMALLINT )
LANGUAGE SQL
RESULT SET 1

BEGIN

    DECLARE SQLCODE INTEGER DEFAULT 0;

    DECLARE c_RegisteredEmplo1 CURSOR WITH RETURN TO CALLER
     FOR SELECT "EMP_ID"
         FROM EMPLOYEES
       WHERE "DEPT_CODE" = p_dept_code
         AND "ACCT_ID" = p_acct_id;

     OPEN c_RegisteredEmplo1;

END
```

There are two options for the WITH RETURN clause:

► WITH RETURN TO CALLER (default): Use this option to return the result set to the invoker, whether the invoker is an application or another procedure.

► WITH RETURN TO CLIENT: Use this option to return the result set directly to the application, bypassing any intermediate nested routines.

To imitate the Oracle example in full, we convert a PL/SQL block that calls the AcountPackage.AccountList procedure to a DB2 SQL procedure, as shown in Example 5-25.

Example 5-25   DB2 store procedure calls AccountPackage.AccountList

```
CREATE PROCEDURE AccountPackage.CALL_AccountList ( )
   LANGUAGE SQL
BEGIN
   DECLARE sqlcode  INT DEFAULT 0;
   DECLARE v_empId  INT DEFAULT 0;
   DECLARE v_empNum INT DEFAULT 0;
   DECLARE v_empCnt INT DEFAULT 0;
   DECLARE loc1     RESULT_SET_LOCATOR VARYING;        - [1]
```

```
      SET v_empNum = 20;

      CALL AccountPackage.AccountList('BA',307);
      ASSOCIATE RESULT SET LOCATOR( loc1) WITH               - [2]
              PROCEDURE   AccountPackage.AccountList;

      ALLOCATE c1 CURSOR FOR RESULT SET loc1;                - [3]

  L1: LOOP
      FETCH FROM c1 INTO v_empID;
      IF (sqlcode = 100) or (v_empCnt > v_empNum)
        THEN LEAVE L1;
      ELSE
        SET v_empCnt = v_empCnt + 1;
        INSERT INTO temp_table (num_col, char_col)
            VALUES (v_empId, 'IN DEPARTMENT  ');
      END IF;
    END LOOP L1;
  END
```

The notes in Example 5-25 are explained as follows; to receive result sets in SQL
procedures, you need to:

► [1] DECLARE result set locators to the stored procedure expected to return
   these result sets.

► [2] ASSOCIATE result set locators to the stored procedure expected to return
   these result sets.

► [3] ALLOCATE each cursor expected to be returned to a result set locator.

After this is done, rows can be fetched from the result sets. The cursor in this
case plays the role of an Oracle nested table, and allows you to pass multiple
variables (the result set from cursor) as one unit.

5.6 Condition handling

This section discusses the various methods of implementing condition handling
conversion.

5.6.1 Condition handling in stored procedure

Both PL/SQL and SQL PL support EXCEPTION HANDLERS to trap SQL errors and handle them. This mechanism permits separation of a procedure's error processing from its main logic.

PL/SQL uses the following syntax for EXCEPTION:

```
EXCEPTION
    WHEN exception_name1 THEN <executable statements>
    WHEN exception_name2 THEN <executable statements>
    ...
    WHEN exception_nameN THEN <executable statements>
    WHEN OTHER <executable statements> ;
```

Where exception_name is one of the predefined exceptions (NO_DATA_FOUND, TOO_MANY_ROWS) or has been defined using the following syntax:

```
exception_name EXCEPTION;
    PRAGMA EXCEPTION_INIT(exception_name, SQLCODE);
```

In DB2, SQL procedures exception handling is accomplished through the use of condition handlers.

A *condition handler* is an SQL statement that is executed when a specified condition is encountered during execution of a statement within the body of a procedure. The handler is declared within a compound statement, and the handler's scope is limited to that compound statement.

The following is the syntax for the condition handler declaration:

```
DECLARE {CONTINUE | EXIT | UNDO} HANDLER FOR <condition>
               SQL-procedure-statement;
```

where *<condition>* is one of the following:

► SQLSTATE value
► SQLEXEPTION (SQLCODE < 0)
► SQLWARNING (SQLCODE > 0)
► NOT FOUND
► Condition name

> **Important:** In DB2, an INSERT, UPDATE or DELETE that affects *no* rows also results in a NOT FOUND condition (+100).
>
> In the absence of a NOT FOUND handler, the SQLWARNING handler will be invoked.

Based on this, to convert the following PL/SQL code to DB2:

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN v_status :=0;
  WHEN OTHER   THEN v_err_flag :=1;
```

Two condition handlers need to be declared:

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET v_status   = 0;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  SET v_err_flag = 1;
```

In this example, the Oracle exception name corresponds to a DB2 condition name, and the predefined NO_DATA_FOUND exception corresponds to the NOT FOUND condition. To match other Oracle predefined exceptions, the appropriate DB2 SQLSTATE values would have to be used, and the condition would be defined using the following syntax:

```
DECLARE  condition_name CONDITION FOR SQLSTATE value;
```

For example, the Oracle predefined exception TOO_MANY_ROWS can be converted using the following statement:

```
DECLARE too_many_rows CONDITION FOR SQLSTATE '21000';
```

Then a HANDLER for this CONDITION can be declared as follows:

```
DECLARE EXIT HANDLER FOR too_many_rows
BEGIN
...
END;
```

The following procedure, shown in Example 5-26, updates title_desc column declared as char(50). If in_title_desc is longer than 50 characters, then SQLEXCEPTION `value is too long` would occur and invoke the declared HANDLER. As a result, the table will not be updated and err_num = -433 will be returned as an output parameter.

Example 5-26 demonstrates the use of CONDITION HANDLERS in SQL procedures.

Note: This example uses a CONTINUE handler. That is, after the handler logic is complete, the flow of control continues from where the condition originally occurred. Without the continue handler, the procedure would have exited early, and returned an exception to the application.

Example 5-26   Condition handler - SQL EXCEPTION

```
CREATE PROCEDURE new_title ( IN in_title_desc varchar(100)
                            ,OUT err_num INT )
LANGUAGE SQL
P1: BEGIN
    DECLARE SQLCODE INTEGER DEFAULT 0;
    DECLARE SQLSTATE CHAR(5) DEFAULT ' ';

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
      SET err_num = SQLCODE;

    UPDATE books
    SET title_desc = in_title_desc
    WHERE author = 'JACK LONDON';
END P1
```

Now this procedure can be changed to HANDLE a long value of title_desc and, should a value longer than 50 characters occur, update the books table with a truncated value; refer to Example 5-27.

Example 5-27   Condition handler - handle a long value

```
CREATE PROCEDURE new_title_1 ( IN in_title_desc VARCHAR(100)
                              ,OUT message_out CHAR(70) )
LANGUAGE SQL
P1: BEGIN
    DECLARE SQLCODE INTEGER  DEFAULT 0;
    DECLARE SQLSTATE CHAR(5) DEFAULT ' ';
    DECLARE v_trunc INT      DEFAULT 0;
    DECLARE value_error CONDITION FOR SQLSTATE '22001';

    DECLARE CONTINUE HANDLER FOR value_error
    BEGIN
       UPDATE books
       SET title_desc = substr(in_title_desc,1,50)
       WHERE author = 'JACK LONDON';
       SET v_trunc = 1;
    END;

    UPDATE books
    SET title_desc = in_title_desc
    WHERE author = 'JACK LONDON';

    IF v_trunc = 0 THEN
```

```
            SET message_out = 'TITLE UPDATED WITHOUT PROBLEM';
        ELSE
            SET message_out = 'TITLE UPDATED WITH TRUNCATION'  ;
        END IF;
END P1
```

5.6.2 Condition handling in triggers and functions

Unfortunately, condition handling is not supported in triggers or functions.
However, there are often cases where condition handlers may not be required.
Here we present some examples.

Consider the following Oracle function, which contains a condition handler:

```
CREATE OR REPLACE FUNCTION func_with_handler1
RETURN NUMBER
AS
    v_id NUMBER;
BEGIN
    BEGIN
        SELECT ObjID
        INTO v_id
        FROM T1;
        EXCEPTION WHEN OTHERS THEN
        v_Id := 0;
    END;
    RETURN v_id;
END;
```

When converted through the MTK, this function is converted as a stored
procedure because a condition handler is not supported in DB2 functions. In
some cases, this may be the appropriate conversion. If this function were used in
an SQL statement, however, we would need to make an effort to retain it as a
function in DB2.

Upon closer examination of this function, you can see that the function can be
rewritten without handlers, provided that the column ObjID in table T1 is not
nullable. It can be rewritten as follows:

```
CREATE FUNCTION func_with_handler1()
RETURNS INT
BEGIN ATOMIC
    DECLARE v_id INT;
    SET (v_id) = (SELECT ObjID FROM T1);
    IF (v_id = NULL) THEN
```

```
                SET v_id = 0;
        END IF;
        RETURN v_id;
    END
```

The value of v_id is set to null if the SELECT statement returns no rows.
Assuming the column ObjID is not nullable, if v_id is null after SELECT, it must
be the case that no rows were returned. The same technique can be applied to
triggers.

Of course, it could very well be that the logic is so complex that such a simple
substitution is not possible. In that case, the other option is to change your
application so that the function operates without handlers. If an error does occur,
then the exception will be returned to the function caller (whether a stored
procedure or the application itself) for handling.

5.6.3 Converting RAISE_APPLICATION_ERROR

Oracle PL/SQL permits the issuance of user-defined error messages from stored
procedures using RAISE_APPLICATION_ERROR. Thus, errors can be reported
to the application.

To call raise_application_error, you can use the following syntax:

```
raise_application_error(error_number, message[, {TRUE | FALSE}]);
```

DB2 SQL procedures support SIGNAL SQLSTATE statements to provide a
similar functionality.

Example 5-28 shows how to rewrite a new_title procedure to report a detection of
input value longer than 50 characters.

Example 5-28   Condition handler - SIGNAL SQLSTATE

```
CREATE PROCEDURE new_title_2 (IN in_title_desc VARCHAR(100))
LANGUAGE SQL
P1: BEGIN
    DECLARE SQLCODE INTEGER   DEFAULT 0;
    DECLARE SQLSTATE CHAR(5)  DEFAULT '  ';

    IF (length(in_title_desc) > 50) THEN
       SIGNAL SQLSTATE '71001'
       SET MESSAGE_TEXT = 'Value for update is too long';
    ELSE
       UPDATE books
       SET title_desc = in_title_desc
```

```
        WHERE author = 'JACK LONDON';
      END IF;
END P1
```

In the ORACLE PL/SQL exception handler, functions SQLCODE and SQLERRM can be used to determine which error occurred and to get the associated error message.

DB2 SQL PL supports the GET DIAGNOSTICS statement to obtain information related to the SQL statement just executed. This statement can be used within the CONDITION HANDLER declaration to return a message associated with the error:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
GET DIAGNOSTICS EXCEPTION 1 out_err_msg = MESSAGE_TEXT;
```

Together with the error message, GET DIAGNOSTICS permits retrieval of the number of rows that were affected by the previous UPDATE, DELETE, or INSERT statement:

```
GET DIAGNOSTICS  v_rowcount = ROW_COUNT;
```

The RETURN value of a nested procedure can be retrieved by using:

```
GET DIAGNOSTICS  v_retcode  = RETURN_STATUS;
```

5.7 Package initialization

With Oracle you can define initialization in PL/SQL packages, which is executed only one time when starting a new database session. Example 5-29 shows the initialization within an Oracle package.

Example 5-29   Oracle package with initialization

```
CREATE OR REPLACE PACKAGE BODY pkg_init_v1 AS

   -- function / procedure definition
   -- ...

-- Body initialization
BEGIN
   -- ...
END pkg_init_v1;
```

In DB2, each SQL PL block is a function or procedure beginning with CREATE FUNCTION or CREATE PROCEDURE. The MTK is able to convert Example 5-29 if you define a procedure with the initialization commands.

Example 5-30 shows an alternative of Example 5-29 with the same behavior. In the initialization, as illustrated in the example, part is now only the call of the procedure `init`. Within init is the source of the initialization.

Example 5-30   Oracle package with initialization as procedure

```
CREATE OR REPLACE PACKAGE BODY pkg_init_v2 AS

   -- function / procedure definition
   -- ...

-- initialization procedure
PROCEDURE init IS
BEGIN
   -- ...
END;

-- Body initialization
BEGIN
   init;
END pkg_init_v2;
```

Within your application, you have to add the call of the procedure init in order to perform the initialization every time you connect to the database.

5.8 Global variables

Global variables in Oracle are distinct for each connected user. The variables are global to the connection. The appropriate conversion is to use DB2 global temporary tables.

Example 5-31 shows a simple Oracle package with the definition and initialization of two global variables.

Example 5-31   Definition of global variables in Oracle

```
CREATE OR REPLACE PACKAGE pkg_gv IS

   global_variable_1  VARCHAR2(128) := NULL;
   global_variable_2  INTEGER       := 1;
```

```
END pkg_gv;
```

Definition and initialization

Example 5-32 is a DB2 stored procedure with the definition of a global temporary table. The table contains all the global variables you need within a session. Each column of the table corresponds to a global variable. The table needs only to have one row with the respective values.

The INSERT statement is necessary to initialize the global variables.

Example 5-32   Temporary table with global variables

```
CREATE PROCEDURE init_global_variables
LANGUAGE SQL
BEGIN
    -- declare temporary table for global variables
    DECLARE GLOBAL TEMPORARY TABLE session.global_variables (
        global_variable_1    VARCHAR(128)
       ,global_variable_2    INTEGER )
    ON COMMIT PRESERVE ROWS;

    -- initialize global variables
    INSERT INTO session.global_variables (
        global_variable_1
       ,global_variable_2 )
    VALUES ( null
           ,0 );
END!
```

Note: ON COMMIT PRESERVE ROWS indicates that rows of the table will be preserved after ending a transaction with COMMIT.

Using a procedure to initialize the temporary table yields two key benefits:

► The procedure developer does not have to hunt through application code (which may be maintained by another person) to find the DDL of the temporary table.

► The definition of the table is centralized at one place. If the global variables require changes, you do not have to search for all the declarations. Instead, you simply change the definition in one place.

As mentioned before, the values of the global variables are distinct for each connection. This means that you have to define the DB2 global temporary table

at the beginning of each session. To do this with the definition of Example 5-32, you need to call the stored procedure init_global_variables after you connect to the DB2 database in your application.

When developing or converting other procedures that rely on these global variables, you will have to run the init_global_variables procedure first to define the temporary table in the current connection. Otherwise, DB2 will not be able to resolve references to the temporary table at build time.

Setting values of global variables

To set a new value to a global variable, use an UPDATE statement to the corresponding column in the temporary table GLOBAL_VARIABLES as shown:

```
UPDATE session.global_variables
    SET global_variable_1 = new_value;
```

Getting values of global variables

To get the value of a global variable, use a SELECT statement to the corresponding column in the temporary table GLOBAL_VARIABLES, as shown:

```
SELECT global_variable_1
    INTO gv_value
    FROM session.global_variables
    FETCH FIRST 1 ROWS ONLY;
```

Under normal circumstances, you only have one row in the GLOBAL_VARIABLES table. To ensure that you get no more than one row as a result of the SELECT statement, use the FETCH FIRST 1 ROWS ONLY clause.

To prevent users from direct accessing the global temporary table, you may optionally encapsulate the statements to set or get values in stored procedures. In this case, you have to implement a procedure for each global variable. After that, you have to grant the user the appropriate authority.

Using INOUT parameters

When using only a few global variables shared by a few procedures, it might be more practical to simply convert the global variables as parameters. In this case, change the parameter definition of the procedures (which uses global variables) by adding an INOUT parameter for each of the global variables.

5.9 Hierarchical queries

The Oracle CONNECT BY ... START WITH ...clause can be used to select data that has a hierarchical relationship, usually some sort of parent - child relationship. In Example 5-33, the table EMPLOYEES consists of just these

attributes: parent and child. We make sure (by means of a unique constraint) that the child is unique within the table.

Example 5-33   Oracle hierarchical query

```
SELECT  substr(lpad(' ', level * 2) || emp_id,1,20)  AS emp_id
      , last_name
      , emp_mgr_id
      , level
FROM  employees
CONNECT BY PRIOR emp_id = emp_mgr_id
START WITH emp_mgr_id IS NULL;


EMP_ID                LAST_NAME              EMP_MGR_ID    LEVEL
--------------------  --------------------   ----------   ----------
  10000               Sands                                  1
  10001               Marcus                  10000          2
  10004               Polite                  10001          3
  10005               Tenor                   10001          3
  10002               January                 10000          2
  10008               Even                    10002          3
  10010               December                10002          3
  10011               August                  10002          3
  10003               March                   10000          2
  10006               Blonde                  10003          3
  10007               Damon                   10003          3
  10009               Ration                  10003          3
```

In the following example we provide a DB2 UDF to get the same result. The identity of each row is actually its encoded position in the hierarchy (this is arbitrary). The solution also assumes that the resulting sorting can be done based on the path. Here this works as long as no more than nine siblings exist on any given level. However, a simple formatting of the path to a specific number of digits per level can solve this problem.

The script uses SQL table functions. This capability, however, is only used for encapsulation and is not required for function.

The function get_direct_childs() collects all immediate children of a node in the hierarchy and returns them together with their relative position to each other and in the tree; see Example 5-34.

Example 5-34   Computing of direct child data

```
CREATE FUNCTION get_direct_childs(code VARCHAR(30), parent INTEGER)
RETURNS TABLE(code VARCHAR(30), id INTEGER)
```

```
READS SQL DATA
DETERMINISTIC
NO EXTERNAL ACTION
RETURN
  SELECT code || '.' || RTRIM(CHAR(RANK() OVER (ORDER BY child_id))),
  child_id
    FROM (SELECT empno FROM emp
          WHERE emp.mgr
                = get_direct_childs.parent)
        AS T(child_id)!
```

The function get_rec_childs() contains the recursive logic. It starts with the root, which is provided by the caller, and then collects children until no more new children can be found; see Example 5-35.

Example 5-35   Hierarchical query with entry point

```
CREATE FUNCTION get_rec_childs(root INTEGER)
RETURNS TABLE(code VARCHAR(30), id INTEGER)
READS SQL DATA
DETERMINISTIC
NO EXTERNAL ACTION
RETURN
  WITH rec(code, id)
    AS (VALUES(CAST('1' AS VARCHAR(30)), root)
        UNION ALL
        SELECT t.code, t.id
          FROM rec, TABLE(get_direct_childs(rec.code, rec.id)) AS T)
    SELECT code, id FROM rec !
```

The function get_level() computes the hierarchy level of the current data; see Example 5-36.

Example 5-36   Compute hierarchy level

```
CREATE FUNCTION get_level(code VARCHAR(30))
RETURNS INTEGER
DETERMINISTIC
NO EXTERNAL ACTION
RETURN
  (length(code) - length(replace(code, '.', '')))!
```

You get the hierarchy tree with the UDF get_rec_childs(). As a parameter, you have to specify the root.

Example 5-37 shows the use of the hierarchy function and its output with an adequate format.

Example 5-37   Sample use of hierarchical query

```
SELECT  T.code
       ,T.id
       ,substr(  (space(2 * get_level(code)) || employees.last_name)
                , 1
                , 20 ) as last_name
       ,emp_mgr_id
FROM  TABLE(get_rec_childs(10000)) AS T
     ,employees
where T.id = employees.emp_id
ORDER BY code!

CODE        ID          LAST_NAME            EMP_MGR_ID
----------- ----------- -------------------- -----------
1                10000  Sands                          -
1.1              10001  Marcus                     10000
1.1.1            10004  Polite                     10001
1.1.2            10005  Tenor                      10001
1.2              10002  January                    10000
1.2.1            10008  Even                       10002
1.2.2            10010  December                   10002
1.2.3            10011  August                     10002
1.3              10003  March                      10000
1.3.1            10006  Blonde                     10003
1.3.2            10007  Damon                      10003
1.3.3            10009  Ration                     10003
```

The article "Port CONNECT BY to DB2" at the following Web site explains how to map recursive queries from Oracle to DB2 using recursive common table expressions:

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0510rie lau/

5.10 Print output messages

When converting the call statement of Oracle function dbms_output.put_line(), the MTK provides only a function skeleton for you. You still need to implement the function that will best suit your application needs.

We offer you a user-defined function (UDF) solution called PUT_LINE() that will enable file output from pure SQL, such as an SQL stored procedure. You can use it as a tool for debugging stored procedures. However, it also allows you to write to a specified file for other purposes.

You can find the sources of the function and an installation guide as additional material on the IBM Redbooks Internet site; see details in Appendix G, "Additional material" on page 701.

5.11 Implicit casting in SQL

In addition to syntax differences, here we also highlight differences in how Oracle and DB2 handle data type casting. Whether you know it or not, Oracle performs implicit casting of data types (as required), while DB2 is strongly typed. Consider the following example:

```
[1] create table t1 (c1 number);
[2] insert into t1 ('1')
[3] select * from t1 where c1='1'
```

In the first line we create a table, t1, which has a numeric column. In line two, however, the character value 1 can be inserted into t1 without error. The value was implicitly cast by Oracle from varchar2 to number.

In line three, we have yet another example of implicit casting because the predicate c1 (numeric type) and 1 (character type) are allowed to be compared. Certainly there are many more cases than this. Implicit casting can happen in a variety of cases.

Because DB2 is strongly typed, SQL may have to be slightly rewritten to perform explicit casts. Implicit casting can raise positive and negative issues. It certainly makes SQL programming easier, but it can also be dangerous, and incurs some overhead, which is why purists will discourage its use.

Following the previous example, we can rewrite the series of SQL as follows:

```
[1] create table t1 (c1 INT);
[2] insert into t1 (1)
[3] select * from t1 where c1=1
```

Of course, these are the easiest cases to solve because it is obvious where the problem lies. Another common area where implicit casting must be resolved is in functions and operators. For example:

```
create procedure echo_input (v_num  in numver(9,0), v_echo out
varchar2)
```

```
    as
    begin
        v_echo := ' Input number is: ' || v_num;
    end;
```

In this case, the concatenation operator (||) in Oracle will automatically cast v_num to varchar2 so that the strings can be combined. In DB2, v_num will have to be explicitly cast to a character type.

If you convert this code through the MTK, it will help you identify all places where implicit casting is occurring by providing code to perform explicit casting. If you decide to perform a conversion without tools, it will take more time to find them. DB2 supports the CAST function to help you with datatype conversion.

Here how that code fragment will look in DB2:

```
    create procedure echo_input (v_num  in integer, v_echo out
    varchar(80))
    as
    begin
        v_echo := ' Input number is: ' || cast (v_num as CHAR(7));
    end;
```

Based on past experience, among the most time-consuming porting activities will revolve around converting implicit casting to explicit casting in dynamic SQL. Because dynamic SQL, in general, cannot be fully resolved until runtime, the MTK is unable to determine the proper casting functions to apply.

Consider the following example:

```
    create procedure dyn_cast
    as
        val varchar2(100) := '100';
    begin
        EXECUTE IMMEDIATE 'CREATE TABLE T1 (C1 number)';
        EXECUTE IMMEDIATE 'INSERT INTO T1 VALUES ( ''' || val || ''')';
    end;
```

Here we create a table T1 with a numeric column C1. When you convert to DB2, the procedure will build with very few changes. The final SQL statement as submitted to the database engine will look like this:

```
    INSERT INTO T1 VALUES ('100');
```

However, it will fail at runtime because implicit casting is occurring in the INSERT statement.

Dealing with implicit casting will be quite troublesome at first, but as you troubleshoot these problems, you will learn to quickly identify these situations. The upside is that it will ultimately make your applications cleaner and in some cases, better performing (especially in high volume SQL statements).

5.12 Outer join

Both Oracle and DB2 support outer join. DB2 supports the ANSI SQL syntax for three types of outer join: right, left, and full. Oracle supports the same syntax, starting in version 9i. Oracle also has proprietary left and right outer join syntax that DB2 does not support. Table 5-4 demonstrates how to map this old syntax to the DB2 equivalent for simple examples.

Table 5-4   Mapping of join definition

Oracle	DB2

`SELECT A.last_name, A.id,B.name`
`FROM emp A, Customer B`
`WHERE A.id (+) = B.sales_rep_id;`
	`SELECT A.last_name,A.id,B.name`
	`FROM emp A`
	`RIGHT OUTER JOIN customer B`
	`ON A.id = B.sales_rep_id;`

`SELECT A.last_name, A.id,B.name`
`FROM emp A, Customer B`
`WHERE A.id = B.sales_rep_id (+);`
	`SELECT A.last_name,A.id,B.name`
	`FROM emp A`
	`LEFT OUTER JOIN customer B`
	`ON A.id = B.sales_rep_id;`

`SELECT A.last_name, A.id,B.name`
`FROM emp A, Customer B`
`WHERE A.id (+) = B.sales_rep_id`
`(+);`
	`SELECT A.last_name,A.id,B.name`
	`FROM emp A`
	`FULL OUTER JOIN customer B`
	`ON A.id = B.sales_rep_id;`

The MTK provides basic support for Oracle outer joins, with the following restrictions:

► Only the equality (=) operator is supported.

► The (+) operator cannot follow a complex expression; it can follow a column reference only.

In some cases, the MTK will not be able to convert complex outer join syntax. The following example shows how a complex SQL statement involving multiple outer joins can be mapped from Oracle to DB2 syntax.

It is important to realize that in Oracle, outer joins are defined in the WHERE clause. By contrast, in DB2 they are defined in the FROM clause. Further, the

outer join condition of the two tables must be specified in the ON clause, not in the WHERE clause.

Example 5-38 shows the Oracle outer join syntax.

Example 5-38   Oracle outer joins

```
SELECT
    t1.surname
FROM
    EXAMPLE_TABLE1 t1,
    EXAMPLE_TABLE2 t2,
    EXAMPLE_TABLE3 t3,
    EXAMPLE_TABLE4 t4
WHERE
      ((t1.emptype = 1) OR (t1.position = 'Manager'))
    AND (t1.empid = t2.empid(+))
    AND (t2.empid = t3.empid(+))
    AND (t2.sin   = t3.sin(+))
    AND (t3.jobtype(+) = 'Full-Time')
    AND (t2.empid = t4.empid(+))
    AND (t2.sin = t4.sin(+))
ORDER BY
    t1.emptype, t2.other
```

Example 5-39 shows the DB2 conversion.

Example 5-39   DB2 outer join conversion

```
SELECT
    t1.surname,
FROM
      EXAMPLE_TABLE1 t1 LEFT OUTER JOIN
        EXAMPLE_TABLE2 t2 ON (t2.empid = t1.empid) LEFT OUTER JOIN
          EXAMPLE_TABLE3 t3 ON (t3.sin = t2.sin)
                                  AND (t3.empid = t2.empid)
                                  AND (t3.jobtype = 'Full-Time')
                              LEFT OUTER JOIN
            EXAMPLE_TABLE4 t4 ON (t4.sin = t2.sin)
                                  AND (t4.empid = t2.empid)
WHERE
    ((t1.emptype = 1) OR (t1.position = 'Manager'))

ORDER BY
    t1.emptype, t2.other
```

5.13 Decode statement

The Oracle DECODE statement can be converted to a DB2 CASE statement in two ways:

```
DECODE (condition, case1, assign1, case2, assign 2....,default)
```

This can be converted to the simple CASE statement:

```
CASE condition
    WHEN case1 THEN assign 1
    WHEN case2 THEN assign 2
    ....
    ELSE default
END
```

Or, to the searched CASE statement:

```
CASE
    WHEN condition THEN assign 1
    WHEN condition THEN assign 2
    ....
    ELSE default
END
```

For example, here is the Oracle code:

```
SELECT AVG(DECODE(Grade, 'A', 1,
                         'B', 2,
                         'C', 3,
                         'D', 4,
                         'E', 5))
INTO v_Grade FROM Students
WHERE DEPARTMENT = p_Department AND Course_ID = p_Course_ID;
```

This can be converted to DB2 code:

```
SELECT AVG(CASE GRADE WHEN 'A' THEN 1
                      WHEN 'B' THEN 2
                      WHEN 'C' THEN 3
                      WHEN 'D' THEN 4
                      WHEN 'E' THEN 5
                      END) INTO v_Grade
FROM Students
```

```
        WHERE DEPARTMENT = p_Department
          AND Course_ID = p_Course_ID;
```

5.14 Rownum

Oracle uses the ROWNUM pseudo-column to control the number of rows returned from an SQL statement. In DB2, you determine the number of rows to read with the FETCH FIRST n ROWS ONLY statement.

Table 5-5 shows how different statements can be converted.

Table 5-5   Mapping of the ROWNUM function

	Oracle	DB2
SELECT	select * from tab1 where ROWNUM < 10	select * from tab1 FETCH FIRST 9 ROWS ONLY;

UPDATE	update tab1
	set c1 = v1
	where c2 = v2
	and ROWNUM <= 10
		FOR lv as temp_cur CURSOR FOR
		   SELECT *
		     FROM tab1
		     WHERE c2 = v2
		     FETCH FIRST 10 ROWS ONLY
		   FOR UPDATE DO
		     UPDATE tab1
		     SET c1 = v1
		     WHERE CURRENT OF temp_cur;
		END FOR;
		
		With FixPak 4 or above:
		UPDATE (select c1 from
		tab1where c2=v2 fetch first 10
		rows only) set c1=v1

DELETE	delete from tab1
	where ROWNUM <= 100
		FOR lv as temp_cur CURSOR FOR
		   SELECT * FROM tab1
		   FETCH FIRST 100 ROWS ONLY
		   FOR UPDATE DO
		     DELETE FROM tab1
		     WHERE CURRENT OF temp_cur;
		END FOR;
		
		With FixPak 4 or above:
		DELETE FROM (SELECT 1 FROM tab1
		FETCH FIRST 100 ROWS ONLY)

5.15 INSERT, UPDATE, DELETE returning values

DB2 has introduced several new features that let you use SELECT and SELECT INTO statements to retrieve result sets from SQL data-change operations (INSERT, UPDATE, and DELETE) embedded in the FROM clause. This feature can be used to migrate Oracle code using a similar feature. Example 5-40 is sample Oracle code using a RETURNING INTO statement to retrieve a value after updating a table.

Example 5-40   Oracle code using RETURNING INTO

```
Update staff
Set salary =10000.0
where id =p_id
returning name into p_name;
```

Example 5-41 and Example 5-42 show two ways to convert this Oracle code into DB2 code; here SELECT INTO is shown.

Example 5-41   DB2 code using SELECT INTO

```
SELECT name INTO p_name
FROM NEW TABLE (
     UPDATE staff
     SET salary = 10000.0
     WHERE id = p_id);
```

Here, using SELECT is shown.

Example 5-42   DB2 code using SELECT

```
set p_name = (SELECT name
FROM NEW TABLE (
     UPDATE staff
     SET salary = 10000.0
     WHERE id = p_id));
```

5.16 Select from DUAL

Oracle provides a dummy table called DUAL, which is frequently used to retrieve system information. When converting references to DUAL, you have several options:

► Change the SELECT statement to a VALUES statement
► Directly assign special registers to variables (in SQL PL)

- Create a table or view called DUAL to mimic the Oracle DUAL table (DUAL would need to be created or aliased under all schemas)
- Use the DB2 dummy table SYSIBM.SYSDUMMY1, which has a single row and one column, IBMREQD, with a value of Y
- Create a synonym of SYSIBM.SYSDUMMY1 called DUAL

Table 5-6 illustrates dummy table usage.

Table 5-6   Use of dummy table for system information

Oracle	DB2

`select SYSDATE from DUAL`	`VALUES(CURRENT TIMESTAMP) INTO <variable>`
	
	or
	
	`select CURRENT TIMESTAMP from SYSIBM.SYSDUMMY1`

In some circumstances it may be too costly to comb through all source code to convert references to DUAL and Oracle system variables to use DB2 syntax. As an alternative, you can preserve your existing SQL by defining a view named DUAL with the (column) value(s) you need; see Example 5-43.

Example 5-43   DB2 dummy view for system information

```
create view dual (sysdate)
as select CURRENT TIMESTAMP from SYSIBM.SYSDUMMY1
!
```

Here is the result:

```
db2 => select sysdate from dual

SYSDATE
--------------------------
2003-10-15-18.03.59.399071

   1 record(s) selected.
```

5.17 Manipulating date and time

Both Oracle and DB2 have date and time data types and functions to get the date and time from the system or convert the date and time into different formats, and perform arithmetic on dates. Oracle has DATE, which can be mapped to

DB2 TIMESTAMP. DB2 has two other date and time functions known as DATE and TIME.

If your applications only use the date portion of Oracle's date data type, it will be more efficient to convert these types to DB2's DATE type (rather than TIMESTAMP).

Here we show examples of date manipulation.

▶ **Getting dates**

In Oracle, you use SELECT to get the date:

```
SELECT sysdate from dual;
```

In DB2, you use VALUES to get the date:

```
SELECT current timestamp FROM sysibm.sysdummy1;
SELECT current date FROM sysibm.sysdummy1;
SELECT current time FROM sysibm.sysdummy1;
```

▶ **Converting dates**

To convert the date in Oracle, TO_CHAR is used:

```
to_char(sysdate,'YYYY-MM-DD')
to_char(sysdate,'MM/DD/YYYY')
```

DB2 V8.1 supports functions TO_CHAR and TO_DATE but for only one format, as illustrated:

```
TO_CHAR (timestamp_expression,'YYY-MM-DD HH24:MI:SS')
TO_DATE (string_expression, 'YYY-MM-DD HH24:MI:SS')
```

In addition, you can use the CHAR function to convert the date and specify the localized format such as:

```
CHAR(current date,ISO);
CHAR(current date,USA);
```

The following additional examples demonstrate how DB2 dates can be converted to different formats:

```
char(current date) = '10/01/2003'
char(current date + 3 days) = '10/04/2003'
char(current date,ISO) = '2003-10-01'
char(current date,EUR) = '01.10.2003'
char(current date,JIS) = '2003-10-01'
char(current time,USA) = '02:21 PM'
char(current time + 2 hours,EUR) ='16.21.23'
```

Note: For more information about using the CHAR function for date and time conversion, refer to the DB2 publication *SQL Reference*.

- ► **Dates arithmetic**

   Dates arithmetic is frequently used. Here is an Oracle example:

   ```
   add_months(sysdate,16)
   ```

   In DB2, similar function can be implemented as follows:

   ```
   current date + 16 months
   ```

   Here are other examples of how arithmetic manipulation can be done with DB2 dates:

   ```
   current date = 10/02/2003
   current date + 3 days = 10/05/2003
   current timestamp + 2 years = 2005-10-02-12.33.27.667000
   current timestamp - 2 months = 2003-08-02-12.33.27.667002
   current time + 5 minutes = 12:38:27
   ```

   DB2 also provides other functions to manipulate with dates, such as *days, dayname, monthname,* and much more.

- ► **Using UDF**

   Certain Oracle functions not supported by DB2 can be easily duplicated by writing UDFs. For example, the following UDF can be used to convert the Oracle built-in function last_day:

   ```
   CREATE FUNCTION last_day(v_date date)
   RETURNS DATE
   SPECIFIC lastday
   LANGUAGE SQL
   CONTAINS SQL
   NO EXTERNAL ACTION
   DETERMINISTIC
   RETURN (v_date + 1 MONTHS) - DAY(v_date + 1 MONTHS) DAYS;
   ```

   You can find the UDF samples to convert Oracle DUMP(date), NEW_TIME, NEXT_DAY(), TRUNC() and other sample UDFs for migration at the following IBM Web site:

   http://www7b.software.ibm.com/dmdd/library/samples/db2/0205udfs/

   Another example of using DB2 UDF in dates arithmetic is converting the Oracle months_between function:

   ```
   months_between(sysdate,v_date)
   ```

   If you use the MTK to automate your conversion, it will implement months_between as a User Defined Function and automatically deploy it into the database. Here is the source code for this function:

   ```
   CREATE FUNCTION months_between(d1 TIMESTAMP, d2 TIMESTAMP)
    RETURNS FLOAT
    LANGUAGE SQL
    DETERMINISTIC
   ```

```
                    NO EXTERNAL ACTION
                    CONTAINS SQL
                    RETURN 12*(year(d1) - year(d2)) + month(d1) - month(d2)
                          + (TIMESTAMPDIFF(2,CHAR(d1 - (d2 + (12*(year(d1) - year(d2))
                            + month(d1) - month(d2)) MONTHS))) / 2678400.0)
```

This function employs the DB2 built-in function TIMESTAMPDIFF. Details for TIMESTAMPDIFF can be found in the DB2 manual *SQL Reference.*

Here we provide another version of a months_between UDF:

```
CREATE FUNCTION NoCASE.months_between(d1 TIMESTAMP, d2 TIMESTAMP)
RETURNS FLOAT
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
RETURN 12*(year(d1) - year(d2)) + month(d1) - month(d2)
+ SIGN( ABS(day(d1) - day(d2)) * (day(d1 + 1 day)*day(d2 + 1 day) -
1) )
*( (day(d1) - day(d2))*86400 + midnight_seconds(d1) -
midnight_seconds(d2) ) / 2678400.0
END
```

For more information about manipulation with dates, refer to the article on the DB2 developer domain:

http://www7b.software.ibm.com/dmdd/library/techarticle/0211yip/0211yip3
.html

5.18 Set operations

Set operators can be used to combine result sets in both Oracle and DB2. Table 5-7 lists the differences between the products.

Table 5-7   Mapping of set operations

Oracle	DB2
UNION	UNION
UNION ALL	UNION ALL
MINUS	EXCEPT
INTERSECT	INTERSECT

DB2 supports the ALL option on each operator, allowing duplicates to be preserved. Oracle allows ALL only on UNION.

Null values conversion

In Oracle, the function NVL provides a conversion of NULL values to non-null values:

```
NVL(TO_CHAR(MANAGER_ID),'No Manager')
```

This statement converts all of the NULL values in the manager_id column to the string No Manager.

In DB2, use the COALESCE function to convert nulls, as shown here:

```
COALESCE(MANAGER_ID,'No Manager')
```

5.19 Function that returns rowtype

Oracle allows result sets to be returned from functions if their return type is an REF cursor, as illustrated in Example 5-44. You will likely convert such functions as procedures in DB2, because this minimizes the number of application changes required (you would simply add the word CALL in front of the function call).

The (less desirable) alternative is to convert this as a table function, but table functions in DB2 have limited functionality, and the application must convert the function call into a SELECT statement.

Example 5-44   Oracle function with a REF cursor

```
CREATE OR REPLACE PACKAGE ReturnRtype AS
  TYPE t_RefCur IS REF CURSOR;

  -- Selects from employees based on the supplied department,
  -- and returns the opened cursor variable.
  FUNCTION EmployeesQuery(p_Department IN VARCHAR2)
    RETURN t_RefCur;
END ReturnRtype;

CREATE OR REPLACE PACKAGE BODY ReturnRtype AS

  -- Selects from employees based on the supplied department,
  -- and returns the opened cursor variable.
  FUNCTION EmployeesQuery(p_Department IN VARCHAR2)
    RETURN t_RefCur IS
    v_ReturnCursor t_RefCur;
    v_SQLStatement VARCHAR2(500);
  BEGIN
    v_SQLStatement := 'SELECT * FROM employees WHERE department = :m';
```

```
    -- Open the cursor variable, and return it.
    OPEN v_ReturnCursor FOR v_SQLStatement USING p_Department;
    RETURN v_ReturnCursor;
  END EmployeesQuery;
END ReturnRtype;
```

The converted DB2 code is shown in Example 5-45.

Example 5-45   Conversion to a procedure with a Result Set in DB2

```
Create Procedure EmployeesQuery (IN p_Department varchar(30))
LANGUAGE SQL
DYNAMIC RESULT SETS 1
BEGIN
    DECLARE C1 cursor with return to client for
    Select * from employees where department = p_Department;
    OPEN C1;
END!
```

5.20 Local functions

Oracle allows procedures or functions—called Local Procedures or Local
Functions—to be created and referenced within the body of a stored procedure.
When converting this to DB2, the Local Procedure or Local Function must be
created outside of the Stored Procedure that references it, before it can be used.

Example 5-46 shows how a Local Function created and referenced in an Oracle
stored procedure would be converted to DB2. It shows the Oracle source code of
the function.

Example 5-46   Oracle procedure with Local function

```
CREATE OR REPLACE PROCEDURE callLocalFunc AS
  /* Local declarations, which include a cursor, variable, and a
     function. */
  CURSOR c_AllEmployees IS
    SELECT first_name, last_name
      FROM employees;

  v_FormattedName VARCHAR2(50);
```

```
    /* Function which will return the first and last name
       concatenated together, separated by a space. */
    FUNCTION FormatName(p_FirstName IN VARCHAR2,
                        p_LastName IN VARCHAR2)
      RETURN VARCHAR2 IS
    BEGIN
      RETURN p_FirstName || ' ' || p_LastName;
    END FormatName;

-- Begin main block.
BEGIN
  FOR v_EmployeeRecord IN c_AllEmployees LOOP
    v_FormattedName :=
      FormatName(v_ EmployeeRecord.first_name,
                 v_ EmployeeRecord.last_name);
    DBMS_OUTPUT.PUT_LINE(v_FormattedName);
  END LOOP;
END callLocalFunc;
```

Example 5-47 shows the DB2 conversion of the function.

Example 5-47   DB2 Conversion of Oracle Procedure with Local Function

```
CREATE FUNCTION FormatName(p_FirstName VARCHAR(20),              -[1]
                           p_LastName VARCHAR(20))
    RETURNS VARCHAR(41)
LANGUAGE SQL
BEGIN ATOMIC
    RETURN p_FirstName || '' || p_LastName;
END!


CREATE PROCEDURE callFunc ()
LANGUAGE SQL

BEGIN
 DECLARE c_AllEmployees CURSOR WITH RETURN FOR
   SELECT FormatName(first_name, last_name) FROM employees;

 OPEN c_AllEmployees;
END!
```

The note in Example 5-47 is explained as follows:

[1] The function must be created before the procedure that references it.

5.21 Partitioning and MDC

If you are using partitioned tables on Oracle, you will need to decide how to partition your data in DB2. In this section we introduce the various partitioning features provided by DB2 and compare them to the partitioning techniques on Oracle.

In a single database partition, DB2 automatically organizes data on disk by distributing data in a round robin fashion across all containers of a table space. This method of data organization is the default behavior on DB2 and does not require any type of further definition. However, DB2 can be designed to organize data in other ways as well. These different data organization schemes can be specified at the database or table level.

The data organization methods available on DB2 are:

► Table partitioning
► Database partitioning
► Multidimensional clustering
► Combined organization schemes

Table partitioning

Table partitioning in DB2 is also referred to as *range partitioning* or *data partitioning*. This data organization scheme is one in which table data is divided across multiple storage objects called *data partitions* or *ranges* according to values in one or more table columns. Each data partition is stored separately and can be in different table spaces.

Example 5-48 is an example of table partitioning on DB2. The example demonstrates the use of a "shorthand" notation that automatically generates 24 partitions of uniform size; that is, one partition for each month over a two-year period. Note that MINVALUE and MAXVALUE will catch all values that fall below and above the defined ranges.

Example 5-48   DB2 table partitioning

```
CREATE TABLE orders
(
   l_orderkey        DECIMAL(10,0) NOT NULL,
   l_partkey INTEGER,
   l_suppkey INTEGER,
```

```
    l_linenumber      INTEGER,
    l_quantity        DECIMAL(12,2),
    l_extendedprice  DECIMAL(12,2),
    l_shipdate        DATE
) PARTITION BY RANGE(l_shipdate)
   (STARTING MINVALUE,
    STARTING '1/1/1992' ENDING '12/31/1993' EVERY 1 MONTH,
    ENDING AT MAXVALUE);
```

Example 5-49 illustrates table partitioning using manual syntax, which is required when the partitioning key is composed of a composite column.

Example 5-49   Table partitioning using manual syntax

```
CREATE TABLE sales
              (
               year INT,
              month INT
              )
   IN tbsp1, tbsp2, tbsp3, tbsp4, tbsp5, tbsp6, tbsp7, tbsp8
   PARTITION BY RANGE (year, month)
   (STARTING FROM (2001, 1)
            ENDING (2001,3)  IN tbsp1,
            ENDING (2001,6)  IN tbsp2,
            ENDING (2001,9)  IN tbsp3,
            ENDING (2001,12) IN tbsp4,
            ENDING (2002,3)  IN tbsp5,
            ENDING (2002,6)  IN tbsp6,
            ENDING (2002,9)  IN tbsp7,
            ENDING AT MAXVALUE );
```

Oracle's range partitioning is conceptually comparable to table partitioning on DB2. The differences between them lie mainly in the syntax used to define how the table is partitioned.

As a comparison, the DB2 example (Example 5-49) has been rewritten to show how range partitioning might be written on Oracle (Example 5-50).

Example 5-50   Oracle range partition

```
CREATE TABLE  sales
(
    year     int,
    month    int
)
```

```
    PARTITION BY RANGE (year, month)
   (PARTITION p1 VALUES LESS THAN (2002,4)        tablespace tbsp1,
    PARTITION p2 VALUES LESS THAN (2002,7)        tablespace tbsp2,
    PARTITION p3 VALUES LESS THAN (2002,10)       tablespace tbsp3,
    PARTITION p4 VALUES LESS THAN (2002,13)       tablespace tbsp4,
    PARTITION p5 VALUES LESS THAN (2003,4)        tablespace tbsp5,
    PARTITION p6 VALUES LESS THAN (2003,7)        tablespace tbsp6,
    PARTITION p7 VALUES LESS THAN (2003,10)       tablespace tbsp7,
    PARTITION p8 VALUES LESS THAN (MAXVALUE)      tablespace tbsp8 );
```

Note that on Oracle each partition is given a name. In contrast, on DB2 the partitions in these examples are not named. On DB2, if the partition is not explicitly named, a system name is generated by default. However, because the system name can be long and cumbersome, it is recommended that you explicitly name the partition. This can be done using the PART or PARTITION keyword.

Also note that on Oracle each partition contains values less than, and not including, the value that defines that partition. On DB2, the values defined for each partition are included within that partition.

DB2 provides a method of table partitioning that is based on a generated expression of a column. Depending on the situation, table partitioning on a generated column may be used in a similar way to list partitioning on Oracle.

Example 5-51 is an example of Oracle's list partitioning.

Example 5-51   Oracle list partitioning

```
CREATE TABLE customer
(
     cust_id        int,
     cust_prov    varchar2(2)
)
    PARTITION BY LIST (cust_prov)
    (PARTITION p1 VALUES ('AB', 'MB") tablespace tbsp_ab,
     PARTITION p2 VALUES ('BC')       tablespace tbsp_bc,
     PARTITION p3 VALUES ('SA')       tablespace tbsp_mb,
               ….
     PARTITION p13 VALUES ('YT')      tablespace tbsp_yt,
     PARTITION p14 VALUES(DEFAULT)    tablespace tbsp_remainder );
```

Example 5-52 shows how Oracle's list partitioning can be written as DB2 table partitioning based on a generated column.

Example 5-52   DB2 conversion of Oracle list partition

```
CREATE TABLE customer
(
    cust_id         INT,
    cust_prov   CHAR(2),
    cust_prov_gen GENERATED ALWAYS AS
    (CASE
        WHEN cust_prov = 'AB' THEN 1
        WHEN cust_prov = 'BC' THEN 2
        WHEN cust_prov = 'MB' THEN 1
        WHEN cust_prov = 'SA' THEN 3
                    ...
        WHEN cust_prov = 'YT' THEN 13
    ELSE 14
     END)
)
IN tbsp_ab, tbsp_bc, tbsp_mb, .... tbsp_remainder
    PARTITION BY RANGE (cust_prov_gen)
                (STARTING 1 ENDING 14 EVERY 1);
```

In Example 5-52, numeric values are generated based on values for CUST_PROV. The numeric values populate the generated column, CUST_PROV_GEN, on which table partitioning is based.

Because an automatic version of the syntax is used in this example, it is sufficient to list the table spaces for each partition with a single IN clause. If the manual version of the syntax was used and each partition was defined individually, then an IN clause for *each* partition would be required (as shown in Example 5-49 on page 228).

Database partitioning

On DB2, database partitioning is used when the database is created as a multiple partitioned database. This feature of DB2 is an optional feature and is known as the Database Partitioning Feature (DPF). DPF is mostly used for large, data warehousing applications although it can be used in some types of OLTP applications as well. When database partitioning is used, the multiple database partitions appear and work together as a single unit, This architecture allows complex data access tasks to run on different parts of the data in parallel.

When this feature is enabled, data organization is based on a hashing algorithm that distributes table data across the multiple database partitions. Each database partition can reside on a separate physical machine. Data is hashed according to a distribution key that is either explicitly defined in the table using the DISRIBUTE BY HASH clause, or is defaulted to the first qualified column. Ideally,

a distribution key is chosen that can hash the table data evenly across all database partitions.

Example 5-53 is an example of how a table is defined when database partitioning is used.

Example 5-53   Define a table in a partitioned database

```
CREATE TABLE partition_table
 (partition_date date NOT NULL,
  partition_data VARCHAR(20) NOT NULL
  )
IN tbsp_parts
DISTRIBUTE BY HASH (partition_date);
```

The DISTRIBUTE BY HASH clause of a table is only used in a multiple database environment. To partition data in a single database partitioned environment, table partitioning or multidimensional clustering organization is used.

Hash partitioning on Oracle is done in a single database environment. Example 5-54 shows an example of how hash partitioning syntax on Oracle compares to DB2.

Example 5-54   Oracle hash partitioning

```
CREATE TABLE hash_table
  (
      hash_part           date,
      hash_data           varchar2(20)
  )
  PARTITION BY HASH(hash_part)
(partition p1 tablespace tbsp1,
partition p2 tablespace tbsp2
);
```

Multidimensional clustering

Multidimensional clustering, also known as an MDC table, is a method of data organization that clusters data together on disk according to dimension key values. A *dimension* is a key or attribute, such as product, time period, or geography, used to group factual data into a meaningful way for a particular application.

A dimension can consist of a composite of two or more columns. A desirable characteristic of dimension values is that they are of a low cardinality and consist of a minimal number of unique values.

Example 5-55 is an example of an MDC table definition.

Example 5-55   MDC table definition

```
CREATE TABLE SALES
(
STORE INT NOT NULL,
SKU INT NOT NULL,
DIVISION INT NOT NULL,
QUANTITY INT NOT NULL
)
ORGANIZE BY DIMENSIONS (STORE, SKU);
```

As shown in Example 5-55, a table is created that specifies that division and quantity is to be organized by two dimensions, STORE and SKU. All data in the table will be stored on disk in blocks of data organized by STORE and SKU values. Each block on disk will only contain rows of data based on a unique set of dimension values.

When dimension keys are used as predicates in the WHERE clause of a SELECT statement, query performance is usually greatly improved because many rows are retrieved with fewer I/Os. In addition, performance benefits are gained from the smaller block index that is used with MDC tables. Because all rows in a block are referenced by the same dimensions, only one index entry per dimension is needed to locate all the rows in that block.

Oracle does not have an data organization scheme that is similar to the MDC table.

Combining methods of data organization

Just as Oracle has composite partitioning, a variety of data organization schemes can be combined on DB2.

These combinations are:

► Database partitioning with a sublevel of table partitioning

► Database partitioning with a sublevel of MDC data organization

► Database partitioning with a sublevel of table partitioning followed by a sublevel of MDC data organization

► Table partitioning with a sublevel of MDC data organization

Example 5-56 shows an example of combining database partitioning, table partitioning, and MDC organization.

Example 5-56   Combining database partitioning, table partitioning, and MDC

```
CREATE TABLE orders
(
order_id INTEGER,
ship_date DATE,
region SMALLINT,
category SMALLINT
)
IN tbsp1, tbsp2, tbsp3, tbsp4
DISTRIBUTE BY HASH (order_id)
PARTITION BY RANGE (ship_date)
(STARTING FROM ('01-01-2005') ENDING ('12-31-2006') EVERY (1 MONTH))
ORGANIZE BY DIMENSION (region, category);
```

In Example 5-56, the data is distributed over multiple database partitions using a hashed value of ORDER_ID. Within each database partition, the table is partitioned by the SHIP_DATE month, and within each table partition the data is organized in blocks by dimensions REGION and CATEGORY.

On Oracle, composite partitioning is used to combine the following types of partitioning methods.

These combinations are:

► Range partitioning with hash subpartitioning
► Range partitioning with list subpartitioning

The composite partitioning on Oracle is used when partitioning by a range alone does not provide enough granularity for managing a partition. On DB2, to break down a table partition into smaller units, you can use a composite column as a range partitioning key. The range partition key is defined by the PARTITION BY RANGE clause as shown in Example 5-57.

Example 5-57   Using PARTITION BY RANGE clause

```
CREATE TABLE sales
              (
               year INT,
              month INT
              )
  IN tbsp1, tbsp2, tbsp3, tbsp4, tbsp5, tbsp6, tbsp7, tbsp8
  PARTITION BY RANGE (year, month) ….
```

If adding a secondary column to the partitioning key is not possible, then use a generated column to complete the composite column.

Table 5-8 summarizes comparisons between Oracle's partitioning methods with DB2.

Table 5-8   Mapping Oracle data organization schemes to DB2

Oracle partitioning	DB2 data organization	Oracle 10g syntax	DB2 9.1 syntax
No equivalent	Round robin	None	Default: occurs automatically on single partition database
Range partitioning	Table partitioning	PARTITION BY RANGE	PARTITION BY RANGE
Hash partitioning	Database partitioning	PARTITION BY HASH	DISTRIBUTE BY HASH
List partitioning	Table partitioning with generated column	PARTITION BY LIST	PARTITION BY RANGE
Composite partitioning: hash-range hash-list	Combination of: database partitioning, table partitioning, multidimensional clustering	PARTITION BY RANGE, SUBPARTITION BY HASH SUBPARTITION BY LIST	DISTRIBUTE BY HASH PARTITION BY RANGE ORGANIZE BY DIMENSIONS
No equivalent	Multidimensional clustering	None	ORGANIZE BY DIMENSIONS

Indexes

With the table partitioning feature, each index on a partitioned table can be placed into its own table space regardless of the underlying table space for the table. The CREATE INDEX statement supports the IN tablespace-name clause as follows:

```
CREATE INDEX index1 ON table1 IN tablespace1;
```

This means that if the partitioned table uses an SMS table space, each index can be placed into its own table space. If no table space for an index is provided, the index will be placed into the first table space specified for a table.

Indexes that are created on partitioned tables are global; that is, the index holds entries for all partitions on the table.

Roll-in and roll-out of data

DB2 supports attaching a new partition to an existing partitioned table (roll-in) and the detaching of a partitioned table into a single table (roll-out). This functionality is achieved by using the ATTACH PARTITION and DETACH PARTITION clauses of the ALTER TABLE statement. By attaching a new partition to a table, you facilitate the adding of a new range of data to a partitioned table. A new partitioned range can be added anywhere in the table, and not only to the high end of the table.

To attach a partition, the data is loaded into a newly created table and then that table is attached to the existing partitioned table. Example 5-58 shows the newly created table DEC03 that has been loaded with data rolled into the partitioned table STOCK.

Example 5-58   Roll-in

```
ALTER TABLE stock ATTACH PARTITION dec03
STARTING FROM '12/01/2003' ENDING AT '12/31/2003'
FROM dec03;
COMMIT WORK ;
```

The new table that is attached must match the existing table in several ways. The source and target tables must match in column order and definitions, default values, nullability, compression and table space types used.

When a source is newly attached it is offline and remains offline until the SET INTEGRITY statement is executed. The following example shows the SET INTEGRITY statement:

```
SET INTEGRITY FOR stock ALLOW WRITE ACCESS
IMMEDIATE CHECKED FOR EXCEPTION IN stock USE stock_ex;
COMMIT WORK;
```

SET INTEGRITY validates the data in the newly attached data partition. The COMMIT WORK is needed to end the transaction and to make the table available for use.

In a similar way, an existing table can have a partition detached into a separate table by using the ALTER statement:

```
ALTER TABLE stock DETACH PART dec01 INTO stock_drop;
DROP TABLE stock_drop;
```

In addition, partitioned tables may be modified using the ADD PARTITION and DROP PARTITION options of the ALTER TABLE statement. The ADD PARTITION clause is used to add an empty partition with a new range to an

existing partitioned table. After it is added, the partitioned table can be loaded with data.

5.22 %ROWTYPE and %TYPE

The %TYPE and %ROWTYPE PL/SQL declarations are used to define PL/SQL variables to inherit the data type definition from other variable or database columns. These declarations are particularly useful for variables used to hold database data values. The %TYPE declaration is used to inherit a single column data type definition. The %ROWTYPE declaration is used to retrieve a record type that represents all columns in a table or cursor result set.

The MKT does not automatically convert %TYPE and %ROWTYPE declarations that use collection or cursor variable as object. (For a more detailed discussion of collection conversion, refer to 5.5, "Collections" on page 193.)

In this section, we discuss how the MTK handles the conversion of %TYPE and %ROWTYPE variable declarations.

%TYPE

In Oracle, you can refer to the type of an existing variable, field, or column by using the %TYPE attribute in PL/SQL. The MTK supports %TYPE for variables, columns, and fields of record variables. The MTK replaces these references directly by the type they refer to.

Example 5-59 shows some Oracle %TYPE examples.

Example 5-59   Oracle %TYPE

```
-- Base table: employees (emp_id int, first_name char(20))

-- %TYPE referencing to a table column definition
v_id employee.emp_id%TYPE;
v_fname employee.emp_name%TYPE;

--%TYPE referencing to a variable definition
v_id2 var_id%TYPE;
v_fname2 var_name%TYPE;

--%TYPE referencing to a table variable
TYPE Typ_Rec1 IS RECORD (emp_id INT, first_name CHAR(20));
var1 Typ_Rec1;
```

The converted DB2 code is shown in Example 5-60.

Example 5-60   DB2 conversion of %TYPE

```
-- Base table: employee (emp_id INTEGER, first_name CHAR(20))

DECLARE V_ID INTEGER;
DECLARE V_FNAME CHAR(20);

DECLARE V_ID2 INTEGER;
DECLARE V_FNAME2 CHAR(20);

DECLARE VAR1_EMP_ID INTEGER;
DECLARE VAR1_FIRST_NAME CHAR(20);
```

%ROWTYPE

On Oracle, you can use %ROWTYPE in PL/SQL to refer to the record type that represents a row, a cursor, or a cursor variable. PL/SQL scripts containing the %ROWTYPE attribute are handled in the same way as a record type that has been previously declared.

During the conversion process, the MTK produces a list of DB2 variables matching the record fields, and creates a variable for each of these record fields. In addition to the variable declaration, you must also change how the variables are referenced in the code. Oracle uses dot notation to reference the field in the record, and after the conversion you must directly use the new variable created by the MTK.

Example 5-61 shows an example of Oracle %ROWTYPE usage.

Example 5-61   Oracle %ROWTYPE declaration and usage

```
-- Base table: employee (emp_id int, emp_name char(40))
declare
   -- ROWTYPE declaration
   v_emp employees%ROWTYPE ;
begin
   -- Assigning values
   v_emp.emp_id     := 1000  ;
   v_emp.first_name := 'John';
end;
/
```

Example 5-62 shows the %ROWTYPE converted to DB2.

Example 5-62   %ROWTYPE conversion to DB2

```
-- Base table: employee (emp_id INTEGER, emp_name CHAR(40))
```

```
BEGIN ATOMIC

DECLARE V_EMP_EMP_ID INTEGER;
DECLARE V_EMP_EMP_NAME CHAR(40);

SET V_EMP_EMP_ID = 1000 ;
SET V_EMP_EMP_NAME = 'John';

END!
```

5.23 MERGE

The MERGE statement is used to merge data from a source, such as table, view, or query, into a target, such as table or view. Depending on whether the target already has the data to be merged, you can specify different actions, for example, insert the new rows or update or delete existing ones.

The MTK translates the Oracle MERGE statement to DB2 with the exception of the following clauses:

► Optimizer hints
► error_logging_clause

Because the Oracle optimizer hints are not applicable to DB2, if used in a MERGE statement, you have to manually convert the MERGE statement. The error_logging_clause is not supported in DB2.

DB2 and Oracle MERGE statements perform differently when a DELETE clause is included that contains an UPDATE clause. DB2 rows that were updated by the MERGE statement containing an UPDATE clause will not be deleted if they match the condition in the DELETE clause. When you use a translated MERGE statement (by the MTK), you need to keep this difference in mind.

We show a conversion example of a MERGE statement with the DELETE clause. We use two tables in our example: STAGE_TABLE and PRODUCTS. STAGE_TABLE is our source table, and its data will be merged into the PRODUCTS table, our target table.

These two tables have the following data:

► STAGE_TABLE

```
select * from stage_table;

    PROD_NO DESCRIPTION                     QUANTITY
```

```
---------- ----------------------------- ----------
         1                                        7
         2                                        2
         4                                        1
         5 Hoad                                   9
         6 Cap                                   20
```

► PRODUCTS

```
SQL> select * from products;

PROD_NO DESCRIPTION                         QUANTITY
---------- ----------------------------- ----------
         1 Shoes                                 10
         2 Socks                                  5
         3 Shirts                                25
         4 Paints                                 3
```

Example 5-63 shows the MERGE statement in Oracle, and the result of the MERGE statement.

Example 5-63   Oracle MERGE example

```
MERGE INTO products pd
  USING (SELECT prod_no, description, quantity
           FROM stage_table) st
        ON (pd.prod_no = st.prod_no)
  WHEN MATCHED THEN
       UPDATE SET pd.quantity = pd.quantity + st.quantity
       DELETE WHERE pd.quantity > 15
WHEN NOT MATCHED THEN
       INSERT (prod_no, description, quantity)
           VALUES (st.prod_no, st.description, st.quantity);

5 rows merged.

SQL> select * from products;

PROD_NO DESCRIPTION                         QUANTITY
---------- ----------------------------- ----------
         2 Socks                                  7
         3 Shirts                                25
         4 Paints                                 4
         5 Hoad                                   9
         6 Cap                                   20
```

In Example 5-63, the DELETE clause of the MERGE statement deletes only matched rows with quantity greater than 15 after the update operation. Note that although products 3 and 6 have quantities greater than 15, they are not deleted because they are not updated by the MERGE statement. The DELETE clause will not be executed against any row affected by the UPDATE clause.

Example 5-64 shows how to achieve similar functionality in DB2. In this example, we first merge two tables and set the quantity accordingly. Since the Oracle MERGE statement only deletes rows after they have been updated, after merge, we use the DELETE statement to remove the records that have a quantity greater than 15.

Example 5-64   DB2 MERGE conversion example

```
MERGE INTO products pd
  USING (SELECT prod_no, description, quantity
            FROM stage_table) st
        ON (pd.prod_no = st.prod_no)
  WHEN MATCHED THEN
      UPDATE SET
            pd.quantity = pd.quantity + st.quantity
  WHEN NOT MATCHED THEN
      INSERT (prod_no, description, quantity)
            VALUES (st.prod_no, st.description, st.quantity)


DELETE FROM products pr WHERE pr.prod_no IN (SELECT pd.prod_no FROM
products pd INNER JOIN stage_table st ON (pd.prod_no = s
t.prod_no) WHERE (pd.quantity + st.quantity) > 15 AND pd.quantity <>
st.quantity )

db2 => select * from products

PROD_NO     DESCRIPTION                       QUANTITY
----------- ----------------------------- -----------
          2 Socks                                   7
          3 Shirts                                 25
          4 Paints                                  4
          5 Hoad                                    9
          6 Cap                                    20

  5 record(s) selected.
```

5.24 Index conversion

Indexes are used to speed up queries by providing the optimizer another way of retrieving data other than a sequential scan. Most indexes in the Oracle database are translated by the MTK, but there are a few differences between Oracle and DB2.

INCLUDE column index

INCLUDE column index is introduced in DB2 9. When creating a unique index, you have the option to include extra columns to the index using the INCLUDE clause. The INCLUDE columns are stored with the index, but will not be sorted and considered for uniqueness. Use of INCLUDE columns improves the performance of data retrieval when index access is involved. DB2 does not need to access the data page because the data value is already available on the index page.

Creating an index with INCLUDE columns is shown in Example 5-65.

Example 5-65   Creating index with INCLUDE columns

```
CREATE UNIQUE INDEX ix1 ON employee
      (name ASC)
       INCLUDE (dept, mgr, salary, years)
```

5.24.1 Differences between Oracle and DB2

Both Oracle and DB2 use index for query performance optimization. There are differences in the implementation of indexes between Oracle and DB2.

Different meaning of cluster index

In Oracle, a cluster index means an index on a clustered table or a partitioned table.

In DB2, if the index is created with the CLUSTER option, the index is the clustering index of the table. While creating the clustering index, the data in the table is rearranged in the same order as that of the index.

The clustering index provides performance enhancements when a query scans most of the data in the same order as that of the index. When a new row is inserted, an attempt is made to keep the new row physically close to rows that have key values logically closed in the index-key sequence. Over time, updates and inserts may make the table less well clustered in relation to the index. You might need to periodically reorganize the table to make the data clustered again.

Example 5-66 provides an example of the command that is used to create a clustering index.

Example 5-66   DB2 clustering index

```
CREATE INDEX inxcls_emp_empno
ON employee(empno ASC)
CLUSTER
PCTFREE 10
MINPCTUSED 40;
```

Bitmap indexes

The Oracle bitmap index is not available in DB2. This type of index is aimed at data warehousing and is suitable for an index where there are very few key values (low cardinality)—for example, gender or state.

Although you can't explicitly create a bitmap index in DB2, the DB2 optimizer may create dynamic bitmap indexes during the execution of certain types of queries.

Functional indexes

An Oracle function-based index computes the value of the function or expression and stores it in the index. The function-based index is not available in DB2. To achieve similar functionality, you should create a computed column with the generated values for the function expression, and then create an index on this column.

5.25 Oracle database links

A table that resides in another database can be accessed like a local table through the features delivered by the database management systems. In Oracle, the database link provides this capability, while in DB2, the Homogeneous Federation Feature delivers the ability to access database objects in different DB2 data servers.

You can have unified access to the data managed by multiple data servers, including DB2 and Informix with DB2 Homogeneous Federation Feature. This allows applications to access and integrate diverse data—mainframe and distributed—as though it were a DB2 table, regardless of where the information resides, while retaining the autonomy and integrity of the data sources. The WebSphere Federation Server significantly expands the choice of data sources to any kind of data including database management systems on various platforms, flat files, Excel®, rich media, e-mails, XML, as well as LDAP.

For more information about WebSphere Federation Server, refer to:

http://www.ibm.com/software/data/integration/support/federation_server/

or the DB2 manual *Administration Guide for Federated Systems,* SC19-1020.

DB2 Homogeneous Federation Feature enables you to:

► Gain virtualized real-time access to disparate data sources
► Speed time to market for new projects
► Access more data sources
► Extend your data warehouse or data mart with remote data

Setting up federated databases

Setting up the federated databases is simple. We demonstrate the steps by an example. In this example, we want to join the LOCAL_DEPARTMENT table in the DB2_EMP database with the EMPLOYEE table in database SAMPLE. Since the query will be executed on DB2_EMP, it will be the federated server.

1. Enable the Federation feature.

   The Federation feature is enabled by setting the DB2 database manager configuration (DBM CFG) parameter FEDERATED to YES on the federated server. You can check and set the value as shown in Example 5-67.

Example 5-67   Enabling Federation feature

```
/WORK # db2 get dbm cfg |grep "Federated Database"
Federated Database System Support         (FEDERATED) = NO
/WORK # db2 update dbm cfg using federated yes immediate
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command
completed
successfully.
/WORK # db2 get dbm cfg |grep "Federated Database"
Federated Database System Support         (FEDERATED) = YES
```

Once the FEDERATED value in DBM CFG is changed, it is applied after restarting the database server. See Example 5-68.

Example 5-68   Restart the server

```
/WORK # db2stop force
02/14/2007 10:09:04    0   0   SQL1064N  DB2STOP processing was
successful.
SQL1064N  DB2STOP processing was successful.
/WORK # db2start
02/14/2007 10:09:08    0   0   SQL1063N  DB2START processing was
successful.
```

```
SQL1063N  DB2START processing was successful.
```

2. Configure the component that is needed to federate the table in another database.

   – WRAPPER - A mechanism by which a federated server can interact with certain types of data sources. In our example, we create a wrapper for the SAMPLE database from DB2_EMP. You can check the data source from the catalog view syscat.servers.

   – SERVER - A data source to a federated database. In our example, the data source is the SAMPLE database.

   – USER MAPPING - Definition of mapping between an authorization ID that uses a federated database and the authorization ID and password to use at a specified data source.

   – NICKNAME - Alias for a data source object. In our example, we create a nickname, REMOTE_EMPLOYEE, for the EMPLOYEE table of the SAMPLE database. From DB2_EMP, we then use this nickname in the query to join data.

   Example 5-69 shows the script, fed_config.db2, we used to set up the federated system in our lab.

   Example 5-69   Source of fed_config.db2

```
CONNECT TO db2_emp;
-----------------------------------------------
-- create  wrapper,user mapping, and nickname
-----------------------------------------------
--Create wrapper;
CREATE WRAPPER drda;
SELECT * FROM syscat.wrappers;

CREATE SERVER fedserver
  type db2/udb version '9.1'
  WRAPPER "DRDA"
  AUTHID "db2inst1"
  PASSWORD "db2inst1"
  OPTIONS ( dbname 'SAMPLE' );
--
SELECT * FROM syscat.servers
-- Map user

CREATE USER MAPPING FOR USER
  SERVER fedserver
  OPTIONS( REMOTE_AUTHID 'db2inst1', REMOTE_PASSWORD 'db2inst1');
```

```
--
SELECT * FROM syscat.usermappings;

CREATE NICKNAME remote_employee FOR fedserver.db2inst1.employee;
```

Use the following command to execute the script:

/WORK # **db2 -tf** fed_config.db2

You can see the registered federation values in the result of the selection from the system catalog tables.

After you configure a Federation server and link a remote table with a nickname in the local database, you can access the remote table because it is in the local database.

Example 5-70 demonstrates that you can delete and insert records on table EMPLOYEE of the SAMPLE database from DB2_EMP because it is one of the tables in this database.

Example 5-70   Access to a remote table

```
CONNECT TO db2_emp;
------------------------------------------------
-- insert a row in remote table
------------------------------------------------
DELETE FROM remote_employee WHERE empno='999999';
INSERT INTO
remote_employee(empno,firstnme,lastname,workdept,edlevel)
        VALUES ('999999','CARLOS','EDUARDO','E11',20);
SELECT empno,firstnme,workdept
        FROM remote_employee WHERE empno='999999';
------------------------------------------------
-- Result of the SELECT
------------------------------------------------
EMPNO  FIRSTNME     WORKDEPT
------ ------------ --------
999999 CARLOS       E11
```

You also can join the remote table with the local table as shown in Example 5-71.

Example 5-71   Join data in two tables

```
connect to db2_emp;
------------------------------------------------
-- Create a local table and insert 3 rows
------------------------------------------------
CREATE TABLE local_department
```

```
                    (deptno CHAR(3), deptname CHAR(20));

    INSERT INTO local_department VALUES
            ('E01','Operation'),
            ('E10','Sales'),
            ('E11','Global Services');

    SELECT * FROM local_department;
    ----------------------------------------------
    -- Join data in two tables
    ----------------------------------------------
    SELECT empno, firstnme, deptname
            FROM remote_employee r, local_department d
            WHERE r.workdept=d.deptno and empno='999999';
```

The result of joining a remote table (REMOTE_EMPLOYEE) and a local table
(LOCAL_DEPARTMENT) is shown in Example 5-72.

Example 5-72   The result of joining two tables

```
-- Result of the first SELECT (SELECT FROM local_department)

DEPTNO DEPTNAME
------ --------------------
E01    Operation
E10    Sales
E11    Global Services

-- Result of the second SELECT ( Join local_department and
remote_employee)

EMPNO  FIRSTNME     DEPTNAME
------ ------------ --------------------
999999 CARLOS       Global Services
```

5.26 Temporary tables

An Oracle temporary table can be translated as the DB2 global temporary table.

In DB2, the DECLARE GLOBAL TEMPORARY TABLE statement defines a
temporary table for the current session. The description of the declared
temporary table does not appear in the system catalog. It is not persistent and
cannot be shared with other sessions. Each session that defines a declared

global temporary table of the same name has its own unique description of the temporary table. When the session terminates, the rows of the table are deleted, and the description of the temporary table is dropped.

The USER TEMPORARY table space must exist to create a global temporary table.

Example 5-73 shows how to create a DB2 global temporary table.

Example 5-73   Creating a DB2 global temporary table

```
CREATE USER TEMPORARY TABLESPACE demotmp
  PAGESIZE 4 k MANAGED BY SYSTEM
  USING ('/db2/demotmp');

DECLARE GLOBAL TEMPORARY TABLE E1
   LIKE employee
   ON COMMIT PRESERVE ROWS NOT LOGGED IN demotmp;

INSERT INTO session.e1
   SELECT * FROM employee WHERE workdept = 'D21';

SELECT empno, workdept, salary  FROM session.e1;
```

The result of this script is shown in Example 5-74.

Example 5-74   Data in the global temporary table

```
EMPNO  WORKDEPT SALARY
------ -------- -----------
000070 D21         96170.00
000230 D21         42180.00
000240 D21         48760.00
000250 D21         49180.00
000260 D21         47250.00
000270 D21         37380.00
200240 D21         37760.00
```

5.27 Concurrency and transaction

Both concurrency control and locking are used to ensure the data integrity in any DBMS. The concurrency can be categorized into *read concurrency* and *update concurrency*.

First of all, the default concurrency processing is different for Oracle and DB2. So when you migrate an application to DB2, you need to keep in mind the fundamental differences shown in Table 5-9.

Table 5-9   Summary of different default settings between Oracle and DB2

	Oracle	DB2
Lock mode	Set lock node to wait	Set lock mode to not wait
Isolation level	Set isolation to dirty read	Set isolation to cursor stability

5.27.1 Read concurrency

Oracle uses a feature called *read consistency,* which lets a query return a result based on the state of the data when the query starts, regardless of other processes such as update or delete on the same data while the query is running. This mechanism is called Multi-Version Read Consistency and is implemented by *undo data* in the undo segments.

DB2 does not support this type of processing. The application should be analyzed to determine if this difference is acceptable. If not, then it must be determined what the requirement is: readers not blocking writers, query results based on the state of the data when the query began (read consistency), or both.

DB2 implements various isolation levels to support read concurrency.

Uncommitted Read

Uncommitted Read (UR), also known as dirty read, is the lowest level of isolation. It is the least restrictive, but provides the greatest level of concurrency. However, it is possible for a query executed under UR to return data that has never been committed to the database.

Cursor Stability

Cursor Stability (CS) is the default isolation mode. It is used when no isolation is set in an application. In this isolation mode, only the row on which the cursor is currently positioned is locked. This lock is held until a new row is fetched or the unit of work (UOW) is terminated. If a row is updated, the lock is held for the duration of the transaction.

Read Stability

Under Read Stability (RS) isolation, locks are only placed on the rows that an application retrieves within a unit of work. Applications cannot read uncommitted data and no other application can change the rows locked by the Read Stability

application. It is possible to retrieve phantom rows if the application retrieves the same row more than once within the same unit of work.

The Read Stability isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking are used. One of the objectives of the Read Stability isolation level is to provide both a high degree of concurrency as well as a stable view of the data. To assist in achieving this objective, the optimizer ensures that table level locks are not obtained until lock escalation occurs.

Repeatable Read

Repeatable Read (RR) is the highest level of isolation and has the lowest level of concurrency. Locks are held on all rows processed (scanned) for the duration of a transaction. Because so many locks are required for repeatable read, the optimizer might choose to lock the entire table instead of locking individual rows. The same query issued by the application more than once in a unit of work gives the same result each time (no phantom reads). No other application can update, delete, or insert a row that affects the resulting table until the unit of work is completed.

Table 5-10   DB2 isolation level summary

Isolation level	Read a data locked by other users	Lock placed on data rows
Uncommitted Read	yes	No
Cursor Stability	no	Only a row currently fetched by cursor, until fetching the next row.
Read Stability	no	Rows retrieved within a UOW.
Repeatable Read	no	All rows processed (scanned) within a transaction.

In DB2, the isolation level can be specified for a session, a client connection, or an application before a database connection and is set to control the type of locks and the degree of concurrency allowed by the application. For embedded SQL, the level is set at bind time; and for dynamic SQL, the level is set at run time.

The isolation level is set by the SET ISOLATION statement, as shown in Example 5-75.

Example 5-75   Setting the isolation level

```
SET ISOLATION TO ur ;
```

```
SET ISOLATION TO rs ;
```

Or you can set the isolation level in a SELECT statement, as shown in Example 5-76.

Example 5-76   Set the isolation level in a SELECT statement

```
SELECT deptno, deptname, mgrno
FROM dept
WHERE admrdept ='a00'
WITH UR;
```

Read scanners can skip noncommitted INSERTs or DELETEs

To improve concurrency, DB2 permits the deferral of row locks for Cursor Stability or Read Stability isolation scans in certain situations, until a record is known to satisfy the predicates of a query. By default, when row locking is performed during a table or an index scan, DB2 locks each row before determining whether the row qualifies for the query. To improve concurrency of scans, it may be possible to defer row locking until after determining that a row qualifies for a query. This lock deferral feature was introduced using the registry variable DB2_EVALUNCOMMITTED.

In addition, you might improve concurrency by setting the registry variables DB2_SKIPDELETED and DB2_SKIPINSERTED. These registry variables permit scans to unconditionally skip uncommitted deletes and inserts, respectively.

The default value for all these registry variables is OFF.

DB2_SKIPINSERTED=ON

DB2 will treat the uncommitted INSERTs (for CS and RS isolation levels only) as though they had not yet been inserted. This feature provides increased concurrency without sacrificing isolation semantics. DB2 implements the ability for scanners to skip uncommitted inserted rows versus waiting, when in conflict through lock attributes and feedback on lock requests.

DB2_SKIPDELETED=ON

This acts in the same manner for deletions as DB2_SKIPINSERTED does for insertions. When set to ON, it allows uncommitted deletions to be ignored for cursors using the CS or RS isolation levels. DB2 will unconditionally bypass any uncommitted rows in a query.

DB2_EVALUNCOMMITTED=ON

DB2 allows scans with isolation level Cursor Stability (CS) or Read Stability (RS) to avoid or defer row locking until a data row is known to satisfy predicate

evaluation. Rows that do not satisfy your query are bypassed. With this variable enabled, predicate evaluation can occur on uncommitted data.

For more information, refer to the white paper "Lock avoidance in DB2 UDB V8" at:

`http://www.ibm.com/developerworks/db2/library/techarticle/dm-0509schuetz/`

Recommendation for the appropriate isolation level

Choosing the appropriate isolation level to use for a transaction is very important. The isolation level not only influences how well the database supports concurrency, but it also affects the overall performance of the application containing the transaction. That is because the resources needed to acquire and free locks vary with each isolation level.

Generally, when more restrictive isolation levels are used, less concurrency support is provided and the overall performance may be decreased because more resources are required. However, when deciding on the best isolation level to use, the deciding factor should be which phenomena are acceptable and which phenomena are not. The following heuristic can be used to help you decide which isolation level to use for a particular situation:

- ► If you are executing queries on read-only databases or if you are executing queries and do not care if uncommitted data values are returned, use the Uncommitted Read isolation level. (The Read-only transactions are needed. High data stability is not required.)

- ► If you want maximum concurrency without seeing uncommitted data values, use the Cursor Stability isolation level. (The Read/Write transactions are needed. High data stability is not required.)

- ► If you want concurrency and you want qualified rows to remain stable for the duration of an individual transaction, use the Read Stability isolation level.

- ► If your application uses a cursor which selects a few data rows and fetches to perform some additional processing, you need to set the isolation level as Read Stability or Repeatable Read to avoid a phantom select.

- ► Even though a cursor in your application has only one row, if your isolation level is set to Repeatable Read, make sure that your query uses a proper index to fetch a row to avoid locking all rows in the table caused by the table scan.

5.27.2 Update concurrency

When an update such as an INSERT, DELETE, or UPDATE statement occurs in the database, a lock is used to support the update concurrency. In Oracle, locks placed on the table elements can be on individual rows, or on pages of rows in

the table. In DB2, the database manager imposes locks on objects such as rows, tables, and table spaces as they are required. The default used by DB2 is *row locking*.

In DB2, databases, table spaces, and tables can be explicitly locked. Here are some examples of commands that can be used to lock the database objects:

► Database lock

```
CONNECT TO database IN EXCLUSIVE MODE
```

► Table space lock

```
QUIESCE tablespace FOR TABLE tablename INTENT FOR UPDATE;
```

► Table lock

```
LOCK TABLE tablename IN EXCLUSIVE MODE;
```

In DB2, databases, tables, and rows can be implicitly locked. For example:

► Databases are locked during full database restore.
► Tables are locked during lock escalation.
► Rows are locked through normal data modification.

SELECT FOR UPDATE

If you convert the Oracle UPDATE cursor or SELECT FOR UPDATE statement to DB2, you can use a DB2 cursor.

Example 5-77 shows the Oracle SELECT FOR UPDATE code sample.

Example 5-77   Oracle - SELECT FOR UPDATE

```
EXEC SQL select * from staff where id=p_id for update;
...
EXEC SQL update staff set salary = 10000.0
     where id =p_id ;
```

Example 5-78 shows how this can be converted to DB2.

Example 5-78   DB2 Update cursor sample in ESQL/C program

```
EXEC SQL  DECLARE C1 CURSOR FOR
    select * staff where id=:p_id for update of salary;
EXEC SQL  OPEN C1;
EXEC SQL  FETCH C1 INTO ...    ;
   if ( strcmp (change, "YES") == 0 )
     EXEC SQL  update staff set salary = :salary
          WHERE CURRENT OF C1;
```

```
EXEC SQL  CLOSE C1;
```

If SELECT FOR UPDATE is used for holding the update lock on a specific row
but not to update the data on Oracle, you can use SELECT with the USE AND
KEEP UPDATE [SHARE|EXCLUSIVE] LOCKS clause in DB2 to achieve the
purpose, as shown in Example 5-79:

Example 5-79   Using the DB2 USE AND KEEP UPDATE clause

```
EXEC SQL select * from staff where id=:p_id
   FOR READ ONLY WITH RS USE AND KEEP EXCLUSIVE LOCK;
```

In Example 5-79, SELECT retrieves data from the STAFF table with exclusive
lock on the data. Because the data will be updated later with a searched update,
and should be locked when the query is executed.

5.27.3 Miscellaneous differences

Unlike in Oracle, in DB2 locks are stored in the memory and not in data pages.
The LOCKLIST database configuration parameter can be used to configure the
memory available for locks, while the MAXLOCKS configuration parameter
defines the maximum amount of memory for a particular application's locks.

Also In DB2 9, LOCKLIST can be set to AUTOMATIC to enable self tuning. This
allows the memory tuner to dynamically size the memory area controlled by this
parameter as the workload requirements change. The value of LOCKLIST is
tuned together with the MAXLOCKS parameter. Therefore, disabling self tuning
of the LOCKLIST parameter also disables self tuning of the MAXLOCKS.
Enabling self tuning of the LOCKLIST parameter automatically enables self
tuning of the MAXLOCKS.

5.27.4 Transaction

Most transaction statements—COMMIT, ROLLBACK, and SAVEPOINT—in
Oracle are converted to DB2 by the MTK.

Different COMMIT behavior

The Oracle COMMIT statement does not close currently open cursors. The DB2
COMMIT statement does close currently open cursors (except those declared
using WITH HOLD). Manual conversion might be required if open cursors are
used after a COMMIT in the Oracle source code. COMMIT in both Oracle and
DB2 releases the locks held by the cursors, except in DB2 for WITH HOLD
cursors.

Multiple SAVEPOINT

You can convert an Oracle multiple SAVEPOINT into DB2, as shown in Example 5-80.

Example 5-80   DB2 multiple SAVEPOINT

```
CREATE TABLE DEPT_SAVEPOINT (
    DEPTNO   CHAR(6),
    DEPTNAME VARCHAR(20),
    MGRNO    INTEGER);

INSERT INTO DEPARTMENT VALUES ('A20', 'MARKETING', 301);

SAVEPOINT SAVEPOINT1 ON ROLLBACK RETAIN CURSORS;

INSERT INTO DEPARTMENT VALUES ('B30', 'FINANCE', 520);

SAVEPOINT SAVEPOINT2 ON ROLLBACK RETAIN CURSORS;

INSERT INTO DEPARTMENT VALUES ('C40', 'IT SUPPORT', 430);

SAVEPOINT SAVEPOINT3 ON ROLLBACK RETAIN CURSORS;

INSERT INTO DEPARTMENT VALUES ('R50', 'RESEARCH', 150);
```

At this point, the DEPT_SAVEPOINT table exists with rows A20, B30, C40, and R50.

If you now issue the ROLLBACK TO SAVEPOINT SAVEPOINT3, row R50 is no longer in the DEPARTMENT table.

If you then issue ROLLBACK TO SAVEPOINT SAVEPOINT1, the DEPARTMENT table still exists, but the rows inserted since SAVEPOINT1 was established (B30 and C40) are no longer in the table.

For more information about Concurrency and Transaction, refer to "Managing concurrency" in the DB2 manual *Performance Guide,* SC10-4222.

5.28 Encryption

In DB2, the encryption and decryption module are bundled, so you can implement an encrypted database using encryption and decryption built-in functions. You can encrypt a column with an encryption key (password). The

data can then be decrypted using the password. You also can include a hint for the key. A function is provided to get the password hint.

Only a data of CHAR and VARCHAR data type can be encrypted. The column data type to store an encrypted value is VARCHAR FOR BIT DATA. To encrypt a column, you need to declare a column with sufficient size to contain the encrypted value.

► When the hint parameter is specified, the length attribute of the result is equal to the total of the following:

  – Length attribute of the unencrypted data
  – 8 bytes
  – Number of bytes to the next 8-byte boundary
  – 32 bytes for the length of the hint

► When the hint parameter is not specified, the length attribute of the result is equal to the total of the following:

  – Length attribute of the unencrypted data
  – 8 bytes
  – Number of bytes to the next 8-byte boundary

You can use the encrypt function to find the length of the result before you create a table, as shown in the Example 5-81:

Example 5-81

```
values length(encrypt('IBM Redbook','los gatos'))
1
-----------
         24

  1 record(s) selected.

values length(encrypt('IBM Redbook','los gatos','a city of the cat'))
1
-----------
         41

  1 record(s) selected.
```

Example 5-82 illustrates how to store encrypted data; an error in the password is also included.

Example 5-82   Creating data encryption

```
CREATE TABLE encryptions
```

```
    (
    id INT NOT NULL,
    data CHAR(30) FOR BIT DATA
    );

-- Insert few rows with encryption function
-- These inserts will success.

INSERT INTO encryptions VALUES
    (1,encrypt('100-1111','WheiZen')),
    (2,encrypt('200-2222','Carlos')),
    (3,encrypt('300-3333','Fraser','CA'));

-- This insert does not have a valid password

INSERT INTO encryptions VALUES
    (4, ENCRYPT('400-4444','Anna'));

DB21034E  The command was processed as an SQL statement because it was
not a valid Command Line Processor command.  During SQL processing it
returned:
SQL20144N  The encryption password is invalid because the length of the
specified password was less than 6 bytes or greater than 127 bytes.
SQLSTATE=428FC
```

Example 5-83 shows how to use the decryption function to retrieve data.

Example 5-83   Retrieve encrypted data

```
-- Select a encrypted value and actual value with decryption function
--
SELECT data FROM encryptions
WHERE id=1;

DATA
----------------------------------------------------------------
x'0C09F7FF0333D5A350B39B651F58A96E248F7E53E44708B9202020202020'

1 record(s) selected.

-- select a data with a key
SELECT id,DECRYPT_CHAR(data,'WheiZen') FROM encryptions
    where id=1

ID          2
```

```
          ---------- ---------------------
                   1 100-1111

  1 record(s) selected.
--
-- select data with wrong key
--
select id, DECRYPT_CHAR(data,'AnnaChoi')
  FROM encryptions
  WHERE id=2;
SQL20145N  The decryption function failed. The password used for
decryption does not match the password used to encrypt the data.
SQLSTATE=428FD

-- select a hint for decryption
--
SELECT GETHINT(data) From encryptions
  where id=3;
1
--------------------------------
CA

  1 record(s) selected.
```

You can also set an encryption password in the session level. See Example 5-84. It will affect every insert if no encryption key is specified.

Example 5-84   Set encryption password in the session level

```
set encryption password = 'The Pride';
insert into encryptions values
  (20,encrypt('500-5555')),
  (30,encrypt('600-6666'));
```

5.29 Oracle multitable, conditional, and pivot insert

Oracle multitable, conditional, and pivot insert are not converted to DB2 statements by the MTK. To convert, you need to separate an Oracle multitable, conditional, or pivot insert SQL statement into appropriate SQL statements.

Example 5-85 shows an Oracle multitable insert SQL statement.

Example 5-85   Oracle multitable insert

```
INSERT  ALL
INTO emp_dept VALUES(empno,firstnme,workdept)
INTO emp_salary VALUES(empno,firstname,salary,comm)
SELECT empno,firstnme,workdept,salary,comm
FROM  employee
WHERE edlevel < 16;
```

Example 5-86 shows the DB2 conversion code.

Example 5-86   Conversion of Oracle multitable insert to DB2

```
INSERT INTO emp_dept
SELECT empno,firstnme,workdept
FROM  employee
WHERE edlevel < 16;

INSERT INTO emp_salary
SELECT empno,firstnme,salary,comm
FROM  employee
WHERE edlevel < 16;
```

5.30 Additional considerations

External procedures and functions are also frequently seen in Oracle and DB2
applications. Complete information about building and running external
procedures and functions is beyond the scope of this book. We provide here two
examples of building routines using C and Java. For complete information about
building and running external procedures and functions, consult the following
IBM DB2 9 documents:

► *Getting Started with Database Application Development*, SC10-4252
► *Developing SQL and External Routines*, SC10-4373

5.30.1 Building C/C++ routines

We now show an example of creating a stored procedure written in C. First, here
are some basic steps that must be followed when creating any C procedure (or
function):

- Create the external procedure/function and save it on your file system. If the procedure/function contains *embedded* SQL, then it should be saved with the extension .sqc, if not save it with the extension .c

- Create the export file (AIX ONLY)

  AIX requires you to provide an *export file* that specifies which global functions in the library are callable from outside it. This file must include the names of all routines in the library. Other UNIX platforms simply export all global functions in the library. This is an example of an AIX export file for the procedure *outlanguage* that exists in the file spserver.sqc:

  ```
  #! spserver export file
  outlanguage
  ```

  The export file `spserver.exp` lists the stored procedure `outlanguage`. The linker uses `spserver.exp` to create the shared library `spserver` that contains the `outlanguage` stored procedure.

- On Windows, a DEF file is required which has a similar purpose. See *sqllib/samples/c/spserver.def* for an example.

- Run the **bldrtn** script which creates the shared library:

  ```
  bldrtn my_routine my_database_name
  ```

  The script copies the shared library to the server in the path sqllib/function.

- Catalog the routines by running the *catalog_my_routine* script on the server.

Notes

- After a shared library is built, it is typically copied into a directory from which DB2 will access it. When attempting to replace a routine shared library, you should either run /usr/sbin/slibclean to flush the AIX shared library cache, or remove the library from the target directory, and then copy the library from the source directory to the target directory. Otherwise, the copy operation may fail because AIX keeps a cache of referenced libraries and does not allow the library to be overwritten.

- DB2 provides build scripts for precompiling, compiling, and linking C stored procedures. These are located in the sqllib/samples/c directory, along with sample programs that can be built with these files. This directory also contains the **embprep** script used within the build script to precompile a *.sqc file. The build scripts have the .bat (batch) extension on Windows, and have no extension on UNIX platforms. For example, **bldrtn.bat** is a script to build C/C++ stored procedure on Windows platform; **bldrtn** is the equivalent on UNIX.

Example

Here is a simple example of creating and cataloging a stored procedure written in C. This procedure queries the sysprocedures table from the DB2 system catalog to determine in which language (JAVA, C, etc.) that a procedure TWO_RESULT_SETS is written.

► The C source file (Example 5-87), with embedded SQL, is created and saved as outlanguage.sqc.

Example 5-87   Stored procedure in C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>
SQL_API_RC SQL_API_FN outlanguage(char language[9]){
  struct sqlca sqlca;
  EXEC SQL BEGIN DECLARE SECTION;
    char out_lang[9];
  EXEC SQL END DECLARE SECTION;
  /* Initialize strings used for output parameters to NULL */
  memset(language, '\0', 9);
  EXEC SQL SELECT language INTO :out_lang
    FROM sysibm.sysprocedures
    WHERE procname = 'TWO_RESULT_SETS';
  strcpy(language, out_lang);
  return 0;
} /* outlanguage function */
```

► The .exp file is created and saved as outlanguage.exp. Here are the contents of that file:

```
outlanguage
```

► The file outlanguage_crt.db2, which catalogs the procedure, is created and saved. Here are the contents:

```
CREATE PROCEDURE outlanguage (OUT language CHAR(8))
DYNAMIC RESULT SETS 0
LANGUAGE C
PARAMETER STYLE SQL
NO DBINFO
FENCED NOT THREADSAFE
MODIFIES SQL DATA
```

```
PROGRAM TYPE SUB
EXTERNAL NAME 'outlanguage!outlanguage'!
```

▶ The build script **bldrtn** for the outlanguage.sqc file is executed using the db2_emp database:

bldrtn outlanguage db2_emp

▶ A connection is made to the database:

db2 connect to db2_emp

▶ The script to catalog the procedure is executed:

db2 –td! –vf outlanguage_crt.db2 **>** message.out

▶ DB2 responds with the message:

DB20000I The SQL command completed successfully.

The message.out file should be viewed for messages, especially if any other message than The SQL command completed successfully is returned.

▶ The procedure is tested:

db2 "call outlanguage(?)**"**

Results:

```
Value of output parameters
--------------------------
Parameter Name  : LANGUAGE
Parameter Value : JAVA
Return Status = 0
```

5.30.2 Building Java routines

Here are the basic steps to create an external Java User-Defined Function (UDF) from the DB2 Command Window:

▶ Compile your_javaFileName.java to produce the file your_javaFileName.class with this command:

javac your_javaFileName.java

▶ Copy your_javaFileName.class to the sqllib\function directory on Windows operating systems, or to the sqllib/FUNCTION directory on UNIX.

▶ Connect to the database:

db2 connect to your_database_name

▶ Register the your_javaFileName library in the database using the CREATE FUNCTION SQL statement.

db2 –td! –vf <your_create_function_statement.db2>

Example

Here is an example of a Java UDF that will retrieve the system name from the DB2 Registry variable DB2SYSTEM.

► The Java source file as shown in Example 5-88 is saved as db2system_nameUDF.java.

Example 5-88 UDF Java source

```
import java.io.*;
public class db2system_nameUDF {
  public static String db2system_name() {
    Runtime rt = Runtime.getRuntime();
    Process p=null;
    String s = null;
    String returnString = "";
    try {
       // WINDOWS: **** uncomment and compile the following for Windows
       // p = rt.exec("cmd /C db2set DB2SYSTEM");
       // UNIX: **** uncomment and compile the following for UNIX
       p = rt.exec("db2set DB2SYSTEM");
       BufferedInputStream buffer =
          new BufferedInputStream(p.getInputStream());
       BufferedReader commandResult =
          new BufferedReader(new InputStreamReader(buffer));
       try {
          while ((s = commandResult.readLine()) != null)
              returnString = returnString.trim() + s.trim() ;
          // MAX number of chars for the DB2SYSTEM variable is 209
characters
          commandResult.close();
          // Ignore read errors; they mean process is done
       } catch (Exception e) {
       }
    } catch (IOException e) {
        returnString = "failure!";
    }
    return(returnString);
  }
}
```

► Compile the Java source. The compile command is:

javac db2system_nameUDF.java

► Copy the .class file to the /sqllib/function directory:

$ **cp** *db2system_nameUDF.java /home/db2inst1/sqllib/function*

► Construct the Create Function file and save it as db2system_name.db2:

```
DROP FUNCTION DB2SYSTEM_NAME !
CREATE FUNCTION DB2SYSTEM_NAME ()
RETURNS VARCHAR(209)
EXTERNAL NAME 'db2system_nameUDF!db2system_name'
LANGUAGE JAVA
PARAMETER STYLE JAVA
NOT DETERMINISTIC
NO SQL
EXTERNAL ACTION!
```

► Connect to the database:

db2 connect to db2_emp

► Execute the script to register the UDF with the database:

db -td! -vf db2system_name.db2

► Test the UDF:

db2 "values db2system_name()"

Result:

```
-----------------------------------------------
smpoaix
```

1 record(s) selected.

6

Data conversion

Data conversion is a sensitive task in the porting project. You have to ensure that all data is moved to the target database, both correctly and in time.

In this chapter, we discuss the data conversion methods for deploying the data from Oracle to the DB2 database. The data can be transformed by the following methods:

► Using the IBM Migration Toolkit (MTK) generated scripts and data files

► Using the MTK to move data online

► Exporting the data manually from Oracle to flat file and importing or loading to DB2

► Using operating system named pipes

► Using WebSphere Federation Server

Furthermore, we give some hints for time planning of the data movement process.

6.1 Data conversion process

The data conversion process is quite complex. Before you define a porting method, you should do some tests with only a portion of the data to verify that the chosen method works successfully for your database environment. Generally, it is a good idea that tests cover all potential cases. For these reasons, we recommend that you start early with the testing.

The tasks of the test phase are:

► Calculate the source data size and calculate the required space for the files on disk.

► Select the tools and the conversion method.

► Test the conversion using the chosen method with only a small amount of data.

With the result of the test, you should be able to:

► Estimate the time for the complete data conversion process.

► Create a plan for the development environment conversion.

► Create a plan for the production environment conversion, using the information from the development environment conversion.

► Schedule the time for the data conversion.

The following factors influence the time and complexity of the process:

► Amount of data and data changes

   The more data that you have to move, the more time you need. Consider the data changes as well as the timestamp conversions.

► System availability

   You can run the data movement either when the production system is down or when the business process is running, by synchronizing source and target database. Depending on the strategy you choose, you will need less or more time.

► Hardware resources

   Be aware that you need up to three times the disk space during the data movement for:

   – The source data in Oracle
   – The unloaded data stored in the file system
   – The loaded data in the target DB2

6.2 Time planning

After testing the data movement and choosing the proper tool and strategy, you should create a detailed time plan. This time plan should include the following tasks:

- ▶ Depending on the data movement method:
    - – Implementing or modifying scripts for data unload and load
    - – Learning how to use the chosen data movement tools
- ▶ Data unload from Oracle
- ▶ Data load to DB2
- ▶ Backup of the target DB2 database
- ▶ Testing the loaded data in DB2 for completeness and consistency
- ▶ Switching of the applications, including database interfaces
- ▶ Fallback process in case of incidents

The most sensitive environment is a production system with a 7x24 hour availability requirement. Figure 6-1 shows the way to move data to the target database in a high availability (HA) environment. The dark color represents the new data, the light color represents the converted and moved data. If possible, export the data from a standby database or mirror database to minimize the impact on the production environment. The following tasks should be performed to move data from an HA environment:

1. Create scripts that export all data up to a defined timestamp.

2. Create scripts that export changed data since the last export. This includes new data as well as deleted data.

3. Repeat Step 2 as often as necessary, until all data is moved to the target DB2 database.

4. Define a fallback strategy and prepare fallback scripts.

Figure 6-1   Data movement strategy in a high availability environment

When the data is completely moved to the target DB2 database, you can switch the application and database. Prepare a well-defined rollout process for the applications and the interfaces belonging to DB2. It is always a good idea to allow some buffer time for any unplanned incidents.

6.3 Data movement through flat files

The main principle of data movement is to export the Oracle data to flat files in a well-defined format, and then to LOAD or IMPORT the data to DB2. We can use any tool or write any application to achieve this. This section discusses the data movement through flat files.

The script examples included in this section can be downloaded from the IBM Redbook Web site. See Appendix G, "Additional material" on page 701 for details.

Before writing data into a flat file, ensure that the maximum file size of your operating system is big enough. On AIX, you can get the actual file size limit in blocks with the following command:

```
ulimit -a
```

To set the file size limit to *unlimited* on AIX, enter the following command as the root user:

```
ulimit -f -1
```

6.3.1 Moving data using the MTK

In 4.7, "The Generate Data Transfer Scripts task" on page 122, we explain how to use the MTK to generate scripts for data unload and data load; the correlation of scripts and table definitions of the source and target are defined in the MTK. The MTK also allows you to move (deploy) data through its GUI, online, without the need for the generated scripts.

6.3.2 Using shell scripts

We use UNIX shell scripts in our example, which make use of the Oracle SQL*Plus utility to extract the data from Oracle tables to flat files. These scripts can only be run on UNIX platforms, and should be run under a user that has the Oracle environment set. Additionally, these scripts will only work against Oracle tables containing CHAR, VARCHAR2, and NUMBER data types. For exporting LOB columns, special programs must be written using Oracle APIs. Example 6-1 shows the main data_unload.sh script. This script reads the table name from the table_list_file parameter specified when invoking the script, constructs the dynamic query, and estimates the line size using two awk scripts. Once the query is constructed, it is fed to the SQL*Plus utility, the output is spooled and stored in the output file table_name.DAT as delimited ASCII (DEL).

Example 6-1   The data_unload.sh script

```
#!/bin/ksh
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# Shell script:  data_unload.sh
#
# Syntax:        data_unload.sh  <table_list_file>
#
# Starting from an flat file containing a list of all the table,
# extracts data from Oracle for each table and writes data into
# a file named table_name.DAT, formatted in columns
#
# This script uses the awk command with the following awk command files:
# desc.awk   formats the query command files using RTRIM and DECODE
#            to obtain a column-formatted output
# count.awk  computes the total length of a record
#
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

#
```

```
# Define the environment variables for the oracle user and password
ORACLE_USR=user1
ORACLE_PWD=user1
#
# Start of main program

# Loop on all the tables listed in the input file
for i in `cat $1`
do
  # Define some environment variables for temporary files
  export OUTFILE=$i.DAT
  DSCFILE=$i.dsc
  SQLFILE=$i.sql
  VARFILE=$i.var
  ALLFILE=$i.all
  POSFILE=$i.pos
  rm -f $OUTFILE
  rm -f $DSCFILE
  rm -f $SQLFILE

  # Extract the table description from Oracle catalog
  sqlplus -s $ORACLE_USR/$ORACLE_PWD <<EOF >/dev/null 2>&1
  clear columns
  clear breaks
  set pagesize 100
  set newpage 1
  set feedback off
  spool $DSCFILE
  desc $i
  EOF

  # Cut head and tail from the file containing the descriptions of the tables
  # Change also the NOT NULL clause in a blank string
  # and cut the blanks in the first column
  tail +3 $DSCFILE | sed 's/NOT NULL/ /; s/^ //' > $DSCFILE.tmp1
  NL=`wc -l < $DSCFILE.tmp1`
  NLM1=`expr $NL - 1`
  head -$NLM1 $DSCFILE.tmp1 > $DSCFILE.tmp2
  cp $DSCFILE.tmp2 $VARFILE

  # Change the data types, leaving in the file the respective lengths
  # It is assumed that 41 bytes are enough to contain the significative
  # part of the NUMBER fields
  sed -e 's/ VARCHAR2(/ /' \
  -e 's/ NUMBER(/ /' \
  -e 's/ NUMBER/ 41/' \
  -e 's/ INTEGER(/ /' \
  -e 's/ INTEGER/ 41/' \
  -e 's/ CHAR(/ /' \
```

```
      -e 's/ CHAR/ 1/' \
      -e 's/ RAW(/ /' \
      -e 's/ VARCHAR(/ /' \
      -e 's/)//' \
      -e 's/\([0-9]*\)\,\([0-9 ]\)*/\1/' \
      $DSCFILE.tmp2 > $DSCFILE.tmp3
      mv $DSCFILE.tmp3 $DSCFILE
      rm -f $DSCFILE.tmp*

      # Compute the record length of the table
      # using the count.awk awk script
      LS=`awk -f count.awk $DSCFILE`

      # Prepare the heading of the query statement on the table
      # by echoing the statements into the sql file
      echo "clear columns" > $SQLFILE
      echo "clear breaks" >> $SQLFILE
      echo "set pagesize 50000" >> $SQLFILE
      echo "set linesize $LS" >> $SQLFILE
      echo "set feedback off" >> $SQLFILE
      echo "set heading off" >> $SQLFILE
      echo "set space 0" >> $SQLFILE
      echo "set newpage 1" >> $SQLFILE
      echo "spool $OUTFILE" >> $SQLFILE
      echo "select '' " >> $SQLFILE

      # Append to the query statement file the list of the table fields
      # to obtain the column layout, using the desc.awk awk script
      awk -f desc.awk $VARFILE >> $SQLFILE

      # Append to the query statement file the "from" clause
      # and the closing instructions
      echo "from $i;" >> $SQLFILE
      echo "spool off" >> $SQLFILE
      echo "quit" >> $SQLFILE

      # Execute the query statement
      sqlplus -s $ORACLE_USR/$ORACLE_PWD @$SQLFILE >/dev/null 2>&1

      # Cut the first line from the output file
      tail +2 $OUTFILE > $OUTFILE.tmp
      mv $OUTFILE.tmp $OUTFILE

      # Change the DATE data type into its DB2 external length, 26 bytes
      sed 's/ DATE/ 26/' $DSCFILE > $DSCFILE.tmp1
      mv $DSCFILE.tmp1 $DSCFILE
done
```

The table name is read from table_list_file and described using the DESCRIBE TABLE command, and output is directed to a describe file TABLE_NAME.dsc. From the describe file, the SELECT query is constructed using the awk script file `desc.awk`, which produces the TABLE_NAME.sql file. Example 6-2 shows the `desc.awk` script. The TABLE_NAME.sql file contains the SELECT statement which, when executed, produces the result that can be exported to a delimited ASCII file type, which can be used for the LOAD or IMPORT utilities. The character data types are enclosed by the ~ character and the DATE data types are converted to the equivalent DB2 TIMESTAMP data type. The query uses the concatenation string || to concatenate the column values to a single line.

Example 6-2   The desc.awk script

```
BEGIN {}
{
  if ($2 == "DATE")
    print " || rtrim(DECODE("$1",NULL,' ',TO_CHAR("$1",'YYYY-MM-DD-HH24.MI.SS')
|| '.000000'),26)"
  if (substr($2,1,4) == "CHAR")
    print " ||'~'||rtrim("$1")||'~,' "
  if (substr($2,1,8) == "VARCHAR2")
    print " ||'~'||rtrim("$1")||'~,' "
  if (substr($2,1,6) == "NUMBER")
    print " ||rtrim("$1")||','"
}
```

The data_unload.sh script also uses another awk script, count.awk, to count the length of each column to estimate the output line size for the SQL*Plus utility. Example 6-3 shows this script. Once the SELECT statement and the line size is ready, the SQL*Plus environment is set using the SET PAGESIZE, and SET LINESIZE commands. The commands SET PAGESIZE, SET LINESIZE, and SET FEEDBACK manipulate the output. The SET LINESIZE is set with the output produced by the count.awk script. Then the SQL*Plus runs the TABLE_NAME.sql file and spools the output to the TABLE_NAME.dat output flat file. This file is in delimited ASCII (DEL) format and can be used by either the LOAD or IMPORT utility. For example, to export the ACCOUNTS table, enter the table name into a file, such as table.lst, edit the data_unload.sh script, and enter the Oracle user name ORACLE_USR and password ORACLE_PWD, and then invoke using the command:

```
sh data_unload.sh table.lst
```

This produces the accounts.dsc, accounts.sql, accounts.var, and accounts.DAT files. To load the data using LOAD utility, use the command:

```
DB2 LOAD FROM accounts.DAT OF DEL MODIFIED BY CHARDEL~ INSERT INTO
ACCOUNTS
```

To load the data using the IMPORT utility, use the command:

```
DB2 IMPORT FROM accounts.DAT OF DEL MODIFIED BY CHARDEL~ INSERT
INTO ACCOUNTS
```

Example 6-3   The count.awk script

```
BEGIN { total=0 }
{
  if ($2 == "DATE")
    total +=26
  else
    total += $2+4
}
END { print total }
```

6.3.3 Using Oracle's stored procedures

In this example, we explain an Oracle stored procedure, export_table, written by the authors of this book, to demonstrate how to unload the data from Oracle using this stored procedure, and to load the data into DB2. As in our previous example, this stored procedure can only be used for CHAR, VARCHAR2, NUMBER, and DATE data types. As in the shell script, this stored procedure gets the table name as the input parameter, constructs the SELECT query for output, and exports the table data to an output flat file. This output file format is also delimited in an ASCII file format.

The advantage of using this stored procedure is that it can be used on both Windows and UNIX platforms. The output file is placed under the directory specified by the UTL_FILE_DIR initialization parameter, which specifies the directory for PL/SQL file I/O. So, it is a must that this initialization parameter be specified in the Oracle instance before using this stored procedure. Example 6-4 shows the export_table stored procedure script.

Example 6-4   Procedure to export_ table data

```
/***********************************************************/
/* This stored procedure accept the table name as input   */
/* and exports the data into flat file identified by the   */
/* UTL_FILE_DIR with the format acceptable by the DB2      */
/* IMPORT utility or LOAD utility as Delimited ASCII file  */
/* Note : this procedure can be used for the table with data */
/* types CHAR,VARCHAR2 and NUMBER.                         */
/***********************************************************/
CREATE OR REPLACE PROCEDURE export_table(
   i_table_name IN VARCHAR2                 -- table name to be exported
)
IS
```

```
            stmt_1      VARCHAR2(4000) := 'select ';          -- first part of select
            stmt_2      VARCHAR(50) := ' as linecol from '; -- second part of the select
            stmt_cursor INTEGER;                             -- statement handle
            linecol     VARCHAR2(4000);                      -- output buffer for utl_file
            ret         INTEGER;                             -- dbms_sql handle
            filepath    VARCHAR(40):='c:\oracle';            -- path for output file
            filename    VARCHAR(40);                         -- output filename
            filemode    CHAR(1):='w';                        -- output file mode for write
            filelnsz    INTEGER := 4000;                     -- max file line size
            dtype_excp  EXCEPTION;
            fhandle     utl_file.file_type;                  -- file handle for utl_file
            CURSOR col_crsr(tab_col_name IN VARCHAR2) IS
               SELECT column_name, data_type
               FROM user_tab_columns
               WHERE table_name = tab_col_name;

        BEGIN
            stmt_1 := stmt_1||'/*parallel('||i_table_name||',4)*/'||''''||'''';

            /**********************************************************/
            /* Build the select statement                           */
            /**********************************************************/
            FOR my_rec IN col_crsr(i_table_name) LOOP
              IF my_rec.data_type = 'DATE' THEN
                stmt_1 := stmt_1 || '|| rtrim(DECODE('|| my_rec.column_name
                                  || ',NULL,'|| ''' '''|| ',TO_CHAR('
                                  || my_rec.column_name ||','''
                                  || 'YYYY-MM-DD-HH24.MI.SS'||''')))';
              ELSIF my_rec.data_type = 'CHAR' THEN
                stmt_1 := stmt_1 || '||'''||'~'||''''||'||rtrim('
                                  || my_rec.column_name||')||'''||'~,'||'''';
              ELSIF my_rec.data_type = 'VARCHAR2' THEN
                stmt_1 := stmt_1 || '||'''||'~'||''''||'||rtrim('
                                  || my_rec.column_name||')||'''||'~,'||'''';
              ELSIF my_rec.data_type = 'NUMBER' THEN
                stmt_1 := stmt_1 || '||rtrim('||my_rec.column_name
                                  || ')||'''||','||'''';
              ELSE RAISE dtype_excp;
              END IF;
            END LOOP;
            stmt_2 := stmt_2 || i_table_name;
            stmt_1 := stmt_1 || stmt_2;

            /**********************************************************/
            /* Execute the statement and open the cursor              */
            /**********************************************************/
            stmt_cursor := dbms_sql.open_cursor;
            dbms_sql.parse(stmt_cursor,stmt_1,dbms_sql.native);
            dbms_sql.define_column(stmt_cursor,1,linecol,4000);
```

```
        ret := dbms_sql.execute(stmt_cursor);
        filename:=i_table_name||'.DAT';
        fhandle:= utl_file.fopen(filepath,filename,filemode,filelnsz);

        /*******************************************************/
        /* Fetch the rows and write it to output file          */
        /*******************************************************/
        WHILE dbms_sql.fetch_rows(stmt_cursor)>0 LOOP
          dbms_sql.column_value(stmt_cursor,1,linecol);
          utl_file.put_line(fhandle,linecol);
        END LOOP;

        /*******************************************************/
        /* Close the cursor and file                           */
        /*******************************************************/
        dbms_sql.close_cursor(stmt_cursor);
        utl_file.fclose(fhandle);

        EXCEPTION
        WHEN dtype_excp THEN dbms_output.put_line('Invalid Data type');
END;
```

This stored procedure uses the Oracle DBMS_SQL package to construct the
SELECT statement and retrieve the result set. It uses the UTL_FILE Oracle
package to create the output file, open it, and write the output data to the output
file. The UTL_FILE can only write to output files created under the
UTL_FILE_DIR identified directory. The *filepath* variable in the stored procedure
has to be edited and given the value for the UTL_FILE_DIR initialization
parameter. In our example, it points to C:\Oracle as the UTL_FILE_DIR
parameter. The output file created will be named as TABLE_NAME.DAT file. For
example, to export the data in the ACCOUNTS table, the stored procedure is
called using the command:

    CALL EXPORT_TABLE('ACCOUNTS')

Note that the table name should be entered in uppercase. When the stored
procedure is called successfully, it produces the output file ACCOUNTS.DAT. To
load the data using the LOAD utility, use the command:

    DB2 LOAD FROM *ACCOUNTS.DAT* OF DEL MODIFIED BY CHARDEL~ INSERT INTO
    ACCOUNTS

To load the data using the IMPORT utility, use the command:

    DB2 IMPORT FROM *ACCOUNTS.DAT* OF DEL MODIFIED BY CHARDEL~ INSERT INTO
    ACCOUNTS

6.4 Alternative ways for moving data

Besides the MTK, there are many other tools and products for data movement. Here we show a couple of them. There are also a number of third-party tools that work with both Oracle and DB2, but they are not covered here. A quick Internet search can identify many of them. You should choose the tool according to your environment and the amount of data that will be moved.

6.4.1 Data movement through named pipes

As described in 6.1, "Data conversion process" on page 266, you need additional disk space during the data movement process. To avoid the space for the flat files, you can use named pipes on UNIX-based systems. To use this function, the writer and reader of the named pipe must be on the same machine. You must create the named pipe on a local file system before exporting data from the Oracle database.

Because the named pipe is treated as a local device, there is no need to specify that the target is a named pipe. The following is an AIX example:

1. Create a named pipe:

   `mkfifo /u/dbuser/mypipe`

2. Use this pipe as the target for a data unload operation:

   `<data unload routine> > /u/dbuser/mypipe`

3. Load the data into DB2 from the pipe:

   `<data load routine> < /u/dbuser/mypipe`

The commands in step 2 and 3 show only the principle of using the named pipes. To unload and load the data, use the routines discussed previously in this chapter.

> **Note:** It is important to start the pipe reader after starting the pipe writer. Otherwise, the reader will find an empty pipe and exit immediately.

6.4.2 WebSphere Federation Server

In a high availability environment, you will usally need to move the data during production activity. A practical solution is to use the federated access that WebSphere Federation Server provides.

IBM WebSphere Federation Server provides integrated, real-time access to diverse data as if it were a single database, regardless of where it resides. You

can hold the same data both in Oracle and in DB2, and you are free to switch to the new DB2 database when the functionality of the ported database and application is guaranteed.

The WebSphere Federation Server, formerly known as *WebSphere Information Integrator*, lets users manage data movement strategies between mixed relational data sources including distribution and consolidation models.

Data movement can be managed table-at-a-time, such as for warehouse loading during batch windows, or with transaction consistency for data that is never offline. It can be automated to occur on a specific schedule, at designated intervals, continuously, or as triggered by events. Transformation can be applied inline with the data movement through standard SQL expressions and stored procedure execution.

For more information, visit the WebSphere Federation Server product site:

http://www.ibm.com/software/data/integration/federation_server/

7

Application conversion

In this chapter, we discuss some aspects of converting an application from an Oracle environment to a DB2 environment. The following topics are covered:

► Planning

► Conversion of self-build applications
   – Written with Oracle Pro*C
   – Written in Java
   – Based on Oracle OCI
   – Based on ODBC
   – Based on Perl
   – Based on PHP
   – Based on .NET

► Package application migration

   – SAP

7.1 DB2 application development introduction

To develop applications that access the DB2 database, embed the data access method of the high-level language into the application. IBM DB2 provides various programming interfaces for data access and manipulation.

There are various methods for performing data interaction from your application, including: embedded static, dynamic SQL, native API calls, and methods provided by DB2 drivers for a specific application environment.

7.1.1 Embedded SQL

The SQL statement can be embedded within a host language where SQL statements provide the database interface while the host programming language provides all remaining functionality. Embedded SQL applications require a specific precompiler for each language environment in order to preprocess (or translate) the embedded SQL calls into the host language.

Building embedded SQL applications involves two prerequisite steps prior to application compilation and linking. This is different from building applications with Oracle database access, as Oracle database applications do not have the concept of binding applications to a database prior to runtime. It also explains why the static embedded SQL method is efficient and can yield good performance.

The two prerequisite steps for building DB2 embedded SQL applications are:

1. Preparing the source files containing embedded SQL statements using the DB2 precompiler

   The PREP (PRECOMPILE) command is used to invoke the DB2 precompiler. The precompiler reads the source code, parses and converts the embedded SQL statements to DB2 run-time services API calls, and writes the output to a new modified source file. The precompiler produces access plans for the SQL statements, which are stored together as a package within the database.

2. Binding the statements in the application to the target database

   Binding is done, by default, during precompilation (the PREP command). If binding is to be deferred (for example, running the BIND command later), then the BINDFILE option needs to be specified at PREP time in order for a bind file to be generated.

Figure 7-1 illustrates the precompile-compile-bind process for creating a program with embedded SQL.

Figure 7-1   Precompile-compile-bind process for creating embedded SQL applications

DB2 supports the C/C++, FORTRAN, COBOL, and Java (SQLJ) programming languages for embedded SQL.

Embedded SQL applications can be categorized as follows:

► Static embedded SQL

In embedded SQL, you are required to specify the complete SQL statement structure. This means that all the database objects (including columns and table) must be fully known at the precompile time with the exception of objects referenced in the SQL WHERE clause. However, all the host variable data types still must be known at the precompiler time. Note that host variables should be declared in a separate EXEC SQL DECLARE section and be compatible with DB2 data types.

Example 7-1 shows a fragment of a COBOL program with static embedded SQL.

Example 7-1   A COBOL static embedded SQL program

```
move "Clerk" to job-update.
EXEC SQL UPDATE staff SET job=:job-update
         WHERE job='Mgr'
END-EXEC.
move "UPDATE STAFF" to errloc.
```

- ► Dynamic embedded SQL

  If not every database object in the SQL statement is known at the precompile time, use dynamic embedded SQL. The dynamic embedded SQL statement accepts a character string host variable and a statement name as arguments. These character string host variables serve as placeholders for the SQL statements to be executed later. Note that dynamic SQL statements are prepared and executed during program runtime.

  Example 7-2 is a fragment of a C program with a dynamic SQL statement.

  Example 7-2   A dynamic SQL C program

  ```
  EXEC SQL BEGIN DECLARE SECTION;
  char st[80];
  char parm_var[19};
  EXEC SQL END DECLARE SECTION;

  strcpy( st, "SELECT tabname FROM syscat.tables" );
  strcat( st, " WHERE tabname <> ? ORDER BY 1" );

  EXEC SQL PREPARE s1 FROM :st;
  EXEC SQL DECLARE c1 CURSOR FOR s1;
  strcpy( parm_var, "STAFF" );
  EXEC SQL OPEN c1 USING :parm_var;
  ```

  Note that host variable PARM_VAR still needs to be declared in EXEC SQL DECLARE SECTION.

7.1.2 Driver support

DB2 supports numerous drivers for developing more complex applications.The driver manager defines a set of methods, variables, and conventions that provide a consistent database interface specifically for the DB2 database. Applications utilizing drivers are compiled and linked with the driver manager's libraries to invoke standardized APIs.

DB2 currently supports a large number of drivers, including CLI/ODBC, ADO and OLEDB, JDBC, SQLJ, PERL DBI, PHP, and the .NET Data Provider.

Perl database interface
To better understand how the interface works, let us examine the PERL database interface (DBI). A Perl program uses a standard API to communicate with the DBI module for Perl, which supports only dynamic SQL. It defines a set of methods, variables, and conventions that provide a consistent database interface independent of the actual database being used. DBI gives the API a consistent interface to any database that the programmer wishes to use.

DBD::DB2 is a Perl module which, when used in conjunction with DBI, allows Perl to communicate with the DB2 database.

Figure 7-2 illustrates the Perl/ DB2 environment.

Figure 7-2   Perl/DB2 environment

Installation

You can acquire Perl as follows:

▶ The source can be downloaded from

   http://www.perl.com

   and compiled.

▶ A binary version called ActivePerl, available for most operating systems, can be downloaded from ActiveState at:

   http://perl.about.com/gi/dynamic/offsite.htm?zi=1/XJ&sdn=perl&zu=htt
   p%3A%2F%2Fwww.activestate.com%2F

Note: The latest version of ActivePerl, at the time of this writing, is 5.8.8.820

System requirements

The following requirement information is available at the ActiveState Web site:

   http://perl.about.com/gi/dynamic/offsite.htm?zi=1/XJ&sdn=perl&zu=htt
   p%3A%2F%2Fwww.activestate.com%2F

▶ General

   – Recommended 90 MB hard disk space for typical install
   – Web browser for online help

▶ AIX

   – PowerPC®
   – Minimum AIX 5.1

- HP-UX
  - PA-RISC: Minimum HP-UX 11.00
  - Itanium: Minimum HP-UX 11.22
- Linux
  - x86
  - libc-2.1.x+ (e.g. Red Hat 6.x+, Debian 2.2+)
- Solaris
  - SPARC: Minimum Solaris 2.6
  - x64/x86: Minimum Solaris 10
  - You must use GNU tar to unpack the ActiveState Installer Package
- Windows
  - x86
  - All Windows platforms: IE 5.5+
  - Windows 95: DCOM for Windows 95
  - Windows NT: Service Pack 5+

In addition to Perl, two additional modules need to be downloaded and installed in order to enable the Perl driver for DB2:

- DBI
- DBD::DB2

These modules can be downloaded from the Comprehensive Perl Archive Network (CPAN) at:

> http://CPAN.org

Information regarding installation of these modules may also be found at this location.

> **Note:** For the latest information about PERL and DB2, and related PERL modules, refer to the following Web site:
>
> > http://www.ibm.com/software/data/db2/perl/

PHP extensions

IBM supports access to DB2 databases from PHP applications through two extensions that offer distinct sets of features:

- ibm_db2

  This is an extension written, maintained, and supported by IBM for access to DB2 databases. It offers a procedural API that, in addition to the normal create, read, update, and write database operations, also offers extensive

access to the database metadata. This extension can be compiled with either PHP 4 or PHP 5.

► PDO_IBM and PDO_ODBC

These are drivers for the PHP Data Objects (PDO) extension that offers access to DB2 databases through the standard object-oriented database interface. PDO_IBM is an IBM database driver. Both PDO_IBM and PDO_ODBC extensions can be compiled directly against the DB2 libraries to avoid the communications overhead and potential interference of an ODBC driver manager.

Installation

An easy method of installing and configuring PHP on Linux, UNIX, or Windows operating systems is Zend Core for IBM, which can be downloaded and installed for use in production systems from the following Web site:

http://www.zend.com/downloads

Additionally, precompiled binary versions of PHP are available for download from:

http://php.net/

Most Linux distributions include a precompiled version of PHP. On UNIX operating systems that do not include a precompiled version of PHP, your own version of PHP may be compiled.

Requirements

For setting up the PHP environment on Linux, UNIX, or Windows operating systems refer to *Developing Perl and PHP applications*; SC10-4234.

7.2 Application migration planning

Application migration is another major step in the migration project. This process includes:

► Check of software and hardware availability and compatibility
► Education of developers and administrators
► Analysis of application logic and source code
► Setting up the target environment
► Change of database specific items
► Application test
► Application tuning
► Roll-out
► User education

The planning includes the creation of a project plan. Plan enough time and resources for each task. IBM and our business partners can help you with questions in order to define a well prepared project.

For the applications developed in-house, the migration effort is on the shoulder of the migration team. For package applications, you can contact the vendor for a recommended migration process. In 7.4, "Package applications migration planning" on page 330, we explain the recommended migration process of some packages.

Check software and hardware availability and compatibility

The architecture profile is one of the outputs of the first task of migration planning assessment. While preparing the architecture profile, you need to check the availability and compatibility of all involved software and hardware in the new environment.

Education of developers and administrators

Ensure that the staff has the skills for all products and the system environment you will use for the migration project. Understanding the new product is essential for analyzing the source system.

Analyzing application logic and source code

In this analysis phase you should identify all the Oracle proprietary features and the affected sources. Examples of Oracle proprietary features are direct SQL queries to the Oracle Data Dictionary, Optimizer hints and Oracle joins, which are not supported by DB2. You also need to analyze the database calls within the application for the usage of database API.

Setting up the target environment

The target system, either the same or a different one, has to be set up for application development. The environment can include:

► The Integrated Development Environment (IDE)
► Database framework
► Repository
► Source code generator
► Configuration management tool
► Documentation tool

A complex system environment usually consists of products from different vendors. Check the availability and compatibility before starting the project.

Change of database-specific items

Regarding the use of the database API, you need to change the database calls in the applications. The changes include:

► Language syntax changes
The syntax of database calls varies in the different programming languages. In 7.3, "Self-build application" on page 288, we discuss the varieties of C/C++ and Java applications. For information regarding other languages, contact IBM Technical Sales.

► SQL query changes
Oracle supports partly nonstandard SQL queries such as including of optimizer hints or table joins with a (+) syntax. To convert such queries to standard SQL, you can use the MTL SQL Translator.

You need to modify the SQL queries to the Oracle Data Dictionary as well, and change them to select the data from the DB2 Catalog.

► Changes in calling procedures and functions
Sometimes there is a need to change procedures to functions and vice versa. In such cases, you have to change all the calling commands and the logic belonging to the calls in other parts of the database and the applications.

► Logical changes
Because of architectural differences between Oracle and DB2, changes in the program flow might be necessary. Most of the changes are related to the different concurrency models.

Application test

A complete application test is necessary after the database conversion and application modification to ensure that the database conversion is completed, and all the application functions work properly.

It is prudent to run the migration several times in a development system to guarantee the process, then run the same migration on a test system with existing test data, then a copy or subset of productions data, before eventually running the process in production. Chapter 10, "Testing" on page 447 discusses the testing steps in detail.

Application tuning

Tuning is a continuous activity for the database since data volume, number of users, and applications change from time to time. After the migration, application tuning should be concerned with the architectural differences between Oracle and DB2. For the details, see Chapter 10, "Testing" on page 447, in *DB2 Performance-tuning Guidelines*, SC10-4222, and *DB2 V7.1 Performance Tuning Guide*, SG24-6012.

Roll-out

The roll-out procedure varies and depends on the type of application and the kind of database connection you have. Prepare the workstations with the proper driver (for example, DB2 Runtime Client, ODBC, and JDBC) and server according to the DB2 version.

User education

In case of changes in the user interface, the business logic, or the application behavior because of system improvements, user education is required. Be sure to provide enough user education since the acceptance of the target system is corresponding to the skills and satisfaction of the users.

7.3 Self-build application

Self-build (in-house developed) applications are unique in every case. There are a variety of languages used in applications and each one can have its unique way of using APIs. In this section we explain the necessary steps in converting self-build applications from Oracle to DB2, and we provide some examples in C/C++ and Java, which show you how to convert the database calls.

Note that the examples included in this chapter are excerpts from the actual programs, and cannot be compiled and executed by themselves.

> **Note:** Since some differences exist in functionality and syntax between SQL statements in Oracle and DB2, all converted SQL statements should be tested in the target (DB2) environment before incorporating them in the converted DB2 application.

7.3.1 Converting Oracle Pro*C applications to DB2

While many aspects of DB2 application development underwent changes in recent years (stored procedures from C/COBOL/Java to SQL procedure language, support for PL/SQL in user-defined functions, triggers, and in-line SQL, and an enriched set of built-in functions, etc.), support for embedding SQL into other host languages (C/C++) practically has not changed.

This chapter explains the steps necessary during application conversion to programs with embedded DB2 SQL calls.

Connecting to the database

There is a difference in how C programs connect to the database. In Oracle each instance (service name) can manage only one database. DB2 instances can be

used to manage multiple databases, thus the database name should be implicitly provided by a connection statement.

In order to connect to the Oracle database, you need to specify the Oracle user and the password for that user:

```
EXEC SQL CONNECT :user_name IDENTIFIED BY :password;
```

In DB2, you need to specify the database name, user ID, and password for that user ID. So, the above statement will be converted to:

```
EXEC SQL CONNECT TO :dbname USERID :userid PASSWORD :password;
```

Note that `dbname`, `userid`, and `password` need to be declared as host variables.

Host variable declaration

Host variables are C or C++ language variables that are referenced within SQL statements. They allow an application to pass input data to and receive output data from the database manager. After the application is precompiled, host variables are used by the compiler as any other C/C++ variable.

Host variables should not only be compatible with DB2 data types (accepted by the DB2 precompiler), but also must be acceptable for the programming language compiler.

As the C program manipulates the values from the tables using host variables, the first step is to convert Oracle table definitions to DB2 data types; see Appendix A, "Data types" on page 597 for details. Note that this mapping is one-to-many because it depends on the actual usage of data. For example, Oracle DATE data can be converted to DB2 DATE, if it only stores the actual date, but it needs to be converted to DB2 TIMESTAMP if it stores DATE and TIME.

The next step is to match DB2 data types with C data types. The table in Appendix A, "Data types" on page 597 shows mapping between data types.

All host variables in a C program need to be declared in a special declaration section, so that the DB2 precompiler can identify the host variables and the data types:

```
EXEC SQL BEGIN
DECLARE SECTION;
   char emp_name[31] = {'\0'};
   sqlint32  ret_code = 0;
EXEC SQL END DECLARE SECTION;
```

Within this declaration section, there are rules for host variable data types that are different from Oracle precompiler rules. Oracle precompiler permits host variables to be declared as VARCHAR. VARCHAR[n] is a pseudo-type recognized by the Pro*C precompiler. It is used to represent blank-padded, variable-length strings. Pro*C precompiler converts it into a structure with a 2-byte length field followed by an n-byte character array. DB2 requires usage of standard C constructs. So, the declaration for the variable emp_name VARCHAR[25] needs to be converted as follows:

```
struct emp_name {
    short  var_len;
    char   var_data[25] };
```

Or, as mentioned above, the use of a char emp_name[n] is also permitted for VARCHAR data. Variables of user-defined types (using typedef) in PRO*C need to be converted to the source data type. For example, type theUser_t has been declared to host values from Oracle object type:

```
typedef struct  user_s
        {short int userNum;
          char userName[25];
          char userAddress[40];
        } theUser_t;
```

In Pro*C program, you can have host variables declared as theUser_t:

```
EXEC SQL BEGIN DECLARE;
         theUser_t *myUser;
      EXEC SQL END DECLARE SECTION;
```

To use this host variable for DB2, you would need to take it out of EXEC SQL DECLARE SECTION and define the host variable MyUser as a structure.

DB2 allows for the host variable to be declared as a pointer with the following restriction: If a host variable is declared as a pointer, no other host variable may be declared with that same name within the same source file.

The host variable declaration char *ptr is accepted, but it does not mean a null-terminated character string of an undetermined length. Instead, it means a pointer to a fixed-length, single-character host variable. This may not be what was intended for the Oracle host variable declaration.

It is recommended that sqlint32 and sqlint64 be used for INTEGER and BIGINT host variables, respectively. By default, the use of long host variables results in the precompiler error SQL0402 on platforms where long is a 64-bit quantity such as 64 BIT UNIX. Use the PREP option LONGERROR NO to force DB2 to accept

long variables as acceptable host variable types and treat them as BIGINT variables.

Starting with Version 9, DB2 supports XML type for host variables. In the declaration section of the application, declare the XML host variables as LOB data types as shown in the following:

```
EXEC SQL BEGIN DECLARE;
    SQL TYPE IS XML as CLOB(N) my_xml_var1;
    SQL TYPE IS XML as BLOB(N) my_xml_var2;
EXEC SQL END DECLARE SECTION;
```

You can learn more on handling XML types within C applications from *DB2 Express-C: The Developer Handbook for XML, PHP, C/C++, Java, and .NET,* SG24-7301.

Oracle host tables

In Pro*C programs, you can declare host variables using arrays, then declare a cursor you want to get results from. You can then issue a fetch statement that will get all rows from the cursor into that host array.

Here is a fragment of PRO*C that demonstrates this method:

```
EXEC SQL BEGIN DECLARE SECTION;

long int    dept_numb[10];
char        dept_name[10][14];
char        v_location[12];

EXEC SQL END DECLARE SECTION;
/* …… */

EXEC SQL DECLARE CUR1 CURSOR FOR
    SELECT DEPTNUMB, DEPTNAME
     FROM org_table
    WHERE LOCATION = :v_location;
/*……. */

EXEC SQL FETCH CUR1 INTO :dept_num, :dept_name;
```

The last statement will get all 10 rows from the cursor into arrays.

Because DB2 does not support arrays for the host variable declaration, the above code needs to be converted as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
sqlint32      h_dept_numb = 0;
char          h_dept_name[14] = {'\0'};
char          v_location[12] = {'\0'};

EXEC SQL END DECLARE SECTION;
/* move array out of DECLARE section - just C variables */
long int      dept_numb[10];
char          dept_name[10][14];
short int     i = 0;

/* …… */

EXEC SQL DECLARE CUR1 CURSOR FOR
     SELECT DEPTNUMB, DEPTNAME
       FROM org_table
      WHERE LOCATION = :v_location;

/*we need Fetch one row at the time and move to corresponding
     member of array */

for (i=0;i<11;i++){
    EXEC SQL FETCH CUR1 INTO :h_dept_num, :h_dept_name;
    if (SQLCODE == 100) {
      break;
    }
    dept_numb[i] = h_dept_numb;
    strcpy(dept_name[i], h_dept_name);
}
```

Exception handling

The mechanisms for trapping errors are quite similar between Oracle and DB2, using the same concept of separating error routines from the mainline logic. There are three different WHENEVER statements that could be used to define program behavior in case of an error in DB2:

```
EXEC SQL WHENEVER SQLERROR GOTO error_routine;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND not_found_routine;
```

Although the WHENEVER statement is prefixed by EXEC SQL like other SQL statements, it is not an executable statement. Instead, a WHENEVER statement causes the precompiler to generate code in a program to check the SQLCODE attribute from the SQLCA after each SQL statement, and to perform the action specified in the WHENEVER statement. SQLERROR means that an SQL statement returns a negative SQLCODE indicating an error condition.

SQLWARNING indicates a positive SQLCODE (except +100), while NOT FOUND specifies SQLCODE = +100, indicating that no data rows were found to satisfy a request.

A compilation unit can contain as many WHENEVER statements as necessary, and they can be placed anywhere in the program. The scope of one WHENEVER statement reaches from the placement of the statement in the file onward in the character stream of the file until the next suitable WHENEVER statement is found or end-of-file is reached. No functions or programming blocks are considered in that analysis. For example, you may have two different SELECT statements: one must return at least one row, and the other may not return any. You will need two different WHENEVER statements:

```
EXEC SQL WHENEVER NOT FOUND GOTO no_row_error;
    EXEC SQL SELECT  address
                       INTO  :address
                       FROM   test_table
                     WHERE phone = :pnone_num;
    ……..
    EXEC SQL WHENEVER NOT FOUND CONTINUE;
    EXEC SQL SELECT  commis_rate
                       INTO :rate :rateind
                        WHERE prod_id = :prodId;
     if (rateind == -1) rate = 0.15;
      ……
```

Oracle precompiler also supports DO and STOP statements as actions in a WHENEVER statement; those are not supported by the DB2 precompiler and need to be converted to GOTO.

Another alternative is to check SQLCODE explicitly after each EXEC SQL statement because that allows more context-sensitive error handling.

Error messages and warnings

The SQL Communication Area (SQLCA) data structure in DB2 is similar to the same structure of Oracle. SQLCA provides information for diagnostic checking and event handling.

To get the full text of longer (or nested) error messages, you need the sqlglm() function:

```
sqlglm(message_buffer, &buffer_size, &message_length);
```

where `message_buffer` is the character buffer in which you want Oracle to store the error message; `buffer_size` specifies the size of `message_buffer` in bytes;

Oracle stores the actual length of the error message in `*message_length`. The maximum length of an Oracle error message is 512 bytes.

DB2 provides its user with a special runtime API function to return an error message based on SQLCODE:

```
rc=sqlaintp(msg_buffer, 1024, 80, sqlca.sqlcode);
```

Where 80 stands for the number of characters after which a line break will be inserted in the message. DB2 will search for work boundaries to place such a line break. 1024 specifies the length of the message buffer, for example `char msg_buffer[1024]`. As a result of invoking this function, the allocated buffer will contain the descriptive error message, for example:

```
SQL0433N Value "TEST VALUES" is too long. SQLSTATE=22001.
```

If you need more information about a particular error, DB2 provides an API function that returns an extended message associated with the specific SQLSTATE:

```
rc=sqlogstt(msg_sqlstate_buffer, 1024, 80, sqlca.sqlcode);
```

As a result of invoking this function, `char msg_sqlstate_buffer[1024]` will contain, for example, the following message:

```
SQLSTATE 22001: Character data, right truncation occurred; for
example, an update or insert value is a string that is too long for
the column, or datetime value cannot be assigned to a host variable,
because it is too small.
```

Passing data to a stored procedure from a C program

In Oracle, in order to invoke a remote database procedure, the following statements are used:

```
EXEC SQL EXECUTE
        BEGIN
          Package_name.SP_name(:arg_in1,:arg_in2, :status_out);
        END;
END-EXEC;
```

The value transfer between the calling environment and the stored procedure may be achieved through arguments. You can choose one of three modes for each argument: IN, OUT or INOUT. For example, the above stored procedure may be declared as:

```
CREATE PACKAGE package_name IS
     PROCEDURE  SP_name(
              agr_in1 IN NUMBER ,
              arg_in2 IN CHAR(30),
```

```
                status_out OUT NUMBER);
```

When this stored procedure is invoked, values passed from the calling program will be accepted by the stored procedure correspondingly.

The DB2 client application invokes a stored procedure by using the CALL statement, which can pass parameters to the stored procedure and receive parameters returned from the stored procedure. It has the following syntax:

```
CALL procedure_name (:parm1, …:parmN);
```

As with all SQL statements, you prepare CALL statement with parameter markers and then supply values for the markers using SQLDA:

```
CALL procedure_name USING DESCRIPTOR host_var;
```

The SQLDA is very helpful if you have an unknown number of host variables or very many variables—such as 100 or more. Managing single variables in those cases can be very troublesome.

In order to invoke a stored procedure from C client, the following need to be in place:

► A stored procedure needs to be created and registered with the database.

► A host variable or parameter marker to each IN and INOUT parameter of the stored procedure should be declared and initialized.

Consider an example. The program must give a raise to each employee whose current salary is less than some value. The program will pass that value to a stored procedure, perform an update, and return back the status. The client code in C will look as shown in Example 7-3.

Example 7-3   Passing data to a stored procedure

```
#include <sqlenv.h>

main()
{
    EXEC SQL BEGIN DECLARE SECTION;
        Sqlint32  salary_val=0;
        Sqlint16 salind=1;
        Sqlint16  status=0;
        Sqlint16  statind=0;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL INCLUDE SQLCA;
    EXEC SQL CONNECT TO sample;
    EXEC SQL WHENEVER SQLERROR GOTO err_routine;
```

```
        salary_val = getSalaryForRaise();
        statind = -1;   /* set indicator variable to -1 */
                        /* for status as output-only variable */

      EXEC SQL CALL raiseSal(:salary_val :salind, :status :statind);
       if (status == 0){
           printf  (" The raises has been successfully given \n ");
           EXEC SQL COMMIT;
        }
        else
           if (status ==1)
               printf  (" NO input  values has been provided.\n ");
           else
               if (status == 2)
                    printf("Stored procedure failed.\n");

    err_routine:
        printf  (" SQL Error, SQLCODE =  \n ", SQLCODE);
        EXEC SQL ROLLBACK;
}
```

Note that all host variables that are used as parameters in the statement are declared and initialized in EXEC SQL DECLARE SECTION.

Building a C/C++ DB2 application

DB2 provides sample build scripts for precompiling, compiling, and linking C-embedded SQL programs. These are located in the sqllib/samples/c directory, along with sample programs that can be built with these files. This directory also contains the **embprep** script used within the build script to precompile a *.sqc file.

Build files are provided by DB2 for each language on supported platforms where the types of programs they build are available in the same directory as the sample programs for each language. These build files, unless otherwise indicated, are for supported languages on all supported platforms. The build files have the .bat (batch) extension on Windows, and have no extension on UNIX platforms. For example, **bldmapp.bat** is a script to build C/C++ applications on Windows.

DB2 also provides utilemb.sqc and utilemb.h files, containing functions for error handling. In order to use utility functions, the utility file must first be compiled, and then its object file linked during the creation of the target program's executable. Both the makefile and build files in the sample directories do this for the programs that require error-checking utilities.

For more information on building C applications, see *Application Development Guide: Building and Running Applications V8*, SC09-4825-01.

7.3.2 Converting Oracle Java applications to DB2

For Java programmers, DB2 offers two APIs: JDBC and SQLj.

JDBC is a mandatory component of the Java programming language as defined in the Java 2, Standard Edition (J2SE™) specification. To enable JDBC applications for DB2, an implementation of the various Java classes and interfaces, as defined in the standard, is required. This implementation is known as a JDBC driver. DB2 offers a complete set of JDBC drivers for this purpose. They are categorized as the CLI drivers, the DB2 JDBC Type 2 Driver for Linux, UNIX and Windows (DB2 JDBC Type 2 Driver), or IBM DB2 Driver for JDBC and SQLJ (Type 2 and Type 4). The Type 2 Driver is deprecated.

SQLJ is a standard development model for data access from Java applications. The SQLJ API is defined in the SQL 1999 specification. The DB2 JDBC Type 2 Driver for Linux, UNIX and Windows, or the IBM DB2 Driver for JDBC and SQLJ provide support for both JDBC and SQLJ APIs in a single implementation. JDBC and SQLJ can interoperate in the same application. SQLJ provides the unique ability to develop using static SQL statements and control access at the DB2 package level.

The Java code conversion is rather easy. The API itself is well defined and database independent. For instance, the database connection logic is encapsulated in standard J2EE™ DataSource objects. The Oracle or DB2 specific things such as user name, database name, etc. are then configured declaratively within the application.

However, there is the need to change your Java source code regarding:

► The API driver (JDBC or SQLJ).

► The database connect string.

► Oracle's proprietary SQL, such as CONNECT BY for recursive SQL, the usage of DECODE() or SQL syntax like the (+) operator instead of LEFT/RIGHT OUTER JOIN. MTK provides support here with the SQL Translator.

► Remove or simulate proprietary optimizer hints in SQL queries.

For complete information regarding the Java environment, drivers, programming, and other relevant information, consult *Developing Java Applications*, SC10-4233.

Java access methods to DB2

DB2 has rich support for the Java programming environment. You can access DB2 data by putting the Java class into a module in one of the following ways:

- ▶ DB2 Server
  - – Stored procedures (JDBC or SQLJ)
  - – SQL functions or user-defined functions (JDBC or SQLJ)
- ▶ Browser
  - – Applets based on JDBC (JDBC)
- ▶ J2EE Application Servers (such as WebSphere Application Server)
  - – Java ServerPages (JSPs) (JDBC)
  - – Servlets (SQLJ or JDBC)
  - – Enterprise JavaBeans™ (EJBs) (SQLJ or JDBC)

JDBC drivers for DB2

The drivers that are supported for DB2 9 are:

- ▶ DB2 JDBC Type 2 Driver for Linux, UNIX and Windows
- ▶ IBM DB2 Driver for JDBC and SQLJ

The DB2 JDBC Type 2 Driver for Linux, UNIX, and Windows has been deprecated. Currently, the IBM DB2 Driver for JDBC and SQLJ is the recommended driver. This driver supports both Type 2 and Type 4 connections.

The DB2 JDBC Type 2 Driver supports:

- ▶ Most of the methods that are described in the JDBC 1.2 specification, and some of the methods that are described in the JDBC 2.0 specification.
- ▶ SQLJ statements that perform equivalent functions to all JDBC methods.
- ▶ Connection pooling.
- ▶ Distributed transactions.
- ▶ Java user-defined functions and stored procedures.

The IBM DB2 Driver for JDBC and SQLJ supports**:**

- ▶ All of the methods that are described in the JDBC 3.0 specifications.
- ▶ SQLJ statements that perform equivalent functions to most JDBC methods.
- ▶ Connections that are enabled for connection pooling. WebSphere Application Server or another application server does the connection pooling.
- ▶ Java user-defined functions and stored procedures (IBM DB2 Driver for JDBC and SQLJ type 2 connectivity only).
- ▶ Global transactions that run under WebSphere Application Server Version 5.0 and above.
- ▶ Support for distributed transaction management. This support implements the Java 2 Platform, Enterprise Edition (J2EE) Java Transaction Service (JTS)

and Java Transaction API (JTA) specifications, which conform to the X/Open standard for distributed transactions.

Type 2 connectivity

For IBM DB2 Driver for JDBC and SQLJ Type 2 connectivity, the getConnection method must specify a URL, as shown in the following:

```
getConnection(String url);
```

The following is the syntax of the URL for the Type 2 connectivity.

```
>>-+-+-jdbc:db2:database----------+-+------------------------><
   | +-jdbc:db2os390:database-----+ |
   | +-jdbc:db2os390sqlj:database-+ |
   | +-jdbc:default:connection----+ |
   | +-jdbc:db2os390--------------+ |
   | '-jdbc:db2os390sqlj----------' |
   |    .----------------------.    |
   |    V                      |    |
   '-:---property--=--value--;-+----'
```

Type 4 connectivity

For IBM DB2 Driver for JDBC and SQLJ Type 4 connectivity, the getConnection method must specify a user ID and a password, through parameters or through property values. See Example 7-4.

Example 7-4   getConnection syntax for Type 4 connectivity

```
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

The following is the syntax for a URL for IBM DB2 Driver for JDBC and SQLJ Type 4 connectivity.

```
>>-+-jdbc:db2:------+-//server--+-------+--/database------------>
   '-jdbc:db2j:net:-'           '-:port-'

>--+---------------------------+---------------------------><
   |    .----------------------. |
   |    V                      | |
   '-:---property--=--value--;-+-'
```

Example 7-5 shows how to set the user ID and password in the *user* and *password* parameters.

Example 7-5   Setting the user ID and password in the user and password parameters

```
// Set URL for data source
String url = "jdbc:db2://sysmvs99.stl.ibm.com:6543/san_jose";
```

```
// Create connection
String user = "db2inst1";
String password = "db2inst1";
Connection con = DriverManager.getConnection(url, user, password);
```

JDBC driver declaration

In order to connect from a Java application to an Oracle database using the OCI driver, the following must be done:

1. Import the Oracle driver.
2. Register the driver manager.
3. Connect with a user ID, the password, and the database name.

Example 7-6 shows an Oracle JDBC connection through OCI:

Example 7-6   Oracle JDBC connection

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;

class rsetClient
{
  public static void main (String args []) throws SQLException
  {
    // Load the driver
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // Connect to the database
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@oracle","uid","pwd");

    // ...
  }
}
```

It is *not* necessary to import a JDBC library when connecting to DB2. The registration and connection to DB2 is demonstrated in Example 7-7. Be aware that the parameters for the getConnection method will be determined by the connection type.

Example 7-7   DB2 JDBC connection

```
import java.sql.*;

class rsetClient
{
  public static void main (String args []) throws SQLException  {
```

```
    // Load DB2 JDBC application driver
    try
     {
    // use ONE of the following - depending on the chosen driver:
//IBM DB2 Driver for JDBC and SQLJ
        Class.forName("com.ibm.db2.jcc.DB2Driver");
//DB2 JDBC Type 2 Driver for Linux, UNIX and Windows
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     }
    catch (Exception e)
    {
         e.printStackTrace();
    }
    // Connect to the database
    Connection conn =
    // Use appropriate parameters for getConnection depending on chosen     //
driver.
     DriverManager.getConnection("jdbc:db2://dbname","uid","pwd");
    // ...
  }
}
```

Stored procedure calls

The handling of input and output parameters in stored procedures calls differ
between Oracle and DB2. The following examples explain the different kinds of
procedure calls, and the usage of parameters and result sets.

Stored procedure with an input parameter

A stored procedure has been created in Oracle as:

```
CREATE OR REPLACE  PROCEDURE sp_testcall_1( parm1 IN INTEGER
                                          ,parm2 IN INTEGER)
```

and in DB2 as:

```
CREATE PROCEDURE sp_testcall_1( IN parm1 INTEGER
                               ,IN parm2 INTEGER )
```

The procedures have two input parameters and no output parameters. There is
no difference in the call between Oracle and DB2. In both cases the parameter
values need to be set *before* the stored procedure can be executed. Example 7-8
demonstrates this point.

Example 7-8   Java call of Oracle or DB2 procedure with input parameter

```
String SP_CALL = "{call sp_testcall_1(?,?)}";
```

```
// Connect to the database
Connection conn =
  DriverManager.getConnection (url, userName, password);

CallableStatement stmt;
try {
    stmt = conn.prepareCall(SP_CALL);
    stmt.setInt(1,10);
    stmt.setInt(2,15);
    stmt.execute();
    // ...
}
```

Stored procedure with a result set

The next example shows a procedure without an input parameter, but defines a result set as an output parameter. The result set is an opened cursor defined in the procedure. The rows are fetched in the Java application with a loop.

The Oracle stored procedure is defined as:

```
TYPE CursorType IS REF CURSOR;
CREATE PROCEDURE sp_testcall_3(oCursor OUT CursorType) AS
BEGIN
 open oCursor for select last_name from employees;
END;
```

The output parameter type is registered as CURSOR before the procedure is called.

Example 7-9   Java call of Oracle procedure with result set

```
String SP_CALL = "{call sp_testcall_3}";

// Connect to the database
Connection conn =
  DriverManager.getConnection (url, userName, password);

try {
    CallableStatement stmt = conn.prepareCall(SP_CALL);
    stmt.registerOutParameter (1, OracleTypes.CURSOR);
    stmt.executeUpdate();
    ResultSet rs = (ResultSet) stmt.getObject(1);
    while(rs.next())
    {
        // ...
    }
}
```

The corresponding DB2 procedure looks like this:

```
CREATE PROCEDURE db2_spcall_v3
RESULT SETS 1
LANGUAGE SQL
BEGIN
    DECLARE c1 CURSOR WITH RETURN FOR
       SELECT last_name
       FROM employees;

    OPEN c1;
END!
```

The result set definition in SQL PL is different from Oracle's PL/SQL. You have to specify the amount of expected result sets.

With DB2, you do not need to register the result set with the method registerOutParameter() in the Java application. To get the result set you call the method getResultSet() instead of getObject(), as in Example 7-9.

Example 7-10 Java call of DB2 procedure with result set

```
String SP_CALL = "{call db2_spcall_v3}";

// Connect to the database
Connection conn =
      DriverManager.getConnection (url, userName, password);

try {
        CallableStatement stmt = conn.prepareCall(SP_CALL);
        ResultSet rs = null;
        stmt.execute();
        rs = stmt.getResultSet();
        while(rs.next())
        {
            // ...
        }
}
```

Stored procedure with an input parameter and result set

Example 7-11 is a combination of Example 7-8 and Example 7-9. Note the numbering of the parameters. The first input parameter, value2, is numbered with 2, the result set *rs* is numbered with 1.

Example 7-11 Java call of Oracle procedure with input parameter and result set

```
private static final String SP_CALL = "{call sp_testcall4 (?) }";

CallableStatement stmt1 = conn.prepareCall(SP_CALL);
```

```
Stmt1.registerOutParameter(1, OracleTypes.CURSOR);
Stmt1.execute();
ResultSet rs = (ResultSet) stmt1.getObject(1);

while(rs.next()) {
    int value1 = rs.getInt(1);
    stmt2.setInt(2, value2);
    stmt2.execute();
    ResultSet rs = (ResultSet) stmt1.getObject(1);
    // ...
}
```

In DB2, input parameters and result sets are defined as shown in Example 7-12 and Example 7-13. Input parameters are numbered, beginning with 1, and are independent from the retrieval of result sets.

Example 7-12   Java call of DB2 procedure with input parameter and result set

```
String SP_CALL = "{call db2_spcall_v4(?)}";

Connect to the database
    Connection conn =
      DriverManager.getConnection (url, userName, password);

try {
        CallableStatement stmt = conn.prepareCall(SP_CALL);
        stmt.setInt(1, emp_id);
        ResultSet rs = null;
        stmt.execute();
        rs = stmt.getResultSet();
        while(rs.next())
        {
            System.out.println (rs.getString (1));
            // ...
        }
}
```

Stored procedure converted from a function

Calling an Oracle function is similar to calling a stored procedure with an input parameter and a result set. The function is defined as:

```
CREATE TYPE CursorType IS REF CURSOR;
CREATE FUNCTION sp_testcall_4(v_num IN INTEGER)
                RETURN CursorType
```

Example 7-13   Java of Oracle function with input parameter and result set

```
String SP_CALL = "{? = call sp_testcall_4(?)}";

// Connect to the database
Connection conn =
  DriverManager.getConnection (url, userName, password);

try {
    CallableStatement stmt = conn.prepareCall(SP_CALL);
    stmt.registerOutParameter (1, OracleTypes.CURSOR);
    stmt.setInt(2, 6);
    stmt.execute();
    ResultSet rs = (ResultSet) stmt.getObject(1);
    while(rs.next())
    {
        // ...
    }
}
```

As described in Chapter 5, "Conversion reference" on page 165, Oracle functions may need to be converted to a stored procedure in DB2. The migrated DB2 procedure may look like the one shown in Example 7-13, with an input parameter and a result set as an output parameter.

7.3.3 Converting Oracle Call Interface applications

You may want to consider rewriting applications using the Oracle Call Interface (OCI) by using CLI or ODBC. The OCI is specific to the Oracle database and cannot be used with any other databases.

In most cases, you can replace OCI functions with the appropriate CLI or ODBC functions, followed by relevant changes to the supporting program code. The remaining non-OCI program code should require minimal modification. The examples in this chapter show a comparison of the OCI and CLI or ODBC statements required for establishing a connection to an Oracle and DB2 database.

Introduction to CLI

DB2 Call Level Interface (DB2 CLI) is the IBM callable SQL interface to the DB2 family of database servers. It is a C and C++ API for relational database access that uses function calls to pass dynamic SQL statements as function arguments. It is an alternative to embedded dynamic SQL, but unlike embedded SQL, DB2 CLI does not require host variables or a precompiler.

DB2 CLI is based on the Microsoft Open Database Connectivity (ODBC) specification, and the International Standard for SQL/CLI. These specifications were chosen as the basis for the DB2 Call Level Interface in an effort to follow industry standards, and to provide a shorter learning curve for application programmers already familiar with either of these database interfaces. In addition, some DB2-specific extensions have been added to help the application programmer specifically exploit DB2 features.

The DB2 CLI driver also acts as an ODBC driver when loaded by an ODBC driver manager. It conforms to ODBC 3.51.

Comparison of DB2 CLI and Microsoft ODBC

Figure 7-3 compares DB2 CLI and the DB2 ODBC driver. The left side shows an ODBC driver under the ODBC Driver Manager, and the right side illustrates DB2 CLI, the callable interface designed for DB2-specific applications.

Figure 7-3   DB2 CLI and ODBC

In an ODBC environment, the Driver Manager provides the interface to the application. It also dynamically loads the necessary driver for the database

server that the application connects to. It is the driver that implements the ODBC function set, with the exception of some extended functions implemented by the Driver Manager. In this environment, DB2 CLI conforms to ODBC 3.51.

For ODBC application development, you must obtain an ODBC Software Development Kit. For the Windows platform, the ODBC SDK is available as part of the Microsoft Data Access Components (MDAC) SDK, available for download from:

http://www.microsoft.com/data/

For non-Windows platforms, the ODBC SDK is provided by other vendors.

In environments without an ODBC driver manager, DB2 CLI is a self-sufficient driver, which supports a subset of the functions provided by the ODBC driver. Appendix D, "Oracle Call Interface (OCI) mapping" on page 667 summarizes the two levels of support. The CLI and ODBC function summary provides a complete list of ODBC functions and indicates whether they are supported.

Setting up the CLI environment

Runtime support for DB2 CLI applications is contained in all DB2 clients. Support for building and running DB2 CLI applications is contained in the DB2 Application Development (DB2 AD) Client.

The CLI/ODBC driver will auto bind on the first connection to the database, provided the user has the appropriate privilege or authorization. The administrator may want to perform the first connect or explicitly bind the required files.

Procedure

In order for a DB2 CLI application to successfully access a DB2 database:

1. Ensure that the DB2 CLI/ODBC driver was installed during the DB2 client install.

2. Catalog the DB2 database and node if the database is being accessed from a remote client. On the Windows platform, you can use the CLI/ODBC settings GUI to catalog the DB2 database.

3. Optional: Explicitly bind the DB2 CLI/ODBC bind files to the database with the command:

```
db2 bind ~/sqllib/bnd/@db2cli.lst blocking all messages cli.msg\
          grant public
```

On the Windows platform, you can use the CLI/ODBC settings GUI to bind the DB2 CLI/ODBC bind files to the database.

4. Optional: Change the DB2 CLI/ODBC configuration keywords by editing the db2cli.ini file, located in the sqllib directory on Windows, and in the sqllib/cfg directory on UNIX platforms.

On the Windows platform, you can use the CLI/ODBC settings GUI to set the DB2 CLI/ODBC configuration keywords.

Change of OCI database calls

All Oracle Call Interface (OCI) calls in your application need to be changed to CLI calls. The program flow is retained, but you need to modify the definition and processing of *database handles*. There may not be an exact match in the conversion process. Your program code might require additional revisions to obtain similar functionality.

The following examples show you the different SQL statements in order to connect to a database. In Oracle you need to define variables for the environment handles as well as the database name, username, and password:

```
ociRC = OCILogon( env_hp,              // environment handle
                  err_hp,              // error handle
                  &svc_hp,             // service context
                  user_name,           // username
                  strlen (user_name),  // length of username
                  password,            // password
                  strlen (password),   // length of password
                  db_name              // database name
                  strlen (db_name));   // length of database name
```

In DB2 you also need to specify the connection handle, database name, username, and password. So, the OCI statement will be converted as follows:

```
cliRC = SQLConnect( *pHdbc,            // connection handle
                    db_name,           // database name
                    strlen (db_name),  // length of database name
                    user_name,         // username
                    strlen (user_name), // length of username
                    password,          // password
                    strlen (password)); // length of password
```

Appendix D, "Oracle Call Interface (OCI) mapping" on page 667 gives you a mapping of the most important Oracle8 OCI calls to the closest DB2 CLI equivalents. Refer to the *Oracle8i Server Application Development Guide* and to the *Call Level Interface Guide and Reference, Volume 1*, SC10-4224, and *Volume 2*, SC10-4225 for details on the OCI and CLI functions.

The following classes of OCI functions have no equivalents in DB2 CLI. The functionality must be implemented either in SQL or in C (or C++) directly:

► Navigational functions

```
OCIObject__()
OCICache__()
```

► Datatype mapping and manipulation functions

```
OCIColl__()
OCIDate__()
OCINumber__()
OCIString__
```

and so on.

However, CLI performs conversion of data between data types wherever possible.

► External procedure functions

```
OCIExtProc__()
```

Error handling and diagnostics

Diagnostics refers to dealing with warning or error conditions generated within an application. Two levels of diagnostics are returned when calling DB2 CLI functions:

► Return codes
► Detailed diagnostics consisting of SQLSTATEs, messages, and SQLCA

Each CLI function returns the function return code for a basic diagnosis. The functions SQLGetDiagRec() and SQLGetDiagField() provide more detailed diagnostic information. If the diagnostic originates at the DBMS, the SQLGetSQLCA() function provides access to the SQLCA. This arrangement lets applications handle the basic flow control based on return codes, and uses the SQLSTATES along with the SQLCA to determine the specific causes of failure, and to perform specific error handling.

Table 7-1 lists the mapping of all possible return codes of Oracle OCI functions and DB2 CLI functions.

Table 7-1   Return code mapping from OCI to CLI functions

OCI return code	CLI return code	Explanation
OCI_SUCCESS	SQL_SUCCESS	The function completed successfully, no additional SQLSTATE information is available.

OCI return code	CLI return code	Explanation
OCI_SUCCESS_WITH_INFO	SQL_SUCCESS_WITH_INFO	The function completed successfully with a warning or other information. Call SQLGetDiagRec() or SQLGetDiagField() to receive the SQLSTATE and any other informational messages or warnings. The SQLSTATE will have a class of '01'.
OCI_NO_DATA	SQL_NO_DATA_FOUND	The function returned successfully, but no relevant data was found. When this is returned after the execution of an SQL statement, additional information may be available and can be obtained by calling SQLGetDiagRec() or SQLGetDiagField().
OCI_ERROR	SQL_ERROR	The function failed. Call SQLGetDiagRec() or SQLGetDiagField() to receive the SQLSTATE and any other error information.
OCI_INVALID_HANDLE	SQL_INVALID_HANDLE	The function failed due to an invalid input handle (environment, connection or statement handle). This is a programming error. No further information is available.
OCI_NEED_DATA	SQL_NEED_DATA	The application tried to execute an SQL statement but DB2 CLI lacks parameter data that the application had indicated would be passed at execute time.
OCI_STILL_EXECUTING	SQL_STILL_EXECUTING	The function is running asynchronously and has not yet completed. The DB2 CLI driver has returned control to the application after calling the function, but the function has not yet finished executing.
OCI_CONTINUE	no equivalent	

The OCI function OCIErrorGet() returns the diagnostic record according to the SQLSTATE. Within a CLI application, the functions SQLGetDiagRec() and SQLGetDiagField() return three pieces of information:

► SQLSTATE

► Native error

   If the diagnostic is detected by the data source, this is the SQLCODE; otherwise, this is set to -99999.

► Message text

   This is the message text associated with the SQLSTATE.

SQLGetSQLCA() returns the SQLCA for access to specific fields, but should only be used when SQLGetDiagRec() or SQLGetDiagField() cannot provide the desired information.

Further information

You can find more information about CLI applications and development in:

► *DB2 Call Level Interface Guide and Reference, Volume 1,* SC10-4224; *Volume 2,* SC10-4225

► *Getting Started with Database Application Development*, SC10-4252

► Web site:

   http://www.ibm.com/software/data/db2/udb/ad

7.3.4 Converting ODBC applications

The Open Database Connectivity (ODBC) is similar to the CLI standard. Applications based on ODBC can connect to the most popular databases. Thus, the application conversion is pretty easy. You have to perform the conversion of database-specific items in your application, such as:

► Proprietary SQL query changes
► Possible changes in calling stored procedures and functions
► Possible logical changes

And the test, roll-out, and education tasks as well. Your current development environment will be the same. For a more detailed description of the necessary steps, reference 7.2, "Application migration planning" on page 285.

7.3.5 Converting Perl applications

In this section we discuss using Perl for connecting to Oracle and DB2 databases, and demonstrate the conversion from Oracle to DB2 by some simple Perl programs.

We create a stored procedure and a Perl program to demonstrate the following syntactical differences between Oracle and DB2:

- ► Connecting to a database using Perl
- ► Calling a stored procedure with an input and an output parameter
- ► Returning an output parameter

Example 7-14 is an Oracle stored procedure Greeting. It contains an input parameter *name*, and an output parameter *message*.

Example 7-14   Oracle stored procedure Greeting

```
CREATE OR REPLACE PROCEDURE Greeting (name IN
    VARCHAR2, message OUT VARCHAR2)
    AS
    name2 varchar2(30);
BEGIN
          name2 := UPPER(name);
          message := 'Hello ' || name2 ||', the date is: ' ||
                      SYSDATE;
END;
```

Example 7-15 shows the Perl program oraCallGreeting.pl. This program connects to the Oracle database, binds the input and output parameters, executes the call to the Greeting stored procedure, and returns the output parameter.

Example 7-15   Oracle Perl program oraCallGreeting.pl

```
#!/usr/bin/perl
use DBI;

$database='dbi:Oracle:xp10g';
$user='sample';
$password='sample';


$dbh = DBI->connect($database,$user,$password);
print "  Connected to database.\n";

$name = 'Ariel';
$message;
```

```
$sth = $dbh->prepare(q{
        BEGIN
                Greeting(:name, :message);
          END;
    });

$sth->bind_param(":name", $name);
$sth->bind_param_inout(":message", \$message, 100);
$sth->execute;
print "$message", "\n";

 # check for problems ...
 warn $DBI::errstr if $DBI::err;

$dbh->disconnect;
```

The results of the execution of oraCallGreeting.pl are displayed in Figure 7-4.

Figure 7-4   The result of executing the Perl program oraCallGreeting.pl

Converting Perl application to DB2

In this section, we demonstrate how to connect to DB2 using Perl.

Example 7-16 is a DB2 stored procedure that has the same function and same name as the Oracle stored procedure shown in Example 7-14 on page 312. The procedure Greeting also contains an input parameter *name*, and an output parameter *message*.

Example 7-16   DB2 stored procedure Greeting

```
CREATE PROCEDURE Greeting (IN name VARCHAR(30), OUT message VARCHAR(100))
LANGUAGE SQL
```

```
RESULT SETS 0
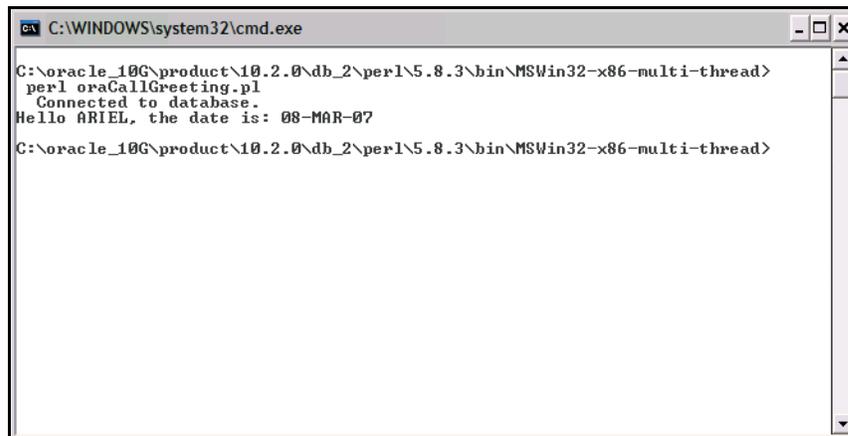

BEGIN

   DECLARE timeofday DATE;

   set name = UPPER(name);
   values CURRENT DATE into timeofday;
   set greeting = 'Hello ' || name || ', the date is: ' ||
                  char(timeofday) || '.';

END
```

Minor changes are necessary to convert the preceding Oracle Perl application
(Example 7-15 on page 312) to use DB2. The steps (besides entering the correct
values for user and password) are:

► Observing the syntax difference in the parameters for the *connect* method,
  and making the necessary changes.

► Observing the syntax differences for calling stored procedures, and making
  the necessary changes.

DB2 Connect method syntax

The syntax for a database connection to DB2 is shown in Example 7-17.

Example 7-17   Generic syntax for a DB2 connection string in a Perl application

```
$dbhandle = DBI->connect('dbi:DB2:dbalias', $userID, $password)
```

The parameters of this connection are as follows:

► `dbhandle` - This represents the database handle returned by the connect
  statement.

► `dbalias` - This represents a DB2 alias cataloged in the DB2 database
  directory.

  Oracle requires the `sid` for the database in the place where DB2 would
  require `dbalias`; the Oracle syntax can be summarized as `dbi:Oracle:sid`. In
  our example this is coded as `dbi:Oracle:xp10g`.

► `userID` - This represents the user ID used to connect to the database.

► `password` - This represents the password for the user ID that is used to
  connect to the database.

Syntax for calling a DB2 stored procedures

In Oracle, a stored procedure is called from an *anonymous block*, that is,
BEGIN...END; within a PREPARE statement. The input and output parameters
of the Oracle stored procedure are defined as host variables, for example,
::name, :message. Example 7-18 demonstrates these points.

Example 7-18   Calling a stored procedure in an Oracle Perl program

```
$sth = $dbh->prepare(q{
   BEGIN
      Greeting(:name, :message);
   END;
});
```

In contrast, a DB2 stored procedure is executed by issuing a CALL statement
from within a PREPARE statement. Also, the stored procedure input and output
parameters are designated as parameter markers (?, ?). This is shown in
Example 7-19.

Example 7-19   Calling a stored procedure in a DB2 Perl program

```
$sth = $dbh->prepare(q{
      CALL Greeting(?,?);
});
```

The complete Perl program, converted to DB2, is shown in Example 7-20.

Example 7-20   DB2 Perl program db2CallGreeting.pl

```
#!/usr/bin/perl
use DBI;

$database='dbi:DB2:sample';
$user='db2inst1';
$password='db2inst1';

$dbh = DBI->connect($database, $user, $password) or die "Can't connect to
$database: $DBI::errstr";

print "  Connected to database.\n";

$name = 'Ariel';
$message;

$sth = $dbh->prepare(q{
      CALL Greeting(?,?);
   });
```

```
$sth->bind_param(1,$name);

$sth->bind_param_inout(2, \$message, 100);

$sth->execute;

 print "$message", "\n";

 # check for problems...
warn $DBI::errstr if $DBI::err;

$sth-> finish;

$dbh->disconnect;
```

The results of executing db2CallGreeting.pl are shown in Figure 7-5.

Figure 7-5   The results of executing the db2CallGreeting.pl program

7.3.6 Converting PHP applications

Oracle supports access to Oracle databases in a PHP application through two extensions:

► OCI8

   The functions in this extension allow access to Oracle 10, Oracle 9, Oracle 8 and Oracle 7 databases using the Oracle Call Interface (OCI). They support binding of PHP variables to Oracle placeholders, have full LOB, FILE and ROWID support, and allow you to use user-supplied define variables. This is the *preferred* extension for PHP connections to an Oracle database.

- PDO_OCI

    This driver implements the PHP Data Objects (PDO) interface to enable access from PHP to Oracle databases through the OCI library.

Information regarding the PHP extensions that are available for DB2 were discussed in the PHP section 7.1.2, "Driver support" on page 282.

In order to demonstrate some differences between PHP programming in Oracle and DB2, we show a sample program that demonstrates the following:

- Connecting to a database through PHP
- Calling a stored procedure with an input and an output parameter
- Returning an output parameter

The same stored procedure, Greeting, that was shown in 7.3.5, "Converting Perl applications" on page 312, is used in the examples in this section.

Connecting to Oracle using PHP (OCI8)

Example 7-21 is the Oracle PHP program oraGreeting.php. This program connects to the Oracle database using the OCI8 extension, binds the input and output parameters, executes the call to the Greeting stored procedure, and returns the output parameter.

Example 7-21   Oracle PHP program oraGreeting.php

```php
<?php
$conn = oci_connect('sample','sample') or die;

$sql = 'BEGIN Greeting(:name, :message); END;';

$stmt = oci_parse($conn,$sql);

//  Bind the input parameter
oci_bind_by_name($stmt,':name',$name,32);

// Bind the output parameter
oci_bind_by_name($stmt,':message',$message,100);

// Assign a value to the input
$name = 'Ariel';

oci_execute($stmt);

// $message is now populated with the output value
print "$message\n";
?>
```

The result of executing oraGreeting.php is shown in Figure 7-6.
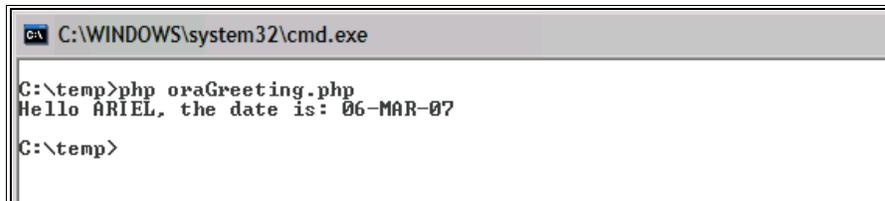
Figure 7-6   The result of executing oraGreeting.php

Connecting PHP applications to DB2

Two extensions, ibm_db2 and PDO_ODBC, can be used to access DB2 databases from a PHP application. See "PHP extensions" on page 284 for details. For the DB2 conversion of the preceding Oracle PHP program, Example 7-21 on page 317, the ibm_db2 extension is used. Example 7-22 shows the source code for this converted program.

Example 7-22   DB2 PHP program db2Greeting.php

```php
<?php
$database = 'sample';
$user = 'db2inst1';
$password = 'db2inst1';
// Next parameters used when making an uncataloged connection
// $hostname = 'localhost';
// $port = 50000;

$conn = db2_connect($database, $user, $password) or die;

// use this connection string for uncataloged connections:
//$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;
//HOSTNAME=$hostname;PORT=$port;PROTOCOL=TCPIP;UID=$user;PWD=$passwo//rd;";

// $conn = db2_connect($conn_string, '', '');

if ($conn) {
 //echo "Connection succeeded.<br />\n";

 $sql = 'CALL Greeting(?, ?)';
 $stmt = db2_prepare($conn, $sql);

 $name = 'Ariel';
 $message = '';

 db2_bind_param($stmt, 1, "name", DB2_PARAM_IN);
 db2_bind_param($stmt, 2, "message", DB2_PARAM_OUT);
```

```
  db2_execute($stmt);

 // $message is now populated with the output value
 print "$message\n";
}

?>
```

As can be seen in this example, there are few changes that need to completed
before the application may use DB2. These can be summarized by observing the
differences between the following functions:

▶ oci_connect and db2_connect
▶ oci_bind_by_name and db2_bind_param
▶ oci_parse and db2_prepare

oci_connect and db2_connect

oci_connect takes the following required and optional parameters (in [ ]):

```
(string $username, string $password [, string $db [, string $charset [, int
$session_mode]]] )
```

> **Note:** Database ($db) is an *optional* parameter. If the database is *not*
> specified, PHP uses the environment ORACLE_SID and TWO_TASK to
> determine the name of the local Oracle instance and the location of
> tnsnames.ora.

oci_connect will be converted to the ibm_db2 function db2_connect, which takes
the following required and optional parameters (in [ ]):

```
(string database, string username, string password, [array options])
```

When connecting to DB2 through PHP, the connection may be made through
either a cataloged or an uncataloged database.

> **Note:** Refer to the DB2 manual *Administration Guide: Implementation*,
> SC10-4221, for detailed information regarding cataloging a DB2 database.

For an uncataloged connection to a database, the *database* parameter
represents a complete connection string in the format shown in Example 7-23.

Example 7-23   Connection string for an uncataloged DB2 database

```
DRIVER={IBM DB2 ODBC DRIVER};DATABASE=database;HOSTNAME=hostname;
PORT=port;PROTOCOL=TCPIP;UID=username;PWD=password
```

This is demonstrated by the code shown in Example 7-24.

Example 7-24   Connection string for uncataloged database used in the example

```
$database = 'sample';
$user = 'db2inst1';
$password = 'db2inst1';
$hostname = 'localhost';
$port = 50000;

$conn_string = "DRIVER={IBM DB2 ODBC DRIVER};DATABASE=$database;
HOSTNAME=$hostname;PORT=$port;PROTOCOL=TCPIP;UID=$user;PWD=$password;";

$conn = db2_connect($conn_string, '', '');
```

oci_bind_by_name and db2_bind_param

oci_bind_by_name takes the following required and optional parameters (in [ ]):

```
( resource $statement, string $ph_name, mixed &$variable [, int $maxlength [,
int $type]] )
```

oci_bind_by_name will be converted to the ibm_db2 function *db2_bind_param* which accepts the following required and optional [ ] parameters:

```
(resource stmt, int parameter-number, string variable-name, [int
parameter-type, [int data-type, [int precision, [int scale]]]])
```

The parameter information is as follows:

▶ `stmt` - A prepared statement returned from db2_prepare().

▶ `parameter-number` - Specifies the 1-indexed position of the parameter in the prepared statement.

▶ `variable-name` - A string specifying the name of the PHP variable to bind to the parameter specified by parameter-number.

▶ `parameter-type` - A constant specifying whether the PHP variable should be bound to the SQL parameter as an input parameter (DB2_PARAM_IN), an output parameter (DB2_PARAM_OUT), or as a parameter that accepts input and returns output (DB2_PARAM_INOUT). This parameter is optional.

▶ `data-type` - A constant specifying the SQL data type that the PHP variable should be bound as: one of DB2_BINARY, DB2_CHAR, DB2_DOUBLE, or DB2_LONG. This parameter is optional.

▶ `precision` - Specifies the precision with which the variable should be bound to the database. This parameter is optional.

▶ `scale` - Specifies the scale with which the variable should be bound to the database. This parameter is optional.

With this information, binding the stored procedure input and output parameters is converted as shown in Example 7-25.

Example 7-25   Converting oci_bind_by_name to db2_bind_param

ORACLE:
```
//  Bind the input parameter
oci_bind_by_name($stmt,':name',$name,32);

// Bind the output parameter
oci_bind_by_name($stmt,':message',$message,100);
```

DB2 conversion:
```
//  Bind the input parameter
db2_bind_param($stmt, 1, "name", DB2_PARAM_IN);

// Bind the output parameter
db2_bind_param($stmt, 2, "message", DB2_PARAM_OUT);
```

---

oci_parse and db2_prepare

The oci_parse function prepares a query. The function accepts the following parameters:

```
( resource $connection, string $query )
```

oci_parse will be converted to db2_prepare, which accepts the following required and optional [ ] parameters:

```
(resource connection, string statement, [array options])
```

The parameters are defined as follows:

- ► `connection` - A valid database connection resource variable as returned from db2_connect() or db2_pconnect().
- ► `statement` - An SQL statement, optionally containing one or more parameter markers.
- ► `options` - [optional] An associative array containing statement options. You can use this parameter to request a scrollable cursor on database management systems that support this functionality.

With this information, calling a DB2 stored procedure is converted as shown in Example 7-26.

Example 7-26   Converting oci_parse to db2_prepare

Oracle:

```
$sql = 'BEGIN Greeting(:name, :message); END;';
```

```
$stmt = oci_parse($conn,$sql);
```

DB2 conversion:

```
$sql = 'CALL Greeting(?, ?)';
$stmt = db2_prepare($conn, $sql);
```

After the changes described above have been implemented, the application is fully converted to DB2. The result of executing this program is shown in Example 7-7.
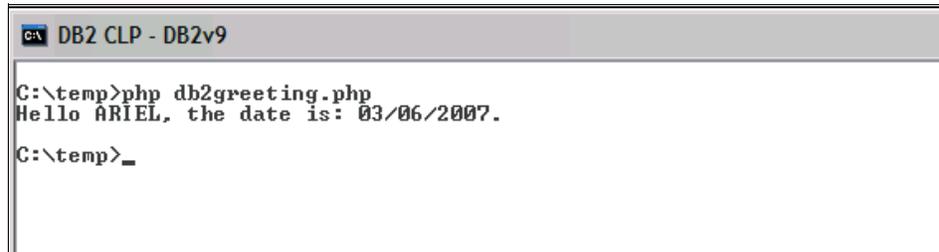
```
▣ DB2 CLP - DB2v9

C:\temp>php db2greeting.php
Hello ARIEL, the date is: 03/06/2007.

C:\temp>_
```

Figure 7-7   The result of executing db2Greeting.php

> **Note:** For complete information regarding PHP, consult *Developing Perl and PHP Applications*, SC10-4234.

PHPEclipse and the Developer Workbench

Developer Workbench is built on the Eclipse framework. This framework allows the installation of IDE plug-ins that are created to support various application development APIs. One such plug-in, PHPEclipse, is available for PHP. To acquire the PHPEclipse plug-in, complete the following steps:

1. Open the Eclipse IDE on your development desktop.

2. From the file menu in Eclipse, select **Help** → **Software Updates** → **Find/Install**.

3. Select **Search for new features to install**.

4. Click **New Remote Site**.

5. Enter `PHP SourceForge` as the name, and enter the URL as:

   `http://phpeclipse.sourceforge.net/update/releases`

6. Click **OK** → **Finish**.

6. A list of features will be presented. Open the list and check the one labeled phpeclipse. Click **Next**.

7. Follow the on-screen instructions to finish the automatic installation.

After restarting Eclipse, switch to the IDE perspective specific to PHP:

1. Under the Window menu, choose **Open Perspective**.

2. Select **Other**.

3. Select **PHP** and click **OK**.

Example 7-8 shows the PHPEclipse IDE in Developer Workbench.
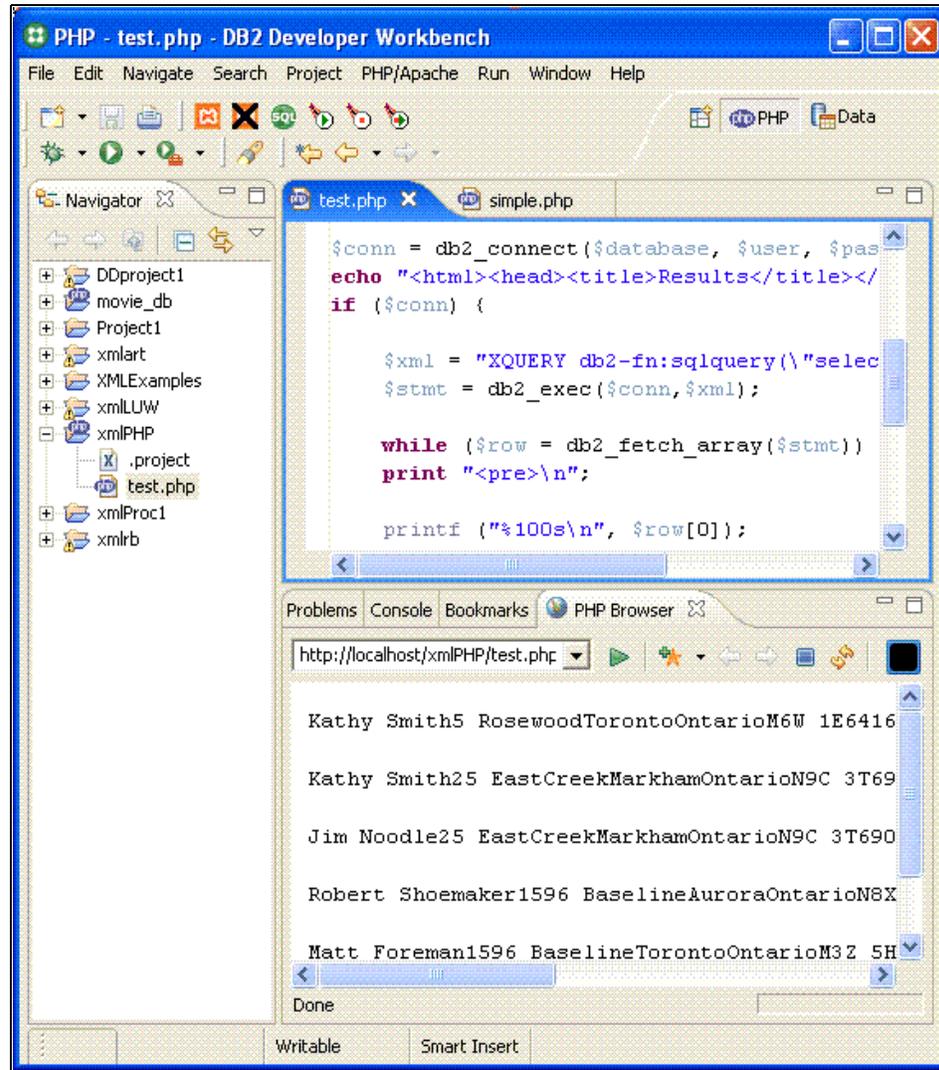
Figure 7-8   PHPEclipse IDE in Developer Workbench

7.3.7 Converting .NET applications

The supported operating systems for developing and deploying .NET Framework 1.1 applications are:

- ► Windows 2000
- ► Windows XP (32-bit edition)
- ► Windows Server® 2003 (32-bit edition)

The supported operating systems for developing and deploying .NET Framework 2.0 applications are:

- ► Windows 2000, Service Pack 3
- ► Windows XP, Service Pack 2 (32-bit and 64-bit editions)
- ► Windows Server 2003 (32-bit and 64-bit editions)

Supported development software for .NET Framework applications

In addition to a DB2 client, you need one of the following options to develop .NET Framework applications:

- ► Visual Studio® 2003 (for .NET Framework 1.1 applications)

- ► Visual Studio 2005 (for .NET Framework 2.0 applications)

- ► .NET Framework 1.1 Software Development Kit and .NET Framework Version 1.1 Redistributable Package (for .NET Framework 1.1 applications)

- ► .NET Framework 2.0 Software Development Kit and .NET Framework Version 2.0 Redistributable Package (for .NET Framework 2.0 applications)

In addition to a DB2 client, the following two options are needed to deploy .NET Framework applications:

- ► .NET Framework Version 1.1 Redistributable Package (for .NET Framework 1.1 applications)

- ► .NET Framework Version 2.0 Redistributable Package (for .NET Framework 2.0 applications)

.NET Data Providers

DB2 for Linux, AIX, and Windows includes three .NET Data Providers:

- ► DB2 .NET Data Provider

   A high performance, managed ADO.NET Data Provider. This is the *recommended* .NET Data Provider for use with DB2 family databases. ADO.NET database access using the DB2 .NET Data Provider has fewer restrictions, and provides significantly better performance than the OLE DB and ODBC .NET bridge providers.

► OLE DB .NET Data Provider

A bridge provider that feeds ADO.NET requests to the IBM OLE DB provider (by way of the COM interop module). This .NET Data Provider is *not recommended* for access to DB2 family databases. The DB2 .NET Data Provider is faster and more feature-rich.

► ODBC .NET Data Provider

A bridge provider that feeds ADO.NET requests to the IBM ODBC driver. This .NET Data Provider is *not recommended* for access to DB2 family databases. The DB2 .NET Data Provider is faster and more feature-rich

In addition to the DB2 .NET Data Provider, IBM also provides a collection of add-ins to the Microsoft Visual Studio .NET IDE. The add-ins simplify the creation of DB2 applications that use the ADO.NET interface. The add-ins can also be used to develop server-side objects, such as SQL stored procedures and user-defined functions. The DB2 Visual Studio add-ins can be obtained at the following Web site:

http://www-306.ibm.com/software/data/db2/windows/dotnet.html

The IBM.Data.DB2 namespace contains the DB2 .NET Data Provider. To use the DB2 .NET Data Provider, you must add the `Imports` or `using` statement for the IBM.Data.DB2 namespace to your application .DLL, as shown in Example 7-27.

Example 7-27   Examples of the required Imports or using statement

```
[Visual Basic]
Imports IBM.Data.DB2

[C#]
using IBM.Data.DB2;
```

Also, references to IBM.Data.db2.dll and IBM.Data.DB2.Server.dll must be added to the project.

VB .NET conversion example

In general, converting a .NET application from Oracle to DB2 is quite simple. In most cases it will entail *replacing* the classes that are available in the Oracle .NET Data Provider with functionally equivalent classes that are available in the DB2 .NET Data Provider, for example, OracleConnection with DB2Connection or OracleCommand with DB2Command, and so on.

In this section, we demonstrate this point using a simple VB .NET application that connects to a database, executes a SELECT, and returns a result set. This example is demonstrated in Oracle and then converted to DB2. In the DB2

example, the changes that are necessary for converting from Oracle to DB2 are outlined.

Components of the GUI for the conversion example

The GUI, used for both examples, consists of several CONTROLS:

- ► A RUN QUERY button - When this button is clicked, the code in the Click event will:
  - Connect to the database
  - Execute the query
- ► A Query Results Text box (for aesthetic purposes only)
- ► A List Box - Once the query has been executed, the results are displayed in this list box.
- ► A Qui**t** button - Clicking this button ends the application.

Example 7-9 shows the GUI used to demonstrate the VB .NET application conversion example.
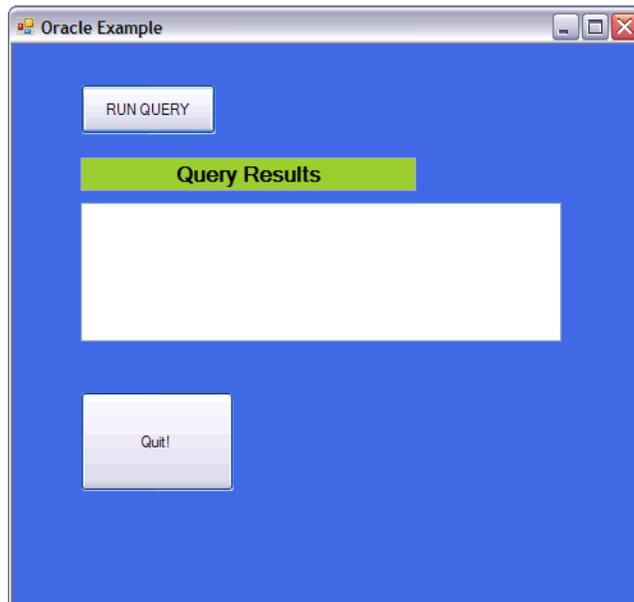
Figure 7-9   GUI for the VB .NET application conversion example

The essential components of this application are contained within the Click Event for the RUN QUERY control button. Example 7-28 shows the code in the Button1_Click event as it might appear in an Oracle application.

Example 7-28   The Button1_Click event

```
Imports Oracle.DataAccess.Client ' ODP.NET Oracle managed provider [1]
_____
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

Dim oradb As String = "Data Source=(DESCRIPTION=(ADDRESS_LIST=" _ +
"(ADDRESS=(PROTOCOL=TCP)(HOST=9.10.11.12)(PORT=1521)))" _ +
"(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=ora10g)));" _ + "User
Id=ora_usr;Password=ora_usr;"                                    [2]

Dim conn As New OracleConnection(oradb)                          [3]
conn.Open()

Dim cmd As New OracleCommand                                     [4]
cmd.Connection = conn
cmd.CommandText = "select first_name, last_name from employees
                         where dept_code = 'IT'"

cmd.CommandType = CommandType.Text

Dim dr As OracleDataReader = cmd.ExecuteReader()                 [5]

        While dr.Read()
            ListBox1.Items.Add("The name of this employee is: " +
dr.Item("first_name") + dr.Item("last_name"))                    [6]
        End While

        conn.Dispose()

    End Sub
```

Notes:

[1] IMPORT Oracle.DataAccess.Client is added to the application .DLL.

[2] A String, OraDb, is declared as the connection string for the Oracle database.

[3] A connection (conn) is defined as an OracleConnection

[4] A command (cmd) is defined as an OracleCommand and populated with the text of the query.

[5] A DataReader (dr) is defined as an OracleDataReader and the query is executed.

[6] The List Box is populated with the results of the query.

When the application executes, clicking **RUN QUERY** yields the results that are shown in Figure 7-10.
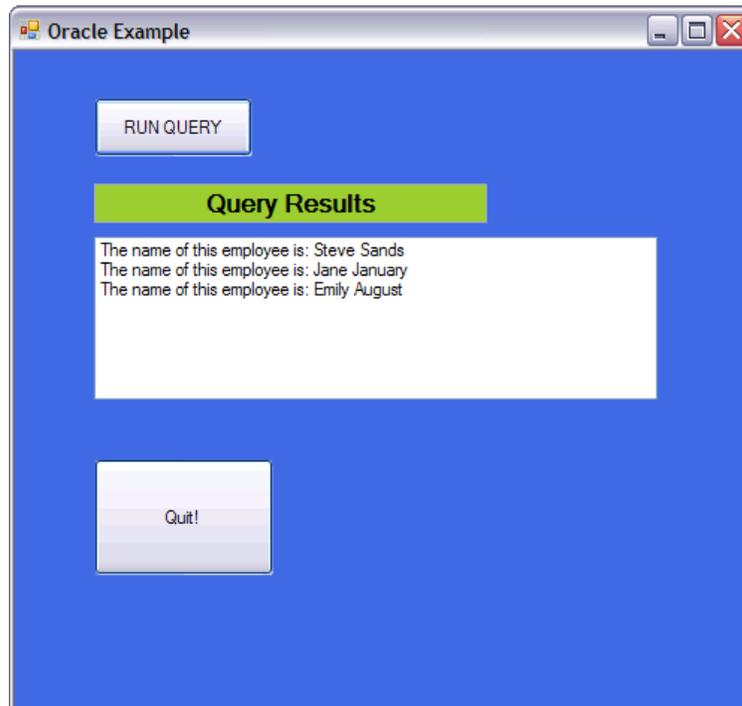
Figure 7-10   The results of the Oracle Example are displayed in the List Box

DB2 Example (conversion)

The GUI for the DB2 conversion is shown in Figure 7-9 on page 326.

Since the essential components of this application are contained within the Click Event for the RUN QUERY control button, the focus of the conversion centers on this control. Example 7-29 shows the code in the Button1_Click event as it will appear after conversion to DB2. Some explanations of the changes are documented in the *Notes* that appear after the code example.

Example 7-29   The code in the Button1_Click event after conversion

```
Imports IBM.Data.DB2                                              [1]
_____

Public Class Form1
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim db2db As String =                                          [2]
"Server=localhost:50000;Database=testdb;UID=db2inst1;PWD=db2inst1"

Dim conn As New DB2Connection(db2db)                           [3]

conn.Open()

Dim cmd As New DB2Command                                      [4]
cmd.Connection = conn

cmd.CommandText = "select first_name, last_name from employees where dept_code
= 'IT'"

cmd.CommandType = CommandType.Text

Dim dr As DB2DataReader = cmd.ExecuteReader()                  [5]
        While dr.Read()

    ListBox1.Items.Add("The name of this employee is: " +
                        dr.Item("first_name") + dr.Item("last_name"))

        End While                                             [6]

conn.Dispose()
    End Sub
```

Notes:

[1] To use the DB2 .NET Data Provider, you must add the `Imports` (VB) or `using` (C#) statement for the IBM.Data.DB2 namespace to your application .DLL.

[2] A String, `db2db`, is declared and populated as the connection string for the DB2 database (converted from OraDb).

[3] A connection (`conn`) is defined as DB2Connection (converted from OracleConnection).

[4] A command (cmd) is defined as DB2Command. (converted from OracleCommand).

[5] A DataReader (dr) is declared as a DB2DataReader (converted from OracleDataReader).

[6] The List Box is populated with the results of the query.

Once the changes are effected, clicking **RUN QUERY** yields the results shown in Example 7-11.
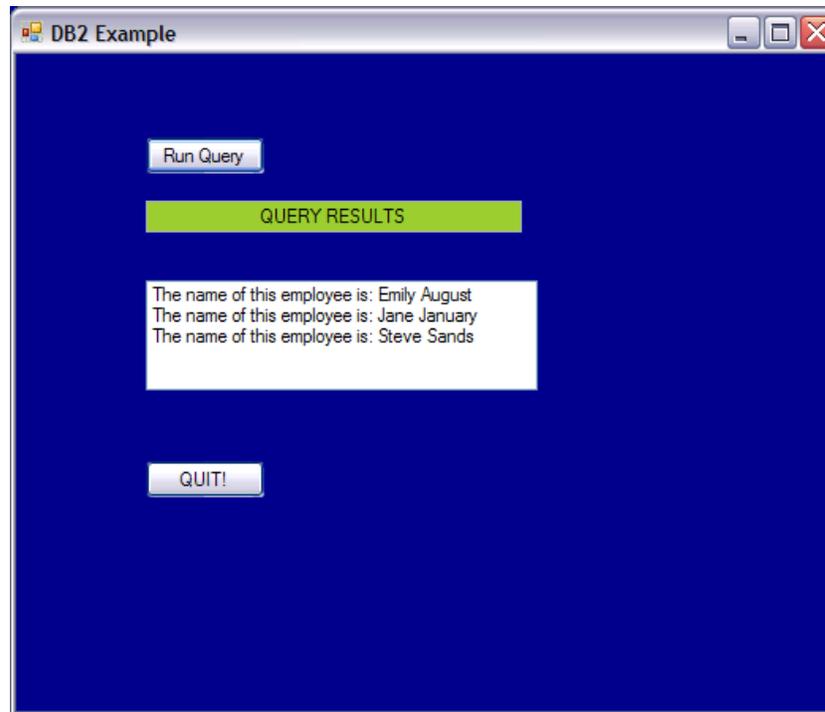
Figure 7-11   The results of executing the DB2 Example application

Note: For complete information about the DB2 .NET provider, consult the information at the following URL:

http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.dndp.doc/htm/frlrfIBMDataDB2.htm

For in-depth information on .NET programming, refer to *Developing ADO.NET and OLE DB Applications*, SC10-4230.

7.4 Package applications migration planning

For package applications, the vendor delivers the application and database based on DB2. The migration is limited to:

► Checking software and hardware availability and compatibility
► Education of developers and administrators

- Analyzing of customized changes in the application and database
- Setting up the target environment
- Changing of customized items
- Testing of data migration and customized items
- Roll-out

To keep the support from the vendor, you have to meet the prescribed migration plan and process. We now show the migration approach for SAP, Siebel®, and PeopleSoft environments.

7.4.1 SAP

An SAP system is divided into layers. The application and business logic is independent of the database. SAP uses only common database types and functionality. However, in planning for the migration efforts, you have to take into account your self-made customizations.

Migration requirements

For database migration, SAP requires you to use the SAP Migration Service in order for the system to be supported after the migration is complete. The SAP Migration Service is fee-based and will provide you with the following:

- Cross-check of the database migration project plan
- Migration tools
- Migration keys
- Remote Going Live Migration check
- Support in case of migration tool problems

The SAP R/3® software for your new IBM DB2 system is delivered by SAP after the necessary contractual or licensing changes.

Migrating an SAP R/3 system to a different operating system or database needs to be planned and approved by SAP, and therefore SAP requires that you use SAP migration tools.

To perform a migration you need a migration key. This key will be provided to you by SAP. The key is mandatory for the data export step and data import using SAP's tools.

The migration project plan

To successfully perform a migration, you need to follow a well-defined process using the steps depicted in Figure 7-12. Therefore, SAP demands a project plan for your migration project. This plan is devised by you and the migration partner. SAP Migration Service will check this plan to ensure that your migration will be successful.

Sample project plans can be found at:

http://service.sap.com/osdbmigration

The migration project

Figure 7-12 shows an overview of an SAP R/3 database migration project.
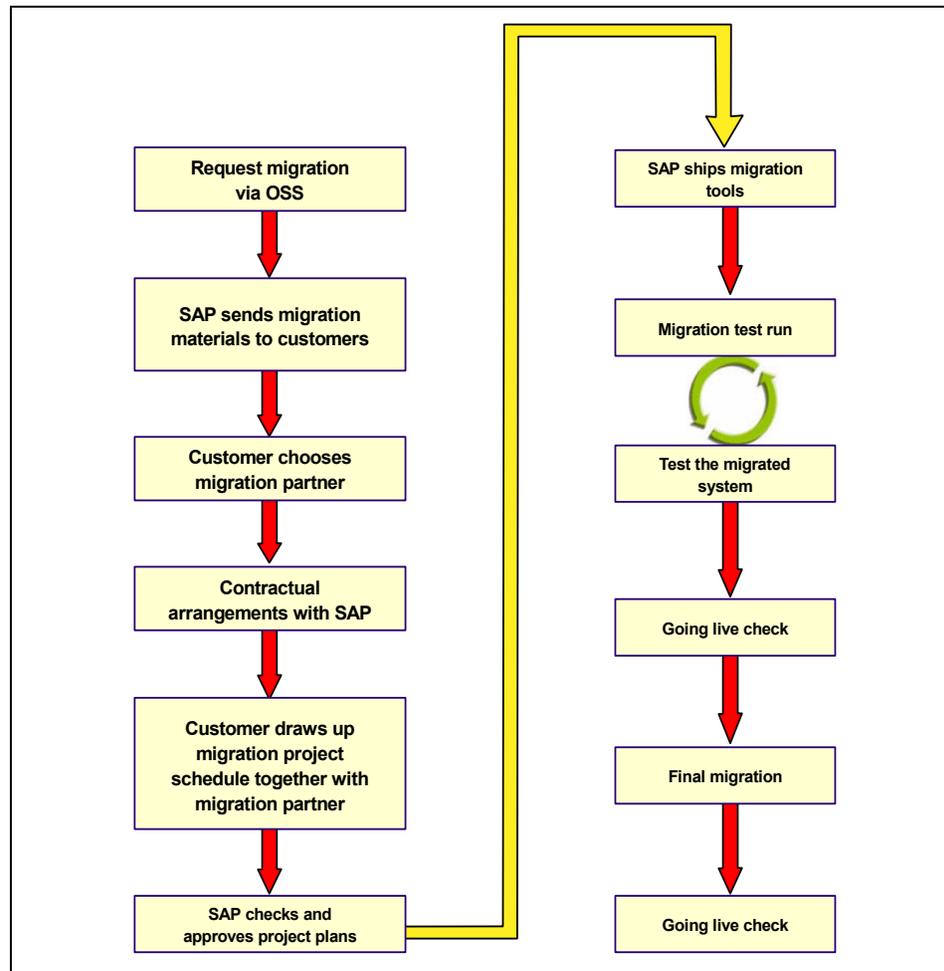
Figure 7-12   SAP R/3 database migration project

Because database migration comprises several tasks, we recommend that you begin planning three to four months in advance.

Figure 7-13 provides a typical time line for planning a migration.
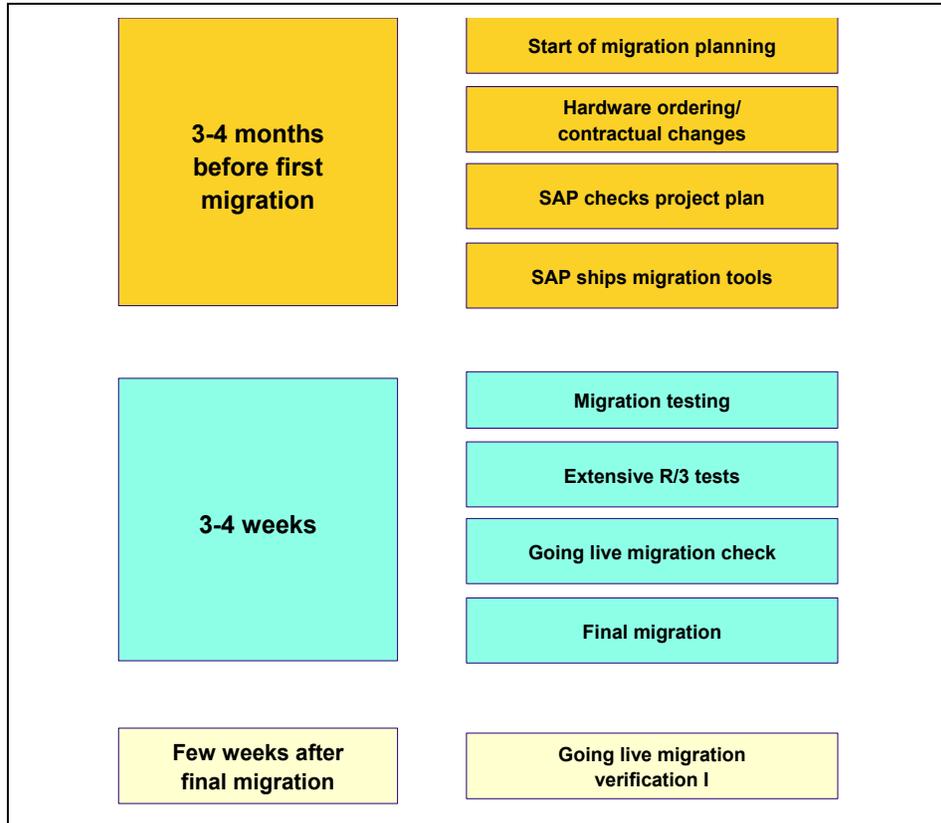
Figure 7-13   SAP R/3 database migration timeline

Migration test and check

Before the final migration, SAP requires at least one migration test run. This will ensure that all your SAP R/3 data and functionality are moved correctly to the target system.

After the first functional test of the migrated system, end users should be involved in the test process as well. This ensures that the target system is tested comprehensively.

The *Going live migration check* is part of the migration services from SAP. It should be scheduled after the test migration, and three to four weeks before the final migration.

Final migration and verification

This step is performed after a successful test of the target system and SAP's *Going live migration check*.

The final migration is usually scheduled for a weekend because downtime is required while the switch to a new platform occurs. The steps are the same as with the test run, but, in addition, you should do a full backup afterwards.

The *Going live migration verification* is again a part of the migration services from SAP. The migration verification step is done twice. The first time should be scheduled one to two weeks after the final migration; the second time it should be scheduled six to eight weeks after the final migration. This should ensure that all daily, monthly, and other reports and functions are operating correctly.

8

XML conversion

DB2 9 introduces a new generation of data server with its revolutionized support for XML data alongside relational data. No longer is it necessary to store XML data as a CLOB or shred to relational tables. The new XML data type stores XML data in its natural hierarchy and supports both XQuery and SQL to query the data.

In this chapter, we describe how to enable your Oracle applications to take advantage of the new DB2 9 XML data type.

This chapter describes the following:

- ► The new DB2 XML data type
- ► Converting an Oracle XML data model to DB2
- ► Moving XML data from Oracle to DB2
- ► Converting Oracle queries on XML data to DB2
- ► Converting Oracle indexes on XML data to DB2
- ► Converting Oracle XML data accessing in Java application to DB2
- ► Converting Oracle stored procedures using XML to DB2
- ► Tools and utilities for working with XML
- ► Best practices

8.1 DB2 XML data type introduction

DB2 9 introduces a new XML data store that enables well-formed XML documents to be stored in their native hierarchical form. The XML data type is used to define columns of a table that will store the XML data, allowing for XML data to be stored alongside relational data. This approach is significantly different from storing XML data as text in a CLOB field or mapping (shredding) to different relational tables.

XML support is fully integrated into DB2 9, supporting efficient search and retrieval of XML through XQuery, SQL, or SQL/XML. SQL functions that enable XML data to be constructed or published from values obtained from the database, including relational data.

DB2 9 also introduces XML decomposition, where XML data can be mapped to target relational table-column pairs. Though in this case, if the target column is not the XML data type, the data will be considered traditional relational data and lose its native hierarchical format.

For information about performance differences between the XML storage options in DB2 9, see the following article:

"A performance comparison of DB2 9 pureXML and CLOB or shredded XML storage" at:

`http://www.ibm.com/developerworks/db2/library/techarticle/dm-0612nicola`

The use of the DB2 9 XML data type and related native data store support is available as a separate licensing feature of DB2 9. For details regarding this feature, see:

`http://www.ibm.com/software/data/db2/9/editions_features_purexml.html`

8.1.1 DB2 pureXML native storage

Supporting DB2 9 pureXML implementation is the new XML data type. This data type is used to define columns of a table that will store well-formed XML documents. The XML data type can be used alongside the traditional relational data types, such as INTEGER and VARCHAR, within the same table.

Using the XML data type allows XML documents to be stored in a true hierarchical structure, giving DB2 a full understanding of the internal structure of the XML document, a manner that is optimal for querying and leveraging the flexibility inherent in XML itself.

In DB2 9.1, all XML data is stored in UTF-8, which currently implies that in order to use the XML data type, the database must be created as a Unicode database. A Unicode database can be created by issuing the following statement:

```
create database dbname using codeset utf-8 territory us
```

A table using the XML data type could then be created in the database with the CREATE TABLE command. The following command creates a CUSTOMER table with one INTEGER column and two XML data type columns:

```
create table customer(cid bigint not null primary key,
                       info xml,
                       history xml)
```

Data stored in an XML column is not stored as a CLOB, BLOB, or any other varying-length or fixed-length character format and is not comparable to a string value unless it is transformed to a string before the comparison. Instead, XML data is stored in its native, preparsed, hierarchical node structure.

Native XML values can be explicitly transformed into a string value using the XMLSERIALIZE function. Conversely, a string value representing XML data can be transformed into a native XML value using the XMLPARSE function. XML values are implicitly parsed or serialized when exchanged with application string and binary data types.

There is no size restriction on the XML document stored in an XML column; however, there are string size restrictions. XML data that is transformed to a string cannot exceed the maximum string size of 2 GB.

XML documents can be inserted, updated, and deleted using SQL. Validation of an XML document against an XML schema is typically performed during IMPORT, INSERT, or UPDATE and is done using one of the schemas stored in the XML Schema Repository (XSR), which is integrated into DB2. XML documents are annotated with type information after having been validated against an XML schema. XML documents stored in a single XML column can belong to the same or different XML schemas—there is no restriction.

There are two internal system-generated indexes associated with XML columns that are automatically created by DB2, namely an XML column path index and an XML regions index.

An XML column path index is automatically created by DB2 for each XML column in a table. The XML column path index records all unique paths that exist within XML documents stored in the associated XML column. A single XML regions index is created for all the XML columns in a table. The XML regions index captures how an XML document is divided up internally into regions, which

are sets of nodes within a page. Both the XML column path and the XML regions indexes are recorded in the SYSCAT.INDEXES view.

These internal indexes associated with XML columns are separate and apart from the indexes created over XML columns for application-specific performance improvements. These types of indexes are added or removed using the CREATE INDEX and DROP INDEX statements.

8.1.2 DB2 decomposition

Decomposition, sometimes referred to as "shredding", is the process of storing content from an XML document into columns of relational tables. This storage approach is significantly different from storing the complete XML document in a single XML column, as discussed above.

Decomposition is primarily designed for users who have an existing relational database (and applications running on it) and who would now like the ability to consume data from XML documents into a relational structure. There is no restriction on using both the pureXML and decomposition storage approaches in the same database. Additionally, both storage techniques can even be used on the same XML document.

Decomposition is accomplished by annotating an associated XML schema, using annotations to instruct how decomposition is to occur. XML documents can be wholly or partially decomposed into relational tables. These annotations are very flexible in their mapping of XML schema structure to relational table structure, and specify details such as the target table and column name that the XML data is to be stored in, as well as any transformation of the content before it is stored.

Decomposition can be specified for new or existing XML schemas, can occur on new or existing tables, and can decompose different elements or attributes (even from different XML schemas) to the same tables or rows. Additionally, previous decomposition annotations can be modified without affecting previously decomposed XML data.

An XML schema is used for mapping information because it is an open standard and requires fewer additional proprietary features in the mapping language specification. Since annotations added to the XML schema do not participate in the validation of the corresponding XML documents, the same XML schema can be used for the mapping and validation of XML documents. Validation can be performed on the entire XML document before decomposition, or it can be limited to the decomposed elements or attributes only. Additionally, DB2 supports multiple XML schemas in the same database, table, and even column, so changes will not affect the existing database design.

SQL is primarily used to query the XML data that has been decomposed to non-XML data types, since the data ceases to be XML unless stored in an XML data type. XQuery is natively supported (not translated) and processed for XML data stored in XML data type columns. However, XQuery can still be performed against relational data by using the XMLTABLE function, which turns non-XML data into XML.

Decomposition has certain advantages:

► Excellent fit into existing relational environments.
► Optimal approach for tabular data that is not required to be retained as XML.
► Easy data updates.
► Fast SQL searches of data.
► Works well with existing reporting tools.

At the same time, decomposition has certain disadvantages:

► Mapping can be complex and must be predefined for every XML document that is to be stored.

► After the XML document is decomposed, it ceases to be XML data, loses any digital signature, and can become difficult and expensive to reconstruct.

► Parent and child relationships inherent in an XML structure may require generation of values to represent foreign key values for those relationships.

► Mapping typically applies only to a single schema of a document. If the underlying relational schema changes or the XML schema changes, the mapping might cease to be valid or it might require a complex change process.

8.2 Converting the XML data model

Although native XML technology is supported by both DB2 and Oracle databases, each database uses a different model to store XML data. As a result, the term "native XML" has a different meaning for each database.

In most cases, when converting from either unstructured or structured storage in Oracle to DB2, you will choose to use the XML data type to store your XML documents. DB2 decomposition is also discussed for applications which also shred XML data into relational tables.

Oracle structured storage requires that an XML schema be used. It is not necessary to use an XML schema for storage with DB2 pureXML. If you wish to use an XML schema with DB2, for validation purposes, in the section "Obtaining and converting XML schemas" on page 343 and "XML schema registration in DB2" on page 346, we look at how to convert and register XML schemas in DB2.

Conversion from both Oracle unstructured and structured storage models to DB2 XML data type is also covered. Finally, this section ends with a discussion on DB2 decomposition.

8.2.1 XML data type differences

In DB2, the native XML data store enables collections of well-formed XML documents to be stored in their hierarchical form within one or more columns of a table. By "well-formed", it is meant that the XML document must conform to syntactical rules as specified by the World Wide Web Consortium (W3C) standard. Collections of XML documents that contain different structures, have different schemas, or have no schema at all, can all be stored in the same column.

XML columns are defined using the XML data type. XML data is not stored as a text string or shredded to a relational data model. Rather, they are stored in a hierarchical structure of nodes in DOM-like representation. The XML data type can also be used as a data type for host variables in languages such as C, Java, and COBOL and can also be used as a parameter and local variable within DB2 SQL procedures, and external procedures written in C or Java.

DB2 stores and manipulates XML data into a parsed format that reflects the hierarchical nature of the original XML document. Thus, when querying an XML document, the internal storage model allows direct access to any portion of the document without requiring a read of the entire document into memory. Indexing is not based on byte offsets into a document. Therefore, index entries are only changed for relocated nodes and not for all nodes.

Since the unit of storage is a node, a node exists on a page along with other nodes either from the same or different documents. Each node is linked to its parent or children. Navigating nodes amounts to pointer transversal. Nodes can be relocated to other pages without rewriting the entire document. The association of schemas to document is per document and not per column.

If an XML schema is registered with DB2 and used to validate data that is inserted into a column, DB2 annotates all nodes in the XML hierarchy with information about the schema types. Otherwise, if no XML schema exists, DB2 annotates the nodes with default type information. Upon storage, DB2 will preserve the internal structure of the document by converting its tag names and other information into INTEGER values as an internal representation. The internal representation is not something that users need to be aware of when accessing the data. The mapping of tags to INTEGER values is maintained in a new catalog table named SYSXMLSTRINGS. Replacing tags with string IDs reduces space consumption and allows for higher performance of navigational queries. DB2 automatically splits nodes of a document across multiple data

pages as needed. To manage this, DB2 automatically generates and maintains a "regions" system index to provide an efficient means of tracking the physical representation of the entire document.

The native XML data store is fully integrated into the DB2 database system and can be accessed and managed by leveraging DB2 functionality. XQuery, SQL or SQL/XML can be used to query XML data alone or XML data together with relational data. The first version of the XQuery standard focuses on only database read activities. At the present, there is no standard within XQuery for write activities and as a result, SQL is used to either insert or import XML into DB2 at this time.

The Oracle XMLType data type is used for storing well-formed XML documents. XMLType is an abstraction layer that represents two types of storage models, structured data and unstructured data. Structured storage allows for five different ways of dealing with repeating elements, which are called "collections" in Oracle. These five options allow repeating elements to be stored as a collection of CLOBs, LOBs, nested tables, separate XMLTypes, and XMLTypes in a nested table. All of these options can be migrated to a single DB2 pureXML column, with DB2 supporting equivalent XML usage patterns to that of Oracle.

XML data using structured storage is parsed, shredded and stored as relational data. These operations are performed transparently to the user although storing these types of documents is dependent on and is defined by a schema. In fact, because the XML is shredded into object-relational tables, all collections of XML data that is stored within a single column or table must be in the same format and represented by the same schema. The user can specify names for the SQL object types and tables where the XML content will be loaded. If the schema changes, then the internal storage structure of the XML document must also change. This is accomplished by exporting and importing the document. If XML is stored in schema based XMLTypes, then Oracle attempts to transparently rewrite the XPath expression into equivalent SQL, as the data is stored in a relational format. If the XPath to SQL rewrite is not possible, then an external Oracle Java processor is used to interpret the XPath.

The second storage model that XMLType supports is referred to as unstructured storage. unstructured storage is stored byte for byte in text format as a CLOB. The data is not pre-parsed and no schema is required. As a result, documents that belong to different schemas may all be stored in the same column or table. At runtime, these documents are converted into a document object model (DOM) tree object and parsed in memory whenever XPath expressions or XML functions are used to process elements of a document.

The XMLType data type is used as a column in a table or as a table itself, and includes several proprietary member functions to create, extract, update, and process XML data. The XMLType data type can be used to declare variables in

PL/SQL procedures, functions, and scripts and can be used in a trigger body or as a host variable in applications such as C and Java.

8.2.2 XML schema conversion and registration

An XML Schema Definition (XSD) is an instance of an XML schema written in the W3C XML schema language. An XSD defines a type of XML document, specifying constraints upon what elements and attributes may appear, their data types, relationship to each other, and more. An XSD can be used in validation to ascertain whether a particular XML document is of that type and to produce a Post-Schema Validation Infoset (PSVI).

In Oracle, aside from validation, XML schemas are required in order to store XML in structured storage. Structured storage requires that the XML schema be annotated with several Oracle XML Database-defined attributes. These annotation attributes control the mapping of the XML document to SQL object types, collection types, and column-table paris in the database. The XML schema must be registered before being able to store XML data in structured storage. If you do not manually specify the annotations, then Oracle will provide default annotations to register the XML schema and map the elements and attributes to an automatically generated relational schema. Unstructured storage does not require the use of an XML schema. The dbms_xmlschema.registerschema stored procedure is used to register an XML schema in Oracle.

Oracle uses partial validation of XML, by default, when an XML document is inserted into an XMLType table or column. This partial validation checks that all mandatory elements and attributes are present and that there are no unexpected elements and attributes. This ensures that the document can be stored in the SQL types of the associated relational schema.

A DB2 XML schema can have components from more than one namespace and can consist of multiple XML schema documents. XML schemas are registered in the DB2 XML Schema Repository (XSR), which is a repository that allows DB2 to manage dependencies on externally referenced XML artifacts in XML instance documents within a database. Once registered with the XSR, XML schemas have a unique identifier and can be used to validate XML instance documents.

XML schemas in DB2 are primarily used for validation. XML validation adds type annotations to element nodes, attribute nodes, and atomic values, and strips off ignorable whitespace in the XML document. Validation is optional. Full validation is performed using the XMLVALIDATE function, usually on INSERT or UPDATE of an XML document; however it can also be used to validate an XML document that is not in a database. Before validation is possible, all schema documents that make up the XML schema must be registered in the XSR.

DB2 also supports annotated XML schemas for decomposition—the storing of XML data in SQL data types belonging to a relational structure. Decomposition is an alternative to the DB2 pureXML data type approach. Annotated XML schemas are not necessary for DB2 to store XML documents natively. Furthermore, XML schemas are only required if you wish to perform validation or decomposition. To take advantage of the XML data type in DB2, all you need to do is insert the XML document into an XML type column of a table.

This section outlines how to extract registered XML schemas from an Oracle database, how to convert the XML schema for use in DB2, and finally, how to register the XML schema in DB2. Again, you only need to register an XML schema if you intend to perform validation or decomposition, otherwise, no XML schema is required.

Obtaining and converting XML schemas

XML schema can be obtained for use in DB2 in several ways. The following are the most typical approaches:

► Use the original XML schema that registration was performed on for the Oracle database.

► Use the DBMS_XMLSCHEMA.generateschema function to produce an XML schema based on an Oracle type.

► Use the Oracle Enterprise Manager (OEM) to view registered XML schemas.

► Use a third-party tool to extract the XML schema from the Oracle database.

First you need to identify which XML schemas reside in the Oracle database. One way this can be done is by querying the DBA_XML_SCHEMAS system catalog view. A statement similar to the following can help with that:

```
select owner, schema_url
  from dba_xml_schemas
```

You may also find it useful to identify the XML schema associated with XMLType columns. The following query will list such details:

```
select owner, table_name, column_name, xmlschema, schema_owner
  from dba_xml_tab_cols
```

To list the XML schemas associated with XMLType tables, issue:

```
select owner, table_name, xmlschema, schema_owner
  from dba_xml_tables
```

Now that you have your inventory of XML schemas, you need the actual XML schema document.

The first approach of simply using the same XML schema that was originally registered with the Oracle database is the most straightforward. This approach does not require any conversion or editing for the XML schema to work with DB2, but requires that you still have access to the original XML schema. This is the recommended approach.

The second approach is to use the generateschema function of the DBMS_XMLSCHEMA package. Unfortunately, this function does not seem to work for the more complex XML schemas that have mapped to several Oracle types. This function can be used in a manner similar to the following:

```
select DBMS_XMLSCHEMA.generateschema('ORA_USR', 'CUSTOMER_T')
    from DUAL;
```

In this case, ORA_USR is the schema of the CUSTOMER_T type, where XML data is stored. The output of this command can be spooled to an .xsd file, which then needs to be edited before use in DB2.

The XML schema fragment shown in Example 8-1 is the kind of output you can expect from the above command.

Example 8-1   generateschema function output

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://posample.org"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           xmlns:oraxdb="http://xmlns.oracle.com/xdb"
           oraxdb:flags="291"
           oraxdb:schemaURL="http://posample.org"
           oraxdb:schemaOwner="ORA_USR"
           oraxdb:numProps="15">
    <xs:element name="customerinfo"
                oraxdb:propNumber="3234"
                oraxdb:global="true"
                oraxdb:SQLName="customerinfo"
                oraxdb:SQLType="customerinfo201_T"
                oraxdb:SQLSchema="ORA_USR"
                oraxdb:memType="258">
        <xs:complexType oraxdb:SQLType="customerinfo201_T"
                        oraxdb:SQLSchema="ORA_USR">
            <xs:sequence>
                <xs:element name="name"
                            type="xs:string"
                            minOccurs="1"
                            oraxdb:propNumber="3221"
                            oraxdb:global="false"
```

```
                                        oraxdb:SQLName="name"
                                        oraxdb:SQLType="VARCHAR2"
                                        oraxdb:memType="1"
                                        oraxdb:SQLInline="true"
                                        oraxdb:MemInline="true"
                                        oraxdb:JavaInline="true" />
                                        ...
```

In order for this XML schema to work with DB2, all Oracle oraxdb annotations
must be removed. For example, the same XML schema fragment shown in
Example 8-1 would look like the one shown in Example 8-2, after removing the
Oracle annotations:

Example 8-2   Converted DB2 XML schema

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://posample.org"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
    <xs:element name="customerinfo">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name"
                            type="xs:string"
                            minOccurs="1"/>
                            ...
```

The third approach of using the OEM is quite similar to the
DBMS_XMLSCHEMA.generateschema approach. First, start OEM and log into
your database. Under the Administration tab, click **XML schemas**. On the next
panel, you can optionally restrict your search to a specific "Schema" or "Object
Name". Fill in the details as desired, or leave them blank, and click **Go**. You will
now see a list of the XML schemas that matched your search criteria.

Now you need to select your XML schemas, one at a time, and click **View**. The
XML schema can now be copied from the Schema Text window and pasted into
a flat file with an .xsd extension. You then need to remove all Oracle oraxdb
annotations, as was done in the DBMS_XMLSCHEMA.generateschema
approach above. Repeat for each XML schema.

The fourth approach of using a third-party tool is not covered here, but should be
similar to the approaches discussed above.

Now that you have a W3C compliant XML schema, it can be registered in DB2.

XML schema registration in DB2

Before an XML schema can be used for validation or annotated decomposition, it must first be registered with the XSR. Registration with the XSR creates an XSR object. If you have chosen to annotate for decomposition, all tables and columns referenced in the annotated XML schema must exist in the database before registration will succeed. An XML schema document is not checked for correctness when adding documents to the XSR. Document checks are performed only when you complete schema registration.

Registration of an XML schema can be done in several ways:

► Command Line Processor (CLP)
► Stored procedure
► JDBC function

XML schema registration consists of the following steps:

1. Register the primary XML schema document in the XSR.

2. Specify additional XML schema documents to be included with the XSR object. This step is only required if the XML schema consists of more than one schema document.

3. Complete the registration process with the XSR.

Each of the DB2 XML schema registration approaches is illustrated.

CLP XML schema registration

From the CLP, registration would go as follows for an XML schema that consists of only one document:

```
register xmlschema 'http://posample.org'
   from 'd:\XMLSchemas\customer.xsd'
   as user1.customer
   complete
```

In this case, `'http://posample.org'` is the schema location of the XML schema document, that is, the external name of the XML schema. The primary document can be identified in the XML instance documents with the `xsi:http://posample.org` attribute. `'d:\XMLSchemas\customer.xsd'` is the location of the XML schema document on the file system, and `user1.customer` is the two-part SQL name that the registered schema will be known as.

If the XML schema had consisted of more than one schema document, the COMPLETE option would be left off of the above command and for each additional XML schema document (except the last), issue the **add xmlschema document** command. For example:

```
register xmlschema 'http://posample2.org'
```

```
   from 'd:\XMLSchemas\porder.xsd'
   as user1.porder

add xmlschema document to user1.porder
   add 'product.xsd'
   from 'd:\XMLSchemas\product.xsd'
```

When you have reached the last schema document of the XML schema, use the COMPLETE XMLSCHEMA command to complete registration.

```
complete xmlschema user1.porder
   with 'd:\XMLSchemas\supplier.xsd'
```

In order for the above multi-XML schema document registration to succeed, the XML schema documents themselves must be connected using an `include` or a `redefine` and the `schemaLocation` should match the URL specified at registration.

Stored procedure XML schema registration

Registration can also be performed via a stored procedure, which is useful if performing XML schema registration from an application. An XML schema document is registered by calling the sysproc.xsr_register stored procedure, as the following demonstrates:

```
call sysproc.xsr_register('user1',
                          'porder',
                          'http://posample2.org',
                          :content_host_var,
                          null)
```

The first input parameter, 'user1', specifies the SQL schema for the XML schema, which is one part of the SQL identifier used to identify this XML schema in the XSR. The second input parameter, 'porder', specifies the name of the XML schema, the other part of the SQL identifier making the complete SQL identifier `user1.porder`. The third parameter, 'http://posample2.org', specifies the schema location of the primary XML schema document. The fourth parameter, `:content_host_var`, is a BLOB that contains the content of the primary XML schema document. Finally, the fifth parameter is a BLOB that indicates the properties for the primary XML schema document. If there are no associated properties for the XML document, `null` is provided.

If the XML schema consists of more than one schema document, call the xsr_addschemadoc stored procedure for each of the schema documents that have not yet been registered. The input parameters are similar to the xsr_register stored procedure illustrated above:

```
call sysproc.xsr_addschemadoc('user1',
```

```
                                    'porder',
                                    'product.xsd',
                                    :content_host_var,
                                    null)
```

Regardless as to whether the XML schema consists of one or multiple schema documents, complete the registration by calling the sysproc.xsr_complete stored procedure:

```
call sysproc.xsr_complete ('user1',
                           'porder',
                           null,
                           0)
```

The third parameter, null in this case, is a BLOB which represents properties associated with the XML schema. The last parameter, isusedfordecomposition, can be set to indicate whether the XML schema will be used for decomposition, 1 for "yes" and 0 for "no".

Java XML schema registration

The IBM DB2 driver for JDBC and SQLJ also provides methods that let you register XML schema documents from a Java application program. The DB2Connection.registerDB2XMLSchema method registers an XML schema in DB2, using one or more XML schema documents. There are two forms of this method: one form for XML schema documents that are input from InputStream objects, and one form for XML schema documents that are in a string.

Example 8-3 demonstrates the use of registerDB2XmlSchema to register an XML schema in DB2 using a single XML schema document, customer.xsd, that is read from an input stream. The SQL schema name for the registered schema is SYSXSR.

Example 8-3   Registering XML schema using registerDB2XmlSchema

```
public static void registerSchema(
  Connection con,
  String schemaName)
  throws SQLException {

  // Define the registerDB2XmlSchema parameters
  String[] xmlSchemaNameQualifiers = new String[1];
  String[] xmlSchemaNames = new String[1];
  String[] xmlSchemaLocations = new String[1];
  InputStream[] xmlSchemaDocuments = new InputStream[1];
  int[] xmlSchemaDocumentsLengths = new int[1];
```

```java
    java.io.InputStream[] xmlSchemaDocumentsProperties = new
InputStream[1];
  int[] xmlSchemaDocumentsPropertiesLengths = new int[1];
  InputStream xmlSchemaProperties;
  int xmlSchemaPropertiesLength;

  //Set the parameter values
  xmlSchemaLocations[0] = "";
  FileInputStream fi = null;
  xmlSchemaNameQualifiers[0] = "SYSXSR";
  xmlSchemaNames[0] = schemaName;
  try {
    fi = new FileInputStream("customer.xsd");
    xmlSchemaDocuments[0] = new BufferedInputStream(fi);
  } catch (FileNotFoundException e) {
         e.printStackTrace();
  }
  try {
     xmlSchemaDocumentsLengths[0] = (int) fi.getChannel().size();
     System.out.println(xmlSchemaDocumentsLengths[0]);
  } catch (IOException e1) {
         e1.printStackTrace();
  }
  xmlSchemaDocumentsProperties[0] = null;
  xmlSchemaDocumentsPropertiesLengths[0] = 0;
  xmlSchemaProperties = null;
  xmlSchemaPropertiesLength = 0;
  DB2Connection ds = (DB2Connection) con;

  // Invoke registerDB2XmlSchema
  ds.registerDB2XmlSchema(
    xmlSchemaNameQualifiers,
    xmlSchemaNames,
    xmlSchemaLocations,
    xmlSchemaDocuments,
    xmlSchemaDocumentsLengths,
    xmlSchemaDocumentsProperties,
    xmlSchemaDocumentsPropertiesLengths,
    xmlSchemaProperties,
    xmlSchemaPropertiesLength,
    false);
}
```

Additional Information

For more information on XML schema registration in DB2, refer to the technical article "XML Schema Registration and Validation" at:

http://www.ibm.com/developerworks/wikis/download/attachments/1824/XMLSchema+Registration+and+Validation.pdf

8.2.3 Oracle unstructured and structured storage to DB2 pureXML

This section outlines the steps necessary to move from Oracle unstructured and structured storage to the DB2 XML data type. In the case of unstructured storage, this applies to cases where XML data is being stored in an XMLType column or table that is not associated with a registered XML schema. This also applies to cases where XML data is being stored in a CLOB in Oracle. Structured storage requires a registered XML schema in Oracle. The conversion of XML schemas is outlined in 8.2.2, "XML schema conversion and registration" on page 342. DB2 does not require a registered XML schema to be able to store XML documents in an XML data type column. Registered XML schemas are only necessary to perform validation or decomposition.

The first step you need to perform is to make an inventory of all the Oracle tables that are storing XML documents. If you already know the tables, then you may decide to skip the queries below. If you are unaware of the tables, as a user with a DBA role (so that you can see all tables), you can use the following commands.

To list user tables using unstructured storage XMLType columns:

```
select distinct table_name, owner
  from dba_xml_tab_cols
  where storage_type = 'CLOB' and owner not in ('MDSYS','XDB')
        and table_name not like 'BIN$%';
```

To list unstructured storage XMLType user tables:

```
select distinct table_name, owner
  from dba_xml_tables
  where storage_type = 'CLOB' and owner != 'XDB'
        and table_name not like 'BIN$%';
```

If you are storing XML in any CLOB columns, the following query will identify user tables making use of a CLOB column. These tables may not necessarily contain XML data—you will need to query their contents to make sure:

```
select distinct table_name, owner
  from dba_tab_columns
  where data_type = 'CLOB' and owner not like '%SYS%'
        and owner != 'OUTLN';
```

To list tables with structured storage XMLType columns:

```
select distinct table_name, owner
  from dba_xml_tab_cols
  where storage_type = 'OBJECT-RELATIONAL'
       and owner not in ('MDSYS','XDB')
       and table_name not like 'BIN$%'
```

And finally, to list structured storage XMLType tables

```
select distinct table_name, owner
  from dba_xml_tables
  where storage_type = 'OBJECT-RELATIONAL'
       and owner != 'XDB' and table_name not like 'BIN$%';
```

Using these queries, we can take things one step further, to generate the Oracle DDL of such tables. It is important to note that this is Oracle DDL and will not run in DB2 as-is. In order to store the XML content in DB2, make sure to create the columns that will store the XML documents with the DB2 XML data type.

Example 8-4 creates a file with the Oracle DDL of all tables using structured or unstructured storage XMLType column, as well as XMLType tables using structured or unstructured storage.

Example 8-4   Creating Oracle DDL of all tables using XMLType columns

```
set pagesize 0
set long 90000
set feedback off
set echo off
spool all_xml_tables.sql
select dbms_metadata.get_ddl('TABLE',u.table_name)
  from dba_tables u
  where u.table_name in (select distinct table_name
                          from dba_xml_tab_cols
                          where storage_type in
                               ('CLOB','OBJECT-RELATIONAL')
                              and owner not in ('MDSYS','XDB')
                              and table_name not like 'BIN$%'
                        union
                        select distinct table_name
                          from dba_xml_tables
                          where storage_type in
                               ('CLOB','OBJECT-RELATIONAL')
                              and owner != 'XDB'
                              and table_name not like 'BIN$%');
```

```
spool off
```

For example, as a result, one of the tables we might get the DDL for is the customer_us table shown in Example 8-5.

Example 8-5   customer_us table DDL

```
CREATE TABLE "ORA_USR"."CUSTOMER_US"
   ("CID" NUMBER(*,0) NOT NULL ENABLE,
    "INFO" "ORA_USR"."XMLTYPE",
    "HISTORY" "ORA_USR"."XMLTYPE"
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USER_DATA_TBS"
 XMLTYPE COLUMN "INFO" STORE AS CLOB (
 TABLESPACE "USER_DATA_TBS" ENABLE STORAGE IN ROW CHUNK 8192 PCTVERSION 10
 NOCACHE LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT))
 XMLTYPE COLUMN "HISTORY" STORE AS CLOB (
 TABLESPACE "USER_DATA_TBS" ENABLE STORAGE IN ROW CHUNK 8192 PCTVERSION 10
 NOCACHE LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT))
```

Since we are only concerned with the table structure, you can remove all other information. This leaves the following:

```
CREATE TABLE "ORA_USR"."CUSTOMER_US"
   ("CID" NUMBER(*,0) NOT NULL ENABLE,
    "INFO" "ORA_USR"."XMLTYPE",
    "HISTORY" "ORA_USR"."XMLTYPE")
```

An equivalent table in DB2, using the XML data type, may look like the following:

```
create table "customer_us"
   (cid integer not null,
    info xml,
    history xml)
```

The XMLType columns (or CLOB columns storing XML) have been mapped to the DB2 XML data type.

8.2.4 Oracle structured storage to DB2 decomposition

DB2 XML decomposition (or shredding) is the process of breaking down an XML document into corresponding columns of one or more relational tables based on the annotations specified in a registered annotated XML schema. The XML data that is decomposed to non-XML data types ceases to be XML. DB2 does not require decomposition to be able to store XML documents.

With decomposition, DB2 gives you the flexibility of controlling exactly what your relational schema will look like and how elements and attributes from the XML documents will be mapped to tables and columns. In most cases, users usually decompose various parts of the XML document to the same table, or small number of tables, which eliminates the need for complex and timely joins later on and also allows the data to be easily worked with using SQL. DB2 annotations are extremely flexible in controlling mapping, so you can decompose XML to any table-column in the database. XML data can be decomposed to any supported SQL type, for example: XML, VARCHAR, INTEGER, TIMESTAMP, and CLOB.

The primary reason you would use decomposition in DB2 is to allow preexisting applications that expect relational data to be able to process XML data as relational. Decomposition can also be used alongside storing the entire XML document in an XML column, by extracting important fields of the XML document to be stored separately. By extracting important fields and storing them alongside the entire XML document in the same table row, you can create relationships among data, such as referential integrity.

DB2 decomposition is a multi-step process:

1. Create the tables that the XML document will be decomposed into.
2. Annotate the XML schema with mapping instructions on how to decompose the XML document into relational tables.
3. Register the XML schema in the DB2 XSR and enable for decomposition.
4. Decompose XML documents with the CLP command or a stored procedure call.

In Oracle, XML schema annotations allow the developer to influence the objects and tables that XML data is decomposed into. There is no direct mapping between the Oracle structured storage and DB2 decomposition approaches, as they serve different purposes. Oracle structured storage is best mapped to the DB2 pureXML implementation, using the XML data type, which was discussed in 8.2.3, "Oracle unstructured and structured storage to DB2 pureXML" on page 350.

The following articles provide details on how to use decomposition in DB2 9:

► "XML tooling for DB2"

Provides information on how to use the DB2 Developer Workbench to graphically define annotations, that is, denote relationships between the XML schema elements and attributes and SQL table and columns. This article can be found at:

`http://www.ibm.com/developerworks/wikis/download/attachments/1824/XML+tooling+in+DB2+DWB.pdf?version=1`

► "Introduction to annotated XML schema decomposition using the DB2 Visual Studio 2005 Add-in"

Details how to graphically define XML schema annotations using DB2 Visual Studio 2005, as opposed to the DB2 Developer Workbench. This article can be found at:

`http://www.ibm.com/developerworks/db2/library/techarticle/dm-0611farahbod/`

► "Default mapping for annotated XML schema"

Demonstrates how to use a Java tool to perform default annotation mappings, which is great for an initial starting point to annotating complex XML schemas. This article can be found at:

`http://www.ibm.com/developerworks/db2/library/techarticle/dm-0604pradhan2/index.html`

8.3 XML data movement

In this section, we discuss the steps involved in moving the XML documents that were stored in the Oracle tables to DB2 target tables. Due to some special considerations for both Oracle and DB2, this step can be somewhat tricky for the beginner.

8.3.1 Exporting XML data from Oracle

Since Oracle does not provide an export utility to generate Comma Separated Value (CSV) files, extracting XML data requires some extra effort. If the combined size of your XML documents and other relational data, per row, is less than 2 MB in size, you can use Oracle to generate SQL INSERT statements for DB2. By spooling these INSERT statements to a file, it makes it possible to bulk insert such data in DB2. If your row size with the XML documents and other relational data is greater than 2 MB in size, per row, an alternate approach is needed.

First, examine the case where the row size is less than 2 MB each. Using the CUSTOMER_SS table as an example, you need to create a SELECT statement

in Oracle that can be used as an insert statement in DB2. This involves the escaping of single quote characters. The SQL to export the CUSTOMER_SS table would look like the following:

```
select 'insert into customer values
        (',cid,',''',info,''',''',history,''');' from customer_ss;
```

Now, to generate a bulk inserting script that will run in DB2, the following can be issued from SQLPlus:

```
set long 2000000
set heading off
set null NULL
set echo off
set feedback off
spool customer_bulk_insert.sql
select 'insert into customer values
        (',cid,',''',info,''',''',history,''');' from customer_ss;
spool off
```

The resulting file, customer_bulk_insert.sql, needs some minor modifications before being able to run in DB2. First, the SELECT statement in the first line of the spooled file needs to be removed. Next, the **spool off** command at the end of the file needs to be removed.

Example 8-6 shows one of the insert statements contained in the spooled file customer_bulk_insert.sql.

Example 8-6   Oracle insert statement for table customer_test

```
insert into customer_test values (      1000 ,'
<customerinfo xmlns="http://posample.org" Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>
','NULL');
```

By default, when Oracle displays a NULL, nothing is displayed. Through the environment command **set null NULL** (above), we instructed Oracle to display nulls as the text NULL. In such cases, the single quotes will need to be removed from around the NULL before the value can be inserted into DB2. After a search

and replace, the record illustrated in Example 8-6 will now look like the one in
Example 8-7.

Example 8-7   Converted DB2 insert statement for customer_test table

```
insert into customer_test values (       1000 ,'
<customerinfo xmlns="http://posample.org" Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>
',NULL);
```

The script containing the insert statements will be used in 8.3.2, "Inserting XML
data into DB2" on page 356 to populate the DB2 target table.

As mentioned previously, there are considerations that need to be made if the
entire row size of the INSERT (including target XML and non-XML columns) of
your data exceeds 2 MB. This stems from the DB2 Command Line Processor
(interface) only allowing the full INSERT statement along with string literal data
(XML and other relational) to not exceed 2 MB in size. In other words, if an
INSERT statement generated above exceed 2 MB, you will not be able to issue
that INSERT from a DB2 command window.

In such cases, you could break the insert statements into separate inserts and
updates. First, perform the INSERT (that is less than 2 MB) and use updates (of
less than 2 MB each) to update the columns that were not populated by the initial
INSERT. Another approach would be to create an application that can connect to
Oracle and DB2, select the Oracle data and insert it into DB2. Using this
approach, you can insert data up to the maximum allowable size for the target
column data type. The XML data type in DB2 supports XML documents up to
2 GB in size.

8.3.2 Inserting XML data into DB2

The SQL INSERT statement is used to populate the XML column of a DB2 table.
In order for the INSERT to succeed, the XML document being inserted must be
well-formed according to the XML 1.0 specification. If the INSERT is being
issued from an application, the application data type containing the XML can be
XML, character, or binary. Host variables are preferred to using literals, as host

variables automatically provide encoding information to DB2 that would otherwise need to be specified when using literals.

Since XML data coming from an application is in a serialized string format, DB2 must convert it to its XML hierarchical format for storage in an XML column. This process is known as XML parsing. DB2 performs implicit parsing when using an XML data type in the application, or when you assign a host variable, parameter marker, or SQL expression with a string data type (character, graphic or binary) to an XML column in an INSERT, UPDATE, DELETE, or MERGE statement. In all other cases, a call to the XMLPARSE scalar function will be required in your statement. Explicit use of XMLPARSE is not covered here.

Using the bulk-insert script that was generated in 8.3.1, "Exporting XML data from Oracle" on page 354, we can now insert our data into DB2. Since the target column for our XML data uses the XML data type, DB2 will perform the parsing implicitly. Use the following steps to insert the generated data into DB2:

1. Connect to the target DB2 database:

   db2 connect to *dbname*

2. Change to the directory containing the bulk-insert file.

3. Execute the script using the statement delimiter ";" (semi-colon):

   db2 -td; -vf customer_bulk_insert.sql

As mentioned, another approach to populate DB2 may involve a custom-built application. Example 8-8 shows a code fragment of a JDBC application that pulls a well-formed customer XML document from a file and inserts it into the CUSTOMER table.

Example 8-8   Moving data using JDBC application

```
PreparedStatement insertStmt = null;
String sqlstmt = null;
int cid = 1005;
sqlstmt = "INSERT INTO customer (CID, INFO) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqlstmt);
insertStmt.setInt(1, cid);
File file = new File("customer1005.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file),
(int)file.length());
insertStmt.executeUpdate();
```

By default, all XML data that is implicitly parsed will have its whitespace stripped out. Specifically, this means that text nodes containing only whitespace characters up to 1000 bytes in length will be stripped, unless the nearest

containing element has the attribute `xml:space='preserve'`. If any text node begins with more that 1000 bytes of whitespace, an error is returned (SQLSTATE 54059).

The INSERT shown in Example 8-9 is used to illustrate the default behavior of stripping whitespace.

Example 8-9   Default behavior of stripping whitespace

```
INSERT INTO customer VALUES (1001 ,'
<customerinfo xmlns="http://posample.org" Cid=" 1001">
  <name>   Kathy Smith</name>
  <addr country="Canada">
    <street  >25 EastCreek</street>
    <city>Markham</city>
    <prov-state>       </prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>
',NULL);


The resulting XML document is as follows:


<customerinfo xmlns="http://posample.org" Cid="  1001">
<name>   Kathy Smith</name><addr country="Canada">
<street>25 EastCreek</street><city>Markham</city>
<prov-state/><pcode-zip>N9C 3T6</pcode-zip>
</addr><phone type="work">905-555-7258</phone></customerinfo>
```

In Example 8-9, text nodes containing only whitespace have been stripped, as shown by the prov-state element (it has also been collapsed into the single element <prov-state/>). Additionally, the whitespace stripping has removed any whitespace between the different elements. However, it did not remove any whitespace of text nodes that contained text.

You can change this behavior by using the CURRENT IMPLICIT XMLPARSE OPTION special register. To instead preserve whitespace, the following command can be issued to DB2 (treated like SQL in applications):

```
set current implicit xmlparse option = 'PRESERVE WHITESPACE'
```

This will now preserve whitespace in any subsequent SQL, until the special register is changed. In this case, the XML document shown in Example 8-10 would have been the result. In this example, the prov-state element's whitespace has been preserved, as well as the whitespace between the elements.

Example 8-10   Behavior of preserving whitespace

```
<customerinfo xmlns="http://posample.org" Cid="  1001">
<name>   Kathy Smith</name> <addr country="Canada">
<street>25 EastCreek</street> <city>Markham</city>
<prov-state>        </prov-state> <pcode-zip>N9C 3T6</pcode-zip>
</addr> <phone type="work">905-555-7258</phone> </customerinfo>
```

8.3.3 Importing XML data into DB2

The use of the IMPORT utility in DB2 is to be able to import data into a DB2 table from data that had been previously exported with the DB2 EXPORT utility. The IMPORT utility can be used to insert XML documents into a table. Only well-formed XML documents can be imported.

XML data involved in the EXPORT and IMPORT utilities must be stored in files separate from the main data file. The XML data, however, is represented in the main data file with an XML Data Specifier (XDS). The XDS is a string represented as an XML tag named XDS, which has attributes that describe information about the actual XML data in the column. This information includes the name of the file that contains the actual XML data, and the offset and length of the XML data within that file.

Specifically, the attributes of the XDS are:

► FIL - The name of the file that contains the XML data

► OFF - The byte offset of the XML data in the file named by the FIL attribute, where the offset begins from 0

► LEN - The length in bytes of the XML data in the file named by the FIL attribute

► SCH - The fully qualified SQL identifier of the XML schema that is used to validate this XML document

The following is an example of an XDS as it would appear in a delimited ASCII data file:

```
"<XDS FIL = ""xmldocs.xml.001"" OFF=""100"" LEN=""300"" />"
```

This entry indicates that the XML data is stored in the file xmldocs.xml.001 beginning at byte offset 100 with a length of 300 bytes. Since this XDS is within an ASCII file delimited with double quotation marks, the double quotation marks within the XDS tag itself must be doubled.

When importing data into an XML table column, you can use the XML FROM option to specify the paths of the input XML data file or files. For example, for an

XML file /home/user/xmlpath/xmldocs.001.xml that had previously been exported, the following command could be used to import the data back into the table.

```
IMPORT FROM t1export.del OF DEL
  XML FROM /home/user/xmlpath
  INSERT INTO USER.T1
```

The XMLVALIDATE option allows XML documents to be validated against XML schemas as they are imported. In the following example, incoming XML documents are validated against schema information that was saved when the XML documents were exported:

```
IMPORT FROM t1export.del OF DEL
  XML FROM /home/user/xmlpath
  XMLVALIDATE USING XDS
  INSERT INTO USER.T1
```

You can use the XMLPARSE option to specify whether whitespace in the imported XML documents is preserved or stripped. In the following example, all imported XML documents are validated against XML schema information that was saved when the XML documents were exported, and these documents are parsed with whitespace preserved.

```
IMPORT FROM t1export.del OF DEL
  XML FROM /home/user/xmlpath
  XMLPARSE PRESERVE WHITESPACE
  XMLVALIDATE USING XDS
  INSERT INTO USER.T1
```

8.3.4 XML validation

XML validation is the process used to determine whether the structure, content, and data types of an XML document are valid according to an XML schema. It is also used to add type annotations to element nodes, attribute nodes, and atomic values, and to strip off ignorable whitespace in the XML document. XML validation is not required, but is quite useful in many cases.

XML validation is performed using the DB2 XMLVALIDATE function. Usually, it is used on the INSERT or UPDATE of an XML document. It can also be used on XML documents not stored in the database, but that is not discussed here.

In order to use XMLVALIDATE, the XML schema document must be registered in the DB2 XSR. This section assumes that the Oracle XML schemas have already been registered in DB2. Registering XML schemas in DB2 was covered in "XML schema registration in DB2" on page 346.

If you do not know what XML schemas to use to validate the XML documents that came from the Oracle database, you can create an inventory of XML schemas related to XMLType tables and columns as mentioned in "Obtaining and converting XML schemas" on page 343.

To list the registered XML schema used for XMLType tables, issue:

```
select owner, table_name, xmlschema, schema_owner
  from dba_xml_tables
```

Similarly, for XMLType columns, issue:

```
select owner, table_name, column_name, xmlschema, schema_owner
  from dba_xml_tab_cols
```

With the DB2 XMLVALIDATE function, you have the option of explicitly specifying the XML schema to be used for XML validation. If no XML schema is specified, DB2 will use the input document's `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` attribute as hints to identify an XML schema to use for validation. Specifying the XML schema explicitly to XMLVALIDATE will override the `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` attributes.

Using the customer example again, we have previously registered the customer XML schema, customer.xsd, with the following command:

```
register xmlschema 'http://posample.org'
  from 'd:\XMLSchemas\customer.xsd'
  as user1.customer
  complete
```

To demonstrate XML validation, imagine a dynamic SQL application. It inserts customer XML documents into a *customer* table, and performs validation against the user1.customer XML schema. The SQL would look like the following:

```
insert into customer (cid, info)
  values ( 1005,
           XMLVALIDATE(? ACCORDING TO XMLSCHEMA ID user1.customer))
```

In this case, the XML document is bound to the parameter marker (?) and validated at insert time. You could also have provided a string literal representing the XML document in place of the parameter marker, as long as the XML document is well formed.

If you need to ensure that you only insert XML documents that are validated into an XML column, you can create a check constraint on the table. The *customer* table could be altered in the following way to enforce this:

```
alter table customer add constraint ck_xml_valid (info is validated)
```

Now, only XML documents that have been validated will be allowed to be inserted into the *info* column of the *customer* table.

8.4 Converting XML queries

Although both SQL/XML and XQuery are each defined by their own particular standards, there are still differences in how Oracle and DB2 have adhered to those standards and as a result, there are differences in how the features that query, access, and generate XML content have been implemented.

8.4.1 SQL/XML

SQL/XML functions are SQL functions that invoke XPath or XQuery expressions and are used in SQL statements such as SELECT. They are used to access portions of an XML document or to generate XML data. Without these functions, an SQL statement can only access a column of XML data at the row level and cannot query at the sub document level.

SQL/XML functions can be categorized into two groups:

► Those that query and access XML content
► Those that generate XML content from SQL data

For those SQL/XML functions that query and access XML content, Oracle provides a set of proprietary functions that use XPath to access XML content. Included in these functions are what are known as XMLType methods. The XMLType functions belong to the XMLType data type. Some examples are getStringVal(), getClobVal(), getNumberVal(), getNamespace() and getBlobVal(). Besides the XMLType methods, Oracle also provides additional proprietary SQL/XML functions. The more important of these are extract(), existsNode(), and extractValue().

To query XML data, Oracle also supports the ISO/IEC standard SQL/XML functions, XMLQuery and XMLTable. These functions, known as the XQuery functions, are supported on DB2 as well and along with the XMLExists predicate are the only SQL/XML functions used on DB2 for XML querying.

In addition to the SQL/XML querying functions, DB2 also supports several other types of functions. Included in these are three casting functions: XMLCast, XMLParse and XMLSerialize. Oracle supports two of these, XMLParse and XMLSerialize. Oracle also supports the proprietary casting function XMLType, which converts an XML string value to an XMLType value. XMLType is most similar to DB2's XMLParse function. Oracle also supports XMLType methods

(such as getStringVal) to cast XML values to scalar string values. The closest DB2 equivalent to these is the XMLCast function.

For those SQL/XML functions that are used to generate XML from relational data, Oracle supports many of the same standard SQL/XML functions that are supported by DB2. These functions are referred to as the "publishing" functions and include: XMLElement, XMLAgg, XMLAttribute, XMLConcat, and XMLForest. Oracle also provides an additional set of proprietary functions that generate XML from SQL data.

Although none of the Oracle-provided SQL/XML functions and methods are found on DB2, their functionality can be mapped to DB2.

Table 8-1 lists some of the more important SQL/XML functions supported by Oracle and maps them to DB2 equivalents.

Table 8-1   SQL/XML function mapping

Oracle SQL/XML	SQL/XML category	Oracle specific	Closest DB2 equivalent	DB2 SQL/XML behavior
existsNode	Access	Yes	XMLEXISTS	Used in WHERE clause to filter rows returned.
extract	Access	Yes	XMLQUERY	Returns an XML sequence.
extractValue	Access	Yes	XMLQUERY & XMLCAST	Returns an XML value and converts to a number or string scalar value.
getstringVal	Access	Yes	XMLCAST	Converts an XML value to a string or numeric scalar value.
getNumberVal	Access	Yes	XMLCAST	Converts an XML value to a string or numeric scalar value.
XMLQUERY	Access	No	XMLQUERY	Returns an XML sequence.
XMLTABLE	Access	No	XMLTABLE	Returns XML values as a table.
XMLPARSE	Casting	No	XMLPARSE	Casts an XML string into the XML data type.
XMLTYPE	Casting	Yes	XMLPARSE	Casts an XML string into the XML data type.
XMLSERIALIZE	Casting	No	XMLSERIALIZE	Casts an XML sequence into an XML string value.
XMLCONCAT	Generate	No	XMLCONCAT	Returns a sequence that concatenates XML values.

Oracle SQL/XML	SQL/XML category	Oracle specific	Closest DB2 equivalent	DB2 SQL/XML behavior
XMLELEMENT	Generate	No	XMLELEMENT	Returns an XML element.
XMLAGG	Generate	No	XMLAGG	Returns a sequence containing non null XML values.
XMLATTRIBUTES	Generate	No	XMLATTRIBUTES	Generates attribute for an element.
XMLFOREST	Generate	No	XMLFOREST	Returns a sequence of element nodes.

To convert Oracle SQL/XML functions to DB2, find the most equivalent function(s) on DB2 and then refer to *DB2 9 XML Guide*, SC10-4254 and *DB2 SQL Reference manual Part 1,* SC10-4249 to assist you with constructing the appropriate syntax for your particular conversion. Syntax differences may even exist when the function is shared by both Oracle and DB2 and part of the ISO/IEC or W3C standard.

The following are some examples that demonstrate these differences.

A sample conversion involving Oracle and DB2 SQL/XML functions is shown in Example 8-11 and Example 8-12.

In this example, both the extract() and existsNode() functions are Oracle proprietary SQL/XML functions; the reference to the namespace value differs from how DB2 references the namespace.

The example for DB2 uses the wildcard notation (*:), which is prefixed to the XML elements. The wildcard will match any namespace specified. Although the wildcard notation is part of the W3C standard, it is not supported by Oracle.

Example 8-11   Using SQL/XML functions in Oracle

```
SELECT
extract(info,'/customerinfo//addr','xmlns="http://posample.org"')
FROM customer_us
WHERE
    existsnode(info,'/customerinfo//addr[city="Aurora"]',
                'xmlns="http://posample.org"')=1;
```

Example 8-12   DB2 conversion of using SQL/XML functions

```
SELECT XMLQUERY('$R/*:customerinfo//*:addr' PASSING info AS "R")
FROM customer
WHERE XMLEXISTS('$R/*:customerinfo//*:addr[*:city="Aurora"]'
                PASSING info as "R");;
```

The following example shows how to specify a namespace in DB2's XMLQuery when not using the wildcard notation:

```
SELECT XMLQUERY ('declare default element namespace
"http://posample.org"; $R/customerinfo//addr' PASSING INFO AS "R")
```

The next example demonstrates casting of XML values on Oracle and DB2.

On Oracle:

```
cityxml := incust.extract('/customerinfo//city');
city := cityxml.extract('//text()').getstringval();
```

On DB2:

```
SET cityXml = XMLQUERY('$cust/customerinfo//city' passing inCust as
"cust");
SET city = XMLCAST(cityXml as VARCHAR(100));
```

When you specify an XPath or XQuery expression in an Oracle SQL/XML function, Oracle executes the expression based on the type of XMLType storage used. If XML is stored in XMLType unstructured storage as a CLOB, then Oracle builds a DOM tree of the XML document in memory to process the XPath expression. If XML is stored in XMLType structured storage which under the covers is represented as object-relational data structures, Oracle rewrites the XPath expression into equivalent SQL statements.

Unlike Oracle, on DB2 the XPath or XQuery expressions are not rewritten into SQL statements and the XML document does not have to be loaded into memory as a DOM tree in order for XML processing by these functions to occur. This is due to the DB2 storage model where each node is already parsed and in a DOM-like, hierarchical format on disk and can easily be traversed by the XPath and XQuery languages.

8.4.2 XQuery

Although standardization was a goal of the World Wide Web Consortium (W3C) when designing the XQuery language, there are differences in how Oracle and IBM have implemented XQuery within their respective database products.

With DB2, XQuery is a case-sensitive, primary language that can be embedded directly within applications that access a DB2 database, or issued interactively

from the DB2 Command Line Processor. An XQuery statement is prefixed with the keyword XQUERY and is not limited to being invoked only from an SQL/XML function. The keyword indicates that the primary language is XQuery. In XQuery, two DB2-defined functions, db2-fn:xmlcolumn and db2-fn:sqlquery, are used in a query to obtain input XML data from a DB2 database. The db2-fn:xmlcolumn function is used to retrieve an XML sequence from the input of an XML column. The db2-fn:sqlquery function is used to retrieve a sequence of XML values based on the input of an SQL fullselect statement.

In Oracle, the XQuery statement cannot be embedded directly within SQL applications. In applications, the XQuery language is executed from the functions XMLQuery() and XMLtable(). The XQuery command can only be executed natively from the SQL*PLUS environment. However, before executing XQuery from SQL*PLUS, the environment must be properly initialized; this is accomplished by running an Oracle-provided script. After running this script and setting some additional parameters, the XQuery command can be used.

Oracle provides several XQuery and XPath extension functions that have a prefix of ora. Some of these are ora:view, ora:contains, and ora:replace. The Oracle XQuery extension functions do not map to the 2 DB2-defined XQuery functions mentioned previously. To convert the Oracle XQuery extension functions to DB2, you will need to rewrite the XQuery expression. For example, ora:view is used to create XML views on relational data so that the data can be manipulated as an XML document. On DB2, this is accomplished by using the SQL/XML publishing functions.

Oracle supports the standard XQuery functions fn:doc and fn:collection. These functions are used to retrieve a single document or a collection of documents that are stored in files on the Oracle XML DB repository. On DB2, since the XML document is always stored in tables, the db2-fn:xmlcolumn function can be used instead.

DB2 supports the use of the XQuery command interactively as well. The XQuery command can be run from the Command Line Processor (CLP). When run from the CLP, no additional setup is required to run XQuery commands.

Example 8-13 compares the differences between an XMLQuery function on Oracle and DB2.

Example 8-13   XQuery differences

```
-- In Oracle -------------------------------------------------------
SELECT XMLQUERY('$i/customerinfo//city'
                PASSING incust AS "i" RETURNING CONTENT) INTO cityxml
       FROM DUAL;
```

```
-- In DB2 ------------------------------------------------------------
SET cityxml = XMLQUERY('$cust/customerinfo//city' PASSING inCust as
"cust") FROM customer;
```

Example 8-14 shows how looping through XML content may be done in an
Oracle application.

Example 8-14   Looping through XML content in Oracle

```
CURSOR cur1(vcity IN VARCHAR2) IS
SELECT info from customer_us
WHERE
existsnode(info,'/customerinfo//addr[city="'||vcity||'"]','xmlns="http:
//posample.org"') = 1;
...
FOR c IN cur1(city) LOOP
customer := c.info.extract('//name','xmlns="http://posample.org"');
...
END LOOP;
```

In the DB2 application, the same iterating through XML content may be done
using the FLWOR expression of the XQuery statement. See Example 8-15.

Example 8-15   Looping through XML content in DB2

```
SET stmt_text = 'XQUERY for $cust
in db2-fn:xmlcolumn("CUSTOMER.INFO")
/*:customerinfo/*:addr[*:city="'||city||'"]
return $cust/../*:name';
...
PREPARE stmt FROM stmt_text;
OPEN cur1;
FETCH cur1 INTO customer;
WHILE (SQLSTATE = '00000') DO
...
FETCH cur1 INTO customer;
END WHILE;
```

Note: XQuery is executed as a dynamically prepared statement when
embedded in DB2 applications.

Example 8-16 and Example 8-17 compare the use of XQuery when used in the
XMLTABLE function on Oracle and DB2. Note that since the XMLTABLE function

is a standard SQL/XML function, the same namespace declaration is used for both.

Example 8-16   Using the XMLTABLE function in Oracle

```
select X.*
from customer_us,
     xmltable (XMLNAMESPACES (DEFAULT 'http://posample.org'),
    'for $m in $col/customerinfo
          return $m'
     passing customer_us.info as "col"
     columns
      "CUSTNAME" char(30) path 'name',
      "phonenum" xmltype path 'phone')
as X;
```

Example 8-17   Using the XMLTBLE function in DB2

```
select X.*
     from xmltable (XMLNAMESPACES (DEFAULT 'http://posample.org'),
      'db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
      columns
        "CUSTNAME" char(30) path 'name',
        "phone" xml path 'phone')
as X;
```

8.4.3 Updates and deletes

The initial release of the XQuery language only provided for querying XML at the subdocument level and did not provide for updating or deleting portions of XML content. Because IBM has chosen to conform to the W3C protocols, DB2 does not yet provide for an XQuery approach to update and delete portions of XML content. Currently, the entire document must be retrieved and subdocument updates and deletes are performed outside of the database. The entire document can be updated, deleted, and inserted using the SQL UPDATE, DELETE and INSERTstatements.

For convenience, IBM has developed a procedure called XMLUpdate that can be used to facilitate updating a portion of an XML document. Refer to *DB2 9 PureXML Guide*, SG24-7315 for details on how to use this procedure or go to:

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0605singh/

Another way to update portions of XML content is by storing XML in shredded format and using the DB2 XML extender, which is delivered with DB2.

Oracle provides the proprietary SQL/XML functions updateXML and deleteXML for purposes of updating and deleting portions of XML content. As a proprietary implementation, it is not part of the ISO/IEC or W3C standard.

8.4.4 Referential Integrity

Since decomposition is typically used to store XML data in non-XML SQL types (for example, INTEGER, VARCHAR, and so on), all SQL modes of enforcing constraints and referential integrity are supported when XML is decomposed into relational data.

For example, if you are decomposing a <CustomerID> element to the ID (INT NOT NULL) column of the table CUSTID_TAB, you could define a unique constraint as follows:

```
ALTER TABLE custid_tab ADD CONSTRAINT uniq_id UNIQUE (id);
```

If you have decomposed fragments of the XML document into XML type columns, there are some restrictions, such as not being able to define referential integrity or use triggers; however, decomposing to non-XML type columns removes that problem.

It is important to realize that in terms of referential integrity, the order of insertion based on referential integrity constraints is not observed. Instead, the following method ensures that rows are inserted in the correct order for the referential integrity:

1. Create your primary key:

   ```
   ALTER TABLE custid_tab ADD CONSTRAINT id_pk PRIMARY KEY (id);
   ```

2. Create your foreign key:

   ```
   ALTER TABLE orders ADD CONSTRAINT custid_fk FOREIGN KEY (custid)
   REFERENCES custid_tab (id);
   ```

3. With two copies of the document's XML schema, annotate one to map only to the table with the primary key and annotate the other to map only to the table with the foreign key.

4. Register both XML schemas with the XSR.

5. Decompose the XML document using the schema that maps to the table with the primary key.

6. Decompose the same XML document with the schema that maps to the table with the foreign key.

8.5 Converting XML indexes

Oracle supports three kinds of indexes on XML documents:

► B-Tree, which indexes the underlying SQL types and can be created on XMLType tables and columns that use structured storage.

► Function-based, which indexes a function or expression and can be defined on structured and unstructured storage XMLType tables and columns.

► Text-based, which indexes text within an XML document and can be defined on structured and unstructured storage XMLType tables and columns.

At index creation time, Oracle uses the provided XPath expression and determines whether it can rewrite the XPath to map to the underlying SQL data types. If the XPath can map to the SQL types, the index will be created as a B-Tree index on the SQL objects. If the XPath cannot be rewritten, a function-based index will instead be created.

In DB2, XML documents are stored with their native tree-based structure intact and DB2 is fully aware of the parent and child relationships among the various XML elements. The new DB2 9 index structure for XML data speeds access to specific XML elements and attributes, including those with repeating groups.

In contrast to traditional relational indexes, where index keys are composed of one or more table columns, a DB2 index over XML data uses a particular XML pattern expression to index paths and values in XML documents stored within a single column. The data type of that column must be XML.

Instead of providing access to the beginning of a document, index entries in an index over XML data provide access to nodes within the document by creating index keys based on XML pattern expressions. Since multiple parts of a XML document can satisfy an XML pattern, DB2 may generate multiple index keys when it inserts values for a single document into the index. DB2 fully supports XML indexes on repeating elements and attributes, and performs no parsing at run-time.

Due to the underlying differences between how Oracle and DB2 store XML data, there are some differences in indexing approaches. DB2 does not support function-based indexes, but does create indexes based on an XPath expression.

You create an index over XML data using the CREATE INDEX statement, and drop an index over XML data using the DROP INDEX statement. The path-specific value index allows indexing of elements and attributes frequently used in predicates and cross-document joins. The GENERATE KEY USING XMLPATTERN clause you include with the CREATE INDEX statement specifies what path-specific value you want to index. Some of the keywords used with the

`CREATE INDEX` statement for indexes on non-XML columns do not apply to indexes over XML data.

In DB2, the UNIQUE keyword has a different meaning for indexes over XML data. For indexes over XML data, the UNIQUE keyword enforces uniqueness within a single XML column across all documents whose nodes are qualified by the XML pattern. The insertion of a single document may cause multiple values to be inserted into a unique index; these values must be unique in that document and in all other documents in the same XML column. Note also that the insertion of some documents may not result in any values being inserted into an index; uniqueness is not enforced for these documents.

To help illustrate the XML indexing available in DB2, we again look at the *customer* table:

```
create table customer (cid bigint not null, info xml, history xml)
```

Example 8-18 shows a sample customer XML document that is stored in the info XML column of the *customer* table.

Example 8-18   Sample customer XML document

```
<customerinfo xmlns="http://posample.org"
              Cid="1005">
    <name>Larry Menard</name>
    <addr country="Canada">
        <street>223 NatureValley Road</street>
        <city>Toronto</city>
        <prov-state>Ontario</prov-state>
        <pcode-zip>M4C 5K8</pcode-zip>
    </addr>
    <phone type="work">905-555-9146</phone>
    <phone type="home">416-555-6121</phone>
    <assistant>
        <name>Goose Defender</name>
        <phone type="home">416-555-1943</phone>
    </assistant>
</customerinfo>
```

Currently, XML index keys can only be stored as VARCHAR, DOUBLE, DATE, and TIMESTAMP. You must ensure that you pick the proper storage type based on the queries you will be issuing. For example, if you have a query that evaluates the Cid attribute of the customerinfo element and uses the string '1005' as the search predicate, index keys will only match if the index is stored as a VARCHAR. If instead your queries search for the number 1005, then your XML index must store the Cid attribute index keys as a DOUBLE.

Example 8-19 illustrates indexing the Cid attribute of the customerinfo element and to store the index keys as a DOUBLE. This index is also created to only index those elements belonging to the default namespace of http://posample.org.

Example 8-19   Indexing the Cid attributes

```
create unique index cust_cid_xmlidx1
  on customer (info asc)
  generate key using xmlpattern
  'declare default element namespace "http://posample.org";
   /customerinfo/@Cid'
  as sql double allow reverse scans
```

A similar index, which will index elements and attributes belonging to any namespace, is shown in Example 8-20.

Example 8-20   Indexing elements and attributes belonging to any namespace

```
create unique index cust_cid_xmlidx2
  on customer (info asc)
  generate key using xmlpattern
  '/*:customerinfo/@Cid'
  as sql double allow reverse scans
```

The following are some common examples used to create indexes in Oracle on XML documents stored in the XMLType data type. The equivalent DB2 XML index is also shown. The main point being illustrated here is that the XPaths used in the Oracle indexes should be the same XPaths provided to the GENERATE KEY USING XMLPATTERN clause in the DB2 indexes.

Oracle indexing a text node:

```
create index cust_name_idx
  on customer_ss
  (extractValue(info, '/customerinfo/name'))
```

DB2 indexing a text node:

```
create index cust_name_idx
  on customer (info asc)
  generate key using xmlpattern
  '/*:customerinfo/*:name'
  as sql varchar(30) allow reverse scans
```

Oracle indexing an attribute node:

```
create index cust_cid_at_idx
```

```
       on customer_ss
       (extractValue(info,'/customerinfo/@Cid'))
```

DB2 indexing an attribute node:

```
  create index cust_cid_at_idx
     on customer (info asc)
     generate key using xmlpattern
     '/*:customerinfo/@Cid'
 as sql double allow reverse scans
```

Oracle unique index:

```
  create unique index cust_cid_uniq_idx
     on customer_ss
     (extract(info,'/customerinfo/@Cid').getStringVal());
```

DB2 unique index:

```
  create unique index cust_cid_uniq_idx
     on customer (info asc)
     generate key using xmlpattern
     '/*:customerinfo/@Cid'
     as sql double allow reverse scans
```

The sample code shown in Example 8-21 can be used to collect the important
XPaths used in the Oracle indexes defined on structured or unstructured storage
XMLType columns and tables.

Example 8-21   Collecting XPaths used in Oracle Indexes

```
set pagesize 0
set long 90000
set feedback off
set echo off
spool all_xml_related_indexes.ddl
select index_name, column_expression, table_name
  from dba_ind_expressions
  where table_name in (select distinct table_name
                         from dba_xml_tab_cols
                        where storage_type in
                                 ('CLOB','OBJECT-RELATIONAL')
                              and owner not in ('MDSYS','XDB')
                              and table_name not like 'BIN$%'
                       union
                       select distinct table_name
                         from dba_xml_tables
                        where storage_type in
```

```
                                 ('CLOB','OBJECT-RELATIONAL')
                        and owner != 'XDB'
                        and table_name not like 'BIN$%');
spool off
```

These captured XPaths can be provided to DB2 in the GENERATE KEY USING XMLPATTERN clause of the CREATE INDEX statement, as demonstrated above. XML index conversion is usually very straightforward.

For more information on DB2 XML indexes, it is recommended that you review the following developerWorks articles:

► "Exploit XML indexes for XML query performance in DB2 9"

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0611nic
ola

► "Indexing XML documents with DB2 9 pureXML"

http://www.ibm.com/developerworks/wikis/download/attachments/1824/in
dexingXMLdocuments.pdf

DB2 also provides an equivalent to Oracle text-based indexes, DB2 Net Search Extender (NSE). Though it comes with DB2 9, NSE is a separate install product and is not covered in detail here. XML full text search is provided by the DB2 Net Search Extender, which has been enhanced for XML. Full-text indexes with awareness of XML document structures can be defined on any native XML column. The documents in an XML column can be fully indexed or partially indexed, for example, if it is known in advance that only a certain part of each document will be subject to full-text search, such as a *description* or *comment* element node. Correspondingly, text search expressions can be applied to specific paths in a document. Administration and management is integrated in the DB2 Control Center. For more details, refer to "XML full-text search in DB2" at `the` developerWorks Web site:

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0606seuber
t/index.html

8.6 Converting XML in stored procedures

With stored procedures come the advantages of expanding SQL with programming constructs such as variables, looping, and if_then_else logic, along with the efficiency of sharing and executing code on the server. So it comes as no surprise that Oracle and DB2 both support XML in their respective procedure languages, PL/SQL and SQL PL.

In this section we compare some of the differences and similarities of XML support in Oracle and DB2 stored procedures.

8.6.1 Comparison overview

Here we demonstrate the following XML processing features, which are common to both procedure languages:

► Passing and returning XML values as parameters

► Local XML data type declarations

► Assignment of XML variables

► SQL/XML functions such as XMLQUERY, XMLTABLE, XMLSERIALIZE, XMLCONCAT

► XQuery or XPath expressions

► Cursor processing of XML documents

► Passing and returning XML values to XQuery or XPath expressions

We also demonstrate how Oracle and DB2 differ with respect to the processing of XML data in procedures. Some of these differences include:

► PL/SQL and SQL PL construct differences

  – Looping and declaring cursors

► XML data type declaration differences

  – XMLTYPE versus XML

► Database-specific XML-extension differences

  – PL/SQL uses ExistsNode() versus the XMLExists predicate used in SQL PL.

► Differences in embedded XQuery support

  – XMLQuery with XQuery expressions in PL/SQL versus embedded XQuery statements in SQL PL

► Differences in the way that namespaces are referenced

  – In PL/SQL the namespace declaration is passed as a parameter in the XMLType function.

  – In SQL PL the namespaces are referenced using a wildcard notation of "*:".

8.6.2 Converting an Oracle procedure with XML to DB2

In the following two sections, stored procedure examples are used to specifically show the XML processing differences just described between Oracle and DB2.

The Oracle and DB2 examples use the same logic flow, input parameter values, and XML documents that are stored in relational tables on each database. As expected, the output produced from each procedure is identical also. The differences are found in the SQL and XML constructs that were used to create each procedure.

The Oracle procedures are written using two different techniques that are available with Oracle:

► The first procedure demonstrates the proprietary extract() function to retrieve data.

► The second procedure demonstrates the standard XMLQuery function to retrieve the data.

The DB2 procedure is written using a single approach, which consists of an embedded XQuery statement using dynamic cursors and standard SQL/XML functions.

Note: DB2's XML data type does not include methods to manipulate XML data. Oracle only supports the use of the XQuery language as expressions within SQL/XML functions. Oracle does not support the embedded XQuery statement. Oracle 10g R2 supports the native XQuery statement as an interactive command in the SQLPLUS environment.

Description of the procedure logic
The Oracle and DB2 procedures are called with identical XML data passed as input parameters.

Each procedure extracts the City ("Markham") from the XML data passed as input. The City is used to find XML documents in the INFO column of the CUSTOMER table that contains a match to the city of "Markham". When a match is found, the customers' Names are extracted and concatenated to each other and to the City value.

The final result is constructed with a $<result>$ tag and returned.

8.6.3 The Oracle procedures

The table, CUSTOMER_US, is created using unstructured storage for XMLType as shown in Example 8-22.

Example 8-22   Creating a Oracle table

```
CREATE TABLE CUSTOMER_US
(
    cid    integer  not null,
    info  xmltype,
    history xmltype
);
```

We use the statements shown in Example 8-23 to insert six rows of XML data into the Oracle table CUSTOMER.

Example 8-23   Inserting XML data

```
INSERT INTO customer_us(cid,info)
      VALUES (1000,
              XMLTYPE('
                        <customerinfo xmlns="http://posample.org"
Cid="1000">
                            <name>Kathy Smith</name>
                            <addr country="Canada">
                               <street>5 Rosewood</street>
                               <city>Toronto</city>
                               <prov-state>Ontario</prov-state>
                               <pcode-zip>M6W 1E6</pcode-zip>
                             </addr>
                            <phone type="work">416-555-1358</phone>
                         </customerinfo>'));
INSERT INTO customer_us(cid,info)
      VALUES (1001,
              XMLTYPE('
                        <customerinfo xmlns="http://posample.org"
Cid="1001">
                            <name>Kathy Smith</name>
                            <addr country="Canada">
                               <street>25 EastCreek</street>
                               <city>Markham</city>
                               <prov-state>Ontario</prov-state>
                               <pcode-zip>N9C 3T6</pcode-zip>
                            </addr>
                            <phone type="work">905-555-7258</phone>
```

```
                                    </customerinfo>'));
        INSERT INTO customer_us(cid,info)
                VALUES (1002,
                        XMLTYPE('
                                <customerinfo xmlns="http://posample.org"
Cid="1002">
                                    <name>Jim Noodle</name>
                                    <addr country="Canada">
                                        <street>25 EastCreek</street>
                                        <city>Markham</city>
                                        <prov-state>Ontario</prov-state>
                                        <pcode-zip>N9C 3T6</pcode-zip>
                                    </addr>
                                    <phone type="work">905-555-7258</phone>
                            </customerinfo>'));
        INSERT INTO customer_us(cid,info)
                VALUES (1003,
                        XMLTYPE('
                                <customerinfo xmlns="http://posample.org"
Cid="1003">
                                     <name>Robert Shoemaker</name>
                                     <addr country="Canada">
                                        <street>1596 Baseline</street>
                                        <city>Aurora</city>
                                        <prov-state>Ontario</prov-state>
                                        <pcode-zip>N8X 7F8</pcode-zip>
                                     </addr>
                                     <phone type="work">905-555-7258</phone>
                                     <phone type="home">416-555-2937</phone>
                                     <phone type="cell">905-555-8743</phone>
                                     <phone type="cottage">613-555-3278</phone>
                            </customerinfo>'));
        INSERT INTO customer_us(cid,info)
                VALUES (1004,
                        XMLTYPE('
                                <customerinfo xmlns="http://posample.org"
Cid="1004">
                                        <name>Matt Foreman</name>
                                        <addr country="Canada">
                                           <street>1596 Baseline</street>
                                           <city>Toronto</city>
                                           <prov-state>Ontario</prov-state>
                                           <pcode-zip>M3Z 5H9</pcode-zip>
                                        </addr>
                                        <phone type="work">905-555-4789</phone>
```

```
                                        <phone type="home">416-555-3376</phone>
                                        <assistant>
                                            <name>Gopher Runner</name>
                                            <phone type="home">416-555-3426</phone>
                                        </assistant>
                    </customerinfo>'));
INSERT INTO customer_us(cid,info)
        VALUES (1005,
                XMLTYPE('
                            <customerinfo xmlns="http://posample.org"
Cid="1005">
                                <name>Larry Menard</name>
                                <addr country="Canada">
                                    <street>223 NatureValley Road</street>
                                    <city>Toronto</city>
                                    <prov-state>Ontario</prov-state>
                                    <pcode-zip>M4C 5K8</pcode-zip>
                                </addr>
                                <phone type="work">905-555-9146</phone>
                                <phone type="home">416-555-6121</phone>
                                <assistant>
                                    <name>Goose Defender</name>
                                    <phone type="home">416-555-1943</phone>
                                </assistant>
                            </customerinfo>'));
```

Creating Oracle procedure #1 - with XMLType methods

Example 8-24 shows the stored procedure with XMLType methods. This
procedure is created using SQLPLUS.

Example 8-24   Oracle stored procedure with XMLType methods

```
CREATE OR REPLACE PROCEDURE xmlproc (incust IN XMLTYPE)
IS
customer xmltype;
vcustomer varchar2(100);
cityxml xmltype;
city varchar2(100);
resxml xmltype;
vresxml varchar2(200);
CURSOR cur1(vcity IN VARCHAR2) IS
SELECT info from customer_us
```

```
WHERE
existsnode(info,'/customerinfo//addr[city="'||vcity||'"]','xmlns="http:
//posample.org"') = 1;
BEGIN
cityxml := incust.extract('/customerinfo//city');
city := cityxml.extract('//text()').getstringval();
DBMS_OUTPUT.put_line('city is '||city);
resxml := cityxml;
FOR c IN cur1(city) LOOP
customer := c.info.extract('//name','xmlns="http://posample.org"');
SELECT XMLCONCAT(resxml,customer) INTO resxml FROM dual;
END LOOP;
SELECT XMLELEMENT("result",resxml) into resxml FROM dual;
SELECT XMLSERIALIZE(CONTENT resxml AS VARCHAR2(200)) INTO vresxml FROM
dual;
DBMS_OUTPUT.put_line('resxml is '||vresxml);
end;
/
```

Creating Oracle procedure #2 - with SQL/XML functions

Example 8-25 is an Oracle procedure that uses a slightly different technique.

Example 8-25   Oracle stored procedure with SQL/XML functions

```
CREATE OR REPLACE PROCEDURE xmlproc (incust IN XMLTYPE)
IS
customer xmltype;
vcustomer varchar2(100);
cityxml xmltype;
city varchar2(100);
resxml xmltype;
vresxml varchar2(200);
CURSOR cur1(vcity IN VARCHAR2) IS
SELECT info from customer_us
WHERE
existsnode(info,'/customerinfo//addr[city="'||vcity||'"]','xmlns="http:
//posample.org"') = 1;
BEGIN
SELECT XMLQUERY('$i/customerinfo//city'
                PASSING incust AS "i" RETURNING CONTENT) INTO cityxml
     FROM DUAL;
city := cityxml.extract('//text()').getstringval();
DBMS_OUTPUT.put_line('city is '||city);
resxml := cityxml;
```

```
FOR c IN cur1(city) LOOP
customer := c.info.extract('//name','xmlns="http://posample.org"');
SELECT XMLCONCAT(resxml,customer) INTO resxml FROM DUAL;
END LOOP;
SELECT XMLQUERY('<result> {$res} </result>' PASSING resxml AS "res"
RETURNING CONTENT)
            INTO resxml FROM DUAL;
commit;
SELECT XMLSERIALIZE(CONTENT resxml AS VARCHAR2(200)) INTO vresxml FROM
dual;
DBMS_OUTPUT.put_line('resxml is '||vresxml);
end;
/
```

The Oracle Call procedure statement

The procedure was called from SQLPLUS as shown in Example 8-26.

Example 8-26   Calling Oracle stored procedure

```
CALL xmlproc(xmltype('<customerinfo Cid="5002"><name>Jim
Noodle</name><addr country="Canada"><street>25
EastCreek</street><city>Markham</city><prov-state>Ontario</prov-state><
pcode-zip>N9C-3T6</pcode-zip></addr><phone
type="work">905-566-7258</phone></customerinfo>'));
```

The output from both Oracle procedures is the same and is displayed using the
Oracle package DBMS_OUTPUT.put_line() shown in Example 8-27.

Example 8-27   Output of Oracle stored procedures

```
DBMS_OUTPUT.put_line():
resxml is <result><city>Markham</city><name
xmlns="http://posample.org">Kathy Smith</name><name
xmlns="http://posample.org">Jim Noodle</name></result>
```

8.6.4 DB2 stored procedure

The procedure uses the CUSTOMER table on the DB2 SAMPLE database. The
table's definition is displayed using the DB2 CLP shown in Example 8-28.

Example 8-28   DB2 table CUSOTMER

```
C:\redprocs>db2 describe table customer
Column                  Type      Type
name                    schema    name       Length   Scale Null
```

```
------------------------- --------- ---------- -------- ----- ----
CID                       SYSIBM    BIGINT           8     0 No
INFO                      SYSIBM    XML              0     0 Yes
HISTORY                   SYSIBM    XML              0     0 Yes
3 record(s) selected.
```

The data shown in Example 8-29 is stored in the CUSTOMER table and used by the DB2 procedure. The data is displayed by issuing a SELECT statement from the DB2 CLP.

Example 8-29   Data in the CUSTOMER table

```
SELECT CID, INFO FROM CUSTOMER;


CID   INFO
----  -------------------------------------------------------------
1000 <customerinfo xmlns="http://posample.org" Cid="1000">
       <name>Kathy Smith</name>
       <addr country="Canada">
         <street>5 Rosewood</street>
         <city>Toronto</city>
         <prov-state>Ontario</prov-state>
         <pcode-zip>M6W 1E6</pcode-zip>
       </addr>
       <phone type="work">416-555-1358</phone>
</customerinfo>
1001 <customerinfo xmlns="http://posample.org" Cid="1001">
       <name>Kathy Smith</name>
       <addr country="Canada">
         <street>25 EastCreek</street>
         <city>Markham</city>
         <prov-state>Ontario</prov-state>
         <pcode-zip>N9C 3T6</pcode-zip>
       </addr>
       <phone type="work">905-555-7258</phone>
     </customerinfo>
1002 <customerinfo xmlns="http://posample.org" Cid="1002">
       <name>Jim Noodle</name>
       <addr country="Canada">
             <street>25 EastCreek</street>
             <city>Markham</city>
             <prov-state>Ontario</prov-state>
             <pcode-zip>N9C 3T6</pcode-zip>
       </addr>
       <phone type="work">905-555-7258</phone>
```

```
          </customerinfo>
1003 <customerinfo xmlns="http://posample.org" Cid="1003">
          <name>Robert Shoemaker</name>
          <addr country="Canada">
             <street>1596 Baseline</street>
             <city>Aurora</city>
             <prov-state>Ontario</prov-state>
             <pcode-zip>N8X 7F8</pcode-zip>
          </addr>
          <phone type="work">905-555-7258</phone>
          <phone type="home">416-555-2937</phone>
          <phone type="cell">905-555-8743</phone>
          <phone type="cottage">613-555-3278</phone>
     </customerinfo>
1004 <customerinfo xmlns="http://posample.org" Cid="1004">
          <name>Matt Foreman</name>
          <addr country="Canada">
              <street>1596 Baseline</street>
              <city>Toronto</city>
              <prov-state>Ontario</prov-state>
              <pcode-zip>M3Z 5H9</pcode-zip>
          </addr>
          <phone type="work">905-555-4789</phone>
          <phone type="home">416-555-3376</phone>
          <assistant>
            <name>Gopher Runner</name>
            <phone type="home">416-555-3426</phone>
          </assistant>
     </customerinfo>
1005 <customerinfo xmlns="http://posample.org" Cid="1005">
          <name>Larry Menard</name>
          <addr country="Canada">
             <street>223 NatureValley Road</street>
             <city>Toronto</city>
             <prov-state>Ontario</prov-state>
             <pcode-zip>M4C 5K8</pcode-zip>
          </addr>
          <phone type="work">905-555-9146</phone>
          <phone type="home">416-555-6121</phone>
          <assistant>
              <name>Goose Defender</name>
              <phone type="home">416-555-1943</phone>
          </assistant>
     </customerinfo>
```

```
6 record(s) selected.
```

DB2 stored procedure

The DB2 procedure is created from the DB2 CLP using the following command:

```
db2 -td@ -vf crtproc.txt -z out
```

The procedure shown in Example 8-30 was contained in the file crtproc.txt.

Example 8-30   DB2 stored procedure

```
CREATE PROCEDURE xmlProc(IN inCust XML, OUT resXML XML)
SPECIFIC xmlProc
LANGUAGE SQL
BEGIN
DECLARE SQLSTATE CHAR(5);
DECLARE stmt_text VARCHAR (1024);
DECLARE customer XML;
DECLARE cityXml XML;
DECLARE city VARCHAR (100);
DECLARE stmt STATEMENT;
DECLARE cur1 CURSOR FOR stmt;
-- Get the city of the input customer
SET cityXml = XMLQUERY('$cust/customerinfo//city' passing inCust as
"cust");
SET city = XMLCAST(cityXml as VARCHAR(100));
-- Iterate over all the customers from the city using an XQUERY cursor
-- and collect the customer name values into the output XML value
SET stmt_text = 'XQUERY for $cust
in db2-fn:xmlcolumn("CUSTOMER.INFO")
/*:customerinfo/*:addr[*:city="'||city||'"]
return $cust/../*:name';
-- Use the name of the city for the input customer data as a prefix
SET resXML = cityXml;
PREPARE stmt FROM stmt_text;
OPEN cur1;
FETCH cur1 INTO customer;
WHILE (SQLSTATE = '00000') DO
SET resXML = XMLCONCAT(resXML, customer);
FETCH cur1 INTO customer;
END WHILE;
set resXML = XMLQUERY('<result> {$res} </result>'
passing resXML as "res");
END@
```

DB2 Call procedure statement

The procedure shown in Example 8-30 is called from the DB2 CLP as illustrated
in Example 8-31.

Example 8-31   Calling a DB2 stored procedure

```
CALL xmlProc(xmlparse(document '<customerinfo Cid="5002">
<name>Jim Noodle</name>
<addr country="Canada">
<street>25 EastCreek</street>
<city>Markham</city>
<prov-state>Ontario</prov-state>
<pcode-zip>N9C-3T6</pcode-zip>
</addr>
<phone type="work">905-566-7258</phone>
</customerinfo>' PRESERVE WHITESPACE),?)@
```

The output of the DB2 procedure is shown in Example 8-32. This output is the
same as the output from the Oracle procedures.

Example 8-32   Output of the DB2 stored procedure

```
Value of output parameters
 --------------------------
 Parameter Name  : RESXML
 Parameter Value : <result><city>Markham</city><name
xmlns="http://posample.org">Kathy Smith</name><name
xmlns="http://posample.org">Jim Noodle</name></result>
```

8.6.5 Other restrictions or limitations

DB2 SQL procedures support the XQuery language in two forms:

► As an argument to a SQL/XML function such as XMLQUERY:

```
XMLQUERY('$cust/customerinfo//city' PASSING inCust AS "cust")
```

In this case, the XQuery expression appears within single quotes and uses a
PASSING clause to pass variables into the XQuery expression.

► As an XQuery query with dynamic cursors, as shown in Example 8-33.

Example 8-33   DB2 XQuery support - with dynamic cursors

```
DECLARE stmt_text VARCHAR (1024);
DECLARE stmt STATEMENT;
DECLARE cur1 CURSOR FOR stmt;
……
```

```
SET stmt_text = 'XQUERY
for $cust in db2-fn:xmlcolumn("CUSTOMER.INFO")
/*:customerinfo/*:addr[*:city="'||city||'"]
return $cust/../*:name';
……
PREPARE stmt FROM stmt_text;
OPEN cur1;
```

In this case, the XQuery query is defined as a string, assigned to a
VARCHAR variable and prepared dynamically before opening the cursor.

The static statement as shown in Example 8-34 cannot be embedded within a
DB2 SQL procedure.

Example 8-34   Static statement

```
DECLARE cur1 CURSOR FOR
XQUERY for $cust
in db2-fn:xmlcolumn("CUSTOMER.INFO")
/*:customerinfo/*:addr[*:city="'||city||'"]
return $cust/../*:name;
```

Oracle XML DB support for the XQuery language in stored procedures is
provided through a native implementation of the SQL/XML functions, XMLQuery
and XMLTable. Unlike DB2, Oracle only supports using XQuery within
XMLQuery and XMLTable functions. Using XQuery as a query without SQL/XML
functions is only available in Oracle when using SQL*Plus.

A restriction to be aware of when using DB2 SQL procedures with XML
parameters and variables involves the use of *commits* and *rollbacks*. During the
execution of SQL procedures, the values assigned to XML parameters and
variables will no longer be available after a commit or rollback. Any attempt to
reference an XML variable after a commit or rollback will raise an error
(SQL1354N 560CE).

8.7 Converting XML in Java applications

Oracle JDBC drivers enable you to access and update XMLType tables and
columns, and call PL/SQL procedures that access Oracle XML DB Repository.
For DOM-based applications, Oracle XML DB supports the use of the DOM API
to access and update XMLType columns and tables. Programming to a DOM
API is more flexible than programming through JDBC, but it may require more
resources at runtime.

DB2 also provides JDBC support to access and update DB2 tables containing XML columns either directly through SQL or via stored procedure calls. The DOM API is also supported with DB2, using classes provided in the xml.jar file. When the JDBC 4.0 draft specification becomes a standard, IBM intends to support this specification. JDBC4.0 will include an SQLXML type and will provide integration with DOM processing.

This section briefly compares JDBC drivers provided by Oracle and IBM, and then discuss retrieval, insert, and update of XML documents with Java applications.

The content of this section is based on Oracle 10G R2.

8.7.1 JDBC drivers

Oracle provides two client side JDBC drivers. The Oracle Call Interface (OCI), also called thick JDBC driver is a Type 2 driver that utilizes C libraries. The thin JDBC driver is a Type 4 driver. Non-schema based XMLType processing can use either driver, while schema-based XMLType processing requires the thick JDBC driver. The only way to use the thin driver with schema-based XMLType is to force a cast from schema-based to non-schema-based XMLType. This can be done by invoking the createNonSchemaBasedXMLType() method on the Schema-Based XMLType object. The Oracle JDBC driver is implemented in classes12.jar.

The DB2 Universal JDBC driver (known as the JCC driver) also provides Type 4 and Type 2 JDBC connectivity. The DB2 JDBC driver is implemented in db2jcc.jar. In the case of the Type 2 driver, the Java classes invoke the native libraries shipped with the DB2 Client. The DB2 JDBC driver is implemented in db2jcc.jar. There is no functional advantage to using the DB2 Type 2 driver over the Type 4 driver, and Type 4 connectivity is recommended. The primary reason for Type 2 connectivity is an application carryover when Type 4 connectivity was not available with DB2.

A simple Oracle JDBC connection, using the DriverManager interface, is as follows:

```
DriverManager.registerDriver (new oracle.jdbc.OracleDriver());
      Connection connection = DriverManager.getConnection(
      "jdbc:oracle:thin:localhost:1521:orcl", "username",
"password");
```

Here 1521 is the listening port and orcl is the System Instance ID (SID).

A simple DB2 JDBC connection, using the DriverManager interface, is as follows:

► Type 4 connection:

```
DriverManager.registerDriver(new com.ibm.db2.jcc.DB2Driver());
                        connection = DriverManager.getConnection(
"jdbc:db2://hostname:50000/mydb","userid","password");
```

Here `50000` is the listening port and sample is the DB2 database.

► Type 2 connection:

```
DriverManager.registerDriver(new com.ibm.db2.jcc.DB2Driver());
                          connection = DriverManager.getConnection(
"jdbc:db2:mydb ","userid","password");
```

8.7.2 XML retrieval

XML data from Oracle tables and columns of type XMLType are typically retrieved into an XMLType object in Java, or directly to one of several Java output data types. XML data from DB2 XML columns can be retrieved into a DB2XML object in Java, or directly to one of several Java output data types. Note that the purpose of the DB2XML Java object is only to provide convenience methods for conversion of the XML document to various output data types. The DB2XML class does not offer comparable features to the Oracle XMLType class with regard to DOM processing and does not provide performance optimization versus using standard ResultSet methods.

DB2 XML retrieval summary

Before discussing approaches for migration of Oracle Java code, the available DB2 JDBC techniques to retrieve XML data need to be summarized. The JDBC result set methods and data types are found at the following Web site and are summarized below:

```
http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.ud
b.apdv.java.doc/doc/c0021817.htm
```

► Use a `ResultSet.getXXX` method other than `ResultSet.getObject` to retrieve the data into a compatible data type.

Example: `resultset.getString`

► Use the `ResultSet.getObject` method to retrieve the data, and then cast it to the `DB2Xml` type and assign it to a `DB2Xml` object. Then use a `DB2Xml.getDB2XXX` or `DB2Xml.getDB2XmlXXX` method to retrieve the data into a compatible output data type.

`DB2Xml.getDB2XmlXXX` methods add XML declarations with encoding specifications to the output data.

For example:

```
DB2Xml data = (DB2Xml) resultset.getObject(1);
String xmlstring1 = data.getDB2XmlString();
```

`getDB2XmlString` results in the following XML declaration being added:

```
<?xml version="1.0" encoding="ISO-10646-UCS-2"?>
```

`DB2Xml.getDB2XXX` methods do not add XML declarations with encoding specifications to the output data.

For example:

```
DB2Xml data = (DB2Xml) resultset.getObject(1);
String xmlstring1 = data.getDB2String();
```

`resultset.getString` and `data.getDB2String` return the same result and do not have an added XML declaration.

Table 8-2 lists the ResultSet methods and the data types for retrieving XML data.

Table 8-2   ResultSet methods and data types for retrieving XML data

Method	Output data type
ResultSet.getAsciiStream	InputStream
ResultSet.getBinaryStream	InputStream
ResultSet.getBytes	byte[]
ResultSet.getCharacterStream	Reader
ResultSet.getObject	DB2Xml
ResultSet.getString	String

Table 8-3 list DB2Xml methods, their output data types, and the type of XML internal encoding declaration added.

Table 8-3   DB2Xml methods, data types, and added encoding specifications

Method	Output data type	Type of XML internal encoding declaration added
DB2Xml.getDB2AsciiStream	InputStream	No XML declaration added, the encoding is US-ASCII
DB2Xml.getDB2BinaryStream	InputStream	No XML declaration added, the encoding is UTF-8
DB2Xml.getDB2Bytes	byte[]	No XML declaration added, the encoding is UTF-8

Method	Output data type	Type of XML internal encoding declaration added
DB2Xml.getDB2CharacterStream	Reader	No XML declaration added, the encoding is the native character encoding of Java (UTF-16)
DB2Xml.getDB2String	String	No XML declaration added, the encoding is the native character encoding of Java (UTF-16)
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	Specified by getDB2XmlBinaryStream targetEncoding parameter
DB2Xml.getDB2XmlBytes	byte[]	Specified by DB2Xml.getDB2XmlBytes targetEncoding parameter
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

Simple Oracle XMLType retrieval into a String/BLOB/CLOB

If Oracle XMLType data is retrieved into a String, BLOB, or CLOB in JDBC using the methods getClobVal(), getStringVal(), or getBlobVal(csid) to obtain the result as an oracle.sql.CLOB, java.lang.String, or oracle.sql.BLOB, you can use DB2 function xmlserialize for CLOB retrieval and resultset.getBinaryStream for binary stream retrieval, to convert the code.

The code in Example 8-35 and Example 8-36 demonstrates this.

Example 8-35   Retrieving XML data into CLOB in Oracle

```
OraclePreparedStatement stmt =
(OraclePreparedStatement)conn.prepareStatement(
"select e.poDoc.getClobVal() poDoc, e.poDoc.getStringVal() poString
from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet oracleset = (OracleResultSet) rset;
while(oracleset.next())
{
// the first argument is a CLOB
```

```
oracle.sql.CLOB clb = resultset.getCLOB(1);
// the second argument is a string..
String poString = oracleset.getString(2);
…
}
```

Example 8-36 performs CLOB retrieval in DB2.

Example 8-36   CLOB retrieval in DB2

```
PreparedStatement stmt =
conn.prepareStatement(
" Select xmlserialize(e.poDoc as clob(2m)) from po_xml_tab1 e");

ResultSet rs = stmt.executeQuery();
while(rs.next())
{
Clob clob = rs.getClob(1);
…
}
```

Example 8-37 does a BinaryStream retrieval in DB2, which is recommended for
XML Parser processing.

Example 8-37   Binary stream retrieval in DB2

```
PreparedStatement stmt =
conn.prepareStatement(
"select e.poDoc  from po_xml_tab e");

ResultSet rs = stmt.executeQuery();
while(rs.next())
{
InputStream inputstream = rs.getBinaryStream(1);
…
}
```

Oracle retrieval into an XMLType object

In Oracle, XMLType data can be retrieved into a Java XMLType object. Once the
XML data is assigned to the XMLType Java object, a number of XMLType
methods are available against the XML data. For example, the getOPAQUE()
call might be used in conjunction with the XMLType.createXML method to create
a Java XMLType object, which might then be converted to a DOM document. To
do the equivalent in DB2, the XML column would be retrieved as a BinaryStream,
and parsed using a DOMParser.

Example 8-38 shows an Oracle code example that retrieves an XML column into a DOM document.

Example 8-38   Oracle - retrieving an XML object

```
OraclePreparedStatement stmt =
(OraclePreparedStatement) conn.prepareStatement(
"select e.poDoc from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet oracleset = (OracleResultSet) rset;

// get the XMLType
XMLType poxmltype = XMLType.createXML(oracleset.getOPAQUE(1));

// now do something with the XMLType object, like creating a DOM
document
Document document = poxmltype.getDocument()
```

The equivalent functionality in DB2 can be accomplished as illustrated in Example 8-39.

Example 8-39   DB2 - retrieving an XML column into a DOM document

```
PreparedStatement selectStmt = conn.prepareStatement("Select poDoc from
po_xml_tab");
ResultSet rs = selectStmt.executeQuery();

InputStream xmlbinarystream = rs.getBinaryStream(1);
DOMParser p = new DOMParser();
p.parse(new InputSource(xmlbinarystream));
Document doc = p.getDocument();
```

Guidelines

When retrieving the XML document from the database to be processed by an XML Parser, use ResultSet.getBinaryStream(). Detailed migration mapping for the XMLType methods including the mapping for Oracle XMLType.getDocument() as well as a code example can be found in 8.7.5, "XMLType object method mapping" on page 404.

Table 8-4 shows the methods and retrieved data type mapping of Oracle and DB2.

Table 8-4   Summary of data type mappings against a resultset object

Oracle Method	Oracle retrieved data type	DB2 Method	DB2 retrieved data type
Resultset.getClobVal()	oracle.sql.CLOB	CLOB not supported - consider using getBinaryStream	java.io.ByteArrayInputStream if getBinaryStream used
Resultset.getStringVal()	oracle.sql.String	getString	java.lang.String
ResultSet.getBlobVal(csid)	oracle.sql.BLOB	BLOB not supported - consider using getBinaryStream	java.io.ByteArrayInputStream if getBinaryStream used
Resultset.getOPAQUE()	Typically used to create an XMLType object as: XMLType poxml = XMLType.createXML(orset.getOPAQUE(1));	Depends on the XMLType method subsequently used against the XMLType object - see 8.7.5, "XMLType object method mapping" on page 404.	Depends on the XMLType method subsequently used against the XMLType object - see 8.7.5, "XMLType object method mapping" on page 404.

8.7.3 Java XML insert

Oracle JDBC programs typically insert XML documents into the database using String, CLOB, and XMLType objects. Some of the older JDBC drivers for Oracle have trouble with Strings that are larger then 4K and use CLOB instead. DB2 does not have a size limit on String length, so there is no need to convert a string to a CLOB prior to insert.

A common way of inserting an XML document from Java is to use the setObject() or setOPAQUE() call in the PreparedStatement to set the whole XMLType instance as shown in Example 8-40. The createXML method accepts a variety of data types for the second parameter besides String. These are enumerated in the mapping table shown in 8.7.5, "XMLType object method mapping" on page 404.

Example 8-40   Oracle - insert XML document

```
import oracle.xdb.XMLType;
    ...
    OraclePreparedStatement stmt =
        (OraclePreparedStatement) conn.prepareStatement(
            "insert into po_xml_tab (docid, xmlpo) values (?,?) ");
    int docid = 1;
```

```
    stmt.setInt(1, docid);

    // the second argument to createXML is a string
    String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";
    XMLType poXML = XMLType.createXML(conn, poString);

    // now bind the string..
    stmt.setObject(2,poXML);

    stmt.execute();
```

With DB2, the string insert could be done using the program shown in
Example 8-41.

Example 8-41   DB2 - insert XML document as string

```
import java.sql.*;
import java.io.*;
public class insertpo
{
public static void main (String[] parameters){
  try {
    int docid = 1;
    String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";
    String query = "insert into po_xml_table (docid, xmlpo) values (?,
?)";

    DriverManager.registerDriver(new com.ibm.db2.jcc.DB2Driver());
    Connection conn = DriverManager.getConnection
("jdbc:db2://localhost:50000/mydb","userid","password");

    PreparedStatement insertStmt = conn.prepareStatement(query);
    insertStmt.setInt(1, docid);
    insertStmt.setString(2, poString);

    // execute the statement
    if (insertStmt.executeUpdate() != 1) {
        System.out.println("No record inserted."); }

    conn.close();
    }
  catch (Exception e) {}
  }
}
```

With DB2, the insert could also have been done using a character stream (useful for large data volumes) as shown in Example 8-42.

Example 8-42   DB2 - insert XML document using a character stream

```
import java.sql.*;
import java.io.*;
public class insertpo_xml_table_characterstream
{
public static void main (String[] parameters){
  try {
    int docid = 1;
    String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";
    String query = "insert into po_xml_table (docid, xmlpo) values (?,
?)";

    StringReader xmlReader = new StringReader(poString);


    DriverManager.registerDriver(new com.ibm.db2.jcc.DB2Driver());
    Connection conn = DriverManager.getConnection
("jdbc:db2://localhost:50000/mydb","userid","password");

    PreparedStatement insertStmt = conn.prepareStatement(query);

    insertStmt.setInt(1, docid);
    insertStmt.setCharacterStream(2, xmlReader, poString.length());

    System.out.println ("about to execute insert statement");
    // execute the statement
    if (insertStmt.executeUpdate() != 1) {
        System.out.println("No record inserted."); }

    conn.close();
    }
  catch (Exception e) {}
  }
}
```

Example 8-43 shows how a file containing an XML document can be inserted into a DB2 XML column, using PreparedStatement.setBinaryStream.

Example 8-43   DB2 - insert XML document from a file

```
import java.sql.*;
import java.io.*;
```

```
public class insertpo_xml_table_file
{

public static void main (String[] parameters){
  try {
    int docid = 1885;
    String fn = "E:/OracleXML/DB2-Java/Client1885.xml";  // input file
    String query = "insert into po_xml_table (docid, xmlpo) values (?,
?)";

    File file = new File(fn);

    DriverManager.registerDriver(new com.ibm.db2.jcc.DB2Driver());
    Connection conn = DriverManager.getConnection
("jdbc:db2://localhost:50000/mydb","userid","password");

PreparedStatement insertStmt = conn.prepareStatement(query);

    insertStmt.setInt(1, docid);
    insertStmt.setBinaryStream(2, new FileInputStream(file),
(int)file.length());
// execute the statement
    if (insertStmt.executeUpdate() != 1) {
        System.out.println("No record inserted."); }

    conn.close();
    }
  catch (Exception e) {}
  }
}
```

The DB2 JDBC options available to insert XML data into DB2 are summarized in the DB2 9 Online help at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.ud b.apdv.java.doc/doc/c0021816.htm

A brief summary of the DB2 JDBC insert options is shown in Table 8-5.

Table 8-5   Methods and data types for updating XML columns

Method	Input data type
PreparedStatement.setAsciiStream	InputStream
PreparedStatement.setBinaryStream	InputStream

Method	Input data type
PreparedStatement.setBlob	BLOB
PreparedStatement.setBytes	byte[]
PreparedStatement.setCharacterStream	Reader
PreparedStatement.setClob	CLOB
PreparedStatement.setObject	byte[], BLOB, CLOB, DB2Xml, InputStream, Reader, String
PreparedStatement.setString	String

Table 8-6 shows the Oracle to DB2 data type and method mappings for XML Insert processing. This table represents the DB2 method that most closely maps to the Oracle source data type. However, the following are best practice guidelines to be followed, when possible:

► If there is a large volume of data, and it is already character-based in the Java application, use setCharacterStream.

► To insert large data with internal encoding, use setBinaryStream.

► To insert data from a file, a message, or any other data source from outside the application, use setBinaryStream, because it avoids transcoding.

► To insert small character data, use setString.

► When possible, avoid BLOB and CLOB due to extra conversion cost in the JDBC driver.

Table 8-6   Oracle to DB2 data type/method mappings for XML Insert processing

Oracle Method	Oracle source data type	DB2 Method
preparedStatement.setString(string)	String	PreparedStatement.setString()
preparedStatement.setObject(clob);	oracle.sql.CLOB	PreparedStatement.setClob()
XMLType.createXML(conn, String);	String	PreparedStatement.setString()
XMLType.createXML(conn, InputStream);	InputStream	PreparedStatement.setAsciiStream() or PreparedStatement.setBinaryStream()

Oracle Method	Oracle source data type	DB2 Method
XMLType.createXML(conn, CLOB);	Oracle.sql.CLOB	PreparedStatement.setClob()

8.7.4 Java XML update

Oracle allows a schema-based XML document to have the XML document in the database updated, that is, change an existing XML document without replacing the entire document. Because there is not yet an XQuery standard for XML updates, DB2 does not provide an update mechanism at this time. However, the DB2 stored procedure DB2XMLFUNCTIONS.XMLUPDATE provides the functional ability to change an XML document. Internally, the stored procedure is retrieving the entire XML document into a DOM tree, making the requested document changes, and re-inserting the entire document. This is the recommended approach, when possible.

Example 8-44 shows a sample Oracle code for performing partial update of an XML document, which changes the discount amount to 10.

Example 8-44   Oracle code - XML partial document update

```
stmt = (OraclePreparedStatement)conn.prepareStatement(
"UPDATE po_xml_tab " +
"SET OBJECT_VALUE = updateXML(OBJECT_VALUE," +
"'/PurchaseOrder/PONO/LINEITEMS/LINEITEM_TYP/DISCOUNT/text()','10'," +
"WHERE existsNode(OBJECT_VALUE, '/PO/[PONO=\"200\"]') = 1)";
stmt.execute();
```

Example 8-45 is an Oracle example that shows the same resulting database change by first retrieving an XML document from the database, performing DOM manipulation, and then updating the document in the database.

Example 8-45   Oracle - XML full document update

```
Statement s = conn.createStatement();
OraclePreparedStatement stmt;
ResultSet rset = s.executeQuery(qryStr);
OracleResultSet oracleset = (OracleResultSet) rset;
while(oracleset.next()){

//retrieve PurchaseOrder xml document from database
XMLType xtype = XMLType.createXML(oracleset.getOPAQUE(1));

stmt.setObject(1,xtype); // bind the XMLType instance
```

```
stmt.execute();

//update "DISCOUNT" element
String newXML = updateXML(xt.getStringVal());

// create a new instance of an XMLtype object from the updated value
xtype = XMLType.createXML(conn,newXML);

// update PurchaseOrder xml document in database
stmt = (OraclePreparedStatement)conn.prepareStatement(
"update po_xml_tab x set x.poDoc =? where "+
"x.poDoc.extract('/PurchaseOrder/PO/PONO/text()').getNumberVal()=200");

// bind the XMLType object
stmt.setObject(1,xtype);
stmt.execute();
```

The Oracle application updateXML method, which manipulates the document, is as shown in Example 8-46.

Example 8-46   Oracle application using the updateXML method

```
static String updateXML(String xmlTypeStr)
{
String outXML = null;
try{
DOMParser parser = new DOMParser();
parser.setValidationMode(false);
parser.setPreserveWhitespace (true);
parser.parse(new StringReader(xmlTypeStr));
XMLDocument doc = parser.getDocument();
NodeList nl = doc.getElementsByTagName("DISCOUNT");
for(int i=0;i<nl.getLength();i++){
XMLElement discount = (XMLElement)nl.item(i);
XMLNode textNode = (XMLNode)discount.getFirstChild();
textNode.setNodeValue("10");
}
StringWriter sw = new StringWriter();
doc.print(new PrintWriter(sw));
outXML = sw.toString();
return outXML;
}
```

Example 8-47 shows the XML document being updated.

Example 8-47   XML document

```
<?xml version = "1.0"?>
<PurchaseOrder>
<PONO>200</PONO>
<CUSTOMER>
<CUSTNO>2</CUSTNO>
<CUSTNAME>Bob Sylvers</CUSTNAME>
</CUSTOMER>
<ORDERDATE>20-June-06</ORDERDATE>
<SHIPDATE>25-June-07 12.00.00.000000 AM</SHIPDATE>
<LINEITEMS>
<LINEITEM_TYP LineItemNo="1">
<ITEM StockNumber="20010">
<PRICE>6750</PRICE>
</ITEM>
<QUANTITY>1</QUANTITY>
<DISCOUNT>5</DISCOUNT>
</LINEITEM_TYP>
<LINEITEM_TYP LineItemNo="2">
<ITEM StockNumber="20255">
<PRICE>499.00</PRICE>
</ITEM>
<QUANTITY>5</QUANTITY>
<DISCOUNT>5</DISCOUNT>
</LINEITEM_TYP>
</LINEITEMS>
</PurchaseOrder>
```

The use of DB2XMLFUNCTIONS.XMLUPDATE to perform the same update in DB2 is shown in Example 8-48. As a Java stored procedure, XMLUPDATE requires that the following DB2 registry variable be set:

```
db2set DB2_USE_DB2JCCT2_JROUTINE=YES
```

Example 8-48   DB2 - XML update using the procedure DB2XMLFUNCTIONS.XMLUPDATE

```
import java.sql.*;
import java.io.*;

public class xmlupdate
{

public static void main (String[] parameters){
```

```
  try {

int errorCode;
String errorMsg ;

Connection conn   = null;
DriverManager.registerDriver(new com.ibm.db2.jcc.DB2Driver());
conn = DriverManager.getConnection
("jdbc:db2:mydatabase","userid","passwrod");



/* parameters to procedure DB2XMLFUNCTIONS.XMLUPDATE in this example
are as follows
(1) Command XML string encapsulating the update command:
    '<updates .....
(2) SQL statement retrieving the document(s) to be updated:
    'Select poDoc from ...
(3) SQL statement to update the documents if they updated documents are
to be
    placed in a different column. Since the documents are being
replaced into
    the same column, an update is not needed and an empty string ''
    is specified.
(4) Output parameter for error Code information:   ? is bound to
errorCode
    variable
(5) Output parameter for error Message information: ? is bound to
errorMsg
    variable
*/

CallableStatement cstmt = conn.prepareCall("Call
DB2XMLFUNCTIONS.XMLUPDATE ("                                   +
"'<updates> "
+
"<update  col=\"1\"
path=\"/PurchaseOrder/LINEITEMS/LINEITEM_TYP/DISCOUNT/text()\"> 10
</update>"             +
"</updates>',"
+
"'Select poDoc from po_xml_tab  where
XMLExists(''$doc/PurchaseOrder/PONO[text() = 200]'' passing poDoc as
\"doc\")','',?,?)");
/* XMLEXISTS XQUERY argument above must be surrounded with 2 single
quotes */
```

```
      // Register output parms
      cstmt.registerOutParameter (1, Types.INTEGER );
      cstmt.registerOutParameter (2, Types.VARCHAR );

      //Execute the stored procedure
      cstmt.execute();

      //get error information
      errorCode= cstmt.getInt(1) ;
      errorMsg = cstmt.getString (2);
      System.out.println ("errorCode" + errorCode + " errorMsg " + errorMsg);

        }  //end try
      catch (Exception e) {System.out.println ("ERROR: " + e.getMessage());}
      }  //end main
      }  //end public class
```

If the XMLUPDATE stored procedure will not be used to update a document, an alternative is to replace the entire document with an SQL UPDATE statement. See Example 8-49. The same JDBC driver methods and Java data types described in 8.7.3, "Java XML insert" on page 393 apply.

Example 8-49   DB2 - full XML document update

```
import java.sql.*;
import java.io.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.xml.sax.InputSource;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.OutputKeys;

public class xmlupdate
{
  public static void main(String[] args) throws Exception, SQLException
  {

    try {
    Connection conn    = null;
```

```
    String xmlstring = null;
    DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
  DocumentBuilder builder = dbf.newDocumentBuilder();


DriverManager.registerDriver(new com.ibm.db2.jcc.DB2Driver());
    conn = DriverManager.getConnection
    ("jdbc:db2://localhost:50000/mydb","userid","password");


    //create query on an updateable statement so each  XML Document can
be retrieved, altered via DOM, and re-inserted
    String query = "Select poDoc from po_xml_tab  where
XMLExists('$xmldoc/PurchaseOrder/PONO[text() = 200]' passing poDoc as
\"xmldoc\")";
    Statement stmt = conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_UPDATABLE);
    ResultSet rs = stmt.executeQuery (query);


    // iterate over the results
        while(rs.next() ){

        InputStream  inputstream = rs.getBinaryStream(1);
        Document doc = builder.parse(new InputSource (inputstream));

/********************************************************************
/* Perform DOM processing here to update XML Document
/* Details not shown
/********************************************************************

    /* convert the DOM document back to a string */

      Transformer serializer =
TransformerFactory.newInstance().newTransformer();

  /* choose XML Declaration, encoding, and indentation options */
  serializer.setOutputProperty(OutputKeys.ENCODING,"utf-8");
  serializer.setOutputProperty(OutputKeys.INDENT,"no");

  /* write Document back to String */
  StringWriter writer = new StringWriter();
  serializer.transform(new DOMSource(doc), new StreamResult(writer));
  String updatedXMLString = writer.toString();


  /* insert updated XML Document into the row */
```

```
                rs.updateString(1, updatedXMLString);

    }  // end while loop processing result set

    conn.close();

    }   //end try
catch (Exception e) { e.printStackTrace();
    }   //end catch
  } //end main
} //end public class
```

8.7.5 XMLType object method mapping

The following are Oracle Java DOM API classes documented in *Oracle XML DB Developers Guide* (Oracle 10g Release 2).

- ► oracle.xdb.dom.XDBDocument
- ► oracle.xdb.dom.XDBCData
- ► oracle.xdb.dom.XDBComment
- ► oracle.xdb.dom.XDBProcInst
- ► oracle.xdb.dom.XDBText
- ► oracle.xdb.dom.XDBEntity
- ► oracle.xdb.dom.DTD
- ► oracle.xdb.dom.XDBNotation
- ► oracle.xdb.dom.XDBNodeList
- ► oracle.xdb.dom.XDBAttribute
- ► oracle.xdb.dom.XDBDOMImplementation
- ► oracle.xdb.dom.XDBElement
- ► oracle.xdb.dom.XDBNamedNodeMap
- ► oracle.xdb.dom.XDBNode

These classes are used in conjunction with the XMLType class. For example, if xmldoc is an XMLType object, then xmldoc.getDocument results in a DOM document object whose class (in the case of the thick client) is oracle.xdb.dom.XDBDocument.

DB2 provides all these classes in the org.w3c.dom package in the `sqllib/java/jdk/jre/lib/xml.jar` file.

The DB2XML class is not comparable to the Oracle XMLType class, and provides completely different functionality than XMLType. The following are code suggestions for providing the functionality of most XMLType methods, if needed. There are some important considerations:

- The code snippets below show a possible approach to achieving the functionality of the XMLType methods but do not represent a complete program. Also, some code snippets provide more functionality than the XMLType method, because when applicable, for purposes of clarity and additional guidance, they show retrieval of the XML document from the database and reinsertion of the XML document into the database.

- Some XMLType methods do not have an equivalent in DB2 because of differences in Oracle XML and DB2 pureXML implementation. For example, the object-relational XML mapping in Oracle requires a specific schema to be associated with an XML column or XML table. Therefore, an XML document is associated with a specific schema. With DB2, there is no mandatory association of an XML document to a specific schema, although use of a schema hint in the instance document will dictate the schema used. These differences are noted as appropriate.

- An XMLType object may have originated outside the database, for example, a DOM document, and is going to be inserted into the database. Alternatively, an XMLType object may have been retrieved from the database. As such, the proper mapping of the Oracle XMLType method to DB2 may depend on how the XML object is being used.

createSchemaBasedXML(schemaURL)

This Oracle method associates an XMLType object with a schema.

DB2 approach

In DB2, you can either explicitly or implicitly validate the XML instance document upon insertion as the following:

Explicit:

```
INSERT INTO purchaseOrder VALUES XMLVALIDATE (XMLPARSE( DOCUMENT
:document) ACCORDING TO XMLSCHEMA URI 'http://www.test.com/po')
```

Implicit:

```
INSERT INTO purchaseOrder VALUES XMLVALIDATE (XMLPARSE( DOCUMENT
:document))
```

createXML(java.sql.Connection conn, oracle.sql.CLOB xmlval)

This Oracle method creates an XMLType object from a CLOB.

DB2 approach

If the objective is to retrieve the CLOB from the database into a DOM document, do the following:

```
xmlstring = ResultSet.getString(1);
```

```
DOMParser p = new DOMParser();
p.parse(new InputSource(new StringReader(xmlstring)));
Document doc = p.getDocument();
```

getStringVal()

This Oracle method gets the string value containing the XML data from the XMLType object.

DB2 approach

In DB2, you can do the following:

► If no internal encoding is needed:

```
String string = ResultSet.getString(1);
```

► To retrieve a string with an XML declaration including an internal encoding declaration, which will always be ISO-10646-UCS-2:

```
DB2Xml data = (DB2Xml) ResultSet.getObject(1);
String string = data.getDB2XmlString();
```

► To retrieve a string with no XML declaration:

```
String string = ResultSet.getString(1);
```

getClobVal()

This Oracle method gets the CLOB value containing the XML data from the XMLType object

DB2 approach

There is no DB2 JCC driver method to retrieve the data into a CLOB object. Instead, retrieve the XML data into an alternative data type depending on subsequent use. For example, to retrieve a String with or without an internal encoding declaration, see "getStringVal()" on page 406.

getBlobVal(character_set_id)

This Oracle method gets the BLOB value containing the XML data from the XMLType object.

DB2 approach

There is no DB2 JCC driver method to retrieve the data into a BLOB object. However, the following will retrieve the XML column from the database into a binary stream with the specified encoding:

```
DB2Xml data = (DB2Xml) ResultSet.getObject(1);
InputStream inputstream = getDB2XmlBinaryStream(targetEncoding);
```

where `targetEncoding` is a valid encoding name that is listed in the IANA Charset Registry. The encoding names that are supported by the DB2 server, and the mappings to Codices are found at the following Web site:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.apdv.embed.doc/doc/r0022549.htm

getInputStream()

This Oracle method gets an InputStream from the XMLType to allow applications to read XML data from the stream.

DB2 approach

For Data Retrieval, there are four input stream options:

▶ ResultSet.getAsciiStream - No XML internal encoding declaration is generated.

▶ ResultSet.getBinaryStream - No XML internal encoding declaration is generated.

▶ DB2Xml data = (DB2Xml) ResultSet.getObject(1);
  DB2Xml.getDB2XmlAsciiStream;

  XML internal encoding declaration of US-ASCII is added.

▶ DB2Xml data = (DB2Xml) ResultSet.getObject(1);
  InputStream inputstream = getDB2XmlBinaryStream(targetEncoding);

  See "getBlobVal(character_set_id)" on page 406 for the targetEncoding description.

writeToOutputStream

This Oracle method writes the contents of the XMLType object to an OutputStream.

DB2 approach

The following code will create an OutputStream for the retrieved XML document:

```
InputStream inputStream = resultSet.getBinaryStream(1);
BufferedOutputStream stream = new BufferedOutputStream (System.out);
byte b = -1;
try {
while ((b = (byte) inputStream.read ()) != -1)        stream.write
(b);
}
```

createXML(connection, domdocument)

This Oracle method creates an XMLType object from an instance of a DOM document.

DB2 approach

In DB2, there are three ways to implement this method:

► If the objective is to retrieve an XML document from the database, update the document via DOM manipulation, then reinsert into the database.

Consider using the DB2XMLFUNCTIONS.XMLUPDATE stored procedure to update the XML document without requiring DOM manipulation (the stored procedure performs the necessary DOM manipulation internally). See 8.7.4, "Java XML update" on page 398 for an example.

► If an XML document must be retrieved from the database, changed through DOM processing, re-inserted to the database, *and* the XMLUPDATE stored procedure cannot be used:

```
//doc is of type Document
//example below assumes 2nd parameter marker is an integer and
uniquely identifies row to be updated
StringWriter out = new StringWriter();
XMLSerializer serializer = new XMLSerializer(out,
    new OutputFormat(doc, "utf-8", true));
serializer.serialize(doc);
String  updatedXMLString  = out.toString();
PreparedStatement updatestmt = conn.prepareStatement("update
xmltable set xmlcolumn = ? where keycolumn = ?");
updateStmt.setString(1, updatedXMLString);
updatestmt.setInt(2, keycolumn_value);
stmt.executeUpdate();
```

► If a DOM document must be inserted into the database and a procedure cannot be used:

```
//doc is of type Document
//example below assumes  the 1st parameter is an integer which
uniquely identifies row to be inserted and  the 2nd parameter marker
is the XML document
StringWriter out = new StringWriter();
XMLSerializer serializer = new XMLSerializer(out,
    new OutputFormat(doc, "utf-8", true));
serializer.serialize(doc);
String  updatedXMLString  = out.toString();
PreparedStatement updatestmt = conn.prepareStatement("insert into
xmltable values (?,?)");
updatestmt.setInt(1, keycolumn_value);
```

```
updateStmt.setString(2, updatedXMLString);
stmt.executeUpdate();
```

extract(xpath_expression, nsmap)

This Oracle function is available only with the thick client to extract the specified set of nodes from the XMLType object. This set of nodes is specified by xpath_eexpression. nsmap. The map of namespaces that resolves the prefixes in the xpath_expression whose format is
"xmlns:a=abc.com xmlns:b=xyz.com".

DB2 approach

If the objective is to update the XML document in the database with the specified nodes, use the DB2XMLFUNCTIONS.XMLUPDATE stored procedure to update the document. The format of the XMLUPDATE call is:

```
DB2XMLFUNCTIONS.XMLUPDATE (commandXML, querySQL, updateSQL,
errorCode, errorMsg)
```

Using the xpath_expression and nsmap parameters for the Oracle extract procedure above, the commandXML parameter of the XMLUPDATE procedure will look like the following:

```
<updates namespaces="a:abc.com; b:xyz.com">
<update using="" col="" action="" path="xpath_expression">update
value</update>
</updates>
```

existsNode(xpath_expression, nsmap)

This Oracle method is a Boolean function to check for the existence of the given set of nodes in the XMLType object. The set of nodes is specified by the xpath_expression. nsmap. The map of namespaces that resolves the prefixes in the xpath expression and whose format is
"xmlns:a=abc.com   xmlns:b=xyz.com".

DB2 approach

In this example, two namespaces are declared, and are assigned prefixes a and b. xpath_expression is an XPath expression. db2_table is a DB2 table name. xmlcolumn is a DB2 XML column name. The data type of fn: boolean is an XQuery Boolean value of true or false, which is returned as a string to the Java program. The parseBoolean Java method converts this to a Java Boolean data type.

```
String querystring = "select xmlquery('declare namespace
a=\"abc.com\"; declare namespace b=\"xyz.com\";
```

```
    fn:boolean($i/xpath_expression)' passing xmlcolumn as \"i\") from
    db2_table";
    ResultSet rs = select.executeQuery (querystring);
    boolean boolvar = Boolean.parseBoolean(rs.getString (1));
```

XMLType transform(XMLType xsldoc, String parammap)

This Oracle function transforms the XMLType object to a new XMLType object
using the given XSL document. xsldoc (data type is XMLTYPE) is the XSL
document to be applied to the original XMLType object parammap (data type is
string)—the top level parameters to be passed to the XSL transformation. This
should be of the format "a=b c=d e=f". This can be NULL.

The return from the method is the transformed XMLType.

DB2 approach

To ensure that the XSL transformation works properly, the XSL document should
be inserted into the database with whitespace being preserved.

The code in Example 8-50 assumes the following:

► No parameters passed to the XSL transformation.

► The XSL document is retrieved from the database.

► The transformed document will be reinserted to the database as an XML
   document.

Example 8-50   DB2 approach for XMLType transform()

```
//(1) if the XML instance document undergoing transformation must first
     be retrieved from the database, then retrieve the document as
     ResultSet1, else proceed to (2)
InputStream xmlbinarystream = rs.getBinaryStream(1);
DOMParser p = new DOMParser();
p.parse(new InputSource(xmlbinarystream));
Document doc = p.getDocument();

//(2) retrieve the XSL document in ResultSet2 from the database
InputStream  xslstream = ResultSet2.getBinaryStream(1);
Source inputXSL = new StreamSource(xslstream);

//(3) create empty target Document
DocumentBuilderFactory dbf =  DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document outdoc = db.newDocument();

//(4) Set up  XSL transformer, use of templates for precomile
```

```
       is optional but recommended
TransformerFactory factory = TransformerFactory.newInstance();
Templates templates = factory.newTemplates(inputXSL);
Transformer transformer = templates.newTransformer();

//(5) Apply XSL transformation
Source domSource = new DOMSource(doc);
Result domResult = new DOMResult(outdoc);
transformer.transform(domSource, domResult);

//(6) Convert transformed document to a string for insertion to the
database
StringWriter out = new StringWriter();
XMLSerializer serializer = new XMLSerializer(out, new
OutputFormat(outdoc, "utf-8", true));
serializer.serialize(outdoc);
String  updatedXMLString  = out.toString();

//(7) Insert transformed document into database.
      1st parameter is an integer which uniquely identifies row to be
     inserted and  the 2nd parameter marker is the XML document
PreparedStatement updateStmt = conn.prepareStatement
("insert into xmltable (keycolumn, xmlcolumn) values (?, ?)");
updateStmt.setInt(1, keyvalue);
updateStmt.setString(2,updatedXMLString);
updateStmt.executeUpdate();
```

isFragment()

This Oracle Boolean function checks if the XMLType is a regular document or a document fragment.

DB2 approach

A well-formed XML document is retrieved from DB2 and a well-formed XML document must be inserted into DB2. An XML document retrieved from DB2 can be manipulated by converting to a DOM document and performing Document or DocumentFragment processing. Although not specifically related to DB2 processing, the following code will return a Boolean indicating if a document node is a document fragment. In this example `mydocnode` is a document node. Note that `getNodeType` returns 9 for a Document Node and 11 for a Document Fragment Node.

```
import org.w3c.dom.Node;
boolean boolvar = (mydocnode.getNodeType() == 11);
```

getDocumentFragment ()

This Oracle function returns a XDBDocFragment if the XMLType instance is a fragment, else null is returned.

DB2 approach

Use the same approach as shown in "isFragment()" on page 411.

isSchemaValid(schema_url, root_element)

This Oracle Boolean function checks if the XMLType is schema-based. It returns TRUE if doc is schema based, else FALSE. The input parameter schema_url is the URL of the schema to be validated against. If this is null then the document's own schema URL is used (if one exists). The parameter root_element is the name of the root element of the schema.

DB2 approach

As with Oracle, an XML document can be validated prior to being inserted into the database. This can be done by implicit validation (via schema hints in the instance document using the attributes xsi:schemaLocation or xsi:noNamespaceSchemaLocation) or by explicit validation (there are several options to explicitly specify the schema).

The following example explicitly validates an XML document in a file by validating against the schema URI, without inserting the document:

```
String xmlfile = "E:/OracleXML/DB2-Java/Client1900.xml";
File file = new File(xmlfile);
stmt.setBinaryStream(1, new FileInputStream(file),
(int)file.length());
String query = "VALUES XMLVALIDATE (XMLPARSE (DOCUMENT CAST(? AS
BLOB(10K))  ) ACCORDING TO XMLSCHEMA URI
'http://www.test.com/client')";
ResultSet rs = stmt.executeQuery();
```

createSchemaBasedXML(schemaURL)

This Oracle method creates an XMLType object with schema schemaURL.

DB2 approach

DB2 does not have a mandatory association of an XML document to a schema. Rather, an XML document can be explicitly validated against a schema, or implicitly validated against a schema using schema hints in the instance document. See DB2 schema validation techniques discussed in "schemaValidate()" on page 413.

schemaValidate()

This Oracle method validates the schema-based document against its own schema. The method has a void data type.

DB2 approach

DB2 does not have an inherent association of an XML document to a schema prior to validation. Schema validation can occur during insert, either by explicit validation against a specified schema or implicit validation against a schema using schema hints in the instance document. Schema validation can also occur after the document already resides in the database. Finally, schema validation can occur for a document that is not in the database. See the code snippet in "isSchemaValid(schema_url, root_element)" on page 412.

Schema validation during insert

The following example has these characteristics:

- ► Explicit validation is done by specifying the URI of the schema.
- ► The document is being inserted into column CLIENTS2.XMLCOL.
- ► The URL of the schema is
  http://www.test.com/client
- ► The input file is specified in string xmlfile.

```
String xmlfile = "E:/OracleXML/DB2-Java/Client1900.xml";
File file = new File(xmlfile);
String query = "INSERT INTO CLIENTS2 (XMLCOL) VALUES(XMLVALIDATE(?
ACCORDING TO XMLSCHEMA URI 'http://www.test.com/client')) ";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setBinaryStream(1, new FileInputStream(file),
(int)file.length());
stmt.executeUpdate();
```

Schema validation after document insert

```
String query = "select XMLVALIDATE (XMLCOL ACCORDING TO XMLSCHEMA
URL 'http://www.test.com/client') from CLIENTS3 WHERE ID=1";
PreparedStatement stmt = conn.prepareStatement(query);
ResultSet rs = stmt.executeQuery();
```

getSchemaURL()

This Oracle method returns the schema URL (the location of the schema to which this XMLType conforms).

DB2 approach

After an XML document has been validated against a schema, information about the schema used for validating the document is stored in the system catalog. The

following example shows how to obtain the location of the schema. In this example, the XML document for which the schema location is needed is in column user_table.xmlcolumn and the row determined by keycolumn = user_keyvalue. The location of this schema will be retrieved into value1.

```
DB2_TABLE.XMLCOLUMN
String query = "SELECT schemalocation  from syscat.xsrobjects where
objectid = (SELECT XMLXSROBJECTID(xmlcolumn) FROM db2_table WHERE
keycolumn = ?)";
PreparedStatement stmt = conn.prepareStatement(query);
updatestmt.setInt(1, keyvalue);
ResultSet rs = stmt.executeQuery();
while (rs.next ())
  {String value1 = rs.getString (1);}
```

getRootElement()

This Oracle method returns the root element name of the XMLType object.

DB2 approach

If the document has already been retrieved from DB2, the root element name can be obtained as follows:

```
InputStream  inputstream = db2xml.getDB2XmlBinaryStream("utf-8");
Document doc = builder.parse(new InputSource (inputstream));
String rootelementname = doc.getDocumentElement().getNodeName();
```

Alternatively, the root element name can be obtained for an XML document in DB2 as follows (where the XML document is in column db2_table.xml_column):

```
Select XMLQUERY('$d/name(/*)' passing t.xml_column as "d") from
db2_table t;
```

isSchemaBased()

This Oracle Boolean function checks if the XMLType object is schema based.

DB2 approach

DB2 provides the IS VALIDATED predicate. The following example is functionally correct only when a single XML document is evaluated. This example presumes that a single row in the clients table will match the predicate id = 1885. ID is an integer column and CONTACTINFO is the XML column.

```
String query = "select case count(*) when 1 then 'true' else 'false'
end from clients where id = 1885 and contactinfo is validated";
PreparedStatement stmt = conn.prepareStatement(query);
 ResultSet rs = stmt.executeQuery();
    while (rs.next ())
```

```
  {
     boolean boolvar =  rs.getBoolean(1);
  }
```

getNamespace()

This Oracle method returns a string as the default namespace of the XMLType document.

DB2 approach

The following example obtains the target namespace for the XML document(s) in db2_table.xmlcolumn.

```
String query = "SELECT targetnamespace  from syscat.xsrobjects where
objectid = (SELECT XMLXSROBJECTID(xmlcolumn) FROM db2_table)";
PreparedStatement stmt = conn.prepareStatement(query);
ResultSet rs = stmt.executeQuery();
while (rs.next ())
  {String value1 = rs.getString (1);}
```

isSchemaValidated

This Oracle Boolean method returns TRUE if the document has been validated against its schema. This method does not validate the document, but checks to see if it is validated. Method setSchemaValidated sets the validated flag for an XMLType to indicate the document is validated. This method is implicitly called by Oracle when the document passes schema validation via explicit calls to either schemaValidate or isSchemaValid.

DB2 approach

DB2 provides the IS VALIDATED predicate. See the code example in "isSchemaBased()" on page 414.

setSchemaValidated (boolean validateFlag)

This Oracle Boolean function sets the VALIDATED flag of the XMLType object (using the value of validateFlag) to indicate whether the XMLType has been validated or not.

DB2 approach

The XML document must go through the validation process to be validated. See code examples for Oracle methods "isSchemaValid(schema_url, root_element)" on page 412 and "schemaValidate()" on page 413, showing validation without inserting the document, validation during document insertion, and validation after document insertion.

getNumberVal()

This Oracle method returns the value of the XMLType instance as a NUMBER. This is only valid if the input XMLType instance contains a simple text node and is convertible to a number.

DB2 approach

Since getNumberVal returns an Oracle NUMBER, the assumption made here is that the Java program needs a numerical representation of the text node. Oracle NUMBER maps to DB2 DOUBLE and most closely maps to Java Double. Thus we map the numerical text node to a Java Double. Although DB2 SQL can perform an XMLCAST of an XML text node to a DB2 DOUBLE, it cannot be directly returned by the JCC driver as a Java Double output type. Therefore, we return the text node as a Java string and use the Java method Double.valueOf to cast the string to a Java Double.

The example here uses a simple XML document of <root> 5.5 </root> stored in table clients3.xmlcol.

```
String query = "SELECT xmlquery('$c/root/text()' passing xmlcol as
\"c\")  from clients3 where id = 2";
PreparedStatement selectStmt = conn.prepareStatement(query);
ResultSet rs = selectStmt.executeQuery();
while(rs.next() ){
Double double_var = Double.valueOf(rs.getString(1));
```

getDocument() and getDOM()

This Oracle method returns the DOM document associated with the XMLType object. This document is the org.w3c.dom.Document. The caller can perform any DOM operations on the document. The getDOM function returns Null if the document is a binary document. getDOM() is for use with the old Java APIs while getDocument() is for the unified APIs.

DB2 approach

An XML document or portion of an XML document can be retrieved from DB2 using XQuery and converted to a DOM document as shown in Example 8-51.

Example 8-51   DB2 - retrieving document

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
documentBuilder builder = dbf.newDocumentBuilder();
```

```
String query = "Select poDoc from po_xml_tab  where
XMLExists('$xmldoc/PurchaseOrder/PONO[text() = 200]' passing poDoc as
\"xmldoc\")";
PreparedStatement selectStmt = conn.prepareStatement(query);
ResultSet rs = selectStmt.executeQuery();
while(rs.next() )
        {
        InputStream  inputstream = rs.getBinaryStream(1);
        Document doc = builder.parse(new InputSource (inputstream));
      //Do desired DOM manipulation
         }
```

Alternatively, the Xerces DOM parser can be used as shown in Example 8-52.

Example 8-52   Alternative way of using Xerces DOM parser

```
import org.apache.xerces.parsers.DOMParser;
import org.xml.sax.InputSource;
String query = "Select poDoc from po_xml_tab  where
XMLExists('$xmldoc/PurchaseOrder/PONO[text() = 200]' passing poDoc as
\"xmldoc\")";
PreparedStatement selectStmt = conn.prepareStatement(query);
ResultSet rs = selectStmt.executeQuery();
while(rs.next() )
        {
           InputStream  inputstream = rs.getBinaryStream(1);
DOMParser p = new DOMParser();
      p.parse(new InputSource(inputstream));
      Document doc = p.getDocument();
          //Do desired DOM manipulation
          }
```

8.8 XML tools and utilities

In this section, we introduce the tools and utilities for working with XML in both
database management systems.

8.8.1 Oracle Enterprise Manager, DB2 Control Center

Oracle Enterprise Manager (OEM) can be used to manage Oracle XML DB.
Through the use of the hierarchical navigational features of OEM, you can:

- ► Create and manage objects that use the XMLType, including tables, columns, views, indexes and schemas
- ► Manage DML operations on XMLType objects
- ► Manage security for your XMLType objects
- ► Manage configuration of XMLtype parameters and resources

The DB2 Control Center is similar to the Oracle Enterprise Manager in that it is an interactive, graphical tool that can be used to create, manage, manipulate, and view relational objects. Along with administrating relational data, the Control Center is used to manage native XML features as well.

Control Center support for the native XML data store includes:

- ► Creating tables with XML columns
- ► Creating indexes over XML columns using the new Create Index wizard
- ► Viewing the contents of XML documents
- ► Working with XML schemas, DTDs, and other entities to validate XML documents
- ► Collecting statistics on tables containing XML columns
- ► Performing Visual Explain on XML queries including XQuery statements

The Control Center includes an XML Document Viewer that can be used to view the structure of an XML document stored in an XML column.

Figure 8-1 shows an example of the INFO column of the CUSTOMER table on the sample database displayed in the Tree View. The Viewer supports to ability to find and examine elements and other characteristics of the document.
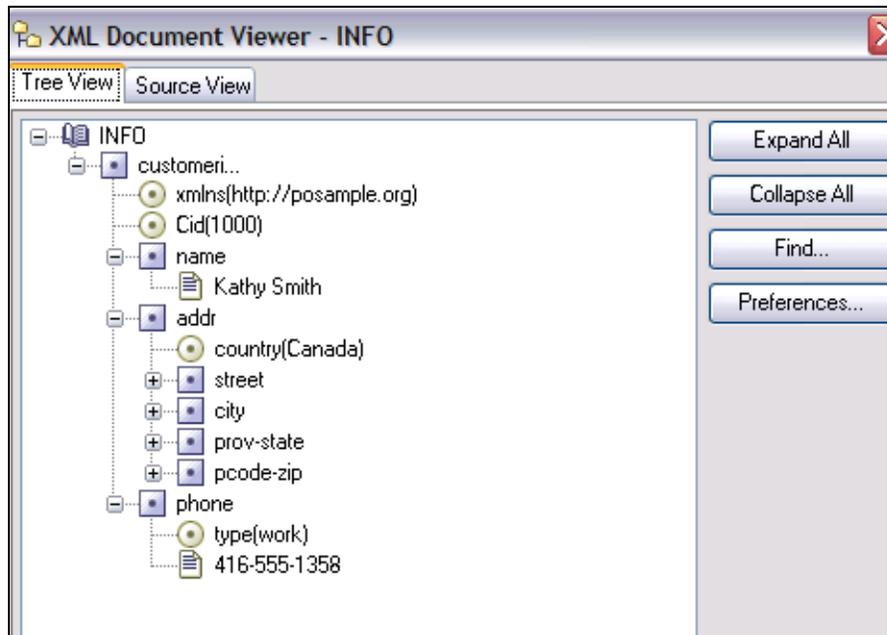
Figure 8-1   XML document Tree View

In addition, the Source View offers a view of the original document as shown in Figure 8-2 on page 420.
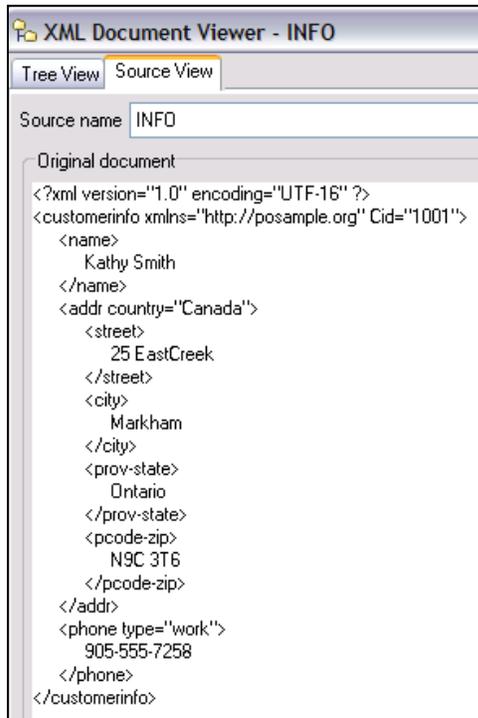
Figure 8-2   XML document Source View

8.8.2 Oracle JDeveloper, DB2 Developer Workbench

JDeveloper is used to support all phases of the application development life cycle for J2EE applications. JDeveloper allows developers to create, edit, and transform XML documents and simplifies the task of working with Java and XML data at the same time.

The DB2 Developer Workbench (DWB) is an Eclipse-based tool that is used for developing, managing, debugging, tuning, and deploying Java applications, stored procedures and user defined functions. In addition, the DWB includes support for XML data in the following ways:

► Building and running stored procedures using the XML data type
► Viewing and manipulating XML data
► Validating and managing XML documents
► Viewing and managing XML schemas
► Annotate XML schema for decomposition
► Run queries using the XML data type
► Providing a drag and drop visual XQuery Builder

8.9 Best practices

For information on best practices and performance considerations with using DB2 9 pureXML, refer to the article "15 best practices for pureXML performance in DB2 9" at:

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0610nicola

9

Script conversion

In this chapter, we discuss the administration script conversion from an Oracle environment to a DB2 environment. We cover the following:

► Data load script conversion
► Data Pump script conversion
► Administration script conversion
► Report tools available to DB2

9.1 Data load scripts

Data from other systems with no direct connection are mostly loaded into the database with a load utility. Oracle uses the SQL*Loader in combination with a control file. With DB2 you can use:

▶ DB2 Load utility

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data. The load process consists of four phases:

– Load data

– Build indexes

– Delete rows with a unique key violation or a DATALINK violation

– Copy index data from a system temporary table space to the original table space

▶ DB2 Import utility

The import utility inserts data from an input file into a table or updatable view. If the table or view receiving the imported data already contains data, you can either replace it or append it to the existing data.

The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs. The load utility does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes).

The Oracle loader utility has two modes: direct path and conventional path. Oracle's direct path closely resembles the DB2 load utility while the conventional path closely resembles the DB2 import utility.

Oracle defines its own Data Definition Language (DDL) to load data from a file into the database. The DDL is different from the DB2 syntax.

For the most frequently used commands in an Oracle SQL*Loader control file, this chapter describes scripts to convert the control files to DB2 load or DB2 import files.

We recommend that you migrate the more complex Oracle control files manually and implement workarounds for the functionality not available in the DB2 load command. For detailed information about the DB2 load command and DB2 import command, see Chapter 6, "Data conversion" on page 265.

9.1.1 Data load migration approach

We propose the following approach in converting Oracle load commands:

1. Examine the Oracle control file, the datafile format, and the target table.

2. Check for alternatives, such as the use of database links, DB2 WebShpere Federation Server, and so on, to improve the data import.

3. Convert the Oracle control file to the proper DB2 command.

In the next section we show an easy way to convert the scripts automatically. Note the variety of DB2 load options.

9.1.2 Loading fixed-format fields

Example 9-1 shows a simple control file to load data into the table accounts of the ORA_EMP database. The file contains a reference to the accounts.dat datafile, the target table, the fixed data positions, and its data types.

*Example 9-1   SQL*Loader control file with fixed-format fields*

```
LOAD DATA
INFILE '/home/ora_usr/accounts.dat'
INTO TABLE accounts
(  acct_id             POSITION(0001:0003) NUMBER
  ,dept_code           POSITION(0004:0006) CHAR
  ,acct_desc           POSITION(0009:0100) VARCHAR2
  ,max_employees       POSITION(0101:0103) NUMBER
  ,current_employees   POSITION(0104:0106) NUMBER
  ,num_projects        POSITION(0107:0107) NUMBER )
```

Example 9-2 shows the corresponding DB2 load command. The details of the load command are almost the same. The most sensitive part during the migration is the correct conversion of the POSITION() specification. To avoid errors, we recommend that you migrate at least this part of the control file automatically. In Appendix E, "Converter for SQL*Loader" on page 673 is the Perl script conv_ctl.pl to convert simple SQL*Loader files to DB2 load files.

Example 9-2   DB2 Load file for table ACCOUNTS

```
LOAD FROM '/home/ora_usr/accounts.dat' of ASC
METHOD L
        ( 0001 0003
         ,0004 0006
         ,0009 0100
         ,0101 0103
         ,0104 0106 )
INSERT INTO accounts
```

```
        ( acct_id
         ,dept_code
         ,acct_desc
         ,max_employees
         ,current_employees );
```

9.1.3 Loading variable-length data

Example 9-3 is a simple Oracle control file to load data with a variable length into
the ACCOUNTS table. The delimiter in this sample is a comma, the fields may be
enclosed in double quotes. The accounts.dat datafile looks like this:

```
101,"ACT","Major Bank Co.",30,11,4
301,"ACT","Large Telco Inc.",30,0,4
101,"IT","Huge Software Co.",50,0,4
203,"MKT","Basic Insurance Co.",15,0,3
```

*Example 9-3   SQL*Loader control file with variable-length fields*

```
LOAD DATA
INFILE '/home/ora_usr/accounts.dat'
INTO TABLE accounts
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
( acct_id
 ,dept_code
 ,acct_desc
 ,max_employees
 ,current_employees
 ,num_projects )
```

The Perl script conv_ctl.pl in Appendix E, "Converter for SQL*Loader" on
page 673 converts the control file to a proper DB2 Load file. Modify the
generated DB2 load command if you need additional features. Example 9-4
shows the DB2 load command.

Example 9-4   DB2 load commend with variable-length fields

```
LOAD FROM '/home/ora_usr/accounts.dat' of ASC
MODIFIED BY COLDEL,
METHOD P (1, 2, 3, 4, 5 )
INSERT INTO accounts
        ( acct_id
         ,dept_code
         ,acct_desc
         ,max_employees
         ,current_employees )!
```

9.1.4 Initializations in the Oracle SQL*Loader control file

In the Oracle SQL*Loader file you are able to specify initialization values for defined data.

Example 9-5 shows the same sample as in Example 9-3, but with conditions for the columns dept_code and current_employees.

*Example 9-5   SQL*Loader control file with conditions for the ACCOUNTS table*

```
LOAD DATA
CHARACTERSET we8pc850
INFILE '/home/ora_usr/accounts.dat'
INTO TABLE accounts
( acct_id            POSITION(0001:0003) NUMBER
 ,dept_code          POSITION(0004:0006) CHAR DEFAULTIF dept_code=" "
 ,acct_desc          POSITION(0009:0100) VARCHAR2
 ,max_employees      POSITION(0101:0103) NUMBER
 ,current_employees  POSITION(0104:0106) NUMBER NULLIF current_employees="0"
 ,num_projects       POSITION(0107:0107) NUMBER )
```

The column dept_code is set to its default value if the loaded data is blank and the variable current_employees is set to null if the loaded data is 0. Such data manipulations are not allowed with DB2 load. To achieve the data manipulation, run a separate UPDATE command after the data is loaded:

```
UPDATE accounts
SET dept_code=DEFAULT
WHERE dept_code=" ");

UPDATE accounts
SET current_employees=NULL
WHERE current_employees="0");
```

In Appendix E, "Converter for SQL*Loader" on page 673 is the Perl script gen_load_update.pl used to generate DB2 UPDATE commands from Oracle control files.

9.1.5 Loading data into multiple tables

To achieve data loads into multiple tables, migrate the Oracle commands in this sequence:

1. Separate the load commands to single commands per script and file.

2. Convert the separated load commands. Consider the suggestions we made in the previous chapters.

3. Control the DB2 commands with scripts (sh, batch, and so on.) according to the needs of your system.

9.2 Oracle Data Pump scripts

Oracle provides Data Pump API for data manipulation between 10g databases. The Data Pump API can be accessed through the expdp and impdb utilities, and the interface closely resembles the export (EXP) and import (IMP) utilities, respectively.

Data Pump can only read and write to a proprietary format file that cannot be read either by Oracle 9i databases or DB2. For flat text file operations, you have to use the UTL_FILE package or SQLPlus utility to unload data, and SQL*Loader or external tables using the DATA_LOADER driver to load data. Data Pump API generates server-side dump files, and it is necessary to use Oracle directories as a storage destination for Data Pump jobs.

The majority of Oracle Data Pump functions can be accomplished using the DB2 data move utility db2move. In conjunction with other DB2 utilities such as db2look, you can convert the scripts using DATA Pump features to DB2. DB2 utilities that provide similar Data Pump functionality are as follows:

► db2move

db2move is used for data movement between DB2 databases through the use of PC/IXF format files. DB2 db2move has four actions:

– EXPORT - Exports all tables that meet the filtering criteria.
– IMPORT - Imports all tables listed in the internal staging file db2move.lst.
– LOAD - Loads all tables listed in the internal staging file db2move.lst.
– COPY - Duplicates one or more schemas into a target database.

db2move calls EXPORT, IMPORT, and LOAD APIs depending on the requested operation. Therefore, the user must have the authorization required by those APIs or the operation will fail.

db2move COPY action is used to directly transfer one schema from a source database to a target database. When using db2move with COPY action, the following are required:

– A list of schemas to copy
– Target database connection with username and password
– Copy mode:
  • DDL_ONLY - Only the objects DDL will be transferred.
  • LOAD_ONLY - Only the data will be transferred.
  • DDL_AND_LOAD - Both DDL and data will be transferred,
– Copy modifier

You can also remap the schema owner of the objects, or even the table spaces that the objects will be stored in. db2move with COPY action facilitates the creation of schema templates that could be easily transferred between different servers. db2move COPY action does not transfer object privileges and statistics. If necessary, you can use db2look to generate scripts for the object privileges and statistics and then apply these scripts into the target database.

DB2 db2move does not allow to transfer schemas within the same database. For this situation, DB2 provides the ADMIN_COPY_SCHEMA procedure.

► db2look

   db2look is used for object Data Definition Language (DDL) extraction. It can also extract authorization DDL (grants, for example), database manager configuration parameters, and database configuration parameters.

► EXPORT

   EXPORT unloads data from DB2 to one or more files stored outside the database.

► IMPORT

   IMPORT reads data from a file stored outside the DB2 database and inserts it into the database using SQL INSERT statements.

► LOAD

   LOAD reads data from a file stored outside the DB2 database and writes the formatted data pages directly into the database.

Data Pump dumpfiles can also be considered as logical backups. If Data Pump scripts are used for logical backup operations, we recommend that you use DB2 BACKUP and RESTORE commands for performance and flexibility reasons, although the utilities listed above can also be used for this purpose. An example of conversion of Data Pump scripts to DB2 BACKUP commands can be found in 9.3.4, "Backup scripts conversion" on page 441.

Both Data Pump and DB2 utilities provide support for parallel data unload and reload. Import jobs can be restarted in both Data Pump and DB2 utilities.

There is no general rule for converting Data Pump scripts to DB2 utilities. Depending on how the Data Pump function is used in a script, one or more DB2 utilities may be required to achieve similar functionality. We recommend that you migrate the Oracle Data Pump scripts manually using correspondent DB2 tools to perform similar activities.

9.2.1 Data Pump migration approach

We propose the following approach to convert Data Pump scripts to DB2:

1. Examine all the operations performed by the Data Pump jobs.

2. Check for alternatives, such as DB2 BACKUP and RESTORE commands for logical backup operations, or DB2 Information Integrator for federated data movement.

3. Rewrite Data Pump scripts using DB2 utilities to achieve similar functionality.

In the following sections, we show how you can convert Oracle Data Pump scripts that perform schema transfer and data export and import to DB2 using a full schema transfer example. We illustrate the full schema transfer using db2move with EXPORT, LOAD, and COPY actions as well as using DB2 function ADMIN_COPY_SCHEMA. You can use db2move COPY, EXPORT, or LOAD options to perform schema transfer. Using db2move with COPY action is more straightforward than the other two actions. Along with the discussion on import and export operations, we list the Data Pump usages and how to implement these tasks using DB2 utilities.

In order to use Oracle Data Pump, you need to use an Oracle directory with proper privileges. We created an Oracle directory called data_pump_dir1 in both source and target databases to be used in our examples using the following commands:

```
create directory data_pump_dir1 as '/oradata/dpump';
grant read, write on directory data_pump_dir1 to ora_usr;
```

9.2.2 Transferring a schema

db2move can be used to transfer a schema between DB2 databases. To duplicate schemas within a database, the ADMIN_COPY_SCHEMA procedure can be used.

db2move with COPY action

db2move with COPY action can be used to transfer a schema between different databases. The great benefit of COPY action over EXPORT and LOAD actions is that you can perform both the unload and the load of the data with a single command, and you can also specify different schemas and table spaces in the target database.

Example 9-6 shows an example of schema export using Data Pump. Two files will be generated: dp_export.dmp for data and dp_export.log for the log of the export operation.

Example 9-6   Full database export using Data Pump

```
expdp ora_usr/ora_usr@DSV dumpfile=dp_export.dmp logfile=dp_export.log
schemas=ORA_USR directory=data_pump_dir1
```

Once the export is completed, the dumpfile must be transferred to the target database server if source and target databases reside on different machines. Moreover, the dumpfile must be accessible through an Oracle directory at the target database, and the user performing the import must have the proper privileges to access the Oracle directory.

Example 9-7 shows an example of Data Pump schema import using the dumpfile generated in Example 9-6.

Example 9-7   Importing a full schema using Data Pump

```
impdb ora_usr/ora_usr@PRD dumpfile=dp_export.dmp logfile=dp_import.log
directory=data_pump_dir1
```

Using DB2 db2move utility with COPY action, you can transfer the schema using a single command, as show in Example 9-8. In this example, the schema *ora_usr* will be copied from *db2_dsv* database to *db2_prd* database using username *user_prd* and password *pwd_prd* in the target database.

Example 9-8   Transfer a schema using db2move with COPY action

```
db2move db2_dsv copy -sn ora_usr -co target_db db2_prd user user_prd
using pwd_prd
```

db2move with COPY action does not transfer grants or statistics from the source database to the target database. If you need this information, you can use the db2look utility in conjunction with db2move, as shown in Example 9-9 .

Example 9-9   Using db2look and db2copy to transfer schema and privileges

```
--1. To extract objects DDL in the source database, execute:

db2look -d db2_src -z ora_usr -xd -m -o ora_usr.sql

--2. To transfer the schema ora_usr, execute, in the source database:
db2move db2_dsv copy -sn ora_usr -co target_db db2_prd user user_prd using
pwd_prd

--3. Change the connection string in the DDL file
--   to connect to the target database.
--   For example, change "CONNECT TO DB2_DSV" to "CONNECT TO DB2_PRD"

--4. To include statistics and authorizations to the schema transfered,
```

```
--   execute the DDL in the target database:
db2 -tvf ora_usr.sql > create_ddl.log
```

The db2look -xd parameter specifies that all the authorization DDL, including object authorizations that were granted by SYSIBM at the object creation time, will be generated. The db2look parameter -m specifies the UPDATE statements required to replicate the statistics on tables, statistical views, columns, and will generate indexes.

The following files are generated by db2move with COPY action:

► COPYSCHEMA.timestamp.msg - Contains the log messages generated during the db2move copy operation.

► COPYSCHEMA.timestamp.err - Contains the errors generated during the db2move copy operation.

► LOADTABLE.timestamp.msg - Contains the log messages generated during the LOAD phase of the db2move copy operation.

► LOADTABLE.timestamp.err - Contains the errors generated during the LOAD phase of the db2move copy operation.

For more information about the **db2move** command, refer to *Data Movement Utilities Guide and Reference,* SC10-4227.

ADMIN_COPY_SCHEMA procedure

The ADMIN_COPY_SCHEMA procedure, introduced in DB2 9, is used to duplicate schemas within the same database. The new target schema objects are created using the same object names as the source with the new schema qualifier. You can transfer tables with or without the original data.

In order for the schema duplication to be successful, the user ID calling this procedure must have the appropriate object creation authorities, including both the authority to select from the source tables, and the authority to perform a load.

You must provide a table schema name and a table name to store error information about objects that cannot be copied during the ADMIN_COPY_SCHEMA execution. If the error information table cannot be created or already exists, the procedure fails and an error message is returned. The error information table must be dropped between ADMIN_COPY_SCHEMA executions, or a new error table must be chosen.

Example 9-10 shows an example of schema copy within a database. In this example, we use COPYSCHEMA table space and the COPYERROR table to store error log messages.

Example 9-10   Transfer a schema using the ADMIN_COPY_SCHEMA procedure

```
CALL SYSPROC.ADMIN_COPY_SCHEMA('ORA_USR', 'ORA_USR2',
   'COPY', NULL, null, null, 'COPYSCHEMA', 'COPYERROR')
```

The output should be similar to the one shown in Example 9-11.

Example 9-11   ADMIN_COPY_SCHEMA function output

```
CALL SYSPROC.ADMIN_COPY_SCHEMA('ORA_USR', 'ORA_USR2', 'COPY', NULL, null, null,
'COPYSCHEMA', 'COPYERROR')
Value of output parameters
  --------------------------
  Parameter Name  : ERRORTABSCHEMA
  Parameter Value : -
  Parameter Name  : ERRORTABNAME
  Parameter Value : -
  Return Status = 0
```

For more information about the ADMIN_COPY_SCHEMA procedure, refer to *Administrative SQL Routines and Views,* SC10-4293.

9.2.3 Export data operations

Table 9-1 shows some Data Pump usage in exporting table data or database metadata, and how it can be implemented in DB2.

Table 9-1   Export functionality mapping

Data Pump usage	DB2 conversion recommended
Full database export	▶ db2move with EXPORT action for data ▶ db2look for objects DDL
Full schema export	▶ In the same database: ADMIN_COPY_SCHEMA procedure or ▶ db2move with COPY action for data [a] ▶ db2look for privileges and statistics or ▶ db2move with EXPORT action for data ▶ db2look for objects DDL
Full table export	▶ db2move with EXPORT action for data ▶ db2look for objects DDL
Partial table export	▶ db2move with EXPORT for data ▶ db2look for objects DDL

Data Pump usage	DB2 conversion recommended
Tablespace export	► db2move with EXPORT action for data ► db2look for objects DDL
Generate DDL files	db2look
Direct path export	DB2 High Performance Unload (HPU) utility

a. db2move with COPY action will perform both export and import operations in a single command. If you only want to export the data, you must use db2move with EXPORT action.

Example 9-12 shows how to perform a full schema export using db2move with EXPORT action and db2look for the privileges associated with the objects. In contrast to Example 9-8, the data will be exported to the local file system where the **db2move** command is being executed, and will not be automatically transferred to the target database.

Example 9-12   Full schema export using DB2 db2move and db2look

```
--Export the objects DDL using db2look:
db2look -d db2_dsv -e -z ora_usr -o ora_usr.sql

--Export the data using db2move:
db2move db2_dsv export -sn ora_usr
```

In Example 9-12, the db2move utility generates an EXPORT.out file for the operation log, the db2move.lst file for the internal staging information, and several files for the exported objects. The parameter *-sn* was specified to export only the tables under the schema *ora_usr*.

The db2look utility will generate the ora_usr_ddl. file containing the DDL for database objects. The parameter *-e* was specified to extract the database object DDL, and the parameter *-z* was specified to limit DDL generation for objects under the schema *ora_usr*.

The DB2 db2move and db2look utilities provide several parameters to control the scope of the data being exported. For more information about the **db2move** command, refer to *Data Movement Utilities Guide and Reference,* SC10-4227. For more information about the **db2look** command, refer to *Troubleshooting Guide*, GC10-4240.

DB2 also provides the DB2 High Performance Unload (HPU) tool for high-speed unloading operations. It is designed for ease of use and flexibility, allowing you to unload and extract data for movement across enterprise systems.

For more information on DB2 HPU features, refer to:

http://www.ibm.com/software/data/db2imstools/db2tools/db2hpu/

9.2.4 Import data functionality

Table 9-2 shows some Data Pump usage in import operations and how it can be implemented in DB2.

Table 9-2   Import functionality mapping

Data Pump usage	DB2 conversion recommended
Full database import	► Execute the DDL file generated by db2look to create all required objects and privileges. ► Use db2move with LOAD action to move data.
Full schema import	► In the same database: ADMIN_COPY_SCHEMA procedure. or ► db2move with COPY action for schema and data. ► Execute the DDL file generated by db2look for privileges and statistics. [a] or ► Execute the DDL file generated by db2look to create all required objects and privileges. ► Use db2move utility to move data.
Partial table import	► Execute the DDL file generated by db2look to create all required objects and privileges. ► Use LOAD or IMPORT utilities to move data.
Table space import	► Execute the DDL file generated by db2look to create all required objects. ► Use db2move with LOAD action to move the data.
Transportable table space	Not supported coping the table space containers as in Data Pump. To achieve similar functionality in DB2: ► Execute the DDL file generated by db2look to create all required objects. ► Use db2move to move the data.
Direct path import	db2move with LOAD action or LOAD utility.

Data Pump usage	DB2 conversion recommended
Remap datafile during import	► Change the DDL file generated by db2look to reference the new table space containers. ► Execute the modified DDL file to create all required objects. ► Use db2move with LOAD action to move the data. Similar functionality can be achieved using DB2 BACKUP and RESTORE utilities.
Remap table space during import	► db2move with COPY action, tablespace_map clause for data ► Execute the DDL file generated by db2look for privileges and statistics or ► Change the DDL file generated by db2look to reference the new table spaces. ► Execute the modified DDL file to create all required objects. ► Use db2move with LOAD action to move the data.
Remap schema during import	► db2move with COPY action, schema_map clause for data ► Execute the DDL file generated by db2look for privileges and statistics or ► Change the DDL file generated by db2look to reference the new schema. ► Execute the modified DDL file to create all required objects. ► Use db2move with LOAD action to move the data.

a. db2move with COPY action performs both export and import operations in a single command. If you already have the data exported, you must use db2move with LOAD or IMPORT actions to load or import the data.

Full schema import using db2move

To perform a full schema import using DB2 db2look and db2move with LOAD action utilities, perform the following steps:

1. Review the DDL generated by db2look. In Example 9-12, the generated file was ora_usr.sql. If necessary, modify any statement in this file for your environment.

2. Run the DDL schema using the DB2 Command Line Processor (CLP) to create objects.

3. Run the **db2move** command to load the data into the database.

Example 9-13 shows an example of schema import using DB2 utilities.

Example 9-13   Importing full schema using DB2 utilities

```
--Create all objects using:
db2 -tvf ora_usr.sql > create_ddl.log

--Import the data using:
db2move db2_prd load -sn ora_usr
```

The DB2 db2move utility provides several parameters to control the scope of the data being imported. For more information on the **db2move** command, refer to *Data Movement Utilities Guide and Reference,* SC10-4227.

9.3 Administration scripts

Database administrators use many administration scripts to schedule and run their day-to-day activities. Conversion of these scripts from Oracle to DB2 also plays an important role in migration. This section discusses how certain administration commands and DDLs can be converted to DB2 commands and DDLs.

9.3.1 Dynamic performance views and table function

Oracle provides dynamic performance views that are updated dynamically by the Oracle instance. Dynamic performance views are prefixed with `V_$` and have public synonyms created with the `V$` prefix. Dynamic performance views are accessed by the database administrators, and are listed in V$FIXED_TABLE. These views are used by the database administrators to monitor the database. The information provided by these views is not static, and is dynamically updated by the database and instance. Examples of such views are V$INSTANCE, V$DATABASE, V$TABLESPACE, V$DATAFILE, and so on.

DB2 administrative views provide an easy-to-use application programming interface to DB2 administrative functions through SQL. The administrative views fall into three categories:

► Views based on catalog views
► Views based on table functions with no input parameters
► Views based on table functions with one or more input parameters

The administrative views are the preferred and only documented interfaces for the views based on catalog views and the views based on table functions, with no input parameters because the table functions do not provide any additional

information or performance benefits. For administrative views based on table functions with one or more input parameters, both the administrative view and the table function can be used, each achieving a different goal.

The DB2 administrative views can be considered to be equivalent to V$ views in Oracle. Though the information provided by the V$ views and the administrative views cannot be exactly the same, some information is common, and both return dynamic data.

For example, to obtain the information about applications connected to the database form Oracle V$ views, use the following query:

```
SELECT * FROM V$SESSION
```

An equivalent query to query data from DB2 administrative views is as follows:

```
SELECT * FROM SYSIBMADM.SNAPAPPL
```

An equivalent query to query data from DB2 administrative views is as follows:

```
SELECT * FROM TABLE (SNAP_GET_APPL(CAST(NULL AS VARCHAR(128)),-1))
AS T
```

Table 9-3 shows some of the V$ views and the equivalent administrative views or table functions.

Table 9-3   Oracle V$ views and DB2 administrative views and table functions

Oracle	DB2
V$INSTANCE	SNAPDBM administrative view or SNAP_GET_DBM table function
V$DATABASE	SNAPDB administrative view or SNAP_GET_DB_V91 table function
V$TABLESPACE	SNAPTBSP administrative view or SNAP_GET_TBSP_V91 table function
V$DATAFILE	SNAPCONTAINER administrative view or SNAP_GET_CONTAINER_V91 table function
V$SESSION	SNAPAPPL administrative view or SNAP_GET_APPL table function

Oracle	DB2
V$SQLTEXT	SNAPSTMT administrative view or SNAP_GET_STMT table function
V$LOCK	SNAPLOCK administrative view or SNAP_GET_LOCK table function
V$SYSSTAT -- information for data buffer	SNAPBP administrative view or SNAP_GET_BP table function
V$SESSION_LONGOPS	LONG_RUNNING_SQL administrative view

Note: The administrative views are catalogued in the SYSIBMADM schema, and table functions are cataloged in the SYSPROC schema. You need SELECT privileges to access these objects.

For more information on DB2 administrative views and table functions, refer to *Administrative SQL Routines and Views,* SC10-4293.

9.3.2 System catalog views

It is common that database administrators (DBAs) have administrative scripts to retrieve information about data dictionary objects. Table 9-4 shows some of the commonly used views available in Oracle Data Dictionary and their DB2 catalog equivalents.

Table 9-4   Useful data dictionary views

Oracle	DB2
DBA_TABLESPACES	SYSCAT.TABLESPACES
DBA_DATA_FILES	SYSIBMADM.CONTAINER_UTILIZATION
DBA_TABLES	SYSCAT.TABLES
DBA_TAB_COLUMNS	SYSCAT.COLUMNS
DBA_TAB_PRIVS	SYSCAT.TABAUTH
DBA_INDEXES	SYSCAT.INDEXES
DBA_IND_COLUMNS	SYSCAT.INDEXCOLUSE
DBA_TRIGGERS	SYSCAT.TRIGGERS

Oracle	DB2
DBA_VIEWS	SYSCAT.VIEWS
DBA_SEQUENCES	SYSCAT.SEQUENCES
DBA_PROCEDURES	SYSCAT.ROUTINES
DBA_CONSTRAINTS	SYSCAT.TABCONST
DBA_CONS_COLUMNS	SYSCAT.COLCHECKS
DBA_TAB_PRIVS	SYSIBMADM.PRIVILEGES

9.3.3 Frequently used commands and DDLs by DBA

It is useful for DBAs to know how certain important administrative commands and DDLs are used in Oracle and their equivalents in DB2. Table 9-5 lists a few frequently used DDLs and commands frequently used by DBAs on a day-to-day basis. This is helpful for the DBAs to create certain maintenance scripts in the DB2 environment.

Table 9-5   Commands and DDL conversion

Oracle	DB2
CREATE DATABASE ORA_EMP MAXLOGFILES 2 MAXLOGMEMBERS 3 LOGFILE GROUP 1 ('/disk1/log1a.log','/disk1/log1b.log','/disk1/log1c.log') SIZE 1M, GROUP 2 ('/disk2/log2a.log','/disk2/log2b.log','/disk2/log2c.log') SIZE 1M DATAFILE '/disk1/system01.dbf' SIZE 100M;	CREATE DATABASE DB2_EMP CATALOG TABLESPACE MANAGED BY DATABASE USING (FILE '/disk1/syscatspace.dbf' 25600); UPDATE DB CFG FOR DB2_EMP USING LOGPRIMARY 2; UPDATE DB CFG FOR DB2_EMP USING NEWLOGPATH '/disk1'; UPDATE DB CFG FOR DB2_EMP USING MIRRORLOGPATH '/disk2'; UPDATE DB CFG FOR DB2_EMP USING LOGFILSIZ 256;
CREATE TABLESPACE USER_DATA_TBS DATAFILE '/disk1/user_data_tbs_01.dbf' SIZE 50M MINIMUM EXTENT 1M PERMANENT	CREATE REGULAR TABLESPACE USER_DATA_TBS MANAGED BY DATABASE USING (FILE '/disk1/user_data_tbs_01.dbf' 12800) EXTENTSIZE 1M

Oracle	DB2
CREATE TABLESPACE USER_TEMP_TBS DATAFILE '/disk1/user_temp_tbs_01.dbf' SIZE 50M MINIMUM EXTENT 1M TEMPORARY	CREATE USER TEMPORARY TABLESPACE USER_TEMP_TBS MANAGED BY DATABASE USING (FILE '/disk1/user_data_tbs_01.dbf' 12800) EXTENTSIZE 1M
CREATE TABLESPACE USER_LOB_TBS DATAFILE '/disk1/user_lob_tbs_01.dbf' SIZE 100M MINIMUM EXTENT 1M PERMANENT	CREATE LARGE TABLESPACE USER_TEMP_TBS MANAGED BY DATABASE USING (FILE '/disk1/user_data_tbs_01.dbf' 25600) EXTENTSIZE 1M
CREATE USER ORA_USR IDENTIFIED BY EXTERNALLY	CREATE SCHEMA DB2_USR AUTHORIZATION DB2_USR -- identifies o/s user db2_usr
GRANT CREATE SESSION, CREATE TABLE TO ORA_USR;	GRANT CONNECT, CREATETAB ON DATABASE TO USER DB2_USR
REVOKE CONNECT FROM ORA_USR	REVOKE CONNECT ON DATABASE FROM USER DB2_USR
ALTER SYSTEM KILL SESSION ('sid','serial') IMMEDIATE	FORCE APPLICATION (appl handle) MODE ASYNC
ALTER SYSTEM SUSPEND	SET WRITE SUSPEND FOR DB
ALTER SYSTEM QUIESCE RESTRICTED	QUIESCE DB database name
ALTER SYSTEM ARCHIVE LOG	ARCHIVE LOG FOR DB database name
ALTER SYSTEM FLUSH SHARED_POOL	FLUSH PACKAGE CACHE DYNAMIC
DBMS_SPACE_ADMIN package	INSPECT database
SET TRANSACTION ISOLATION LEVEL	CHANGE ISOLATION LEVEL
ANALYZE TABLE command	RUNSTATS ON TABLE

9.3.4 Backup scripts conversion

Oracle uses two levels of backups: datafile backup and logical backup using the export or expdp utilities. DB2 uses the BACKUP database command to back up the database. This can be considered as being equivalent to the Oracle export or expdb utilities. So, the logical backup scripts using export or expdp utilities in

Oracle can be converted to DB2 backup scripts. Example 9-14 shows a sample shell script used to export the database using the export utility.

Example 9-14   Export script in Oracle using the export utility

```
#!/usr/bin/ksh
##### -- Oracle daily logical backup script --
today=`date +%C%y%m%d`
dumpfile=/oracle/backup/exp_$today.dmp
logfile=/oracle/backup/exp_$today.log

ORACLE_HOME=/u01/app/oracle/product/10.2.0/db_1
ORACLE_SID=ORA_EMP
NLS_LANG=AMERICAN_AMERICA.WE8ISO8859P1
export ORACLE_SID
export ORACLE_HOME
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_LANG

exp system/manager file=$dumpfile log=$logfile buffer=10485760 full=y
```

Example 9-15 shows a sample shell script used to export the database using the expdp utility.

Example 9-15   Export script in Oracle using expdp utility

```
#!/usr/bin/ksh
##### -- Oracle daily logical backup script using expdpb --
today=`date +%C%y%m%d`
dumpfile=expdp_$today.dmp
logfile=expdp_$today.log
directory=data_pump_dir1

ORACLE_HOME=/u01/app/oracle/product/10.2.0/db_1
ORACLE_SID=ORA_EMP
NLS_LANG=AMERICAN_AMERICA.WE8ISO8859P1
export ORACLE_SID
export ORACLE_HOME
export PATH=$ORACLE_HOME/bin:$PATH
export NLS_LANG

expdp system/manager dumpfile=$dumpfile logfile=$logfile
```

Example 9-16 shows the equivalent shell script used in the DB2 environment for both Example 9-14 and Example 9-15 to back up the database.

Example 9-16   BACKUP database script in DB2

```
#!/usr/bin/ksh
###### DB2 daily backup script ##########
today=`date +%C%y%m%d`
logfile=/db2/backup/db2bkup_log$today.log
BACKUPDIR=/db2/backup
db1=DB2_EMP

DB2INSTANCE=db2inst1
export DB2INSTANCE

db2 backup db $db1 online to $BACKUPDIR with 4 buffers buffer 512>> $logfile;
```

The examples show how to back up the database onto disk. Like Oracle, DB2 also supports backing up the database directly to tape. For more information, refer to *Data Recovery and High Availability Guide and Reference,* SC10-4227.

9.4 Tools and wizards

DB2 ships with many tools and wizards that help the database administrators to maintain and administer the database. These wizards and tools are also used to generate the scripts for database administration. Examples of such wizards are Backup Wizard, Create Database Wizard, Create Tablespace Wizard, Design Advisor, Load Wizard, and more. These wizards and tools can be accessed from the DB2 Control Center. Figure 9-1 shows how the Backup Wizard is used to generate the backup script. The generated script can be exported to the output file using the Export Scripts option.
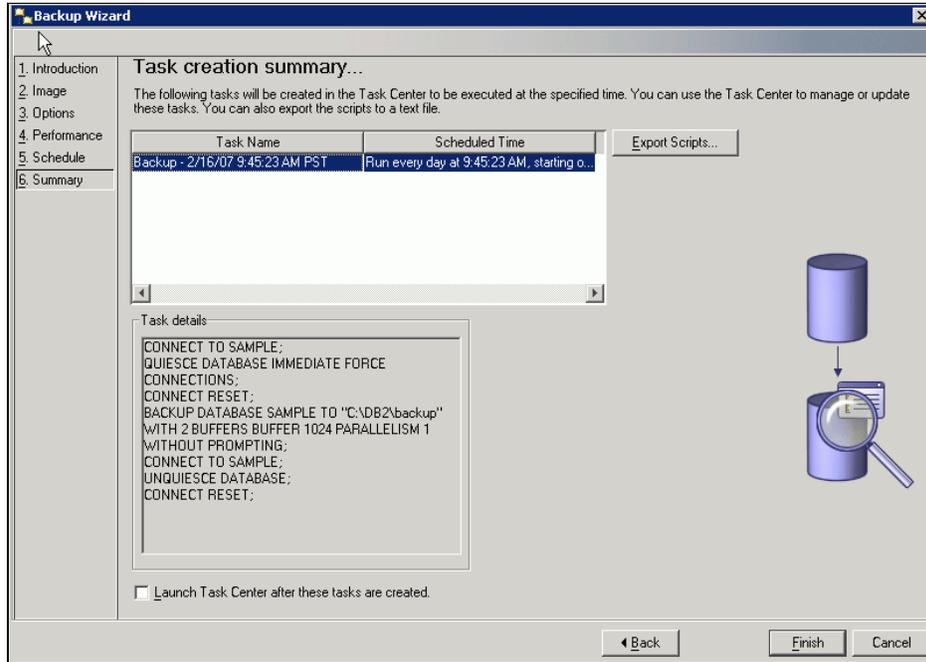
Figure 9-1   Backup Wizard

The scripts or jobs generated by the wizards can be scheduled to run and are maintained in the Task Center. Figure 9-2 shows the Task Center. The Task Center is also used to notify administrators on the status of a completed job. For more information about these wizards and tools, refer to Chapter 7 "Using the DB2 administration tools" in the *Administration Guide: Implementation,* SC10-4221.
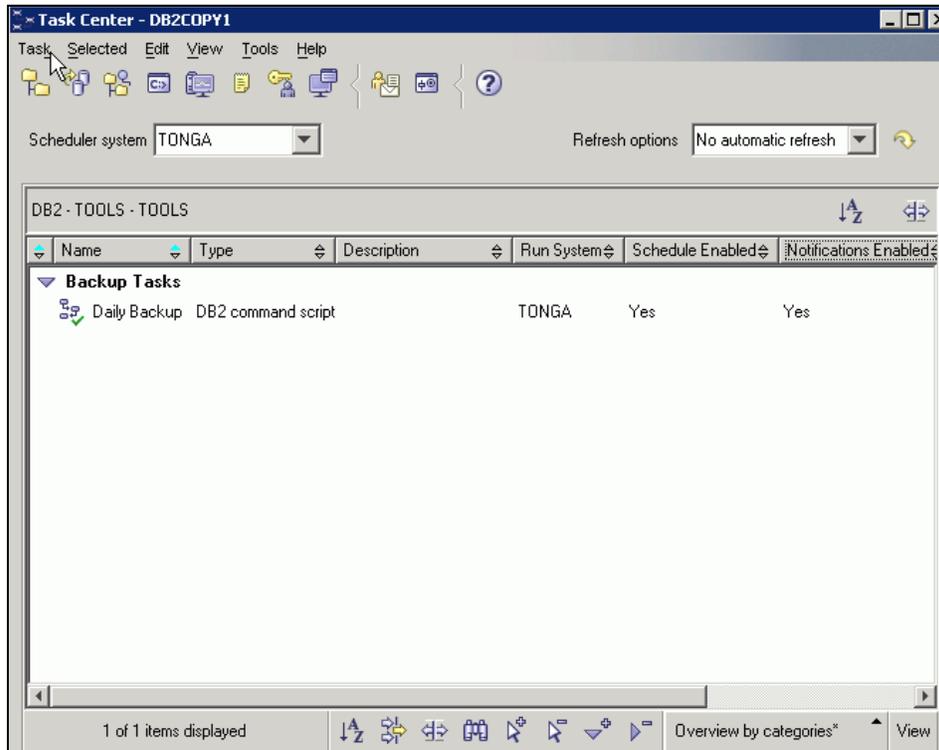
Figure 9-2   Task Center

9.5 Report tools

Reporting in DB2 is based on self-prepared applications or external tools. One of the tools is the DB2 Query Management Facility (DB2 QMF™ for Windows). QMF is an integrated query and reporting tool set that enables you to generate queries and reports, and allows you to store them in multiple DB2 database servers. The prepared queries or reports can be viewed in various formats, such as Microsoft Excel, IBM Lotus® 1-2-3®, or text files. For more information about QMF visit the Web site:

```
http://www.ibm.com/software/data/qmf/
```

Because DB2 relies on open standards, many reporting tools from other vendors (including Business Objects, Cognos, Hyperion, Meta Integration Technology, MicroStrategy, QlikTech, SAS, and Viador) can be used with DB2.

In the DB2 Data Warehouse Edition (DB2 DWE), the DB2 Olap acceleration feature (formerly DB2 Cube views) and the DB2 Alphablox tool can be used together to provide OLAP analytical reporting capabilities. The DB2 DWE OLAP acceleration feature automates the creation of OLAP metadata at the database level so that metadata can be shared among applications that access the database. The generated metadata can be used by the DB2 optimizer for faster performance, and can also be accessed by DB2 Alphablox and other third-party report tools, providing a shared single view of the data.

DBAs can use the DB2 DWE OLAP acceleration feature to aggregate the data into cube-like dimensional charts, allowing users to access the data from different perspectives. DBAs can also import and export DB2 cube models metadata into and out of DB2 databases. DB2 DWE OLAP acceleration feature can also be used to analyze dimensional models and recommend aggregates that improve OLAP performance.

DB2 Alphablox provides the ability to create custom Web-based applications to provide real-time, highly customizable and interactive multidimensional analysis of data. DB2 Alphabox is tightly integrated with the DB2 DWE OLAP acceleration feature. DB2 Alphablox supports the standard J2EE application development model, providing programmatic access to its components through a set of Java application programming interfaces (API). DB2 Alphablox allows users to interactively specify different levels of details and aggregations in multidimensional reports.

DB2 DWE OLAP provides an SQL-based and XML-based API for OLAP tools and application developers. Through CLI, ODBC, or JDBC connections or by using embedded SQL to DB2, applications and tools can use a single stored procedure to create, modify, and retrieve metadata objects. DB2 DWE OLAP also provides a functionality to query and explore the data in a Microsoft Excel spreadsheet.

For more information about DB2 Data Warehouse Edition, refer to:

http://www.ibm.com/software/data/db2/dwe/

10

Testing

All stages of the migration process should be validated by running a series of carefully designed tests. The purpose of the tests is to determine the differences between the expected results (the source environment) and the observed results (the migrated application). The detected changes should be synchronized with the development stages of the project. This chapter describes the test objectives and a generic testing methodology, which can be employed to test migrated applications. Additionally, problem determination techniques and initial tuning strategies are discussed.

447

10.1 Planning

The test planning details the activities, dependencies, and effort required to conduct the test of the converted solution.[1]

10.1.1 Principles of software tests

Keep in mind the principles of software tests in general:

► It is not possible to test a nontrivial system completely.

► Tests are optimizing processes regarding completeness.

► Always test against expectations.

► Each test must have reachable goals.

► Test cases have to contain reachable and non-reachable data.

► Test cases must be repeatable.

► Test cases have to be archived in the configuration management system as well as source code and documentation.

10.1.2 Test documentation

The test documentation is the most important part of the project. The *ANSI/IEEE Standard for Software Test Documentation*, ANSI/IEEE Std 829-1983, describes its content exactly. Here, we provide you a high-level overview.

Scope

State the purpose of the plan, possibly identifying the level of the plan (master, etc.). This is essentially the executive summary part of the plan.

You may want to include any references to other plans, documents, or items that contain information relevant to this project and process. If preferable, you can create a references section to contain all reference documents.

Identify the Scope of the plan in relation to the Software Project plan that it relates to. Other items may include, resource and budget constraints, scope of the testing effort, how testing relates to other evaluation activities (Analysis & Reviews), the process to be used for change control and communication, and coordination of key activities.

As this is the *Executive Summary,* keep information brief and to the point.

---

[1] Source: Gerrard Consulting

Definition of test items

Define the test items you intend to test within the scope of this test plan. Essentially, something you will test is a list of what is to be tested. This can be developed from the software application inventories, as well as other sources of documentation and information.

This section is a technical description of the software, and can be oriented to the level of the test plan. For higher levels, it may be by application or functional area, for lower levels it may be by program, unit, module, or build.

Features to be tested

This is a listing of what is to be tested from the users' viewpoint of what the system does. This is not a technical description of the software, but a users' view of the functions. Users do not understand technical software terminology. They understand functions and processes, as they relate to their jobs.

Set the level of risk for each feature. Use a simple rating scale such as high, medium, and low (H, M, L). These types of levels are understandable to a user. You should be prepared to discuss why a particular level was chosen.

Features not to be tested

This is a listing of what is *not* to be tested from both the users' viewpoint of what the system does and a configuration management view. This is not a technical description of the software, but a users' view of the functions.

Identify *why* the feature is not to be tested; there can be any number of reasons.

Approach

This is your overall test strategy for this test plan. It should be appropriate to the the plan and should be in agreement with plans affecting application and database parts. Overall rules and processes should be identified:

- ▶ Are any special tools to be used and what are they?
- ▶ Will the tool require special training?
- ▶ What metrics will be collected?
- ▶ Which level is each metric to be collected at?
- ▶ How is Configuration Management to be handled?
- ▶ How many different configurations will be tested?
- ▶ Combinations of hardware, software, and other vendor packages.
- ▶ What levels of regression testing will be done and how much at each test level?
- ▶ Will regression testing be based on severity of defects detected?
- ▶ How will elements in the requirements and design that do not make sense or are un-testable be processed?

Item pass and fail criteria

What is the completion criteria for this plan? What is the number and severity of defects located? This is a critical aspect of any test plan and should be appropriate to the level of the plan.

Suspension criteria and resumption requirements

Know when to pause in a series of tests. If the number or type of defects reaches a point where the follow-on testing has no value, it makes no sense to continue the test; you are just wasting resources.

Specify what constitutes stoppage for a test or series of tests, and what is the acceptable level of defects that will allow the testing to proceed past the defects.

Testing after a truly fatal error will generate conditions that may be identified as defects, but are in fact ghost errors caused by the earlier defects that were ignored.

Test deliverables

What is to be delivered as part of this plan?

- ► Test plan document
- ► Test cases
- ► Test design specification
- ► Tools and their outputs
- ► Error logs and execution logs
- ► Problem reports and corrective actions

One thing that is not a test deliverable is the software itself, which is listed under test items, and is delivered by development.

Environmental needs

Are there any special requirements for this test plan, such as:

- ► Special hardware such as simulators, static generators, and so on.
- ► How will test data be provided? Are there special collection requirements or specific ranges of data that must be provided?
- ► How much testing will be done on each component of a multi-part feature?
- ► Special power requirements
- ► Specific versions of other supporting software
- ► Restricted use of the system during testing

Staffing and skills

The staffing depends on the kind of test defined in Chapter 10.1.3, "Test phases" on page 451. In this section, you should define the persons and the education and training needed for executing the test case.

Responsibilities

Who is in charge? This issue includes all areas of the plan. Here are some examples:

► Setting risks

► Selecting features to be tested and not tested

► Setting overall strategy for this level of plan

► Ensuring all required elements are in place for testing

► Providing for resolution of scheduling conflicts, especially if testing is done on the production system

► Who provides the required training?

► Who makes the critical "go/no" decisions for items not covered in the test plans?

10.1.3 Test phases

A series of well designed tests should validate all stages of the migration process. A detailed test plan should describe all test phases, scope of the tests, validation criteria, and specify the time frame. To ensure that the applications operate in the same manner as they did in the source database, the tests plan should include data migration, functional, and performance tests, as well as other post migration assessments.

Data migration testing

The extracting and loading process entails conversion between source and target data types. The migrated database should be verified to ensure that all data is accessible, and was imported without any failure or modification that could cause applications to function improperly.

Functional testing

Functional testing is a set of tests in which new and existing functionality of the system are tested after migration. Functional testing includes all components of the RDBMS system (stored procedures, triggers, user defined functions), networking, and application components. The objective of functional testing is to verify that each component of the system functions as it did before migrating, and to verify that new functions are working properly.

Integration testing

Integration testing examines the interaction of each component of the system. All modules of the system and any additional applications (WEB, supportive modules, Java programs, etc.) running against the target database instance should be verified to ensure that there are no problems with the new environment. The tests should also include GUI and text-based interfaces with local and remote connections.

Performance testing

Performance testing of a target database compares the performance of various SQL statements in the target database with the statements' performance in the source database. Before migrating, you should understand the performance profile of the application under the source database. Specifically, you should understand the calls the application makes to the database engine.

Volume/Load stress testing

Volume and load stress testing tests the entire migrated database under high volume and loads. The objective of volume and load testing is to emulate how the migrated system might behave in a production environment. These tests should determine whether any database or application tuning is necessary.

Acceptance testing

Acceptance tests are carried out by the end users of the migrated system. Users are asked to simply explore the system, test usability, and system features, and give direct feedback. After acceptance, tests are usually the last step before going into production with the new system.

Post migration tests

Since a migrated database can be a completely new environment for the IT staff, the test plan should also encompass examination of new administration procedures like database backup/restore, daily maintenance operation, or software updates.

10.1.4 Time planning and time exposure

The time planning should be based on realistic and validated estimates. If the estimates for the migration of the application and database are inaccurate, the entire project plan will slip, and the testing is part of the overall project plan.

It is always best to tie all test dates directly to their related migration activity dates. This prevents the test team from being perceived as the cause of a delay. For example, if system testing is to begin after delivery of the final build, then system testing begins the day after delivery. If the delivery is late, system testing

starts from the day of delivery, not on a specific date. This is called dependent or relative dating.

Figure 10-1 shows the phases during a typical migration project. The definition of the test plans happen in a very early stage. The test cases, and all its following tasks, must be done for all test phases as described in Chapter 10.1.3, "Test phases" on page 451.
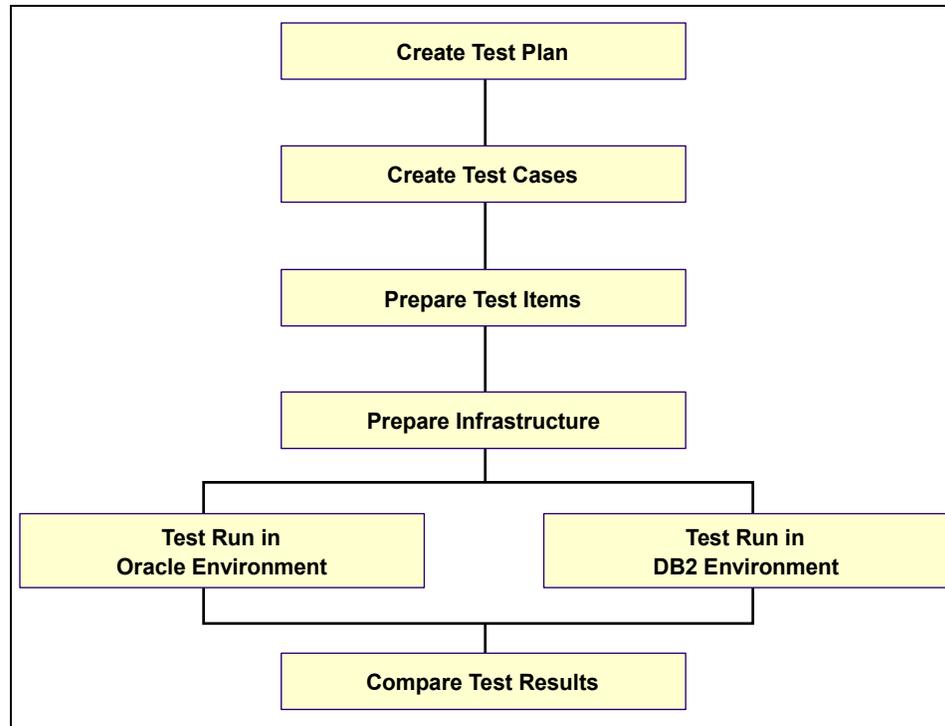
Figure 10-1   Phases during a migration project

The time exposure of tests depends on the availability of an exiting test plan and already prepared test items. The efforts depend also on the degree of changes during the application and database migration.

Note: The test efforts can be between 50% and 70% of the total migration effort.

10.2 Data checking technique

Data movement is the first thing any migration should focus on. Without having all your tables and data properly moved over, all other migration testing is in vain. The test process should detect, if all rows were imported into the target database, verify that data type conversions were successful, and check random data byte-by-byte. The data checking process should be automated by appropriate scripts. When testing data migration you should:

► Check **IMPORT**/**LOAD** messages for errors and warnings.

► Count the number of rows in source and target databases and compare them.

► Prepare scripts that perform data checks

► Involve data administration staff familiar with the application and its data to perform random checks.

10.2.1 IMPORT/LOAD messages

You should always check the messages generated by **IMPORT** or **LOAD** commands. Example 10-1 presents messages generated by the sample import command. You should read not only the summary at the end of the listing, but also pay attention to the warning messages.

Example 10-1   Sample IMPORT messages

```
db2 import from table01.unl of del replace into table01

SQL3109N  The utility is beginning to load data from file "table01.unl".

SQL3148W  A row from the input file was not inserted into the table.  SQLCODE
"-545" was returned.

SQL0545N  The requested operation is not allowed because a row does not satisfy
the check constraint "ARTURW.TABLE01.SQL030812222227680".  SQLSTATE=23513

SQL3185W  The previous error occurred while processing data from row "2" of the
input file.

SQL3117W  The field value in row "3" and column "1" cannot be converted to a
SMALLINT value.  A null was loaded.

SQL3125W  The character data in row "4" and column "2" was truncated because
the data is longer than the target database column.

SQL3110N  The utility has completed processing.  "4" rows were read from the
input file.
```

```
SQL3221W  ...Begin COMMIT WORK. Input Record Count = "4".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "4" rows were processed from the input file.  "3" rows were
successfully inserted into the table.  "1" rows were rejected.


Number of rows read       = 4
Number of rows skipped    = 0
Number of rows inserted   = 3
Number of rows updated    = 0
Number of rows rejected   = 1
Number of rows committed  = 4
```

As shown in the summary, during the import process one record from the input file was rejected, and three were inserted into the database. To understand the nature of the warnings, you should look into the data source file and the table definition (**db2look** command). For Example 10-1, the table definition is presented in Figure 10-2 and the data file in Figure 10-3.

```
CREATE TABLE TABLE01 (
   C1 SMALLINT,
   C2 CHAR(3),
   C3 SMALLINT CHECK( C3 IN (1,2,3)))
```

Figure 10-2   Table "table01" definition for Example 10-1

```
1,"abc",1
2,"abc",4
32768,"abc",2
4,"abcd",3
```

Figure 10-3   Data file "table01.unl" for Example 10-1

The first row from the input file (Figure 10-3) was inserted without any warnings. The second row was rejected because it violated check constraint (warnings SQL3148W, SQL0545N, SQL3185W). A value of 32768 from the third row was changed to null because it was out of the SMALLINT data type range (warning SQL3117W) and string abcd from the last row was truncated to abc because it was longer than the relevant column definition (warning SQL3125W).

The **LOAD** utility generates messages in a similar format, but because it is designed for speed, it bypasses the SQL engine, and inserts data directly into

table spaces without constraint checking. Inserting the same *table01.unl* file (Figure 10-3) into *table01* (Figure 10-2) with the **LOAD** utility generates messages without SQL3148W, SQL0545N, SQL3185W warnings as shown in Figure 10-2.

Example 10-2   LOAD messages

```
db2 load from table01.unl of del replace into table01

[..]
SQL3117W  The field value in row "3" and column "1" cannot be converted to a
SMALLINT value.  A null was loaded.

SQL3125W  The character data in row "4" and column "2" was truncated because
the data is longer than the target database column.

[..]
Number of rows read         = 4
Number of rows skipped      = 0
Number of rows loaded       = 4
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 4
```

A table that has been created with constraints is left by the **LOAD** command in *set integrity pending state*. Accessing the table with SQL queries generates warning SQL0668N Operation not allowed for reason code "1" on table "<TABLE_NAME>". SQLSTATE=57016. The **SET INTEGRITY** SQL statement should be used to move the loaded table into a usable state. Example 10-3 shows a way to validate constraints. All rows that violated constraints will be moved to exception table *table01_e*.

Example 10-3   Turning integrity checking back on.

```
db2 create table table01_e like table01
db2 set integrity for table01 immediate checked for exception in table01 use
table01_e

SQL3602W  Check data processing found constraint violations and moved them to
exception tables.  SQLSTATE=01603
```

The set integrity statement has many options, like turning integrity on only for new data, turning integrity off, or specifying exception tables with additional diagnostic information. For more information about the **SET INTEGRITY** command and exception tables, please refer to:

► "SET INTEGRITY" in *SQL Reference, Volume 2,* SC10-4250
► "Appendix K. Exception tables" in *SQL Reference, Volume 1,* SC10-4249

10.2.2 Data checking scripts

Scripts performing logical data integrity checks automate the data verification process and save administrator effort. For small tables (with less that 50,000 rows) you can write a program that compares data byte-by-byte. The program (preferably ODBC, JDBC, or SQL script) can extract sorted rows from Oracle and DB2 to files in the same ASCII format. The files should then be binary compared (on UNIX use DIFF command) and checked to determine if they are the same. For larger tables, comparing all rows byte by bytes can be very inefficient. The data migration should be evaluated by comparing aggregate values, like the number of rows. To do this you can create a special table for storing the information about the number of rows in the source Oracle database. Table *CK_ROW_COUNT* presented in Example 10-4 can be used for that purpose.

Example 10-4   Table for storing number of rows (Oracle)

```
CREATE TABLE CK_ROW_COUNT (
   TAB_NAME VARCHAR2(30), -- table name
   ROW_COUNT INT, -- number of rows
   SYS_NAME CHAR(3), -- code to distinguish the system: ORA or DB2
   TIME_INS DATE -- time when the count was performed
```

For each table, you should count the total number of rows and store the information in the CK_ROW_COUNT table. The following **INSERT** statement can be used for that purpose:

```
insert into ck_row_count select 'TAB_NAME', count(*), 'ORA',  sysdate from
TAB_NAME
```

The table CK_ROW_COUNTS and its data can be manually migrated to the target DB2 database. Example 10-5 presents DB2 versions of the table.

Example 10-5   Table for storing number of rows (DB2)

```
CREATE TABLE CK_ROW_COUNT (
   TAB_NAME VARCHAR(30),
   ROW_COUNT INT,
   SYS_NAME CHAR(3),
   TIME_INS TIMESTAMP
)
```

On the DB2 system, you should repeat the counting process with the equivalent **INSERT** statement:

```
insert into ck_row_count select 'TAB_NAME', count(*), 'DB2',  CURRENT
TIMESTAMP from  TAB_NAME
```

After performing the described steps, the DB2 table *CK_ROW_COUNT* should contain information about the number of rows counted on Oracle and DB2 databases. The records in the table should look like Example 10-6.

Example 10-6   Sample table CK_ROW_COUNTS contents

```
select TAB_NAME, ROW_COUNT, SYS_NAME, TIME_INS from CK_ROW_COUNT
[...]
TABLE_A 39001 ORA 2007-02-23-10.13.39
TABLE_A 39001 DB2 2007-02-23-10.32.13
TABLE_B 60003 ORA 2007-02-23-10.15.29
TABLE_B 60002 DB2 2007-02-23-10.33.49
[...]
```

Having information about the number of rows in a SQL table is very convenient, because with a single query, you can get the table names that contain a different number of rows in the source and target databases:

```
select tab_name from (select distinct tab_name, num_rows from CK_ROW_COUNT)
as t_temp group by t_temp.tab_name having(count(*) > 1)
```

To manually migrate the data from the Oracle *ck_row_count* table, you can save the results of the following query to a file (*ck_row_count.unl*):

```
select tab_name || ',' || row_count || ',' || sys_name || ',' || to_char(
time_ins, 'yyyy-mm-dd-hh24.mi.ss".000000"') from  ck_row_count
```

and import the file into DB2 using DB2 **IMPORT** command:

```
db2 import from ck_row_count.unl of del insert into ck_row_count
```

The process of creating the statements that count the number of rows is automated by a PL/SQL script presented in Example 10-7. The script retrieves information about the user tables from the Oracle data dictionary, and generates two files count_rows_ora.sql (Oracle version) and count_rows_db2.sql (DB2 version) with **INSERT** statements that can be used for calculating the number of rows for all user tables.

Example 10-7   PL/SQL program that generates scripts for counting rows

```
set heading off;
set echo off;
set feedback off;
set serveroutput on size 100000

---- Generating script for Oracle database
spool count_rows_ora.sql;

BEGIN
```

```
    dbms_output.put_line( 'CREATE TABLE CK_ROW_COUNT ( TAB_NAME VARCHAR2(30),
ROW_COUNT INT, SYS_NAME CHAR(3), TIME_INS DATE);');


  FOR tab_rec IN (SELECT TABLE_NAME from TABS ) LOOP
    dbms_output.put_line( 'INSERT INTO CK_ROW_COUNT SELECT
'''||tab_rec.table_name||''', COUNT(*), ''ORA'',  SYSDATE FROM '||
tab_rec.table_name ||';');
  END LOOP;
  dbms_output.put_line( 'COMMIT; ');
END;
/
spool off;

---- Generating script for DB2 database
spool count_rows_db2.sql;
BEGIN

  dbms_output.put_line( 'CREATE TABLE CK_ROW_COUNT ( TAB_NAME VARCHAR(30),
ROW_COUNT INT, SYS_NAME CHAR(3), TIME_INS TIMESTAMP);');


  FOR tab_rec IN (SELECT TABLE_NAME from TABS ) LOOP
    dbms_output.put_line( 'INSERT INTO CK_ROW_COUNT SELECT
'''||tab_rec.table_name||''', COUNT(*), ''DB2'',  CURRENT TIMESTAMP FROM '||
tab_rec.table_name ||';');
  END LOOP;
  dbms_output.put_line( 'COMMIT; ');
END;
/
spool off;
rollback;
```

The presented approach for comparing the number of rows can be extended for
additional checking, like comparing the sum of numeric columns. Here are the
steps that summarize the technique:

1. Define check sum tables on the source database and characterize scope of
   the computation.

2. Perform the computation and store the results in the appropriate check sum
   tables. Use Oracle data dictionary (TABS and COLS) to generate the
   aggregate queries, or write SQL/PL procedures.

3. Migrate the check sum tables as other user tables.

4. Perform equivalent computations on the target system, and store the
   information in the migrated check sum tables.

5. Compare the computed values.

Table 10-4 provides computations for selected database types. The argument for the DB2 **sum()** function is converted to the DECIMAL type, because in most cases, the **sum()** function returns the same data type as its argument, which can cause arithmetic overflow. For example, when calculating the sum on an INTEGER column, if the result exceeds the INTEGER data type range, error SQL0802N is generated `Arithmetic overflow or other arithmetic exception occurred`. Converting the argument to DECIMAL eliminates the error.

Table 10-1   Aggregations for data migration verification

Data type	Oracle operation	DB2 operation
numeric(<preci son>,<scale>)	sum(val)	sum(cast(val as decimal(31,<scale>)))
date	sum(trunc(val - to_date('0001/01/02', 'yyyy/mm/dd')))	sum(cast(days(val) as decimal(31,1)))
variable length character	sum(length(val))	sum(cast(length(val) as decimal(31,0)))
fixed length character	sum(length(rtrim(val)))	sum(cast(length(rtrim(val)) as decimal(31,0)))

10.3 Code and application testing

The most important part of the testing process is to verify that each component of the system functions as it did before migrating. All components of the RDBMS system, including views, stored procedures, triggers, application components, and security systems should be verified as to whether they are working properly.

10.3.1 View sanity check

The next step after data migration testing is to check all migrated views. Once data is moved over, the views can be tested to make sure they are working in the same way as in the source database. Depending upon the view, different checking scenarios can be created. Basic views can be easily checked, by counting the number of rows the views return, or by comparing calculated values, similarly to the tests done on regular tables. More complicated views need a little more thought on how they should be checked, however, all view should be at a minimum reviewed by executing queries against them. The views created with the intention to modify data should also be tested for inserting, updating, and deleting.

10.3.2 PL/SQL to SQL PL object check

All stored procedures, user defined functions, and triggers should be unit tested individually before they are promoted for further testing. This means that after objects are migrated (either manually, with the MTK, or some combination of that) they need to be checked to make sure they function properly. Basic problems can be discovered by running stored procedures through the Developer Workbench (Figure 10-4).

The Developer Workbench is an Eclipse-based tool that replaces the Development Center used in DB2 Version 8. Developer Workbench is a comprehensive development environment for creating, editing, debugging, deploying, and testing DB2 stored procedures and user-defined functions. You can also use Developer Workbench to develop SQLJ applications, and to create, edit, and run SQL statements, and XML queries.
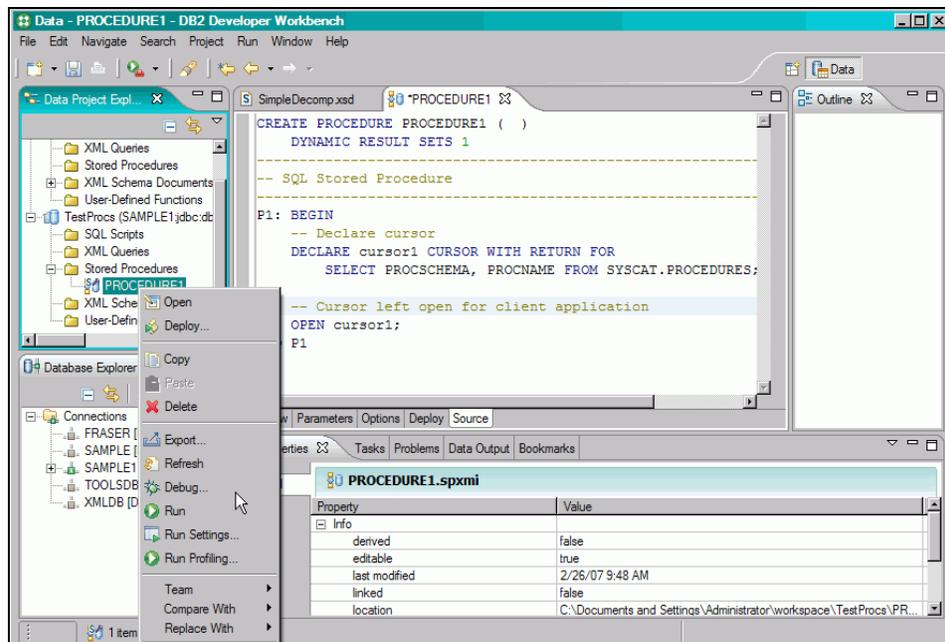
Figure 10-4   Developer Workbench.

The Developer Workbench is a free download, that is installed separately from DB2. For more information about the Developer Workbench, refer to *Developing SQL and External Routines,* SC10-4373.

10.3.3 Application code check

The scope of application testing depends on the migrated application. For systems that were designed to support many versions of databases, like SAP, Siebel, or PeopleSoft, the testing process is usually well defined by the vendors, and offered as a paid service.

For self-built applications, the testing process should be started with the application queries. All queries should be independently tested to ensure that they return the expected results. With the application queries successfully migrated, the surrounding client programs should be rebuilt, and then the application should be tested against the target database. Each module of the application, and possibly each panel form, should be run and checked for errors or improper functionality. All the supported application connectivity interfaces should also be checked.

A very important issue is to document all the test conditions, such as what operations were performed, which application screens were opened, what input data was used for testing, and what was the result. For larger projects, the documenting part can become overwhelming, so specialized software is usually used for those cases. As mentioned earlier, by definition, the new application cannot be fully tested. In the migration project, the application testing is an iterative process of planning, designing the test cases, executing the test cases, and finally evaluating and analyzing the results.

Together with various functional testing, the application should also be checked for performance. Since there are many architectural differences between Oracle and DB2, some SQL operations might require further optimization. Observing the performance differences in early testing stages increases the chance to prepare more optimal code for the new environment.

Before going into production, the migrated database should be verified under high volume and loads. These tests should emulate the production environment, and can determine if further application or database tuning is necessary. The stress load can also reveal other hidden problems, such as locking issues, which can be observed only in a production environment.

10.3.4 Security

Before going into production, security must be checked in detail. Oracle handles security quite differently from DB2, so it is not trivial to compare the user rights between the two systems. Oracle's grants to roles are resolved in DB2 with operating system groups or secondary authorization identifications. A list of Oracle users should be compared to equivalent DB2 operating system users. All of DB2's authorities should be verified to allow proper persons to connect to the

database. Privileges for all database objects should also be verified. Another important issue is to check the identifications of the user who was used for binding stored procedures, because executors of the procedures inherit the binder privileges.

10.3.5 Tools for testing and problem tracking

The software testing process can be a very complex task. All the tests should be synchronized with the development life cycle, and be well documented. For large projects, it might be necessary to use supportive software to improve testing productivity. IBM Rational® Functional Tester and IBM Rational Performance Tester can be used for that purpose.

IBM Rational Functional Tester provides testers with automated capabilities for data-driven testing and a choice of scripting language and industrial-strength editor for test authoring and customization. IBM Rational Performance Tester can create, execute, and analyze tests to validate the reliability of complex e-business applications.

Additionally, there are many other Rational products that may suit your testing needs. For more information about testing products, go to the IBM Rational Web site:

> http://www.ibm.com/software/rational

10.4 Troubleshooting

The first step of problem determination is to know what information is available to you. Whenever DB2 performs an operation, there is a return code associated with that operation. The return code is displayed to the user in the form of an informational or error message. These messages are logged into diagnostic files depending on the diagnostic level set in the DB2 Manager Configuration. In this section, we discuss the DB2 diagnostic logs, error message interpretation, tips that may help with problem determination, troubleshooting, as well as the resolutions to some specific problems.

The following actions should be taken when experiencing a DB2-related problem:

► Check related messages
► Explain error codes
► Check documentation
► Search through available Internet resources
► Review APARs for current FixPak level
► Use available tools to narrow down the problem

► Ask IBM for support

10.4.1 Interpreting DB2 informational messages

Start your investigation with the return code. DB2 provides a return code for every operation performed in the form of `CCCnnnnnS`. The prefix `CCC` identifies the DB2 component that is returning the message; the `nnnnn` is a four or five digit number, which is also referred to as an SQLCODE; and the `S` is a severity indicator. For example, `SQL0289N`: the `SQL` component identifier, represents a message from the Database Manager, the SQLCODE is `0289`, which is an error message, and the `N` is the severity indicator, in this case an error.

Here is the complete list for DB2 error messages prefixes for your reference:

► SQL - Database Manager messages
► ADM - messages generated by many DB2 components. These messages are written in the Administration Notification log file and are intended to provide additional information to system administrators.
► AMI - MQ Application Messaging Interface messages
► ASN - DB2 Replication messages
► CCA - Configuration Assistant messages
► CLI - Call Level Interface messages
► DBA - Database Administration tools messages
► DBI - Installation and Configuration messages
► DBT - Database tools messages
► DB2 - Command Line Processor (CLP) messages
► DQP - Query Patroller messages
► EAS - Embedded Application Server messages
► EXP - Explain utility messages
► GSE - Spatial Extender messages
► LIC - License Manager messages
► MQL - MQ Listener messages
► SAT - Satellite Environment messages
► SPM - Sync Point Manager messages
► XMR - XML Metadata Repository messages

The three severity indicators are:

► W: Warning or informational messages
► N: Error messages
► C: Critical system errors

DB2 also provides detailed information for each message. The full error message describes the nature of the problem in detail, and potential user responses. To display the DB2 return code full message, you can use the DB2 command **db2 ?** *error-code*. In AIX or Linux, since ? (question mark) is a special

character, you need to separate the DB2 command and the error code with a double quote ("). See Example 10-8.

Example 10-8   Explaining error codes

```
db2 "? sql0289"
SQL0289N Unable to allocate new pages in table space
         "<tablespace-name>".

Explanation:

One of the following conditions is true on one or more database
partitions:

 1 One of the containers assigned to this SMS table space has
reached the maximum file size. This is the likely cause of the
error.

 2 All the containers assigned to this DMS table space are full.
This is the likely cause of the error.

[...]
```

You can find full information about the DB2 message format, and a listing of all the messages in *Message Reference Volume 1* and *2* (SC10-4238 and SC10-4239).

10.4.2 DB2 diagnostic logs

DB2 logs every return code in diagnostic logs based on the diagnostic level set in the database manager configuration. When investigating DB2 problems, the essential information can be found in diagnostic log files generated by DB2. These logs are:

► db2diag.log
► Notify files
► Trap files
► Dump files
► Messages files

db2diag.log

The db2diag.log log is the most often used file for DB2 problem investigation. You can find this file in the DB2 diagnostic directory, defined by the DIAGPATH variable in the Database Manager Configuration. If the DIAGPATH parameter is not set, by default the directory is located at:

UNIX:

    $HOME/sqllib/db2dump

where $HOME is the DB2 instance owner's home directory.

Windows:

    <INSTALL PATH>\<DB2INSTANCE>

where <INSTALL PATH> is the directory where DB2 is installed, and <DB2INSTANCE> is the name of DB2 instance.

The database manager configuration parameter DIAGLEVEL controls how much information is logged to db2diag.log. Valid values can range from 0 to 4:

    0 - No diagnostic data captured

    1 - Severe errors only

    2 - All errors

    3 - All errors and warnings (default)

    4 - All errors, warnings and informational messages

Most of the time, the default value is sufficient for problem determination. In some cases, especially on development or test systems, you can set the parameter to 4, and collect all informational messages. However, be aware that depending on the activity, this may cause performance issues due to the large amount of data recorded into the file. Setting DIAGLEVEL to 4 may also make the file very large and harder to read.

The information in db2diag.log includes:

► A diagnostic message (beginning with DIA) explaining the reason for the error.

► Application identifiers, which allow matching up error entries with corresponding application or DB2 server processes

► Any available supporting data, such as SQLCA data structures, and pointers to the location of any extra dump or trap files.

► Administrative events, such as BACKUP/RESTORE start and finish times

Example 10-9 contains extract of db2diag.log taken at DIAGLEVEL 3.

Example 10-9   Example of db2diag.log file

```
(1)  2007-02-26-11.04.28.515000-240  (2) E394355H479    (3) LEVEL: Warning
(4)  PID    : 3840                    (5) TID  : 504     (6) PROC : db2syscs.exe
(7)  INSTANCE: DB2INST1               (8) NODE : 000     (9) DB   : SAMPLE
```

```
(10) APPHDL  : 0-191              (11) APPID: *LOCAL.DB2.061026150526
(12) FUNCTION: DB2 UDB, data management, sqldEscalateLocks, probe:3
(13) MESSAGE : ADM5502W  The escalation of "1133" locks on table "DB2INST1
          .TABLE01" to lock intent "X" was successful.
```

An explanation of db2diag.log entries is included below. Not every log entry will contain all of these parts. Only the first several fields (timestamp to TID) and FUNCTION will be present in all the db2diag.log records. The number in parenthesis (which are used here for illustration purposes, are not part of the actual db2diag.log entry) corresponds to the following numbers:

(1) - A timestamp and time zone for the message.

(2) - The record ID field. The db2diag.log's recordID specifies the file offset at which the current message is being logged (for example, "394355") and the message length (for example, "479") for the platform where the DB2 diagnostic log was created.

(3) - The diagnostic level associated with an error message. For example, Info, Warning, Error, Severe, or Event.

(4) - The process ID

(5) - The thread ID

(6) - The process name

(7) - The name of the instance generating the message.

(8) - For multi-partition systems, the partition generating the message. (In a non-partitioned database, the value is "000".)

(9) - The database name

(10) - The application handle. This value aligns with that used in db2pd output and lock dump files. It consists of the coordinator partition number followed by the coordinator index number, separated by a dash.

(11) - Identification of the application for which the process is working. In this example, the process generating the message is working on behalf of an application with the ID *LOCAL.DB2.061026150526. This corresponds to the LIST APPLICATIONS command output. Each application has a unique application ID.

(12) - The product name ("DB2"), component name ("data management"), and function name ("sqldEscalateLocks") that is writing the message (as well as the probe point ("3") within the function).

(13) - The return code (if any) returned by a called function. In this example this is a administration warning, telling about lock escalation (1133 row locks where successfully replaced by one table lock) on table DB2INST1.TABLE01.

Notification log

DB2 also provides diagnostic information at the point of failure to the administration notification log. On UNIX platforms, the administration notification log is a text file called instance.nfy, where instance is the name of the instance. On Windows, all administration notification messages are written to the Event Log.

The DBM configuration parameter NOTIFYLEVEL specifies the level of information to be recorded:

    0 - No administration notification messages captured (not recommended)

    1 - Fatal or unrecoverable errors

    2 - Immediate action required

    3 - Important information, no immediate action required (default)

    4 - Informational messages

Not only can DB2 write to the notification logs, but also the Health Monitor, the Capture and Apply programs, and user applications using the db2AdminMsgWrite API.

Trap files

Whenever a DB2 process receives a signal or exception (raised by the operating system as a result of a system event) that is recognized by the DB2 signal handler, a trap file is generated in the DB2 diagnostic directory (as specified by the DIAGPATH database manager configuration parameter). The files are created using the following naming convention:

Linux and UNIX:

► tpppppp.nnn

    – ppppppp: the process ID (PID)
    – nnn: the database partition number where the trap occurred (000 on single partition databases)
    – Example: t123456.000

Windows:

► Pxxxxx.yyy or Pxxxxx.TRP

    – xxxxx: the process ID (PID)

– yyy: the database partition number where the trap occurred (000 on single partition databases). If the trap file is generated because of an exception, it will have the extension .TRP.
– Example: P52524.000

There are also diagnostic traps, generated by the code when certain conditions occur which do not warrant crashing the instance, but where it is useful to see the stack. Those traps are named with the PID in hexadecimal format, followed by the partition number (0 in a single partition database).

Depending on the signal received or the exception raised, the existence of these files can indicate different extremes of consequences. These consequences can range from the generation of a simple stack trace back for additional diagnostics, to a complete DB2 instance shutdown due to a serious internal or external problem. A list of all available signals for selected operating systems can be obtained from the following files:

► UNIX: /usr/include/sys/signal.h
► Windows (requires the software development kit): Winnt.h

Dump files

Dump files are created when an error occurs for which there is additional information that would be useful in diagnosing a problem (such as internal control blocks). Every data item written to the dump files has a timestamp associated with it to help with problem determination. Dump files are created in binary format and are located in the DB2 diagnostic path (DIAGPATH DBM CFG). When a dump file is created or appended, an entry will be made to the db2diag.log, indicating the time, type of data written, and the name of the dump file. Generally, dump files are intended for DB2 customer support representatives.

These files are generated with the following format:

Linux and UNIX:

► pppppp.nnn or lpppppp.nnn (for lock list dump)

– pppppp: the process ID (PID)
– nnn: the database partition number where the problem occurred
– Example: 123456.000

Windows:

► pppttt.nnn or lpppttt.nnn (for lock list dump)

– ppp: the process ID (PID)
– ttt: the thread ID (TID)
– nnn: the database partition number where the problem occurred
– Example: 123654.000

Messages files

Some DB2 utilities such as BIND, EXPORT, IMPORT, and LOAD provide an option to produce a messages file in a user-defined location. These files contain useful information to report the progress, success, or failure of the utility that was run. It is recommended that this option be specified, for any utility that supports it.

10.4.3 DB2 support information

Identifying what information is typically required to resolve problems is a very important step. All the conditions that define the problem are essential when reviewing documentation, searching through available Internet resources, or contacting DB2 support.

Maintenance version

The db2level utility can be used to check the current versions of DB2 installed. As shown in Figure 10-5, the utility returns information about the installed maintenance updates (FixPaks), length of word used by instance (32 bit or 64 bit), build date, and other code identifiers. It is a good habit to check periodically if newer FixPaks are available. DB2 maintenance updates are freely available from:

http://www.ibm.com/software/data/db2/udb/support/downloadv9.html

```
                                        ┌──────────────────┐
                                        │   DB2 version    │
                                        └──────────────────┘
$db2level                                        /
                                                /
DB21085I  Instance "db2inst1" uses "64" bits and DB2
code release "SQL09011"
with level identifier "01020107".
Informational tokens are "DB2 v9.1.0.1", "s061104",
"U809676", and Fix Pack "1".

Product is installed at "/opt/IBM/db2/V9.1".


┌──────────────────────────┐    ┌──────────────────────┐
│   This can be matched to │    │    The built date    │
│  related FixPak directory│    │       (YYMMDD)       │
│   on ftp.software.ibm.com│    └──────────────────────┘
└──────────────────────────┘
```

Figure 10-5   Sample db2level output

db2support utility

The db2support utility is designed to automatically collect all DB2 and system diagnostic data. This program generates information about a DB2 server, including information about its configuration and system environment. The output

of this program is stored in one compressed file named db2support.zip, located in the directory specified as part of the command invoked under command line. In one simple step, the tool can gather database manager snapshots, configuration files, and operating system parameters, which should make the problem determination quicker. A sample call of the utility is as follows:

```
db2support . -d db2_emp -c
```

The dot (`.`) represents the current directory where the output file is stored. The rest of the command is optional. `-d` and `-c` instructs the utility to connect to the db2_emp database, and also gather information about database objects such as table spaces, tables, and packages.

DB2 Product Support site

An invaluable place to look when experiencing a problem is the DB2 Product Support site for Linux, UNIX, and Windows, located on the Web at:

http://www.ibm.com/software/data/db2/support/db2_9/

The site has the most recent copies of the documentation, the knowledgebase to search for technical recommendations or DB2 defects, links for product updates, the latest support news, and many other useful DB2 related links.

To find related problems, prepare words that describe the issue such as the commands that were run, the symptoms, and tokens from the diagnostics messages, and use them as a search terms in the DB2 knowledgebase. The knowledgebase offers an option to search through DB2 documentation, Technotes, and DB2 defects (APARs).

Technotes is a set of recommendations and solutions for specific problems.

Authorized Program Analysis Reports (APARs) are defects in the DB2 code discovered by customers that require a fix. APARs have unique identifiers and are always specific to a particular version, but may affect multiple products in the DB2 family running on multiple platforms. Fixes for APARs are provided through DB2 FixPaks. The status of a closed APAR indicates that the resolution for the problem has been verified and included in the FixPaks. Open APARs represent DB2 defects that are currently being worked on or waiting to be included in the next available FixPak. HIPER APARs (High-Impact or PERvasive) are critical problems that should be reviewed to assess the potential impact of staying at a particular FixPak level.

The DB2 Product Support site also offers e-mail notification of critical or pervasive DB2 customer support issues, including HIPER APARs and FixPak alerts, among other DB2 related topics. To subscribe to it, follow the *Request e-mail updates* link which is offered by IBM My Support on the DB2 Product

Support main page. Here you can specify exactly the type of alerts that you would like to be notified with.

Calling IBM support

If the problem seems to be too complex to solve on your own, you can contact IBM Support. In order to understand and resolve your support service request in the most expedient way possible, it is important that you gather information about the problem and have it on hand when talking to the software specialist.

The guidelines and reference materials (which you may need when you require IBM support) as well as the telephone numbers are available on IBM Software Support Handbook at:

http://techsupport.services.ibm.com/guides/handbook.html

10.4.4 Problem determination tools

Tuning and troubleshooting a database can be a complex process. DB2 comes with a great number of tools, functions, and applications that simplefy this task.

Monitoring tools

DB2 monitoring utilities can collect information on many different system activities, such as usage of buffer pools, locks held by applications, sorts preformed by the system, activities on tables, connections, transactions statistics, or statements run on the system. There are two main methods of monitoring:

► Snapshot monitoring
► Event monitoring

Snapshot monitoring

Snapshot monitoring describes the state of database activity at the particular point in time the snapshot is taken. Snapshot monitoring is useful in determining the current state of the database and its applications. Since snapshots provide point in time data, they are usually executed in scripts at regular intervals.

Snapshots can be taken from the command line, using custom API programs, or through SQL using table functions or administrative views. Example 10-10 shows an extract from a sample snapshot invoked from the command line.

Example 10-10   Sample snapshot

```
db2 get snapshot for database on db2_emp

            Database Snapshot
```

```
Database name                            = DB2_EMP
Database path                            =
/home/db2inst1/db2inst1/NODE0000/SQL00001/
Input database alias                     = DB2_EMP
Database status                          = Active
[...]

High water mark for connections          = 3
Application connects                     = 7
Secondary connects total                 = 0
Applications connected currently         = 1
Appls. executing in db manager currently = 0
Agents associated with applications      = 1
Maximum agents associated with applications= 1
Maximum coordinating agents              = 1
[...]

Buffer pool data logical reads           = Not Collected
Buffer pool data physical reads          = Not Collected
Buffer pool temporary data logical reads  = Not Collected
Buffer pool temporary data physical reads = Not Collected
Asynchronous pool data page reads        = Not Collected
[...]
```

The snapshot collects database level information for database DB2_EMP. Some of the returned parameters display point-in-time values, such as the number of currently connected applications:

```
Applications connected currently         = 1
```

Some parameters represent cumulative values, like the number of connect statements issued against the database:

```
Application connects                     = 7
```

Some parameters can contain historical values, like the maximum number of concurrent connections that have been observed on the database:

```
High water mark for connections          = 3
```

The cumulative or historical values relate to the point in time, since the last initialization of the counters. The counters can be reset to zero with the RESET MONITOR command, or by the appropriate DB2 event. In Example 10-10, database deactivation and activation reset all the database level counters. Example 10-11 shows how to reset monitors for the entire instance and for a specific database.

Example 10-11   Resetting snapshot monitor counters

```
db2 reset monitor all
db2 reset monitor for database db2_emp
```

To optimize database performance in a default DB2 configuration, most of the snapshot monitor elements are not collected. As a result, in Example 10-10 the value `Not Collected` was displayed for the buffer pool statistics. DB2 contains monitor switches to provide database administrators with the option of constraining the collection of monitor elements. Current monitor switches set for the session can be displayed from the command line by GET MONITOR SWITCHES, as shown in Example 10-12.

Example 10-12   Displaying monitor switches

```
db2 get monitor switches

            Monitor Recording Switches

Switch list for db partition number 0
Buffer Pool Activity Information  (BUFFERPOOL) = OFF
Lock Information                        (LOCK) = OFF
Sorting Information                     (SORT) = OFF
SQL Statement Information          (STATEMENT) = OFF
Table Activity Information            (TABLE) = OFF
Take Timestamp Information         (TIMESTAMP) = ON 02/26/2007 16:04:18.468029
Unit of Work Information                (UOW) = OFF
```

The monitor switches can be turned on at the instance level or at an application/connection level. To switch the monitors at the instance level, modify the appropriate database manager parameter. After modifying the DFT_MON_BUFPOOL parameter, as shown in Example 10-13, all users with administration authorities will be able to collect buffer pool statistics on any database in the instance.

Example 10-13   Updating monitor switches at the instance level

```
db2 update dbm cfg using DFT_MON_BUFPOOL ON
```

To switch the monitors at the application/connection level, issue UPDATE MONITOR SWITCHES from the command line. The changes will only be applicable to that particular prompt window. Example 10-14 shows how to update the suitable monitor switch for collecting buffer pool information.

Example 10-14   Updating monitor switches at the application/connection level

```
db2 update monitor switches using BUFFERPOOL ON
```

The complete list of monitor switches and related database manager parameters is presented in Table 10-2.

Table 10-2   List of monitor switches and related DBM parameters

Database manager parameter	Monitor switch	Information provided
DFT_MON_BUFPOOL	BUFFERPOOL	Number of reads, writes, time taken
DFT_MON_LOCK	LOCK	Lock wait times, deadlocks
DFT_MON_SORT	SORT	Number of heaps used, sort performance
DFT_MON_STMT	STATEMENT	Start/stop time, SQL statement identification
DFT_MON_TABLE	TABLE	Measure of activity (rows read/written)
DFT_MON_UOW	UOW	Start/end times, completion status
DFT_MON_TIMESTAMP	TIMESTAMP	Timestamps

Sample snapshots

The database manager snapshot (Example 10-15) captures information specific to the instance level. The information centers on the total amount of memory allocated to the instance and the number of agents that are currently active on the system.

Example 10-15   Database manager snapshot

```
db2 get snapshot for database manager
```

The lock snapshot (Example 10-16) is very useful for determining what locks an application is currently holding, or what locks another application is waiting on. The snapshot lists all applications on the system and the locks that each is holding. Each lock and each application is given a unique identifier number.

Example 10-16   Lock snapshot.

```
db2 get snapshot for locks on db2_emp
```

The table snapshot (Example 10-17) contains information about the usage and creation of all tables. This information is quite useful in determining how much work is being run against a table and how much the table data changes. This

information can then be used to decide how your data should be laid out physically.

Example 10-17   Table snapshot

db2 get snapshot for tables on db2_emp

The table space and buffer pool snapshots (Example 10-18) contain similar information. The table space snapshot returns information about the layout of the table space and how much space is being used. The buffer pool snapshot contains information on how much space is currently allocated for the buffer pools and how much space will be allocated when the database is next reset. Both snapshots contain a summary of the way in which data is accessed from the database. This access can be done from a buffer pool, direct from tables on disk, or through a direct read or write for LOBs or LONG objects.

Example 10-18   Table space and buffer pool snapshots

db2 get snapshot for tablespaces on db2_emp
db2 get snapshot for bufferpools on db2_emp

The dynamic SQL snapshot is used extensively to determine how well SQL statements are performing. This snapshot summarizes the behavior of the different dynamic SQL statements that are run. The snapshot does not capture static SQL statements, so anything that was prebound will not show up in this list (use a statement event monitor to capture information about static/prebound SQL). The snapshot is an aggregate of the information concerning the SQL statements. If an SQL statement is executed 102 times, there will be one entry with the summary of the total behavior of the 102 executions.

Example 10-19   Dynamic SQL snapshot

```
db2 get snapshot for dynamic sql on
db2_emp
```

Snapshot table functions and administrative views

As mentioned earlier, DB2 features the capability to capture snapshots using SQL table functions or administrative views. Accessing snapshot information through the SQL interface is very convenient, because the requested information can be filtered and sorted, thereby presented in a more readable format. The snapshot table functions and administrative views can also be very helpful in analyzing system utilization over a period of time.

The difference between the snapshot table functions and snapshot administrative views should be noted. Both the snapshot table functions and snapshot administrative views return equivalent information to the GET SNAPSHOT FOR *object* ON *database* CLP command. However, snapshot table functions allow you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions, or all database partitions. Snapshot administrative views retrieve information for all database partitions of the currently connected database.

Most of the snapshot table functions accept two input parameters. The first is a string representing the database name. Entering a NULL value for the database name parameter instructs the function to get snapshot information for all databases in the instance. The second parameter represents a partition number. To capture a snapshot for the currently connected partition, enter a value of -1 or a NULL.

The query in Example 10-20 uses the table function SNAP_GET_TAB_V91() to retrieve the five table names, which have the most read and write activity on database DB2_EMP.

Example 10-20   Sample snapshot table function

```
db2 "select snapshot_timestamp, tabname, rows_written, rows_read,
          rows_written + rows_read as rows_accessed
     from table (SNAP_GET_TAB_V91('DB2_EMP', -1))as T
    order by rows_accessed desc
    fetch first 5 rows only"


SNAPSHOT_TIMESTAMP         TABNAME       ROWS_WRITTEN ROWS_READ ROWS_ACCESSED
-------------------------- ------------- ------------ --------- -------------
2007-02-26-17.25.20.546798 EMPLOYEE                0       256           256
2007-02-26-17.25.20.546798 STAFF                  35       105           140
2007-02-26-17.25.20.546798 SYSTABLES               0        30            30
2007-02-26-17.25.20.546798 SYSROUTINES             0        10            10
2007-02-26-17.25.20.546798 INTERNAL                0         5             5
```

The query in Example 10-21demonstrates the equivalent approach of taking a table snapshot using the administrative view, SYSIBMADM.SNAPTAB.

Example 10-21   Sample snapshot administrative view

```
db2 "select snapshot_timestamp, tabname, rows_written, rows_read,
          rows_written + rows_read as rows_accessed
     from SYSIBMADM.SNAPTAB
    order by rows_accessed desc
    fetch first 5 rows only"
```

```
SNAPSHOT_TIMESTAMP              TABNAME        ROWS_WRITTEN ROWS_READ  ROWS_ACCESSED
--------------------------      -------------  ------------ ---------- --------------
2007-02-26-17.26.27.124402 EMPLOYEE                      0        256            256
2007-02-26-17.26.27.124402 STAFF                        35        105            140
2007-02-26-17.26.27.124402 SYSTABLES                     0         30             30
2007-02-26-17.26.27.124402 SYSROUTINES                   0         10             10
2007-02-26-17.26.27.124402 INTERNAL                      0          5              5
```

Example 10-22 illustrates a usage of the SNAP_GET_DYN_SQL_V91() function, which is very useful for finding the SQL statements that are taking the most time in the database.

Example 10-22   Sample snapshot table function - taking the time in the database

```
SELECT stmt_text, total_exec_time, num_executions
  FROM TABLE(SNAP_GET_DYN_SQL_V91('DB2_EMP', -1)) as dynSnapTab
 ORDER BY total_exec_time desc
 FETCH FIRST 5 ROW ONLY
```

Example 10-23 finds the five SQL statements with the worst average execution time.

Example 10-23   Sample snapshot table function - worst average execution time

```
SELECT CASE WHEN num_executions = 0
            THEN 0
            ELSE (total_exec_time / num_executions)
       END avgExecTime,
       num_executions,
       stmt_text
FROM TABLE(SNAP_GET_DYN_SQL_V91('DB2_EMP', -1)) as dynSnapTab
ORDER BY avgExecTime desc
FETCH FIRST 5 ROWS ONLY
```

Like snapshot commands, snapshot table functions access point-in-time data kept by monitors in memory. To keep the history of the snapshots, include the SNAPSHOT_TIMESTAMP column in the snapshot query. Create a table based on the snapshot query, such as presented in Example 10-24, and periodically store the results of the query in the table.

Example 10-24   Storing snapshot data in a table

```
db2 create table table_snap_hist as
    (select snapshot_timestamp, tabname, rows_written, rows_read,
            rows_written + rows_read as rows_accessed
       from table (SNAP_GET_TAB_V91('DB2_EMP', -1))as T) definition only
```

```
db2 "insert into table_snap_hist
    select snapshot_timestamp, tabname, rows_written, rows_read,
           rows_written + rows_read as rows_accessed
      from table (SNAP_GET_TAB_V91('DB2_EMP', -1))as T
      order by rows_accessed desc fetch first 5 rows only"
```

Table 10-3 lists the more commonly used snapshot table functions and snapshot administrative views. A complete list and detailed descriptions of snapshot table functions can be found in *System Monitor Guide and Reference,* SC10-4251, and *Administrative SQL Routines and Views,* SC10-4293 for the snapshot administrative views.

Table 10-3   Common snapshot table functions and administrative views

Snapshot table function	Snapshot admin views	Information returned
SNAP_GET_APPL	SNAPAPPL	General application information for each application that is connected. This includes cumulative counters, status information, and most recent SQL statement executed (if the statement monitor switch is set).
SNAP_GET_APPL_INFO	SNAPAPPL_INFO	General application identification information for each application that is connected.
SNAP_GET_BP	SNAPBP	Buffer pool activity counters for the specified database. Requires the buffer pool monitor switch.
SNAP_GET_DB_V91	SNAPDB	Database information. Information is returned only if there is at least one application connected to the database.
SNAP_GET_DBM	SNAPDBM	Database manager information
SNAP_GET_DYN_SQL_V91	SNAPDYN_SQL	Point-in-time statement information from the SQL statement cache for the database.

Snapshot table function	Snapshot admin views	Information returned
SNAP_GET_LOCK	SNAPLOCK	Lock information at the database level, and application level for each application connected to the database. Requires the lock monitor switch.
SNAP_GET_LOCKWAIT	SNAPLOCKWAIT	Application information regarding lock waits for the applications connected.
SNAP_GET_STMT	SNAPSTMT	Application information regarding statements for the applications connected. This includes the most recent SQL statement executed (if the statement monitor switch is set).
SNAP_GET_TAB_V91	SNAPTAB	Table activity information for each table that was accessed by an application. Requires the table monitor switch.
SNAP_GET_TBSP_V91	SNAPTBSP	Information about table space activity at the database level, and the table space level for each table space that has been accessed. Requires the buffer pool monitor switch.

All the snapshot table functions above belong to the SYSPROC schema, whereas all the snapshot administrative views belong to the SYSIBMADM schema.

Similar to snapshot commands, the amount of information returned from table snapshot functions and administrative views is controlled by the monitor switches. Since snapshots can collect large amounts of diagnostic data, enabling all monitor switches (especially DYNAMIC SQL) can have a very negative impact on database performance.

All the monitoring utilities use the memory heap, which is controlled by the MON_HEAP_SZ database manager parameter. This monitoring heap size should be increased when many applications access snapshot data.

Event monitoring

Event monitors are used to monitor the performance of DB2 over a fixed period of time. The information that can be captured by an event monitor is similar to the snapshots, but event monitors examine transition events in the database, and consider each event as an object. Event monitors can capture information about DB2 events in the following areas:

► Database

   An event of database information is recorded when the last application disconnects from the database.

► Tables

   All active table events will be recorded when the last application disconnects from the database. An active table is one which has been altered or created since the database was activated. The monitor captures the number of rows read and written to the table.

► Deadlocks

   A deadlock event is recorded immediately when a deadlock occurs. The information captured by the monitor focuses on the locks involved in the deadlock and the applications that own them. This monitor also has several very useful additional options to help diagnose the cause of the deadlock. The `with details` option will capture additional information, such as what SQL was being executed when the deadlock occurred, and what locks were held by the application that encountered the deadlock. The `history` option will capture all statements in the current unit of work. Finally, the `values` option will show the data values used as input variables for each SQL statement (if parameter markers were used).

► Buffer pools

   A buffer pool event is recorded when the last application disconnects from the database. The information captured contains the type and volume of use of the buffer pool, use of pre-fetchers and page cleaners, and whether or not direct I/O was used.

► Table spaces

   A table space event is recorded when the last application disconnects from the database. This monitor captures the same information as the buffer pool monitor, but the information is summarized at a table space level.

► Connections

   A connection event is recorded whenever an application disconnects from the database.

► Transactions

A transaction event is recorded whenever a transaction finishes. The event will be written out whenever a COMMIT or ROLLBACK occurs. The monitor captures all of the individual statement data, and also information about the transaction, such as its start and stop time.

► Statements

A statement event is recorded when an SQL statement completes. The monitor records statement start and stop time, CPU used, text of dynamic SQL (or package and section number of static SQL), return code of SQL statement, and other metrics such as fetch count.

Event monitors are created with the CREATE EVENT MONITOR SQL statement. Information about event monitors is stored in the system catalog table, and it can be reused later.

Example 10-25 creates a sample event monitor named DEADLOCK_EVMON (using the `with details history values` options, described above). The query in the example accesses the SYSCAT.EVENTMONITORS view and displays the names of event monitors that have been created in the database.

Example 10-25   Creating a sample event monitor

```
db2 create event monitor deadlock_evmon for deadlocks with details
history values write to table manualstart

db2 select evmonname from
syscat.eventmonitors
```

The output of the DEADLOCK_EVMON monitor is recorded in newly created tables. To check in advance what tables are to be created, or to generate syntax that overrides the default table names, use the db2evtbl tool as shown in Example 10-26.

Example 10-26   Generating table syntax for specified event monitor

```
db2evtbl -evm deadlock_evmon deadlocks with details

CREATE EVENT MONITOR deadlock_evmon
          FOR DEADLOCKS WITH DETAILS
          WRITE TO TABLE
                CONNHEADER (TABLE CONNHEADER_deadlock_evmon,
                            INCLUDES (AGENT_ID,
                                      APPL_ID,
                                      APPL_NAME,
                                      AUTH_ID,
```

[...]

Since the DEADLOCK_EVMON monitor was created with a manual start, it remains inactive after creation. To activate an event monitor, change the state of the event monitor to a value of 1 and use the **event_mon_state()** function to check for the current state, as shown in Example 10-27 (when calling event_mon_state(), specify the event monitor name in uppercase). After activation of DEADLOCK_EVMON, each time a deadlock occurs in the database it will be recorded in the event monitor tables.

Example 10-27   Enabling event monitor

```
db2 set event monitor deadlock_evmon state = 1

db2 values
event_mon_state('DEADLOCK_EVMON')
```

To browse the data collected by event monitor, you can directly access the tables, or use the GUI tool db2eva. The sample db2eva screen capture is presented in Figure 10-6. For more information about db2eva, refer to the *Command Reference,* SC10-4226.
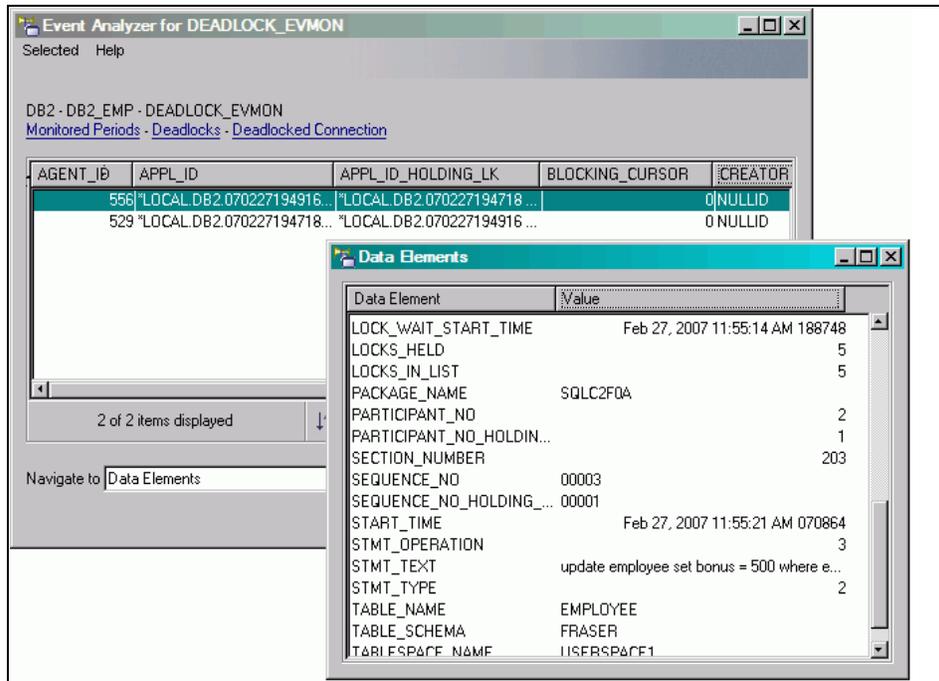
Figure 10-6   Presenting event monitor data using the db2eva GUI tool

Event monitors also offer an option to write the monitored information to a binary file. This option is particularly useful when there is a need to prevent the event monitor from collecting uncontrolled amount of data. Example 10-28 shows the creation of an event monitor that writes the diagnostic data to files (extensions *.EVT) located on the 'c:\tmp\deadlock' directory (Windows example). In this example, if the total amount of collected data exceeds 5000 pages (4 KB) the event monitor will be stopped.

Example 10-28   Creating an event monitor with the file option

```
db2 create event monitor deadlock_evmon for deadlocks with details
    write to file 'c:\tmp\deadlock' maxfilesize 5000
manualstart
```

To convert the event monitor binary files to a user-readable form, use the db2evmon utility, as shown in Example 10-29.

Example 10-29   Formatting event monitor output files

```
C:\tmp>db2evmon -path c:\tmp\deadlock
```

```
Reading c:\tmp\00000000.EVT ...
--------------------------------------------------------------------------
                             EVENT LOG HEADER
  Event Monitor name: DEADLOCK_EVMON
  Server Product ID: SQL09011
  Version of event monitor data: 8
  Byte order: LITTLE ENDIAN
  Number of nodes in db2 instance: 1
  Codepage of database: 1252
  Territory code of database: 1
  Server instance name: DB2
--------------------------------------------------------------------------


--------------------------------------------------------------------------
  Database Name: DB2_EMP
  Database Path: C:\DB2\NODE0000\SQL00001\
  First connection timestamp: 02/27/2007 13:09:48.916779
  Event Monitor Start time:   02/27/2007 13:12:33.277309
--------------------------------------------------------------------------

3) Deadlock Event ...
  Deadlock ID:   4
  Number of applications deadlocked: 2
  Deadlock detection time: 02/27/2007 13:12:35.979437
  Rolled back Appl participant no: 2
  Rolled back Appl Id: *LOCAL.DB2.070227210931
  Rolled back Appl seq number: : 0005
[...]
```

Visual Explain

Visual Explain is used to capture and view information about the access plan chosen by the DB2 optimizer for SQL statements as a graph. An access plan is a cost estimation of resource usage for a query, which is based on available information, such as statistics for tables and indexes, instance and database configuration parameters, bind options and query optimization level, and so on. An access plan also specifies the order of operations for accessing the data.

The access plan acquired from Visual Explain helps to understand how individual SQL statements are executed. The information available from the Visual Explain graph can be used to tune the SQL queries for better performance.

To start Visual Explain, launch the Control Center, right-click the database name and select either the **Explain Query...** or **Show Explained Statements History**

option. You can input an SQL statement manually or import the SQL statement with the Get button available in the Explain Query Statement window. You can also specify the optimization class for the SQL statement in the same window. The optimization class indicates the effort the DB2 optimizer will spend on preparing an execution plan (a higher value means more sophisticated optimization). Figure 10-7 shows an example of an access plan graph.
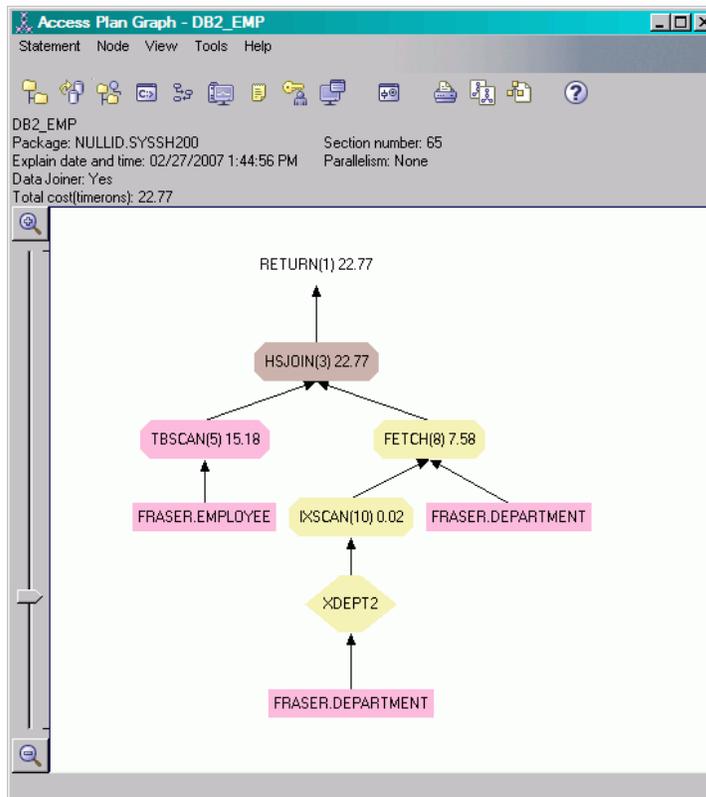
Figure 10-7   A Visual Explain access plan graph

An access plan graph shows details of:

▶ Tables (and their associated columns) and indexes
▶ Operators (such as table scans, sorts, and joins)
▶ Table spaces and functions

To get the details regarding a specific element in the Visual Explain graph, right-click the desired graph element and select **Show Details**. Figure 10-8 shows an example of the details that are listed for the above HSJOIN operator.
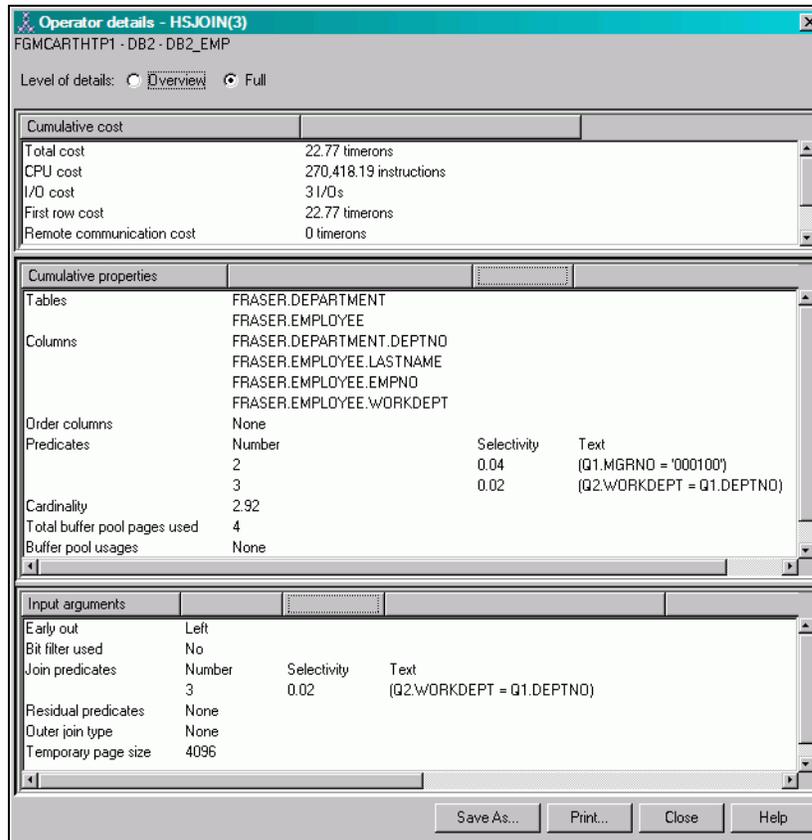
Figure 10-8   Visual Explain operator details

10.5 Initial tuning

The performance of a DB2 database application can be influenced by many factors, such as the type of workload, application design, database design, capacity planning, and instance and database configuration. This section focuses on a number of DB2 performance tuning tips that may be used for initial configuration.

10.5.1 Table spaces

At database creation time, three table spaces are created:

► SYSCATSPACE - Catalog table space for storing information about all the objects in the database.

- ▶ TEMPSPACE1 - System temporary table space for storing internal temporary data required during SQL operations, such as sorting, reorganizing tables, creating indexes, and joining tables.

- ▶ USERSPACE1 - For storing application data.

There are also two additional table spaces, SYSTOOLSPACE and SYSTOOLSTMPSPACE, that support many of the administration tools, SQL administrative routines, and automatic maintenance utilities. They are created the first time any such tool, routine, or utility is used. The SYSTOOLSPACE table space is a user data table space, while SYSTOOLSTMPSPACE is a user temporary table space.

By default, SYSCATSPACE, USERSPACE1, and SYSTOOLSPACE are created as Database Managed Spaces (DMS), while TEMPSPACE1 and SYSTOOLSTMPSPACE are created as System Managed Spaces (SMS).

With DMS table spaces, the database manager controls the storage space, which typically offers better performance than that of SMS for non-temporary data. With SMS table spaces, the operating system's file system manager allocates and manages the space where the table is stored, which is best suited for temporary data.

For SMS table spaces, reading and writing data from tables is buffered by the operating system, and space is allocated according to the operating system conventions: files with .DAT extension for tables and .INX files for table indexes. SMS is almost always a better choice than DMS for temporary table spaces, as there is more overhead in the creation of a temporary table when using DMS. Additionally, the database manager attempts to keep temporary table pages in memory, rather than writing them out to disk. As a result, the performance advantages of DMS are less significant.

For regular (non-temporary) user data, optimal INSERT performance can be achieved with DMS table spaces, since containers are preallocated and management of the I/O operations is shifted to the database engine. Using the AUTORESIZE feature of DMS table spaces means that the DMS table space automatically increases in size when it becomes full. Using automatic storage also has the same effect. Manual commands are also supported that add new containers, drop, or modify the size of existing containers.

For best performance, large volume data and indexes should be placed on DMS table spaces, and, if possible, split to separate containers. Initially, system catalogs and system temporary table spaces should stay on the SMS table spaces. System catalogs contain large objects (LOBs) that are not cached by the DB2 engine, and can be cached by the operating system cache. In an OLTP-like

environment, there is no need for creating large temporary objects to process SQL queries, so the SMS system temporary table space is a good starting point.

10.5.2 Physical placement of database objects

When creating a database, the first important decision is the storage architecture. The ideal situation is to have the fastest disks possible and at least five to 10 disks per processor (for high I/O OLTP workload, use even more). The reality is that the hardware is often chosen based on other considerations, so in order to achieve optimal performance, the placement of database objects should be carefully planned.
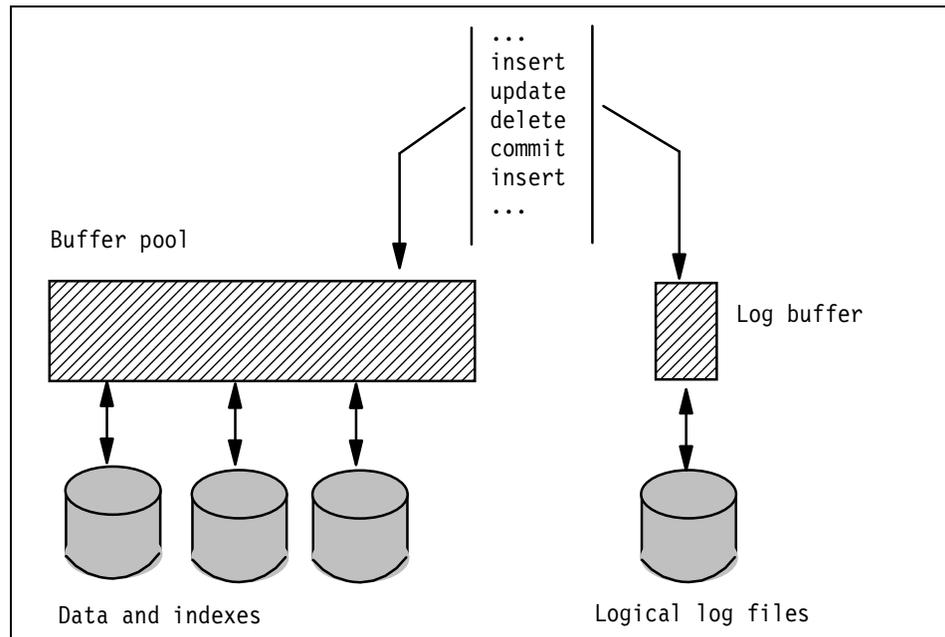
Figure 10-9   Explaining logical log placement

As shown in Figure 10-9, all data modifications are not only written to table space containers, but are also logged to ensure recoverability. Since every INSERT, UPDATE, or DELETE is replicated in the transactional log, the flushing speed of the logical log buffer can be crucial for the entire database performance. To understand the importance of logical log placement, you should keep in mind that the time necessary to write data to disk depends on the physical data distribution on disk. The more random reads or writes are performed, the more disk head movements are required and, therefore, the slowest is the writing speed. Flushing the logical log buffer to disk by its nature is sequential and should not be

interfered with by other operations. Locating logical log files on separate devices isolates them from other processes, and ensures uninterrupted sequential writes.

To change logical log files to a new location, you need to modify the `NEWLOGPATH` database parameter as shown in Example 10-30. The logs will be relocated to the new path during the next database activation (it can take some time to create the files).

Example 10-30   Relocation of logical logs

```
db2 update db cfg for db2_emp using NEWLOGPATH /db2/logs
```

When creating a DMS table space with many containers, DB2 automatically distributes the data across them in a round-robin fashion, similar to the striping method available in disk arrays. To achieve the best possible performance, each table space container should be placed on a dedicated physical device.

For parallel asynchronous writes and reads from multiple devices, the number of database page cleaners (NUM_IOCLEANERS) and I/O servers (NUM_IOSERVERS) should be adjusted. The best values for these two parameters depend on the type of workload and available resources; however, new to DB2 9, both of these database configuration parameters have an AUTOMATIC setting, which is the default.

For NUM_IOCLEANERS, an AUTOMATIC setting bases the number of page cleaners on the number of CPUs and the number of local logical database partitions. Specifically, the following formula is used to calculate the number:

```
number of page cleaners = max( ceil( # CPUs / # local logical DPs )
- 1, 1 )
```

It is recommended to keep the AUTOMATIC setting in most cases. The only time you may want to consider setting the value explicitly is for a read-only environment. In such a case, since there will be no data writes, a value of 0 can save some CPU cycles.

Similarly for NUM_IOSERVERS: the AUTOMATIC setting is recommended. AUTOMATIC will calculate the value based on the parallelism settings of the table spaces in the current database partition (parallelism is controlled by the DB2_PARALLEL_IO profile registry variable, which is discussed shortly). For each DMS table space, the value of this parallelism setting is multiplied by the maximum number of containers in the table space stripe set. For each SMS table space, the value of this parallelism setting is multiplied by the number of containers in the table space. The largest result over all table spaces in the current database partition will be used as the number of prefetchers to start.

Specifically, the following formula is used:

```
number of prefetchers = max( max over all table spaces ( parallelism
setting * [SMS: # containers; DMS: max # containers in stripe set]
), 3 )
```

If a relatively small number of disks are available, it can be difficult to keep database logical logs, data, indexes, system temporary table spaces (more important for processing large queries in warehousing environments), backup files, or operating system paging file on separate physical devices. A compromise solution is to have one large file system striped by disk array (RAID device) and create table spaces with only one container. The load balancing is shifted to hardware, and you do not have to worry about space utilization. In that case to parallel I/O operations on a single container. The DB2_PARALLEL_IO profile registry variable should be set before starting the DB2 engine.

With the following command, I/O parallelism will be enabled within a single container for all table spaces:

```
db2set DB2_PARALLEL_IO="*"
```

The following example enables parallel I/O for only two table spaces: DATASP1 and INDEXSP1:

```
db2set DB2_PARALLEL_IO="DATASP1,INDEXSP1"
```

To check the current value for the parameter, issue:

```
db2set DB2_PARALLEL_IO
```

10.5.3 Buffer pools

DB2 buffer pools are used to cache table and index data pages when they are being read form disk, or when they are being modified. When a row is accessed for the first time, the database manager places the page containing the row into the buffer pool. The next time there is a data request, the buffer pool is first checked for the data before looking on disk. Database performance is greatly improved by allowing data access from memory, as opposed to disk. The less the database manager needs to be read from or write to disk, the greater the performance. For this reason, the proper configuration of buffer pools is one of the most important areas of tuning. Only large objects (LOBs) and LONG data are not manipulated in the buffer pool.

The configuration of buffer pools in DB2 9 has been greatly simplified, thanks to the self-tuning memory manager. Now, achieving optimal buffer pool performance takes very little effort on behalf of the DBA. The self-tuning memory manager redistributes available memory among consumers as workload requirements change. This feature is automatically enabled for any new, single

partition database and can be explicitly controlled by modifying the SELF_TUNING_MEM database configuration parameter. The memory allocated to the database is controlled by the DATABASE_MEMORY database configuration parameter, which is set to AUTOMATIC, by default, on both Windows and AIX, but can also be explicitly set. This controls how much shared memory is available for the database and, in turn, for buffer pools of the database.

By default, buffer pools created in DB2 9 are specified to be AUTOMATIC and therefore are automatically tuned by the database manager. This is the recommended setting, which ensures that memory is available to the buffer pools with greatest need. This is especially important when that need can change over time. If you would rather control buffer pool sizes on your own, you can disable the self-tuning memory manager by setting the database configuration parameter SELF_TUNING_MEM to OFF, or by altering or creating bufferpools without the AUTOMATIC option.

When using the AUTOMATIC option of the buffer pool, the size specified for the buffer pool (either implicitly or explicitly) is used as the initial size. The default initial size for buffer pools is very small: only 250 pages (~ 1 MB) for Windows and 1000 pages (~ 4 MB) for UNIX platforms. You will likely want to increase these sizes to something more acceptable as a starting point for the self-tuning memory manager.

If you are not using the automatic resizing feature of buffer pools, you will definitely want to increase the size of the bufferpools. However, the total buffer pool size should not be set too high, because there might not be enough memory to allocate it. To calculate the maximum buffer size, all other DB2 memory related parameters such as database heap, the agent's memory, storage for locks, as well as the operating system, and any other applications should be considered.

Initially set the total size of buffer pools to 10% to 20% of available memory. You can monitor the system later and correct it. The ALTER BUFFERPOOL statement with the IMMEDIATE option takes effect immediately, except when there is not enough reserved space in the database-shared memory to allocate new space. This feature can be used to tune database performance according to periodic changes in use, for example, switching from daytime interactive use to nighttime batch work.

Having more than one buffer pool is generally recommended, to preserve data in the buffers. For example, let us suppose that a database has many very frequently used small tables, which would normally be in the buffer in their entirety, and thus would be accessible very fast. Now let us suppose that there is a query which runs against a very large table, which uses the same buffer pool and involves reading more pages than the total buffer size. When this query

runs, the pages from the small, very frequently used tables will be lost, making it necessary to re-read them when they are needed again.

At the start you can create an additional buffer pool for caching data and leave the IBMDEFAULTBP for system catalogs. Creating an extra buffer pool for system temporary data also can be valuable for system performance, especially in an OLTP environment, where the temporary objects are relatively small. Isolated temporary buffer pools are not influenced by the current workload, so it should take less time to find free pages for temporary structures, and it is likely that the modified pages will not be swapped out to disk. In a warehousing environment, the operations on temporary table spaces are considerably more intensive, so the buffer pools should be larger, or combined with other buffer pools if there is not enough memory in the system (one pool for caching data and temporary operations).

Example 10-31 shows how to create buffer pools assuming that the additional table space DATASPACE for storing data and indexes was already created, and that there is enough memory in the system. You can take this as a starting buffer pool configuration for a 2 GB RAM system. This example makes use of the AUTOMATIC resizing feature of the buffer pools.

Example 10-31   Increasing buffer pools

```
connect to db2_emp;
    -- creating two buffer pools with initial size of 256 MB and 64 MB
create bufferpool DATA_BP immediate size 65536 automatic pagesize 4k;
create bufferpool TEMP_BP immediate size 16384 automatic pagesize 4k;


    -- changing size of the default buffer pool
alter bufferpool IBMDEFAULTBP immediate size 16384 automatic;


    -- binding the table spaces to buffer pools
alter tablespace DATASPACE bufferpool DATA_BP;
alter tablespace TEMPSPACE1 bufferpool TEMP_BP;


    -- checking the results
select
    substr(bs.bpname,1,20) as BPNAME
    ,bs.npages
    ,bs.pagesize
    ,substr(ts.tbspace,1,20) as TBSPACE
from syscat.bufferpools bs join syscat.tablespaces ts on
     bs.bufferpoolid = ts.
bufferpoolid;
```

The results:

```
BPNAME                 NPAGES      PAGESIZE    TBSPACE
-------------------- ----------- ----------- --------------------
IBMDEFAULTBP                 -2        4096 SYSCATSPACE
IBMDEFAULTBP                 -2        4096 USERSPACE1
DATA_BP                      -2        4096 DATASPACE
TEMP_BP                      -2        4096 TEMPSPACE1
```

The -2 value in the NPAGES column indicates that the buffer pool is using the autoresize feature. A value of -1 indicates that the buffer pool is being sized based on the database configuration parameter BUFFPAGE. Any other value will show the actual number of pages that the buffer pool consists of.

In the case of AUTOMATIC buffer pools, you can use a buffer pool snapshot to obtain the current size of the buffer pool, as Example 10-32 demonstrates.

Example 10-32   Obtaining current buffer pool size for AUTOMATIC buffer pools

```
C:\SQLLIB\BIN>db2 get snapshot for bufferpools on db2_emp


                      Bufferpool Snapshot

Bufferpool name                          = IBMDEFAULTBP
Database name                            = DB2_EMP
Database path                            = C:\DB2\NODE0000\SQL00008\
Input database alias                     = DB2_EMP
Snapshot timestamp                       = 02/28/2007 13:51:12.485208
...


Alter bufferpool information:
 Pages left to remove                    = 0
 Current size                            = 8192
 Post-alter size                         = 8192
```

The Current size shows the size in pages. In this case, the IBMDEFAULTBP buffer pool is 8192 4K pages, which is 32 MB.

If result sets obtained by your application often consist of a sequence of consecutive rows, or you notice that DB2 often performs prefetching of data, you could benefit from the use of a block-based buffer pool. By default, all buffer pools are page-based, which means that prefetches will place contiguous pages on disk into non-contiguous pages in memory. Using a block-based buffer pool allows DB2 to use block I/O to read multiple pages into the buffer pool in a single I/O, which greatly improves the performance of sequential prefetching.

A block-based buffer pool consists of both the standard page area and a block area. The NUMBLOCKPAGES parameter of the CREATE and ALTER BUFFERPOOL statements is used to define the size of the block memory, while the BLOCKSIZE parameter specifies the size of the blocks, and hence the number of pages to read from a disk in a block I/O. Table spaces that share the same extent size should be the only users of a specific block-based buffer pool. Set the BLOCKSIZE equal to the table space's EXTENT SIZE that is using the buffer pool.

In most cases, even allocating a very small amount of your buffer pool to the block-based area can show significant performance gains. A good starting point is around 1 to 3% of the initial buffer pool size.

The CHNGPGS_THRESH database configuration parameter specifies the percentage of changed pages at which the asynchronous page cleaners will be started. Asynchronous page cleaners will write changed pages from the buffer pool to disk. The default value for the parameter is 60%. When that threshold is reached, some users may experience a slower response time. Having larger buffer pools means more modified pages in memory and more work to be done by page cleaners, as shown in Figure 10-10. To guarantee a more consistent response time and also a shorter recovery phase, lower the value to 50 or 40 using the following command:

```
db2 update db cfg for sample using CHNGPGS_THRESH 40
```
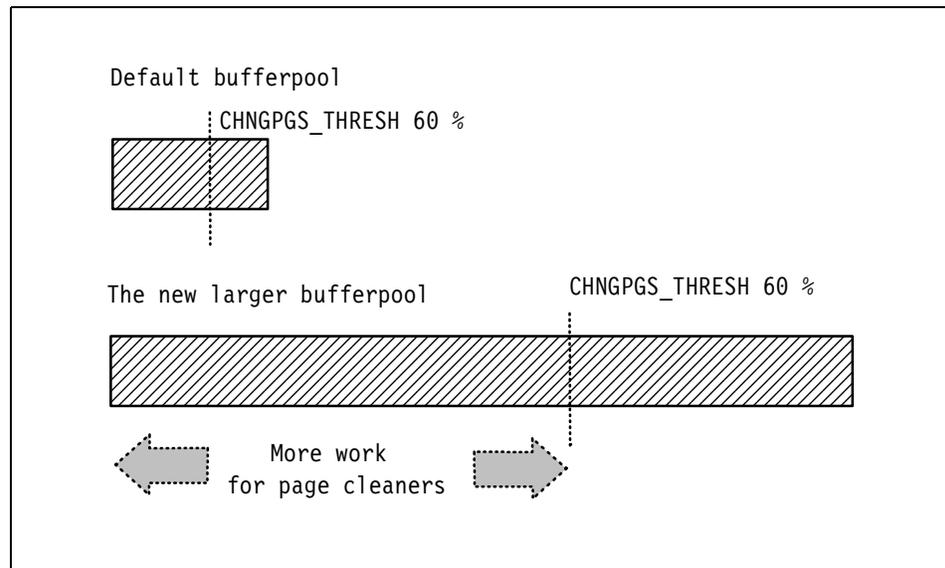

Figure 10-10   Visualizing the CHNGPGS_THRESH parameter

10.5.4 Large transactions

By default, databases are created with a relatively small space for transactional logs, only three log files for each 1000 pages. A single transaction should fit the available log space to be completed. If not, the transaction is rolled back by the database manager (`SQL0964C The transaction log for the database is full`). To process transactions that are modifying large numbers of rows, adequate log space is needed. Current total log space available for transactions can be calculated by multiplying the size of the log files (database parameter LOGFILSIZ) and the number of logs (database parameter LOGPRIMARY).

From a performance perspective, it is better to have a larger log file size, because of the cost of switching from one log to another. When log archiving is switched on, the log size also indicates the amount of data for archiving. In this case, a larger log file size is not necessarily better, since a larger log file size may increase the chance of failure, or cause a delay in archiving or log shipping scenarios. The log size and the number of logs should be balanced.

Example 10-33 allocates 400 MB of total log space.

Example 10-33   Resizing the transactional log

```
db2 update db cfg for sample using LOGFILSIZ 5120
db2 update db cfg for sample using LOGPRIMARY 20
```

Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Each database has its own list of locks (a structure stored in memory that contains the locks held by all applications concurrently connected to the database). The size of the lock list is controlled by the LOCKLIST database parameter.

The default storage for LOCKLIST on Windows and UNIX is set to AUTOMATIC. On 32-bit platforms, each lock requires 48 or 96 bytes of the lock list, depending on whether other locks are held on the object or not. On 64-bit platforms, each lock requires 64 or 128 bytes of the lock list, depending on whether other locks are held on the object or not.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Since the memory tuner trades memory resources among different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

The value of LOCKLIST is tuned together with the MAXLOCKS parameter. Therefore, disabling self tuning of the LOCKLIST parameter automatically disables self tuning of the MAXLOCKS parameter. Enabling self tuning of the LOCKLIST parameter automatically enables self tuning of the MAXLOCKS parameter.

Automatic tuning of this configuration parameter will only occur when self tuning memory is enabled for the database (the SELF_TUNING_MEM database configuration parameter is set to ON).

When the maximum number of lock requests has been reached, the database manager replaces existing row-level locks with table locks (lock escalation). This operation reduces the requirements for lock space, because transactions will hold only one lock on the entire table instead of many locks on every row. Lock escalation has a negative performance impact because it reduces concurrency on shared objects. Other transactions must wait until the transaction holding the table lock commits or rollbacks work. Setting LOCKLIST to AUTOMATIC avoids this situation, as the lock list will increase synchronously to avoid lock escalation or a "lock list full" situation.

To check current usage of locks, use snapshots such as in Example 10-34.

Example 10-34   Invoking snapshot for locks on database db2_emp

```
db2 get snapshot for locks on db2_emp
```

The snapshot collects the requested information at the time the command was issued. In Figure 10-11 you can find sample lock snapshot output. For the time the snapshot was run, there were two applications connected to the database DB2_EMP, and in total 1151 locks were acquired on the database. Issuing the **get snapshot** command later can produce different results because in the meantime the applications may commit the transaction and release locks.

Figure 10-11   Explaining lock snapshot information

To check lock escalation occurrences, look at the db2diag.log file. The lock escalation message should look as shown in Example 10-35.

Example 10-35   Lock escalation message in the db2diag.log file

```
2007-02-28-16.38.28.515000-240 E394355H479          LEVEL: Warning
PID    : 467172              TID  : 1          PROC : db2agent (SAMPLE) 0
INSTANCE: db2inst1           NODE : 000        DB   : DB2_EMP
APPHDL  : 0-191              APPID: *LOCAL.DB2.070228202559
AUTHID  : DB2INST1
FUNCTION: DB2 UDB, data management, sqldEscalateLocks, probe:3
MESSAGE : ADM5502W  The escalation of "1136" locks on table "DB2INST1
          .TABLE01" to lock intent "X" was successful.
```

Logical log buffer

The default size for the logical log buffer is eight pages (32 KB)—often too small for an OLTP database, and not big enough for long-running batch processes. In

most cases the log records are written to disk when one of the transactions issues a COMMIT, or the log buffer is full. Increasing the size of the log buffer may result in more efficient I/O operations, especially when the buffer is flushed to disk, because of the second reason. The log records are written to disk less frequently and more log records are written each time. Initially, set LOGBUFSZ to 128 or 256 4 KB pages. The log buffer area uses space controlled by the DBHEAP database parameter, so consider increasing this parameter also.

Later, use the snapshot for applications to check current usage of log spaces by transactions, as shown in Example 10-36.

Example 10-36   Current usage of log space by applications

```
$db2 update monitor switches using uow on
$db2 get snapshot for applications on db2_emp| grep "UOW log"

UOW log space used (Bytes)              = 478
UOW log space used (Bytes)              = 21324
UOW log space used (Bytes)              = 110865
```

Before running the application snapshot, the Unit Of Work monitor should be switched on. At the time the snapshot was issued, only three applications were running on the system. The first transaction used 478 bytes of log space, the second 21324, and the last used 110865, which is roughly 28 pages, more than the default log buffer size. The snapshot gives only current values from the moment the command was issued. To get more valuable information about the usage of log space by transactions, run the snapshot many times.

Example 10-37 shows how to get information about log I/O activity.

Example 10-37   Checking log I/O activity

```
db2 reset monitor for database db2_emp
   # let the transactions run for a while

db2 get snapshot for database on db2_emp > db_snap.txt
egrep  -i "commit|rollback" db_snap.txt

Commit statements attempted             = 23
Rollback statements attempted           = 2
Internal commits                        = 1
Internal rollbacks                      = 0
Internal rollbacks due to deadlock      = 0

grep "Log pages" db_snap.txt
```

```
Log pages read                              = 12
Log pages written                           = 630
```

Before running the database snapshot, you may have to reset the monitors. The values gathered by snapshot and presented here are cumulated since the last monitor reset or database activation, so wait for a certain period after resetting the counters. For convenience, the snapshot output was directed into a file, and then analyzed using the UNIX grep tool. In the example, 630 pages were written for the period, which gives about `630 / (23+2+1) = 25` pages per transaction. Looking at the value "Log pages written" it is not possible to tell what was the average size of transactions, because the basic DB2 read or write unit is one page (4 KB). Issuing only one small INSERT will force a flush of 4 KB from log buffer to disk. The partially filled log page remains in the log buffer, and can be overwritten to disk more than once, until it is full; this guaranties that the log files are contiguous.

When setting the value for log buffer, also look at the ratio between *log pages read* and *log pages written*. An ideal value is zero log pages read, while seeing a large number of log pages written. When there are too many log pages read, it means a bigger LOGBUFSZ can improve performance.

10.5.5 SQL execution plan

When a query is issued against a database, DB2 prepares an execution plan. The execution plan defines the necessary steps that should be done to get the requested data. In order to prepare an optimal execution plan, the DB2 optimizer considers many elements such as configuration parameters, available hardware resources, or the characteristics of the database objects (available indexes, table relationships, number of records, data distribution). The database characteristics are collected with the RUNSTATS utility, and are stored in special system catalog tables.

Deciding which statistics to collect for a specific workload can be complex, and making sure that the statistics are up to date can be time-consuming. Fortunately, DB2 9 offers automatic statistics collection as part of the Automated Table Maintenance feature, which is enabled by default when a new database is created. DB2 determines when and what statistics need to be collected. When automatic statistics collection is enabled, DB2 automatically runs the RUNSTATS utility in the background to ensure that the most current statistics are available to the optimizer.

Automatic statistics collection minimizes the impact of RUNSTATS in several ways:

► Throttling of RUNSTATS limits the resources consumed based on the current database activity. If database activity increases, RUNSTATS is throttled to run more slowly, reducing resource consumption.

► By default, basic table statistics with distribution information and detailed index sampling is performed. This can also be customized by enabling statistics profiling, where information about previous database activity is used to determine exactly what kind of statistics are required by database workload.

► Only highly active tables (measured by number of inserts, updates, and deletes) are considered for statistics collection. Large tables, consisting of more than 4000 pages, are sampled to determine if statistics are out of date, and collected only if needed.

► RUNSTATS is scheduled to run only during the maintenance window that you specify. You can also limit the automatic statistics collection to only a set of tables.

► Even during automatic statistics collection, the affected tables are available for regular database activity.

If you choose to not use automatic statistics collection and instead perform manual RUNSTATS, it should be used when:

► A table has been loaded with new data

> **Recommendation:** After loading data to DB2 tables, run RUNSTATS before starting testing.

► The appropriate indexes have been created

► There have been extensive updates, deletions, and insertions that affect a table and its indexes (for example, 10% to 20% of the table and index data has been affected).

► Data has been physically reorganized (by running the REORG utility, or adding new containers)

The RUNSTATS command should be run against each table in the database. The DB2 Control Center can be very helpful with running statistics on a group of tables. To run statistics using Control Center, select the desired tables (to select more than one table, press the Ctrl or Shift key while clicking the table names; to select all tables, click any table name and then press Control + A), right-click the selection, and choose the **Run Statistics...** option, as shown in Figure 10-12.
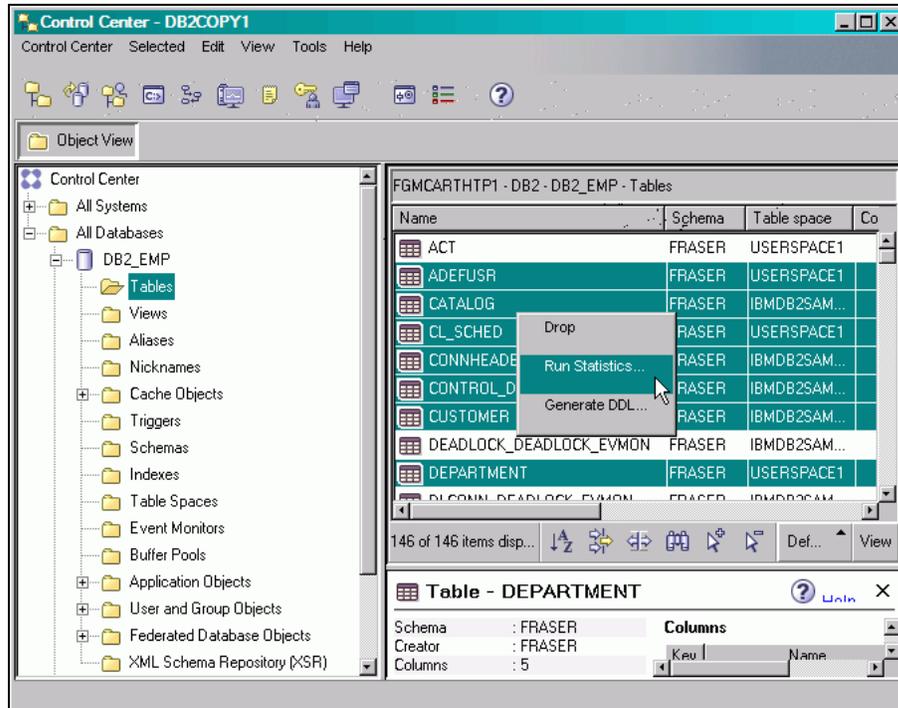
Figure 10-12   Running RUNSTATS on multiple tables

On the second tab, Statistics, you can specify options for the RUNSTATS command. You can start with collecting basic statistics on all columns and indexes, and distribution of values only for key columns, as presented in Figure 10-13. After setting the RUNSTATS option, you can execute the commands by clicking **OK**.
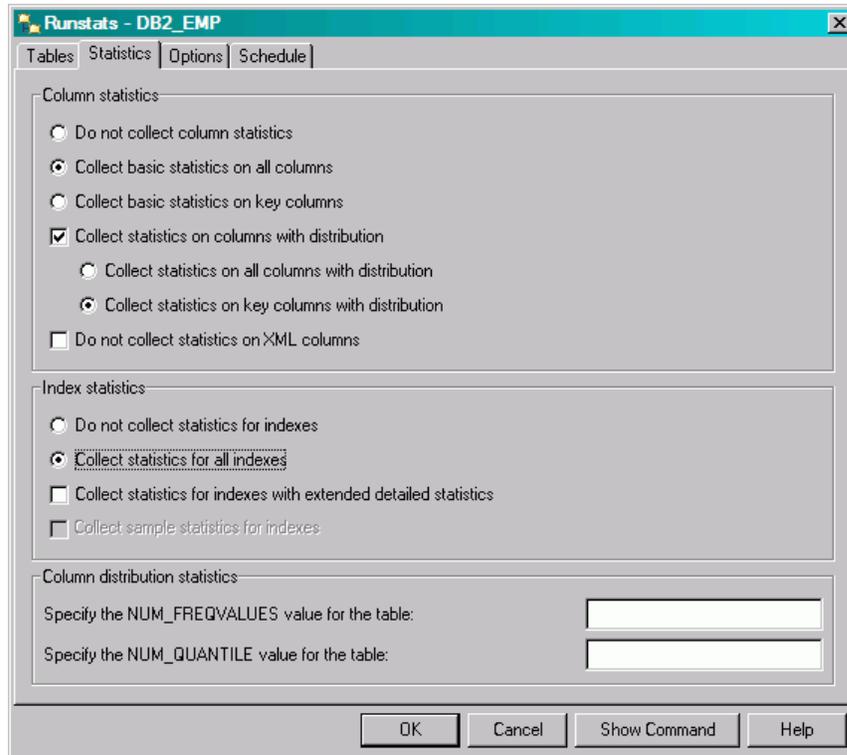
Figure 10-13   RUNSTATS command options

DB2 comes with a very powerful query optimization algorithm. This cost-based algorithm will attempt to determine the cheapest way to perform a query against a database. Items such as the database configuration, database physical layout, table relationships, and data distribution are all considered when finding the optimal access plan for a query. To check the current execution plan, you can use the Explain Query function (described in "Visual Explain" on page 485).

10.5.6 Configuration Advisor

The Configuration Advisor is a tool that can be helpful in preparing an initial DB2 configuration. It is used to generate recommendations for the initial values of buffer pool size, database configuration parameters, and database manager configuration parameters, based on the database environment characteristics, such as CPU speed and workload type. It can be run manually, through a GUI or command line interface, and automatically during database creation.

By default in DB2 9, any new databases will run the Configuration Advisor in the background and have such configuration recommendations automatically applied. To disable this feature, or to explicitly enable it, you must use the **db2set** command, as Figure 10-38 illustrates.

Example 10-38   Configuring default behavior of the Configuration Advisor

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

In any case, the Configuration Advisor can be manually run at any time against a database to update the current configuration, regardless of the DB2_ENABLE_AUTOCONFIG_DEFAULT setting. All recommendations are based on the input that you provide and system information that the Configuration Advisor gathers. The generated recommendations can be applied or simply displayed.

It is important to point out that the values suggested by the Configuration Advisor are relevant for only one database per instance. If you want to use this advisor on more than one database, each database should belong to a separate instance.

The Configuration Advisor can be manually invoked with the AUTOCONFIGURE command from the command line processor (CLP), either standalone or as part of the CREATE DATABASE command. Additionally, it can also be run via a GUI available in the Control Center, or by calling the db2AutoConfig API, and finally by using the ADMIN_CMD stored procedure.

For those new to DB2, the best place to explore the capabilities of the Configuration Advisor is from the GUI that can be launched from the Control Center. In the Control Center, right-click the database you wish to configure and select **Configuration Advisor...**. Within the GUI, you will be presented with a series of interview questions, which collect information regarding the percentage of memory dedicated to DB2, type of workload, number of statements per transaction, transaction throughput, trade-off between recovery and database performance, data volume, number of applications, and isolation level of applications connected to the database. Figure 10-14 shows one of the interview panels.
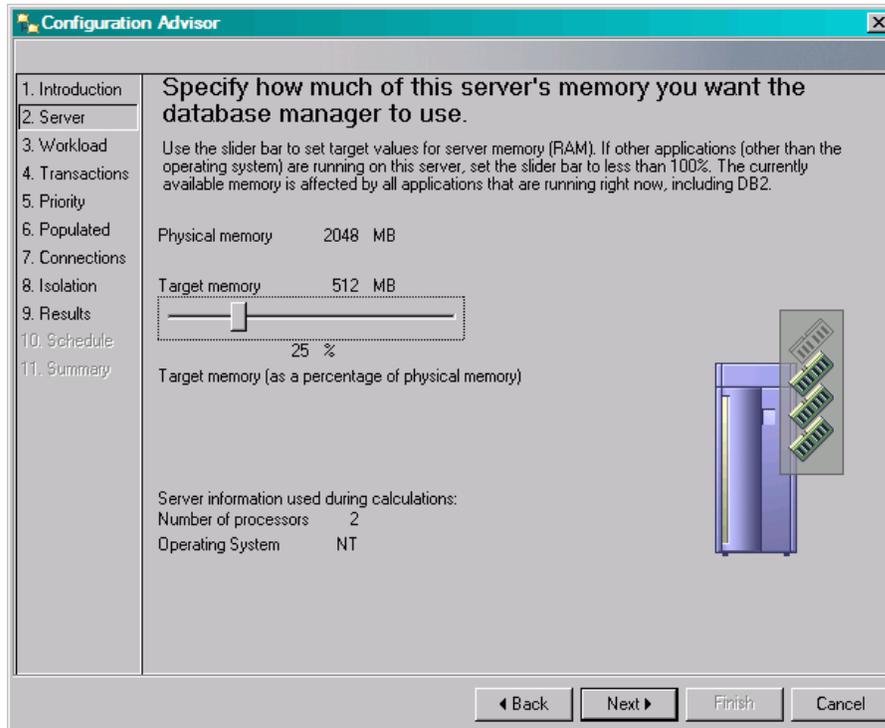
Figure 10-14   Configuration Advisor memory interview screen

Based on the supplied answers, the Configuration Advisor displays the proposed configuration changes, as shown in Figure 10-15.
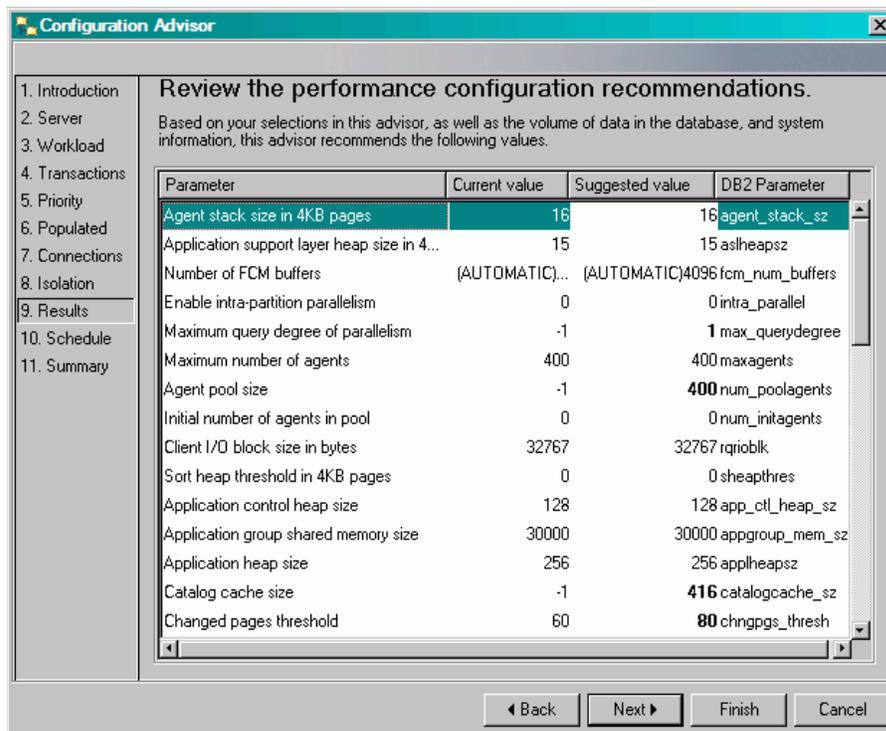
Figure 10-15   Sample of proposed performance configuration recommendations

You are then given the option to save the recommendations to disk, apply the recommendations, or save them to the Task Center for later execution, as shown in Figure 10-16.
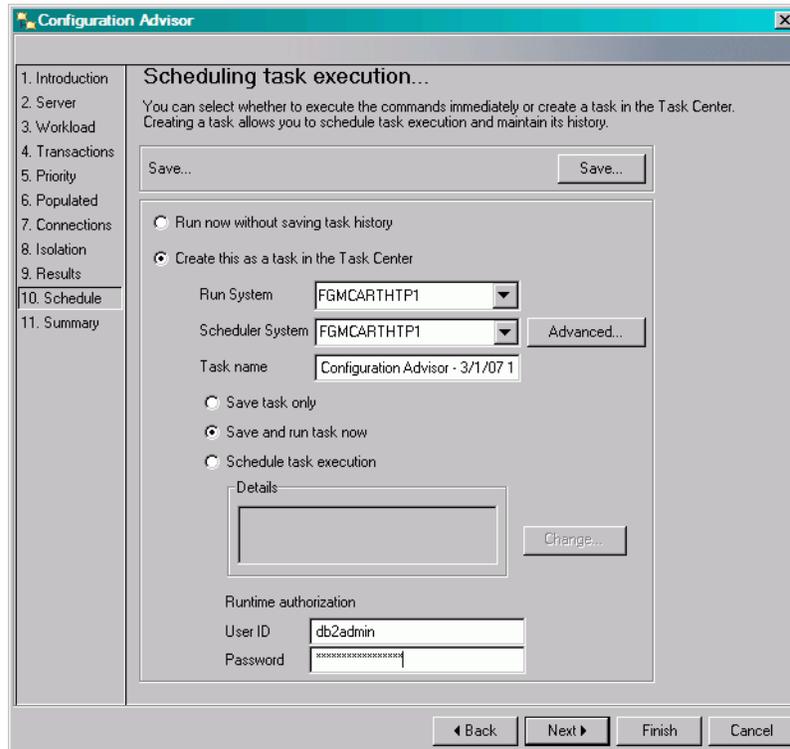
Figure 10-16   Scheduling Configuration Advisor recommendations

Configuration recommendations can also be acquired with the text-based AUTOCONFIGURE command. Example 10-39 shows a sample execution of the command.

Example 10-39   AUTOCONFIGURE command

```
db2 autoconfigure using mem_percent 40 tpm 300 num_local_apps 80 isolation CS apply none

[...]
Current and Recommended Values for Database Configuration

Description                             Parameter  Current Value  Recommended Value
-----------------------------------------------------------------------------------
Max appl. control heap size (4KB)   (APP_CTL_HEAP_SZ) = 4096              128
Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000          9908
Default application heap (4KB)            (APPLHEAPSZ) = 256              256
Catalog cache size (4KB)            (CATALOGCACHE_SZ) = (MAXAPPLS*4)      404
Changed pages threshold             (CHNGPGS_THRESH) = 40                60
Database heap (4KB)                          (DBHEAP) = 600              1461
Degree of parallelism                    (DFT_DEGREE) = 1                1
Default tablespace extentsize (pages)  (DFT_EXTENT_SZ) = 32              32
```

[. . .]

Table 10-4 lists all AUTOCONFIGURE command parameters.

Table 10-4   Parameters of the AUTOCONFIGURE command

Keyword	Values	Explanation
mem_percent	1-100 default: 25	Percentage of memory to dedicate to DB2
workload_type	simple, mixed, complex default: mixed	Type of workload: simple for transaction processing, complex for warehousing
num_stmts	1-1 000 000 default: 10	Number of statements per unit of work
tpm	1-200 000 default: 60	Transactions per minute
admin_priority	performance, recovery, both default: both	Optimize for better performance or better recovery time
is_populated	yes, no default: yes	Is the database populated with data?
num_local_apps	0-5 000 default: 0	Number of connected local applications
num_remote_apps	0-5 000 default: 10	Number of connected remote applications
isolation	RR, RS, CS, UR default: RR	Isolation levels: Repeatable Read, Read Stability, Cursor Stability, Uncommitted Read
bp_resizeable	yes, no default: yes	Are buffer pools re-sizeable?

The Configuration Advisor also complements the self-tuning memory feature. When the self-tuning memory feature is enabled, by setting the database configuration parameter SELF_TUNING_MEM to ON (the default), and when the Database Partitioning Feature (DPF) is not being used, the self-tuning memory feature will dynamically refine some of the Configuration Advisor recommendations based on the demands of the current database workload.

10.5.7 Design Advisor

Well designed indexes are essential to database performance. The Design Advisor can help you significantly improve your workload performance by identifying indexes, materialized query tables (MQTs), multidimensional clustering (MDCs) tables, or database partitions that can benefit your applications. The Design Advisor can also tell you whether existing indexes and MQTs are unused for the workload.

Like other DB2 tools, the recommendations can be implemented immediately or scheduled for a later time. The tool can be invoked through a GUI, available in the Control Center, or through the command line using the DB2ADVIS command.

This utility accepts one or more SQL statements and their relative frequency, known as a workload. It is very useful when planning for or setting up a new database. Additionally, it is the recommended approach for workload performance tuning, as it can help you to:

▶   Improve performance of a particular statement or workload.

▶   Improve performance of the most frequently executed queries, for example, as identified by the Activity Monitor.

▶   Test the performance of an index, without having to create the index in the database.

▶   Find objects that are not used in a workload.

Example 10-40 shows a simple **db2advis** call against the single SQL query; for more options, type db2advis -h from the command line.

Example 10-40   Finding indexes for a particular query

```
db2advis -d db2_emp -s "select firstnme, lastname, deptname from department d,
employee e where d.deptno = e.workdept and e.lastname like 'W%'"

Using user id as default schema name. Use -n option to specify schema
execution started at timestamp 2007-03-01-11.54.16.750000
Recommending indexes...
total disk space needed for initial set [    0.013] MB
total disk space constrained to        [   13.942] MB
Trying variations of the solution set.
Optimization finished.
  1  indexes in current solution
 [ 23.0000] timerons  (without recommendations)
 [ 15.0000] timerons  (with current solution)
 [34.78%] improvement
--
--
-- LIST OF RECOMMENDED INDEXES
```

```
-- ===========================
-- index[1],    0.013MB
   CREATE INDEX "IDX703011954190000" ON "EMPLOYEE"
       ("LASTNAME" ASC, "FIRSTNME" ASC, "WORKDEPT" ASC) ALLOW REVERSE SCANS;
   COMMIT WORK ;
   RUNSTATS ON TABLE "EMPLOYEE" FOR INDEX "IDX703011954190000" ;
   COMMIT WORK ;
--
--
-- RECOMMENDED EXISTING INDEXES
-- ===========================
--
--
-- UNUSED EXISTING INDEXES
-- ===========================
-- DROP INDEX "XDEPT2";
-- DROP INDEX "XDEPT3";
-- DROP INDEX "XEMP2";
-- ===========================
--
7 solutions were evaluated by the advisor
DB2 Workload Performance Advisor tool is finished.
```

Launching Design Advisor in a GUI environment

The Design Advisor can also be invoked as a GUI tool. From the Control Center, expand the object tree to find the Database folder. Right-click the desired database and select **Design Advisor...**. The wizard guides you through the necessary steps, and also helps construct a workload by looking for recently executed SQL queries, or looking through the recently used packages. In order to get accurate recommendations, it is important to have the current catalog statistics. With the Design Advisor there is an option to collect the required basic statistics; however, this increases the total calculation time. Figure 10-17 shows a sample Design Advisor window.
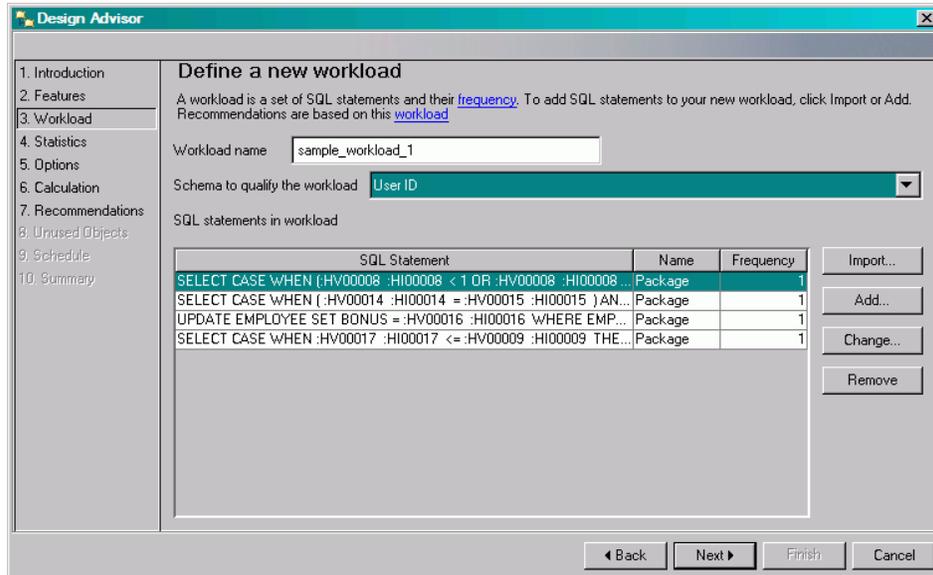
Figure 10-17   The Design Advisor

For more information on the Design Advisor, refer to *Performance Guide,*
SC10-4222.

11

Database administration and management

In this chapter we introduce the tools, wizards, and commands commonly used for database administrative tasks in DB2 management. The intention is to provide an overview of these tools and commands and their equivalents in Oracle, if applicable, to help database administrators (DBAs) who are new to DB2 to become familiar with DB2 quickly, for those operations that are performed daily.

This chapter covers the following sections:

- ► DB2 administration tools
- ► Instance Management commands
- ► Database Management commands
- ► Object Management commands
- ► Automatic Database Management Features introduced in DB2 9
- ► DB2 monitoring commands

Database Partitioning Feature is not covered in this discussion. For complete information about DB2 commands, see *DB2 9.1 Command Reference,* SC10-4226.

11.1 DB2 administration tools

DB2 provides both command line interface (CLI) and graphical user interface (GUI) tools for administering a DB2 environment. You can use both command line and graphic tools to perform most database administrative tasks. However, there are a few administrative activities that require a specific tool; for example, to schedule a job you must use the Task Center GUI tool.

Here we discuss two tools that are commonly used to administer DB2 environments:

► The DB2 command line processor (CLP), which is a CLI tool
► The DB2 Control Center, which is a GUI tool

The DB2 CLP can be considered as being equivalent to the Oracle SQL*Plus. The DB2 Control Center can be considered as being the equivalent of the Oracle Enterprise Manager.

In the following sections, we discuss these tools in more detail.

11.1.1 DB2 command line processor

The DB2 CLP is the tool used to interact with DB2 instances and databases. You can use DB2 CLP to perform almost all administrative tasks related to DB2 instances and databases, and also to obtain online help.

Commands can be entered in both upper case and lower case. However, case-sensitive parameters must be typed in exactly the same case as required by the command.

The DB2 CLP is invoked by using the **db2** utility. DB2 CLP has three operation modes:

► Interactive input mode: You type commands interactively; this is characterized by the presence of $db2 =>$ input prompt.

► Command mode: Each command is prefixed by db2 utility.

► Batch mode: A text file containing DB2 commands is run using the *-f* file input option.

Example 11-1 shows how you can invoke DB2 CLP in each of the three modes and how you can obtain online help for the syntax of LIST TABLES command.

Example 11-1   Invoking DB2 CLP

```
-- Interactive mode
-- At the command prompt, just type db2 and press ENTER
```

```
[db2inst1]/home/db2inst1 # db2

(c) Copyright IBM Corporation 1993,2002
Command Line Processor for DB2 ADCL 9.1.1

You can issue database manager commands and SQL statements from the command
prompt. For example:
    db2 => connect to sample
    db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be the first few keywords
of a database manager command. For example:
 ? CATALOG DATABASE for help on the CATALOG DATABASE command
 ? CATALOG          for help on all of the CATALOG commands.

To exit db2 interactive mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the Online Reference Manual.

db2 => ? LIST TABLES
LIST TABLES [FOR {USER | ALL | SYSTEM | SCHEMA schema-name}] [SHOW DETAIL]

-- Command mode
-- Just type in the command prompt db2 and the command you want to run
--

[db2inst1]/home/db2inst1 # db2 ? list tables
LIST TABLES [FOR {USER | ALL | SYSTEM | SCHEMA schema-name}] [SHOW DETAIL]

-- Batch mode
-- Create a file and run with db2 -f command
[db2inst1]/home/db2inst1/ # echo "? list tables" > db2test.sql
[db2inst1]/home/db2inst1/ # db2 -f db2test.sql
LIST TABLES [FOR {USER | ALL | SYSTEM | SCHEMA schema-name}] [SHOW
DETAIL]
```

The DB2 CLP uses environment variables and configurations stored in the
registry profile to operate. See 11.2.4, "Setting registry variables" on page 533
for more information about how to configure the environment and the registry
variables.

The DB2 CLP consists of two processes: the front-end process, which acts as
the user interface, and the back-end process, which maintains the database

connection. The front-end process is started each time the **db2** command is issued. The back-end process is started by the first **db2** command invocation.

To finish the DB2 CLP processes you can use the QUIT, CONNECT RESET or TERMINATE commands. See "QUIT, CONNECT RESET, and TERMINATE" on page 538, for more details.

With DB2 CLP you can specify options to control the behavior of the CLP, such as to automatically commit SQL statements, to log commands in a history file, or to redirect the output messages to a specific file. For a complete discussion about DB2 CLP features and their usage, refer to Chapter 2 "Command Line Processor (CLP)" in *Command Reference*, SC10-4226.

11.1.2 DB2 Control Center

You can use the DB2 Control Center to manage and administer DB2 systems, instances, databases and database objects, such as tables and views. From the DB2 Control Center you can start other DB2 tools and wizards in order to perform specific tasks.

Some of the key tasks that you can perform with the DB2 Control Center are:

► Manage instances and databases
► Manage database objects
► Manage security
► Manage data
► Perform backup and recovery operations
► Manage applications
► Analyze queries using Visual Explain to look at the access plans
► Launch other tools

To open the Control Center:

► On Windows, click **Start → Programs → IBM DB2 → General Administration Tools → Control Center.**
► On Linux, open the IBM DB2 folder on the desktop and click **Control Center**.

The Control Center interface is available in three views:

► Basic: This view provides basic functionality. You can manage essential objects, such as databases, tables, and stored procedures.

► Advanced: This view displays all objects and actions available in the Control Center.

► Custom: With this view you can specify what options are to be displayed.

Figure 11-1 shows the DB2 Control Center with the Advanced interface.
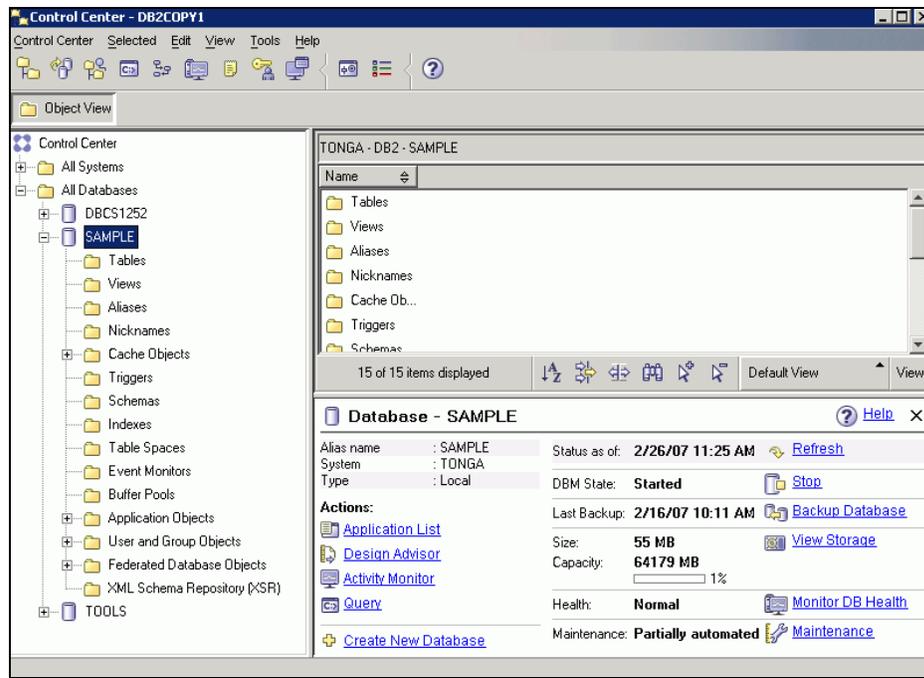
Figure 11-1   DB2 Control Center

When you right-click an object in the left pane, a pop-up menu with available administrative commands for that object is displayed. When you click the desired task, a window or notebook opens to guide you through the steps required to complete the task for the selected object.

Figure 11-2 shows how you can launch the other DB2 tools from the Control Center.
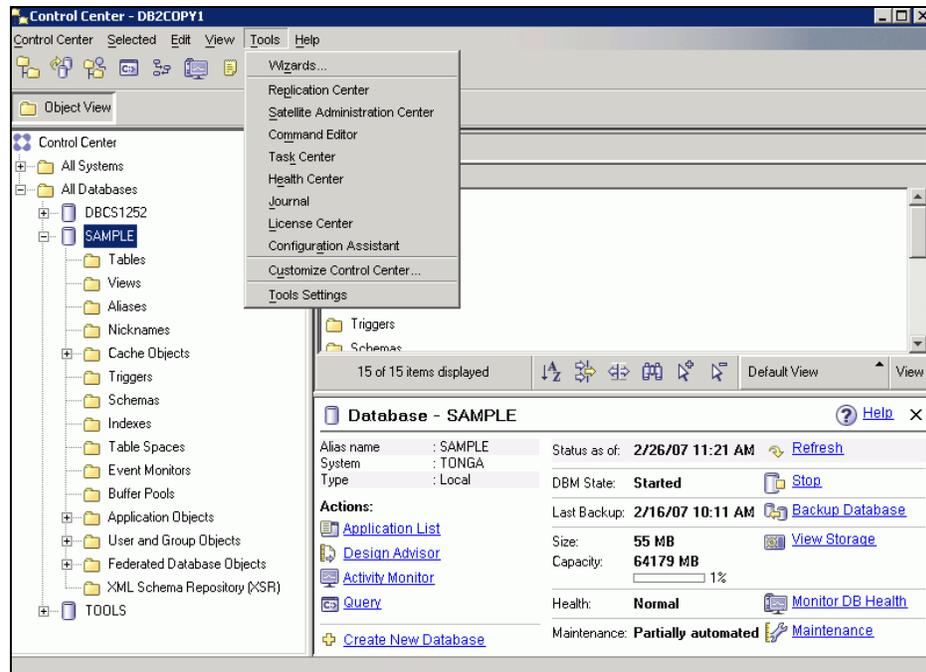
Figure 11-2   Using Control Center to open other tools

Table 11-1 lists the tools that can be launched from the Control Center, and their functionality.

Table 11-1   Tools that can be accessed from Control Center

Tool	Function
Replication Center	Used to configure and administer replication between a DB2 database and another relational database (DB2 or non-DB2).
Satellite Administration Center	Used to administer satellites and information that is stored in the satellite tables.
Command Editor	Used to run commands against a DB2 database. Can be considered as a graphical version of the DB2 CLP.
Task Center	Used to create, schedule and run tasks.

Tool	Function
Health Center	Used to monitor instances. This tool alerts you to potential problems and provides recommendations to resolve those problems.
Journal	Used to list job execution history and view notification log entries.
License Center	Displays license and statistics for DB2 products installed in the server.
Configuration Assistant	Used to configure your DB2 clients to work with database objects in the server.

The DB2 Control Center provides wizards which help you with administrative tasks by providing you step-by-step guidance for completing each task. Table 11-2 shows a list of commonly used wizards.

Table 11-2   DB2 Control Center Wizards

Wizard	Function
Backup Wizard	Backup DB2 databases.
Create database Wizard	Create a database.
Create table space Wizard	Create a table space.
Design Advisor	Analyzes the database workload and makes performance-improving recommendations. This wizard can generate recommendations for creation of indexes, materialized query tables (MQT), conversion of tables to multidimensional clustering (MDC), redistribution of tables, and deletion of unused indexes and MQTs.
Load Wizard	Loads data into a table.
Configuration Advisor	Provides recommendations about configuration parameters changes to improve database performance.
Restore Data Wizard	Restore DB2 databases.
Configure Automatic Maintenance	Configure automatic database maintenance routines, such as backup, data defragmentation, and automatic update statistics.

For more information about DB2 Control Center, refer to Chapter 7 "Using the DB2 administration tools" in *DB2 9.1 Administration Guide: Implementation*, SC10-4221.

11.2 Instance management commands

DB2 provides several commands to create and manage instances. These commands can be categorized into two types:

► System commands, which are run directly from the operating system prompt, or in a shell script
► CLP commands, which are run in a DB2 tool such as the CLP

Next we discuss the scope of these commands and their equivalents in Oracle, if applicable, and give an example of the command usage.

11.2.1 Managing instances

In this section we discuss the commands used for instance management tasks, such as creating, dropping, starting, and stopping instances.

db2icrt

The `db2icrt` command is used to create a DB2 instance.To issue the `db2icrt` command, you must have root privileges on Linux and UNIX-based systems, or Local Administrators' authority on Windows operating systems.

The instance created is related to the DB2 copy installation path from where `db2icrt` is performed. Only one instance can be created under a specific user name on Linux and UNIX-based systems. If you need to create several DB2 instances in the same machine, you must have a user name for each instance. (This restriction does not apply to Windows operating systems.)

Oracle equivalent command

You do not have a specific command to create an Oracle instance. Rather, the instance is a combination of processes and memory allocated when a database is started. The instance parameters are stored in an operating system file.

Examples

Example 11-2 shows how to create an instance on a Linux operating system. This command creates a DB2 instance called db2inst1, and uses the user db2fenc1 to run fenced user-defined functions and fenced stored procedures. The users db2fenc1 and db2inst1 must exist in the operating system before the command is used.

Example 11-2   Creating a DB2 instance in a Linux operating system

```
db2icrt -u db2fenc1 db2inst1
DBI1070I Program db2icrt completed successfully.
```

Example 11-3 shows how to create an instance in a Windows operating system. In this example, the instance db2inst1 is created as a Windows service.

Example 11-3   Creating a DB2 instance in a Windows operating system

```
db2icrt db2inst1
DB20000I  The DB2ICRT command completed successfully.
```

db2idrop

The **db2idrop** command is used to drop a DB2 instance. You can only drop instances located in the same DB2 copy installation path where you perform the **db2idrop** command. To run **db2idrop**, you must have root privileges on Linux and UNIX-based systems, or Local Administrators' authority on Windows operating systems.

The **db2idrop** command does not remove any databases under the instance. It is therefore recommended that if the databases are no longer required they must be removed first.

Oracle equivalent command

You do not have a specific command to drop an Oracle instance. Because an Oracle instance is composed of a set of memory allocation and processes, when you shut down the database the instance is also removed.

Example

Example 11-4 shows how to drop the DB2 instance db2inst1.

Example 11-4   Dropping a DB2 instance

```
db2idrop db2inst1
DB20000I  The DB2IDROP command completed successfully.
```

db2start

The **db2start** command is used to start an instance in DB2. It starts the current database manager instance background processes. It can be run either from the operating system prompt or from the DB2 CLP. On Windows, it starts the Windows service associated with the current instance. You can also start the service directly from the Windows Control Panel, or from the DB2 Control Center.

The user starting the DB2 instance must have SYSADM, SYSCTRL, or SYSMAINT authority on the instance; be the instance owner on Linux and UNIX-based systems; or have privileges to start a service on the Windows operating system. The `db2start` command is equivalent to START DATABASE MANAGER command.

DB2 `db2start admin mode` command is used to start the instance in quiesce mode. Quiescing an instance is useful for maintenance activities. While the instance is quiesced, only users with SYSADM, SYSMAINT, or SYSCTRL authority can attach to the instance. You must manually unquiesce the instance, to allow other users to connect to the instance, by using the UNQUIESCE INSTANCE command.

Oracle equivalent command

DB2 `db2start` can be considered as being equivalent to the Oracle STARTUP command. The DB2 `db2start admin mode` command is equivalent to running the Oracle STARTUP RESTRICT command and then the ALTER SYSTEM QUIESCE DATABASE command.

Example

Example 11-5 shows how to start a DB2 instance on a Linux and UNIX-based operating system directly from the operating system prompt.

Example 11-5   Starting a DB2 instance on Linux and UNIX operating systems

```
db2start
02/22/2007 16:43:17     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
```

Example 11-6 shows how to start a DB2 instance in the admin mode on a Windows operating system using DB2 CLP.

Example 11-6   Starting a DB2 instance in Windows operating system

```
db2 => db2start admin mode
DB20000I  The DB2START command completed successfully.
```

db2stop

The `db2stop` command is used to stop an instance in DB2. It stops the current database manager instance. It can be run either from the operating system prompt, or from the DB2 CLP. On the Windows operating system, it stops the Windows service associated with the current instance. You can also stop the service directly from the Windows Control Panel, or from the DB2 Control Center.

To run the `db2stop` command, you must have SYSADM, SYSCTRL, or SYSMAINT authority on the instance. `db2stop` does not stop an instance if there are any applications connected to the database under the instance. You must manually disconnect all applications using the FORCE APPLICATION command, or issue a `db2stop force` command. The `db2stop` command is equivalent to the STOP DATABASE MANAGER command.

Oracle equivalent command

The DB2 `db2stop` command can be considered as being equivalent to the SHUTDOWN command. The DB2 `db2stop force` command is equivalent to SHUTDOWN IMMEDIATE command.

Example

Example 11-7 shows how to stop an instance on Linux and UNIX-based operating systems directly from the operating system prompt.

Example 11-7   Stopping an instance on Linux and UNIX-based operating systems

```
db2stop

02/22/2007 19:29:08     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.
```

Example 11-8 shows how to stop a DB2 instance with the FORCE option on Windows using DB2 CLP.

Example 11-8   Stopping an instance in Windows using force parameter

```
db2 => db2stop
SQL1025N  The database manager was not stopped because databases are still
active.

db2 => db2stop force
DB20000I  The DB2STOP command completed successfully.
```

attach

The `attach` command is used to connect to an instance. You can attach to any instance that is available, including remote instances.

Oracle equivalent command

The DB2 `attach` command can be considered as being equivalent to the CONNECT command in Oracle. However, the CONNECT command in Oracle is used to connect to both the instance and the database; by contrast, in DB2 the CONNECT command connects only to a database.

Example

Example 11-9 shows how to connect to an instance using the **attach** command.

Example 11-9   Attaching to an instance

```
db2 attach to db2inst1

   Instance Attachment Information

 Instance server       = DB2/LINUX 9.1.1
 Authorization ID      = DB2INST1
 Local instance alias  = DB2INST1
```

detach

The **detach** command is used to remove the logical DB2 instance attachment.

Oracle equivalent command

The DB2 **detach** command can be considered as being equivalent to the DISCONNECT command in Oracle.

Example

Example 11-10 shows how to detach from a current instance.

Example 11-10   Detaching from the current instance

```
db2 detach
DB20000I  The DETACH command completed successfully.
```

db2iauto

The **db2iauto** command is used to enable or disable the auto-start of an instance after each system restart on Linux and UNIX-based systems. It requires *root* authority on the operating system or SYSADM authority in DB2. On the Windows operating system, you can enable automatic startup of DB2 instances by configuring the corresponding Windows service to restart automatically.

Oracle equivalent command

On the Windows operating system, you can also configure the Oracle service to be automatically started after each system restart. On Linux and UNIX-based operating systems, you must change a flag in the /etc/oratab file and then configure the script dbstart to run automatically after each system restart.

Example

Example 11-11 shows how to configure an instance to auto-start.

Example 11-11   Configuring a DB2 instance to restart automatically

```
db2iauto -on db2inst1
```

Example 11-12 shows how to turn off the auto-start configuration of an instance.

Example 11-12   Disabling instance automatic restart

```
db2iauto -off db2inst1
```

11.2.2 Retrieving instance information

In this section we discuss the commands commonly used to retrieve information about DB2 instances and installed products on the server.

db2ilist

The **db2ilist** command is used to list all DB2 instances created from the same DB copy location that you are running the **db2ilist** command on.

Oracle equivalent command

There is no easy way to list all instances (or databases) in Oracle. On Linux and UNIX-based systems, you can check the /etc/oratab file to find out which databases are installed on the server, or run the **ps -ef | grep pmon** command to check for the PMON processes that are running. The drawback of these methods is that a database can exist on a server, but not be included in the /etc/oratab file. There will be no PMON process if the database is not active when the **ps** command is run. On a Windows operating system, you can check the Windows services starting with "OracleService".

Example

Example 11-13 shows how to list all DB2 existing instances from the same DB copy location.

Example 11-13   Listing all DB2 instances

```
db2ilist
DB2
```

db2level

The **db2level** command shows the current version and service level of the installed DB2 products for the current DB copy location. It returns a different set of information than **db2ls**, including, for example, whether the DB2 is running on a 32-bit or a 64-bit operating system.

Oracle equivalent command

The **select banner from v$version;** query produces an output similar to **db2level**.

Example

Example 11-14 shows a **db2level** command.

Example 11-14   Listing the current version of DB2 installed software

```
db2level

DB21085I  Instance "db2inst1" uses "64" bits and DB2 code release "SQL09011"
with level identifier "01020107".
Informational tokens are "DB2 v9.1.0.1", "s061104", "U809676", and Fix Pack
"1".
Product is installed at "/opt/IBM/db2/V9.1".
```

db2ls

The **db2ls** command is used to list all the products and features installed on your system. In Version 9, **db2ls** is the only supported method to query a DB2 product. In prior releases to DB2 9, you can use native operating system tools, such as pkgadd, rpm, or SMIT to query DB2 products. During the installation process, a symbolic link to **db2ls** is created in the /usr/local/bin directory.

The **db2ls** command is not supported on Windows platforms. For Windows platforms, you can use the Default DB2 Selection wizard to list all installed DB2 copies on a machine.

Oracle equivalent command

Oracle stores information about installed products in an inventory. The inventory location is specified using the /etc/oraInst.loc file on Linux and UNIX-based systems, or in the registry for the Windows operating system. To list the product releases you can use the Universal Installer tool, or execute **opatch lsinventory -detail** for each Oracle Home. To list all Oracle Homes, use the **opatch lsinventory -all** command.

However, for Linux and UNIX-based systems, it is possible to have several different inventories on one server, so can be difficult to discover all Oracle software installed in a system.

Examples

Example 11-15 shows how to list all DB2 software copies installed on a Linux or a UNIX-based system and its output. Note that the sample output is formatted and only columns with pertinent data are shown.

Example 11-15   List of all DB2 copy installation software

```
db2ls

Install Path          Level     Fix Pack  Install Date
-----------------------------------------------------------------------
/opt/IBM/db2/V9.1     9.1.0.1        1     Wed Jan 31 17:55:14 2007 CST
```

Example 11-16 shows how to list the installed products on a particular path. For this example, we use the value returned in the Install Path column in Example 11-15.

Example 11-16   List all installed products from a particular path

```
db2ls -q -p -b /opt/IBM/db2/V9.1

Install Path: /opt/IBM/db2/V9.1
Product Response File ID     Level Fix Pack  Product Description
-----------------------------------------------------------------------
ENTERPRISE_SERVER_EDITION  9.1.0.1        1  DB2 Enterprise Server
                                               Edition
```

Example 11-17 shows how to list all DB2 database features installed on a particular path.

Example 11-17   db2ls command used to list all features from a particular path

```
db2ls -q -a -b /opt/IBM/db2/V9.1

Install Path: /opt/IBM/db2/V9.1
-----------------------------------------------------------------------
Feature Response File ID       Level Fix Pack  Feature Description
-----------------------------------------------------------------------
DB2_PRODUCT_MESSAGES_EN        9.1.0.1      1   Product Messages English
BASE_CLIENT                    9.1.0.1      1   Base client support
JDK                            9.1.0.1      1   IBM Software Development Kit
                                                 (SDK) for Java(TM)
DB2_JAVA_HELP_EN               9.1.0.1      1   Java Help (HTML) - English
REPL_QSERVER                   9.1.0.1      1   Replication with MQ Server
BASE_DB2_SERVER                9.1.0.1      1   Run-time Environment
SQL_PROCEDURES                 9.1.0.1      1   SQL procedures
ICU_SUP                        9.1.0.1      1   ICU Utilities
REPL_SERVER                    9.1.0.1      1   SQL Replication Support
```

```
BASE_DB2_ENGINE              9.1.0.1      1    Base server support
CONNECT_SUPPORT              9.1.0.1      1    Connect support
JAVA_SUPPORT                 9.1.0.1      1    Java support
JAVA_COMMON_FILES            9.1.0.1      1    Java Common files
APPLICATION_DEVELOPMENT_TOOLS 9.1.0.1     1    Base application development
                                                 tools
ADMINISTRATION_SERVER        9.1.0.1      1    Administration Server
COMMUNICATION_SUPPORT_TCPIP  9.1.0.1      1    Communication support - TCP/IP
DATABASE_PARTITIONING_SUPPORT 9.1.0.1     1    Parallel Extension
REPL_CLIENT                  9.1.0.1      1    Replication tools
DB2_DATA_SOURCE_SUPPORT      9.1.0.1      1    DB2 data source support
INSTANCE_SETUP_SUPPORT       9.1.0.1      1    DB2 Instance Setup wizard
FIRST_STEPS                  9.1.0.1      1    First Steps
DB2_WEB_TOOLS                9.1.0.1      1    DB2 Web Tools
ESE_PRODUCT_SIGNATURE        9.1.0.1      1    Product Signature for
                                                 DB2 Enterprise Server Edition
XML_EXTENDER_SAMPLES         9.1.0.1      1    XML Extender samples
DB2_SAMPLE_APPLICATIONS      9.1.0.1      1    ADT sample programs
DB2_SAMPLE_DATABASE          9.1.0.1      1    Sample database source
```

11.2.3 Managing instance configuration parameters

In this section we discuss the commands used to retrieve and update the database manager configuration parameters.

Get database manager configuration

The DB2 GET DATABASE MANAGER CONFIGURATION command is used to retrieve the values of the configuration parameters for an instance, the database manager (DBM). The parameter SHOW DETAIL shows the current value of instance parameters, as well the value of parameters the next time you start the instance. Using the show detail parameter, you can easily verify which non-dynamic parameters will be changed in the next instance startup. Attaching to the instance is not required unless the SHOW DETAIL parameter is used.

To simplify entering the command, you can use an abbreviated form of the command: GET DBM CFG. You can also retrieve the DBM parameter information by using the DBMCFG administrative view or the DBM_GET_CFG table function.

Oracle equivalent command

To retrieve information about an instance, you can query the V$PARAMETER view or run the SHOW PARAMETER command in SQL*Plus.

Example

Example 11-18 shows how to retrieve information about the current values of the DBM parameters. Note that the sample output is formatted to fit in the page, and not all rows are displayed.

Example 11-18   Retrieving DBM parameters current values

```
db2 get dbm cfg

Database Manager Configuration

     Node type = Enterprise Server Edition with local and remote clients

 Database manager configuration release level              = 0x0b00

 CPU speed (millisec/instruction)           (CPUSPEED) = 6.612818e-07
 Communications bandwidth (MB/sec)     (COMM_BANDWIDTH) = 1.000000e+02

 Max number of concurrently active databases     (NUMDB) = 8
 Federated Database System Support           (FEDERATED) = YES

...

db2start/db2stop timeout (min)         (START_STOP_TIME) = 10
```

Example 11-19 shows how to retrieve a list of current values for the DBM, as well as values that will be in effect after the instance restart. Note that the sample output is formatted to fit in the page and not all rows are displayed.

Example 11-19   Retrieving DBM parameters current and delayed values

```
db2 get dbm cfg show detail
SQL1427N  An instance attachment does not exist.
db2 attach to db2inst1

   Instance Attachment Information

 Instance server       = DB2/LINUX 9.1.1
 Authorization ID      = DB2INST1
 Local instance alias  = DB2INST1

db2 get dbm cfg show detail

         Database Manager Configuration

     Node type = Enterprise Server Edition with local and remote clients

Description             Parameter    Current Value       Delayed Value
```

```
                -------------------------------------------------------------------------
                Database manager configuration
                         release level                = 0x0b00

                CPU speed (millisec/instruction)
                                      (CPUSPEED) = 6.612818e-07        6.612818e-07
                Communications bandwidth (MB/sec)
                              (COMM_BANDWIDTH) = 1.000000e+02       1.000000e+02

                Max number of concurrently active
                         databases       (NUMDB) = 8                            8
                Federated Database System Support
                                     (FEDERATED) = YES                         YES


                ...


                db2start/db2stop timeout (min)
                             (START_STOP_TIME) = 10                           10
                -------------------------------------------------------------------------
```

Example 11-20 shows how to use the DBM CFG administrative view to retrieve all database manager configuration parameters.

Example 11-20   Retrieving DBM parameters using DBMCFG administrative view

```
db2 => select name, substr(value,1,20) as running, substr(deferred_value,1,20)
as DEFERRED from sysibmadm.dbmcfg

NAME                              RUNNING            DEFERRED
-------------------------------- ------------------ -----------------
agent_stack_sz                   0                  0
agentpri                         -1                 -1
aslheapsz                        15                 15
audit_buf_sz                     0                  0
authentication                   SERVER             SERVER
catalog_noauth                   NO                 NO
clnt_krb_plugin
clnt_pw_plugin
comm_bandwidth                   1.000000e+02       1.000000e+02
conn_elapse                      10                 10
cpuspeed                         6.770267e-07       6.770267e-07
dft_account_str
dft_mon_bufpool                  OFF                OFF
dft_mon_lock                     OFF                OFF
dft_mon_sort                     OFF                OFF
dft_mon_stmt                     OFF                OFF
dft_mon_table                    OFF                OFF
dft_mon_timestamp                ON                 ON
dft_mon_uow                      OFF                OFF
```

```
dftdbpath                           /home/db2inst1      /home/db2inst1
diaglevel                           3                   3
diagpath
...
util_impact_lim                     10                  10
nodetype                            DPF_SERVER          DPF_SERVER
release                             0x0b00              0x0b00
federated_async                     0                   0
fcm_num_channels                    2048                2048
```

Update database manager configuration

The UPDATE DATABASE MANAGER CONFIGURATION command is used to update any parameter in the database manager configuration file. Only users with SYSADM authority can update the DBM parameters. An instance attachment is required only when you update parameters in a remote instance, or when you update parameters online on the local instance.

You can specify if your change is dynamically applied to DBM by using the IMMEDIATE clause. By using the DEFERRED clause, only the DBM configuration file is updated, and the change will take effect on the next DB2 instance restart. You can also let DB2 automatically adjust some DBM parameters by using the AUTOMATIC clause. You can disable the DB2 automatic tuning for the configuration parameter by using the MANUAL clause.

To simplify entering the command, you can use an abbreviated form of the command: UPDATE DBM CFG.

Oracle equivalent command

In Oracle, you use the ALTER SYSTEM command to update instance parameters. Using the SCOPE clause, you can choose whether the update is dynamically applied to the instance, or only written in the initialization file.

Example

Example 11-21 shows how to dynamically update a parameter in the DBM configuration file. As shown in the example, we update the dynamic parameter DIAGLEVEL, and verify whether the change has become effective.

Example 11-21   Change a dynamic parameter

```
db2 => get dbm cfg
...
Diagnostic error capture level              (DIAGLEVEL) = 3
...

db2 => update dbm cfg using DIAGLEVEL 1
```

```
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.

db2 => get dbm cfg
...
Diagnostic error capture level              (DIAGLEVEL) = 3
...
```

In Example 11-22, we update a non-dynamic parameter in the DBM file. Note
that the change is *not* dynamically applied to the instance.

Example 11-22   Updating DBM for non-dynamic parameter

```
db2 => get dbm cfg show detail
...
Database monitor heap size (4KB)        (MON_HEAP_SZ) = 90 90
...

db2 => update dbm cfg using MON_HEAP_SZ 1000
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
SQL1362W  One or more of the parameters submitted for immediate modification
were not changed dynamically. Client changes will not be effective until the
next time the application is started or the TERMINATE command has been issued.
Server changes will not be effective until the next DB2START command.

db2 => get dbm cfg show detail
...
Database monitor heap size (4KB)        (MON_HEAP_SZ) = 90 1000
...
```

In order to have the update become effective, an instance restart is required.
Example 11-23 shows the restart of the instance, and that the updated
parameter has become effective after the restart.

*Example 11-23   Restarting the instance to make non-dynamic parameters become
effective*

```
db2 => db2stop
DB20000I  The DB2STOP command completed successfully.

db2 => db2start
DB20000I  The DB2START command completed successfully.

db2 => attach to db2inst1

   Instance Attachment Information
```

```
 Instance server        = DB2/LINUX 9.1.1
 Authorization ID       = DB2INST1
 Local instance alias   = DB2INST1

db2 => get dbm cfg show detail
...
Database monitor heap size (4KB)          (MON_HEAP_SZ) = 1000 1000
...
```

11.2.4 Setting registry variables

DB2 registry variables are used to control the database environment. Four different registry profiles are available, and each controls the database environment at a different level. The registry profiles available are:

► DB2 instance level profile registry

   This registry stores settings for a particular instance. Values defined in this level override their settings at the global level. When setting the variable, specify the instance level variables using the parameter `-i instance_name`.

► DB2 global level profile registry

   This registry is visible to all instances created under the same DB copy. If an environment variable is not set for a particular instance, this registry is used. When setting the variable, specify the instance level variables using the parameter `-g`.

► DB2 instance node level profile registry

   This registry contains variable settings specific to a database partition in a partitioned database. Values defined in this level override their settings at the instance and global levels. When setting the variable, specify the instance node level variables using the parameter `-i instance_name nodenum`.

► DB2 instance profile registry

   This registry contains a list of all instance names associated with the current copy.

In addition to the registry variables, DB2 also uses specific operating system environment variables to control the database environment. You can use the EXPORT command on Linux and UNIX-based systems, or the **set** command in Windows operating systems, to define an environment variable.

DB2 configures the operating environment by checking the registry values and the environment variables in the following order:

1. Environment variables
2. Instance node level profile

3. Instance level profile
4. Global level profile

db2set

The **db2set** command is used to display, set, or remove DB2 registry variables.

Oracle equivalent command

Oracle uses operating system variables (such as ORACLE_SID, NLS_LANG, and so on) to specify the characteristics of the environment variables. Oracle uses the information stored in the LISTENER.ORA and TNSNAMES.ORA to configure its network.

Examples

Example 11-24 shows how to list all defined registry variables in the profile registry.

Example 11-24   List all registry variables

```
db2set -all

[e] DB2PATH=C:\Program Files\IBM\SQLLIB_2
[i] DB2ACCOUNTNAME=TONGA\db2inst2
[i] DB2INSTOWNER=TONGA
[i] DB2PORTRANGE=60004:60007
[i] DB2INSTPROF=C:\PROGRA~1\IBM\SQLLIB_2
[i] DB2COMM=TCPIP
[g] DB2_EXTSECURITY=YES
[g] DB2SYSTEM=TONGA
[g] DB2PATH=C:\Program Files\IBM\SQLLIB_2
[g] DB2INSTDEF=DB2_01
```

In the **db2set -all** command output, the first letter in the brackets [ ] identifies the scope of the registry variable:

e   Denotes the environment variable
n   Denotes the node level registry
i   Denotes the instance level registry
g   Denotes the global level registry

Example 11-25 shows how to specify a variable at the global level.

Example 11-25   Setting a variable at the global level

```
db2set DB2CODEPAGE=1208 -g
```

> **Note:** If you do not specify a parameter to define the scope when setting a registry variable, the default is instance (-i).

11.3 Database management

DB2 provides several commands to create and manage databases. In this section, we discuss the DB2 commands commonly used by DBAs for their database management tasks.

We discuss the scope of the commands, their equivalent command in Oracle if applicable, and give an example of the command usage.

11.3.1 Managing databases

In this section, we discuss how to create, drop, activate and deactivate a database. We also discuss how to connect and terminate a connection to a database.

Create database

The CREATE DATABASE command is used to create DB2 databases within an instance. You can specify several parameters to control the database characteristics, such as the code set, where the data files are stored, the database page size, and the table space management clauses. You must have SYSADM or SYSCTRL authority to create a database.

Oracle equivalent command

The Oracle CREATE DATABASE command is the equivalent of DB2 CREATE DATABASE command. However, the parameters passed to the command for each RDBMS are completely different.

Example

Example 11-26 shows how to create a database specifying the code set, territory, and automatic storage management.

Example 11-26   Creating a database

```
db2 => create database db2db9 automatic storage yes using codeset utf-8
territory US
DB20000I  The CREATE DATABASE command completed successfully.
```

Drop database

The DROP DATABASE command is used to drop a database. It deletes the database contents and all the log files for the database, uncatalogs the database, and deletes the database subdirectory. Before you drop a database, you must disconnect all users from the database. You must have SYSADM or SYSCTRL authority in order to use this command.

If you only want to remove the database from the system catalog, but not remove the database physically, then use the UNCATALOG DATABASE command instead of DROP DATABASE.

Oracle equivalent command

The Oracle DROP DATABASE command is equivalent to the DB2 DROP DATABASE command.

Example

Example 11-27 shows how to drop a database.

Example 11-27   Dropping a database

```
db2 => drop database db2db9
DB20000I  The DROP DATABASE command completed successfully.
```

Activate database

The ACTIVATE DATABASE command is used to start all necessary database services. If a database is not started, the first application that connects to the database automatically starts the database, but it waits for the database initialization.

You must have SYSADM, SYSCTRL or SYSMAINT authority to activate a database.

Oracle equivalent command

Oracle databases are started through the STARTUP command.

Example

Example 11-28 shows how to activate a database.

Example 11-28   Activating a database

```
db2 activate database db2db9
DB20000I  The ACTIVATE DATABASE command completed successfully.
```

Deactivate database

The DEACTIVATE DATABASE command stops a specific DB2 database. The stop instance command **db2stop** stops all databases under the instance.

You must have SYSADM, SYSCTRL or SYSMAINT authority to activate a database.

Oracle equivalent command

The DB2 DEACTIVATE DATABASE command can be considered to be an equivalent of the SHUTDOWN command in Oracle. However, the Oracle shutdown command stops the instance *and* the database. In contrast, the DB2 DEACTIVATE DATABASE command only stops the database. You must use **db2stop** to stop a DB2 instance.

Examples

Example 11-29 shows how to deactivate a database.

Example 11-29   Deactivating a database

```
db2 deactivate database db2db9
DB20000I  The DEACTIVATE DATABASE command completed successfully
```

Connect to a database

The CONNECT command is used to connect to a database.

Oracle equivalent command

The equivalent Oracle command is CONNECT.

Examples

Example 11-30 shows how to display the current connection information.

Example 11-30   Displaying current connection information

```
db2 => connect

   Database Connection Information

 Database server        = DB2/AIX64 9.1.1
 SQL authorization ID   = DB2INST1
 Local database alias   = DB2DB9
```

Example 11-31 shows how to connect to a database using a user name and a password.

Example 11-31   Connecting to a database specifying user name and password

```
db2 => connect to remotedb user db2inst1 using user_pwd

   Database Connection Information

 Database server        = DB2/AIX64 9.1.1
 SQL authorization ID   = DB2INST1
 Local database alias   = REMOTEDB
```

QUIT, CONNECT RESET, and TERMINATE

Note the following differences between these commands:

► The QUIT command exits the CLP interactive input mode and returns to the operating system command prompt. It does *not* break the database connection or the back-end process.
► The CONNECT RESET command breaks the connection, but does *not* terminate the back-end process.
► The TERMINATE command explicitly terminates the database connection and the back-end process.

Oracle equivalent command

The DB2 TERMINATE command can be considered as being equivalent to the Oracle DISCONNECT command.

Examples

Example 11-32 shows how to terminate the current connection.

Example 11-32   Terminating a back-end process

```
db2 => terminate
DB20000I  The TERMINATE command completed
successfully.
```

11.3.2 Managing node and database directories

DB2 maintains the information on remote nodes locally in the node directory. DB2 also maintains a list of all the configured databases locally in the database directory. In this section, we discuss the commands used to manage these two directories.

Catalog TCPIP node and database

The CATALOG TCPIP NODE command is used to catalog a remote node. When cataloging a remote node, you can specify the host name or IP address, the port, the remote instance and an alias for the remote node.

The CATALOG DATABASE command is used to catalog an existing database in the local database directory. The database being cataloged can be local or remote. To catalog a remote database, you must first catalog the node. You can also use the CATALOG DATABASE command to maintain multiple aliases for the same database.

You must have the SYSADM or SYSCTRL authority to catalog a node or a database. To refresh the catalog operations, you must run the TERMINATE command.

Oracle equivalent command

In Oracle, you can add an alias to the database connect descriptors to the tnsnames.ora file, and use the alias to connect to the databases.

Example

Example 11-33 shows how to catalog a remote node. The remote_hostname node is locally identified as rmt_node.

Example 11-33   Cataloging a remote node

```
db2 => catalog tcpip node rmt_node remote remote_hostname server 60000
remote_instance db2inst1
DB20000I  The CATALOG TCPIP NODE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.

db2 => terminate
DB20000I  The TERMINATE command completed successfully.
```

Example 11-34 shows how to catalog a DB2 database in a remote node.

Example 11-34   Cataloging a database

```
db2 => catalog database DB2DB9 as remotedb at node rmt_node
DB20000I  The CATALOG DATABASE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.

db2 => terminate
DB20000I  The TERMINATE command completed successfully.
```

Uncatalog TCPIP node and database

The UNCATALOG TCPIP NODE command and the UNCATALOG DATABASE command are used to remove the nodes and cataloged databases, respectively, from the local directory. Both local and remote databases can be uncataloged. If you uncatalog a node used by a remote database, it is not possible to connect to the remote database.

You must have SYSADM or SYSCTRL authority to uncatalog a node or a database.

Oracle equivalent command

In Oracle, you can remove the alias from the tnsnames.ora file.

Example

Example 11-35 shows how to uncatalog a remote node.

Example 11-35   Uncataloging a remote node.

```
db2 => uncatalog node rmt_node
DB20000I  The UNCATALOG NODE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache
isrefreshed.

db2 => terminate
DB20000I  The TERMINATE command completed successfully.
```

Example 11-36 shows how to uncatalog a database.

Example 11-36   Uncataloging a database

```
db2 => uncatalog database db2remote
DB20000I  The UNCATALOG DATABASE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.

db2 => terminate
DB20000I  The TERMINATE command completed successfully.
```

List node directory and database directory

The LIST NODE DIRECTORY command and the LIST DATABASE DIRECTORY command are used to list the node directory contents and the database directory contents, respectively. For the LIST NODE DIRECTORY command, the SHOW DETAIL clause shows more detailed information.

Oracle equivalent command

There is no equivalent command in Oracle to list the database directory. You can manually list the tnsnames.ora file contents to verify all databases configured for access.

Example

Example 11-37 shows how to list the node directory contents.

Example 11-37   Showing node directory contents

```
db2 => list node directory

 Node Directory

 Number of entries in the directory = 1

Node 1 entry:

 Node name                    = RMT_NODE
 Comment                      =
 Directory entry type         = LOCAL
 Protocol                     = TCPIP
 Hostname                     = remote_hostname
 Service name                 = 60000
```

Example 11-38 shows how to list the database directory contents.

Example 11-38   Showing database directory contents

```
db2 => list database directory

 System Database Directory

 Number of entries in the directory = 2

Database 1 entry:

 Database alias                       = DB2_EMP
 Database name                        = DB2_EMP
 Local database directory             = C:
 Database release level               = b.00
 Comment                              = A sample database
 Directory entry type                 = Indirect
 Catalog database partition number    = 0
 Alternate server hostname            =
 Alternate server port number         =

Database 2 entry:
```

```
Database alias                          = REMOTEDB
Database name                           = DB2DB9
Node name                               = RMT_NODE
Database release level                  = b.00
Comment                                 =
Directory entry type                    = Remote
Catalog database partition number       = -1
Alternate server hostname               =
Alternate server port number            =
```

11.3.3 Managing database configuration parameters

DB2 stores database configuration parameters in the database configuration file. In this section, we discuss the commands used to retrieve and update these parameters.

GET database configuration parameter

The DB2 GET DATABASE CONFIGURATION PARAMETER command is used to return the values of the configuration parameters of a database, or the database configuration file (DB). The parameter SHOW DETAIL shows the current value of database parameters, as well the value of parameters the next time you activate the database. By using the SHOW DETAIL parameter, you can easily verify which non-dynamic parameters will be changed during the next database activation. No database connection is required to run the command unless the SHOW DETAIL parameter is used.

To simplify entering the command, you can use an abbreviated form of the GET DB CFG command. You can also retrieve the DB parameter information by using the DBCFG administrative view or the DB_GET_CFG table function.

Oracle equivalent command

The DB2 GET DATABASE CONFIGURATION PARAMETER retrieves a series of information for you in one command. In Oracle, you need to query different views to retrieve similar data.

For example, you can select information from V$DATABASE to retrieve information about the database; from V$INSTANCE for information about the instance; from V$PARAMETER for memory parameter information; and from V$LOGFILE for redo log file information.

Example

Example 11-39 shows how to list all database configuration parameters.

Example 11-39   Listing all database configuration parameters

```
db2 => get db cfg

       Database Configuration for Database

Database configuration release level                      = 0x0b00
Database release level                                    = 0x0b00

Database territory                                        = C
Database code page                                        = 1208
Database code set                                         = UTF-8
Database country/region code                              = 1
Database collating sequence                               = UCA400_NO
Alternate collating sequence          (ALT_COLLATE) =
Database page size                                        = 4096

Dynamic SQL Query management          (DYN_QUERY_MGMT) = DISABLE

Discovery support for this database      (DISCOVER_DB) = ENABLE

Restrict access                                           = NO
Default query optimization class       (DFT_QUERYOPT) = 5
Degree of parallelism                     (DFT_DEGREE) = 1
Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO
Default refresh age                   (DFT_REFRESH_AGE) = 0
Default maintained table types for opt (DFT_MTTB_TYPES) = SYSTEM
Number of frequent values retained    (NUM_FREQVALUES) = 10
Number of quantiles retained           (NUM_QUANTILES) = 20

Backup pending                                           = NO

Database is consistent                                   = YES
Rollforward pending                                      = NO
Restore pending                                          = NO

Multi-page file allocation enabled                       = YES

...

Automatic maintenance                     (AUTO_MAINT) = ON
   Automatic database backup          (AUTO_DB_BACKUP) = OFF
   Automatic table maintenance         (AUTO_TBL_MAINT) = ON
     Automatic runstats                 (AUTO_RUNSTATS) = ON
     Automatic statistics profiling   (AUTO_STATS_PROF) = OFF
       Automatic profile updates        (AUTO_PROF_UPD) = OFF
     Automatic reorganization             (AUTO_REORG) = OFF
```

Example 11-40 shows how to retrieve detailed information about the database configuration parameters.

Example 11-40   Listing detailed information about database configuration parameter

```
db2 => get db cfg show detail

        Database Configuration for Database

 Description                              Parameter     Current   Delayed
                                                        Value     Value
 ----------------------------------------------------------------------

 Database configuration release level                  = 0x0b00
 Database release level                                = 0x0b00

 Database territory                                    = US
 Database code page                                    = 819
 Database code set                                     = ISO8859-1
 Database country/region code                          = 1
 Database collating sequence                           = UNIQUE       UNIQUE
 Alternate collating sequence        (ALT_COLLATE) =
 Database page size                                    = 4096         4096

 Dynamic SQL Query management        (DYN_QUERY_MGMT) = DISABLE    DISABLE

 Discovery support for this database  (DISCOVER_DB) = ENABLE      ENABLE

 Restrict access                                       = NO
 Default query optimization class     (DFT_QUERYOPT) = 5                 5
 Degree of parallelism                  (DFT_DEGREE) = 1                 1
 Continue upon arithmetic exceptions
                            (DFT_SQLMATHWARN) = NO                 NO
 Default refresh age              (DFT_REFRESH_AGE) = 0                  0
 Default maintained table types for opt
                            (DFT_MTTB_TYPES) = SYSTEM         SYSTEM
 Number of frequent values retained(NUM_FREQVALUES) = 10                10
 Number of quantiles retained      (NUM_QUANTILES) = 20                20

 Backup pending                                        = NO

 Database is consistent                                = YES
 Rollforward pending                                   = NO
 Restore pending                                       = NO

Multi-page file allocation enabled                = YES
...
Automatic maintenance                 (AUTO_MAINT) = ON                 ON
   Automatic database backup      (AUTO_DB_BACKUP) = OFF                OFF
```

```
Automatic table maintenance     (AUTO_TBL_MAINT) = ON              ON
    Automatic runstats           (AUTO_RUNSTATS) = ON              ON
Automatic statistics profiling(AUTO_STATS_PROF) = OFF             OFF
    Automatic profile updates    (AUTO_PROF_UPD) = OFF             OFF
    Automatic reorganization       (AUTO_REORG) = OFF             OFF
```

Example 11-41 shows how to retrieve the database configuration parameters by using the DBCFG administrative view.

Example 11-41   Retrieving database configuration parameters using DBCFG view

```
db2 => select name, substr(value,1,20) as running, substr(deferred_value,1,20)
as DEFERRED from sysibmadm.dbcfg

NAME                             RUNNING             DEFERRED
-------------------------------- ------------------- ----------------
app_ctl_heap_sz                  128                 128
appgroup_mem_sz                  30000               30000
applheapsz                       256                 256
archretrydelay                   20                  20
auto_maint                       ON                  ON
auto_db_backup                   OFF                 OFF
auto_tbl_maint                   ON                  ON
auto_runstats                    ON                  ON
auto_stats_prof                  OFF                 OFF
auto_prof_upd                    OFF                 OFF
auto_reorg                       OFF                 OFF
autorestart                      ON                  ON
avg_appls                        1                   1
blk_log_dsk_ful                  NO                  NO
catalogcache_sz                  -1                  -1
chngpgs_thresh                   60                  60
database_memory                  18128               18128
dbheap                           1200                1200
dft_degree                       1                   1
...
db_mem_thresh                    10                  10
db_collname                      UCA400_NO           UCA400_NO
restrict_access                  NO                  NO
```

Update database configuration parameter

The UPDATE DATABASE CONFIGURATION command is used to update any parameter in the database configuration (DB CFG) file. Only users with SYSADM, SYSCTRL and SYSMAINT authority can update the parameters in the database. It is recommended to have an active database connection while updating the DB configuration file for an active database. It is also mandatory to update a parameter online.

You can specify whether your change is dynamically applied to DB CFG by using the IMMEDIATE clause. The DEFERRED clause is used to update only the DB configuration file. IMMEDIATE is the default action for DB updates, but requires an active database connection. If you update only the DB configuration file, then it is necessary to deactivate and reactivate the DB2 database for the update to become effective.

You can also let DB2 automatically adjust some DB CFG parameters using the clause AUTOMATIC. You can disable the DB2 automatic tuning for the configuration parameter by using the MANUAL clause.

To simplify entering the command, you can use an abbreviated form of the UPDATE DB CFG command.

Oracle equivalent command

The DB2 UPDATE DB CFG command allows you to update all the database parameters. By contrast, in Oracle you need to use different commands. For example, you can use the ALTER SYSTEM command to alter the archive log destinations, or the ALTER DATABASE command to change database recovery options.

Another important aspect to consider is that you can specify memory parameters for a DB2 database. By contrast, in Oracle these parameters are only specified at the instance level. Therefore, the DB2 UPDATE DB CFG updating memory parameters can be considered as the ALTER SYSTEM commands with Oracle instance memory parameters specified.

Example

Example 11-42 shows how to update a database configuration parameter.

Example 11-42   Update a database configuration parameter

```
db2 => update db cfg using DBHEAP 10000
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully
```

11.3.4 Managing table spaces

In this section we discuss the commands to create, alter, drop, and list table spaces in a DB2 database. The commands for these operations are similar in DB and Oracle.

Create table space

The CREATE TABLESPACE statement is used to create a table space within the database. You can specify all table space parameters, such as containers, type

of table space, page size, and so on. If your database is using automatic storage management, DB2 automatically chooses all the values for you.

You must have SYSADM or SYSCTRL authority to create a table space. A more detailed discussion about table spaces can be found in 2.4.8, "Table space design" on page 56.

Oracle equivalent command
The equivalent Oracle command is CREATE TABLESPACE.

Example
Example 11-43 shows how to create a table space in DB2.

Example 11-43   Creating a regular table space

```
db2 => create tablespace app_tbs managed by database using ( file
'/db2/tbs/app_tbs_1.dbf' 100m )
DB20000I  The SQL command completed successfully.
```

Alter table space
The ALTER TABLESPACE statement is used to modify a table space characteristic. Using the ALTER TABLESPACE statement you can:

► Add, resize, or drop a container from a DMS table space
► Modify the PREFETCHSIZE, BUFFERPOOL, OVERHEAD or TRANSFERRATE settings for a table space
► Modify the file system caching policy for a table space

You must have SYSCTRL or SYSADM authority to alter a table space.

Oracle equivalent command
The equivalent Oracle command is ALTER TABLESPACE.

Example
Example 11-44 shows how to add, drop, and resize a container in an existing DMS table space.

Example 11-44   Adding, dropping and resizing containers in a DMS table space

```
--Adding a container
db2 => alter tablespace APP_TBS add (file '/db2/tbs/app_tbs_2.dbf' 300)
DB20000I  The SQL command completed successfully.
-- Resizing the container
db2 => alter tablespace APP_TBS resize (file '/db2/tbs/app_tbs_2.dbf' 1000)
DB20000I  The SQL command completed successfully.
```

```
--Dropping the container
db2 => alter tablespace APP_TBS drop (file '/db2/tbs/app_tbs_2.dbf')
DB20000I  The SQL command completed successfully.
```

Drop table space

The DROP TABLESPACE statement is used to drop a table space.

Oracle equivalent command

The equivalent Oracle command is DROP TABLESPACE.

Example

Example 11-45 shows how to drop a table space.

Example 11-45   Dropping a table space

```
db2 => drop tablespace APP_TBS
DB20000I  The SQL command completed successfully.
```

List table spaces

The LIST TABLESPACE command is used to list all existing table spaces of the
current database. You can use the clause SHOW DETAILS to retrieve detailed
information about the table spaces. To retrieve similar information, you can also
use the TBSP_UTILIZATION administrative view.

The State column in the output of the LIST TABLESPACE command indicates
the state of the table space in hexadecimal notation. You can use the **db2tbst**
command to obtain the state. 0x0 is the Normal state.

Oracle equivalent command

You can use V$TABLESPACE or DBA_TABLESPACES views to retrieve
information about table spaces.

Example

Example 11-46 shows how to retrieve detailed information about the table
spaces in the current database. The sample output only shows the information
for the first and the last table spaces.

Example 11-46   List all table spaces information using LIST TABLESPACE command

```
db2 => list tablespaces show detail

          Tablespaces for Current Database

 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
```

```
 Type                          = Database managed space
 Contents                      = All permanent data. Regular table
space.
 State                         = 0x0000
   Detailed explanation:
     Normal
 Total pages                   = 16384
 Useable pages                 = 16380
 Used pages                    = 11040
 Free pages                    = 5340
 High water mark (pages)       = 11040
 Page size (bytes)             = 4096
 Extent size (pages)           = 4
 Prefetch size (pages)         = 4
 Number of containers          = 1

...

Tablespace ID                  = 8
 Name                          = APP_TBS
 Type                          = Database managed space
 Contents                      = All permanent data. Large table space.
 State                         = 0x0000
   Detailed explanation:
     Normal
 Total pages                   = 25600
 Useable pages                 = 25568
 Used pages                    = 96
 Free pages                    = 25472
 High water mark (pages)       = 96
 Page size (bytes)             = 4096
 Extent size (pages)           = 32
 Prefetch size (pages)         = 32
 Number of containers          = 1
```

Example 11-47 shows how to retrieve table space information using the
TBSP_UTILIZATION administrative view.

Example 11-47   Listing table space information using TBSP_UTILIZATION administrative view

```
db2 => SELECT TBSP_ID as ID, SUBSTR(TBSP_NAME,1,20) as TBSP_NAME, TBSP_TYPE,
TBSP_CONTENT_TYPE as CONTENT_TYPE, TBSP_STATE FROM SYSIBMADM.TBSP_UTILIZATION

ID TBSP_NAME            TBSP_TYPE CONTENT_TYPE TBSP_STATE
-- -------------------- --------- ------------ ----------
 0 SYSCATSPACE          DMS       ANY          NORMAL
 1 TEMPSPACE1           SMS       SYSTEMP      NORMAL
 2 USERSPACE1           DMS       LARGE        NORMAL
```

```
3 IBMDB2SAMPLEREL     DMS     LARGE       NORMAL
4 IBMDB2SAMPLEXML     DMS     LARGE       NORMAL
5 SYSTOOLSPACE        DMS     LARGE       NORMAL
6 SYSTOOLSTMPSPACE    SMS     USRTEMP     NORMAL
7 DEMOTMP             SMS     USRTEMP     NORMAL
8 APP_TBS             DMS     LARGE       NORMAL
```

List table space container

The LIST TABLESPACE CONTAINER command is used to list all the containers for a specific table space. The required parameter is the table space number of the table space that you want to list the containers for. You can gather this information from the output of the LIST TABLESPACES command. The SHOW DETAIL clause provides more detailed information about the container.

You can also use the CONTAINER_UTILIZATION or SNAPCONTAINER administrative views to retrieve similar information.

Oracle equivalent command

To list the containers (or Oracle datafiles) for a table space, you can use the V$DATAFILE or DBA_DATA_FILES views.

Example

Example 11-48 shows how to list the containers for a specific table space.

Example 11-48   Listing existing containers for a table space

```
db2 => list tablespace containers for 8

          Tablespace Containers for Tablespace 8

Container ID                    = 0
Name                            = /db2/tbs/app_tbs_1.dbf
Type                            = File
```

Example 11-49 shows how to use the CONTAINER_UTILIZATION administrative view to retrieve similar information.

Example 11-49   Listing containers using the CONTAINER_UTILIZATION view

```
db2 => SELECT SUBSTR(TBSP_NAME,1,20) AS TBSP_NAME, INT(TBSP_ID) AS TID,
SUBSTR(CONTAINER_NAME,1,25) AS CONTAINER_NAME, INT(CONTAINER_ID) AS CID,
CONTAINER_TYPE, INT(TOTAL_PAGES) AS TOT_PAGES FROM
SYSIBMADM.CONTAINER_UTILIZATION WHERE TBSP_ID = 8

TBSP_NAME    TID  CONTAINER_NAME          CID  CONTAINER_TYPE  TOT_PAGES
-----------  ---- ----------------------- ---  --------------- ---------
```

```
APP_TBS       8  /db2/tbs/app_tbs_1.dbf   0  FILE_EXTENT_TAG     25600
```

11.3.5 Managing buffer pools

A *buffer pool* is memory used to cache table and index data pages as they are being read from the disk, or being modified. The buffer pool improves database system performance by allowing data to be accessed from the memory instead of from the disk. You can create more than one buffer pool for a database, each one with a different memory size or page size. You can assign a buffer pool to a table space or a temporary table space with the same page.

In this section we discuss how to create, alter, and drop buffer pools.

Create buffer pool

The CREATE BUFFERPOOL statement is used to create a buffer pool. You can specify the size of the buffer pool or use the default value, which is the value of the database configuration parameter PAGESIZE. You can have the memory allocated at the time the buffer pool is created by using the keyword IMMEDIATE. If the keyword DEFERRED is used, only the buffer pool definition is created and memory allocation takes place at the next database restart. IMMEDIATE is the default behavior.

You must have SYSADM or SYSCTRL authority to create a buffer pool.

Oracle equivalent command

A DB2 buffer pool is, in concept, equivalent to the Oracle database buffer cache. In Oracle, you can create one database buffer cache for every different block size (equivalent to DB2 page size). All database buffer caches are instance parameters stored in the instance parameter file (init.ora or spfile).

Example

Example 11-50 shows how to create a buffer pool for a DB2 database. In this example, the buffer pool will be created using the database default page size.

Example 11-50   Creating a buffer pool

```
db2 => create bufferpool BP_TEST size 50000
DB20000I  The SQL command completed successfully.
```

Alter buffer pool

The ALTER BUFFERPOOL statement is used to change buffer pool characteristics, such as the size or for automatic resizing. You must have SYSADM or SYSCTRL authority to alter a buffer pool.

Oracle equivalent command

To alter a database buffer cache you can use the ALTER SYSTEM command, or change the initialization file.

Example

Example 11-51 shows how to alter the size of a buffer pool.

Example 11-51   Altering a buffer pool

```
db2 => alter bufferpool BP_TEST size 10000
DB20000I  The SQL command completed successfully.
```

Drop buffer pool

The DROP BUFFERPOOL statement is used to drop a buffer pool. You must have at least one buffer pool active in a database.

Oracle equivalent command

You can reset the value of database buffer caches. As in DB2, you must have at least the default buffer cache active in an instance.

Examples

Example 11-52 shows how to drop a buffer pool.

Example 11-52   Dropping a buffer pool

```
db2 => drop bufferpool BP_TEST
DB20000I  The SQL command completed successfully.
```

Retrieve buffer pool information

You can use the SNAPBP and SNAPBP_PART administrative views and SNAP_GET_BP and SNAP_GET_BP_PART table functions to retrieve information about DB2 buffer pools.

The SNAPBP administrative view returns information related to buffer pool performance, such as page reads and writes from and to the buffer pool. The SNAPBP_PART administrative view returns information related to the buffer pool itself, such as the current size.

You can also use the GET SNAPSHOT FOR BUFFERPOOLS database command to retrieve information about DB2 buffer pools. All these commands return buffer pool information for the current database.

Oracle equivalent command

You can use the V$PARAMETER view to list the existing database buffer cache values and the V$SYSSTAT view to check for performance information about the database buffer caches.

Examples

Example 11-53 shows how to retrieve information about buffer pools using the SNAPBP_PART administrative view.

Example 11-53   Retrieving buffer pool information using the SNAPBP_PART view

```
db2 => SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(BP_NAME,1,15) AS BP_NAME,
BP_CUR_BUFFSZ, BP_NEW_BUFFSZ FROM SYSIBMADM.SN
APBP_PART

DB_NAME  BP_NAME          BP_CUR_BUFFSZ        BP_NEW_BUFFSZ
-------- --------------- -------------------- --------------------
DB2_EMP  IBMDEFAULTBP                    1012                 1012
DB2_EMP  TESTEBP                        50000                50000
DB2_EMP  BP_TEST                        10000                10000
DB2_EMP  IBMSYSTEMBP4K                     16                   16
DB2_EMP  IBMSYSTEMBP8K                     16                   16
DB2_EMP  IBMSYSTEMBP16K                    16                   16
DB2_EMP  IBMSYSTEMBP32K                    16                   16
```

Example 11-54 shows how to retrieve similar information using GET SNAPSHOT FOR BUFFERPOOLS INFORMATION. The sample output has been formatted to show only information about the BP_TEST buffer pool.

Example 11-54   Retrieving information about buffer pool using GET SNAPSHOT

```
db2 => get snapshot for bufferpools on db2_emp

...

Bufferpool Snapshot

Bufferpool name                          = BP_TEST
Database name                            = DB2_EMP
Database path                            =
/db2/data/db2inst1/NODE0000/SQL00001/
Input database alias                     = DB2_EMP
Snapshot timestamp                       = 02/27/2007 14:51:58.909314

Buffer pool data logical reads           = Not Collected
Buffer pool data physical reads          = Not Collected
Buffer pool temporary data logical reads = Not Collected
Buffer pool temporary data physical reads = Not Collected
```

```
Buffer pool data writes                        = Not Collected
Buffer pool index logical reads                = Not Collected
Buffer pool index physical reads               = Not Collected
Buffer pool temporary index logical reads      = Not Collected
Buffer pool temporary index physical reads     = Not Collected
Buffer pool xda logical reads                  = Not Collected
Buffer pool xda physical reads                 = Not Collected
Buffer pool temporary xda logical reads        = Not Collected
Buffer pool temporary xda physical reads       = Not Collected
Buffer pool xda writes                         = Not Collected
Total buffer pool read time (milliseconds)     = Not Collected
Total buffer pool write time (milliseconds)    = Not Collected
Asynchronous pool data page reads              = Not Collected
Asynchronous pool data page writes             = Not Collected
Buffer pool index writes                       = Not Collected
Asynchronous pool index page reads             = Not Collected
Asynchronous pool index page writes            = Not Collected
Asynchronous pool xda page reads               = Not Collected
Asynchronous pool xda page writes              = Not Collected
Total elapsed asynchronous read time           = Not Collected
Total elapsed asynchronous write time          = Not Collected
Asynchronous data read requests                = Not Collected
Asynchronous index read requests               = Not Collected
Asynchronous xda read requests                 = Not Collected
No victim buffers available                    = Not Collected
Direct reads                                   = Not Collected
Direct writes                                  = Not Collected
Direct read requests                           = Not Collected
Direct write requests                          = Not Collected
Direct reads elapsed time (ms)                 = Not Collected
Direct write elapsed time (ms)                 = Not Collected
Database files closed                          = Not Collected
Unread prefetch pages                          = Not Collected
Vectored IOs                                   = Not Collected
Pages from vectored IOs                        = Not Collected
Block IOs                                       = Not Collected
Pages from block IOs                           = Not Collected
Physical page maps                             = Not Collected

Node number                                    = 0
Tablespaces using bufferpool                   = 0
Alter bufferpool information:
 Pages left to remove                          = 0
 Current size                                  = 10000
 Post-alter size                               = 10000
```

11.3.6 Managing database security

To perform an action within the DB2 database, or to access an object, you must have the required authority or privilege. In this section we discuss how to assign the authorities, grant and revoke privileges, and how to list authority and privilege information.

A discussion about authorities and privileges can be found in 2.4.4, "Authentication and authorization" on page 50.

Instance authority

Instance authority assignments are stored inside the database manager configuration file for the instance. To assign an instance authority to a user or a group, you must update the database manager configuration file. Because this change is not dynamic, you need to restart the instance in order for the change to take effect.

Oracle equivalent command

DB2 instance authorities can be considered as being equivalent to Oracle roles. In Oracle, you use the GRANT command to assign a role to a user.

Example

Example 11-55 shows how to assign instance authority to a group.

Example 11-55   Assigning DB2 instance authority

```
-- Assigning an instance authority
db2 update dbm cfg using SYSADM_GROUP db2grp1
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
```

Grant

The GRANT statement is used to assign database authorities and DB2 privileges to a group or user. You can use the ALL or ALL PRIVILEGES clause to grant all appropriate privileges (except CONTROL) on the base table, view, or nickname used in the ON clause.

Oracle equivalent command

The equivalent Oracle command is GRANT.

Example

Example 11-56 shows how to assign a database authority to a group.

Example 11-56   Assigning database authority

```
db2 => grant dbadm on database  to appgroup
DB20000I  The SQL command completed successfully.
```

Example 11-57 shows how to grant privilege on a table to a user.

Example 11-57   Grant privileges

```
db2 => grant select on sysibmadm.dbcfg to appuser
DB20000I  The SQL command completed successfully.
```

Revoke

The REVOKE statement is used to revoke a database authority or privilege authorized by the **grant** statement.

Oracle equivalent command

The equivalent Oracle command is REVOKE.

Example

Example 11-58 shows how to revoke a database authority from a group.

Example 11-58   Revoking database authority

```
db2 => revoke dbadm on database  from appgroup
DB20000I  The SQL command completed successfully.
```

Example 11-59 shows how to revoke a privilege from a user.

Example 11-59   Revoking privileges

```
db2 => revoke select on sysibmadm.dbcfg from  appuser
DB20000I  The SQL command completed successfully.
```

Listing existing privileges

Starting from DB2 9, you can use administrative views to retrieve information about DB2 authorizations and privileges.

Table 11-3 shows some administrative views used to retrieve privileges and authorization information.

Table 11-3   Administrative views used to retrieve security information

Administrative view	Used for
SYSIBMADM.PRIVILEGES	Returns all explicit privileges for all authorization IDs in the current database.
SYSIBMADM.AUTHORIZATIONIDS	Returns all authorization IDs that have been granted privileges or authorities.
SYSIBMADM.OBJECTOWNERS	Retrieves object ownership information.

For DB2 versions prior to DB2 9, you can use the catalog views listed in Table 11-4 on page 557 to retrieve authorization and privilege information. These views are also applicable to DB2 9.

Table 11-4   Views used to retrieve privilege information

Catalog view	Users or groups with privileges on
SYSCAT.DBAUTH	Database-level authorities
SYSCAT.TABAUTH	Table or view
SYSCAT.PACKAGEAUTH	Package
SYSCAT.INDEXAUTH	CONTROL privilege on an index
SYSCAT.COLAUTH	Column
SYSCAT.SCHEMAAUTH	Schema
SYSCAT.SEQUECEAUTH	Sequence
SYSCAT.PASSTHRUAUTH	Query to a data source
SYSCAT.TBSPACEAUTH	USE privilege on a table space

You can also use the **db2look** command to extract Data Definition Language (DDL) with privileges for database objects.

Oracle equivalent command

You can use `DBA_SYS_PRIVS`, `DBA_ROLE_PRIVS` and `DBA_TAB_PRIVS` views to retrieve information about Oracle privileges.

Example

Example 11-60 shows how to list all the privileges granted to a specific object.

Example 11-60   Extracting privileges from a table

```
db2 => SELECT AUTHID, AUTHIDTYPE, PRIVILEGE FROM SYSIBMADM.PRIVILEGES WHERE
OBJECTNAME = 'DBMCFG' and OBJECTSCHEMA = 'SYSIBMADM'

AUTHID             AUTHIDTYPE  PRIVILEGE
------------------ ----------  --------------
PUBLIC                      G  SELECT
APPUSER                     U  SELECT
```

11.3.7 Managing database backup and recovery

DB2 backup and recovery routines are issued through the use of the BACKUP DATABASE and RECOVER DATABASE commands, respectively. You should have regular schedules for backing up database data to a safe location that will be available for recovery operations when needed. DB2 provides different recovery methods for you to design your database recovery strategy. The recovery methods provided are:

► Version recovery: Restoration of a previous image of a database from a backup.

► Roll forward recovery: Reapplication of transactions recorded in the database log files after a version recovery.

► Crash recovery: Performed using information stored in the current database log files.

When the database is created, database log files are also created to record the database activity. By default, these database log files are used in a circular way, as a ring. The database overwrites the oldest database log file when needed, but does not archive the information recorded in that database log file. This recovery mode is known as *circular logging*.

With circular logging, you cannot roll forward a database; you can use only version recovery or crash recovery with the current database log files.You *cannot* perform online backups with this recovery mode. To back up the database, you need to shut down the database.

To roll forward the database, you need to activate *archive logging*. In this mode, the database does not overwrite the oldest database log file when needed. Instead, it creates a new database log file to continue recording the database activities. You *can* perform online backups with this mode.

A database with circular logging is known as a *non-recoverable database*. A database using archive logging is referred to as *recoverable database*. A DB2 database in circular logging recovery mode is equivalent to an Oracle database

in `NOARCHIVELOG` mode. A DB2 database in archive logging recovery mode is equivalent to an Oracle database in `ARCHIVELOG` mode.

The intention of this section is not to discuss every possible backup and recovery scenario, but rather to introduce the commands frequently used to perform database backup and recovery. We also show how you can look for the backup history of the backup executed.

For more information about DB2 backup and recovery features, refer to *Data Recovery and High Availability Guide and Reference*, SC10-4228.

Back up a database

The BACKUP DATABASE command is used to create a backup image of a database or table space. You can perform offline and online backups for your database. To perform an online backup, you must enable the database parameters LOGRETAIN or USEREXIT; refer to "Activating database log retention logging" on page 561, for an example of activating the LOGRETAIN parameter.

You can also perform an *incremental backup*, by which only the modified data since the last full backup will be copied; or a *delta backup*, by which only the modified data since the last backup of any type will be copied.

Backup images are created at the target location specified when you invoke the backup utility. This location can be the following:

► A directory (for backups to a disk or diskette)
► A device (for backups to tape)
► A Tivoli Storage Manager (TSM) server
► Another vendor's server

The BACKUP DATABASE command automatically establishes a connection to the specified database. If you are connected to the database that you are performing the backup on, you will be automatically disconnected when the backup is started.

You can perform the backup locally or remotely, and the backup image will remain in the local server unless you use a storage management product such as Tivoli Storage Manager. You must have SYSADM, SYSCTRL or SYSMAINT authority to perform a backup.

You can configure a database for automatic backup using either GUI or CLI tools. For more information about automatic backup features, see 11.3.7, "Managing database backup and recovery" on page 558.

Oracle equivalent command

Oracle provides two types of backups:

- ▶ Physical: Run through the `Recovery Manager` (`RMAN`) utility or with the BEGIN BACKUP and END BACKUP commands,
- ▶ Logical: Run though `EXPORT` or `EXPDP` (Data Pump) tools.

You can only roll forward a database (or, in Oracle terms, perform the media recovery) from a *physical* backup. The DB2 BACKUP DATABASE command produces a logical backup of the database, but allows you to roll forward the database.

Example

Example 11-61 shows how to perform a full offline backup from a database.

Example 11-61   Performing a full offline backup

```
db2 => backup database sample to '/db2/backup'

Backup successful. The timestamp for this backup image is : 20070226210406
```

Example 11-62 shows how to perform a full online backup from a database.

Example 11-62   Performing a full on line backup

```
db2 => backup database db2_emp online to  '/db2/backup/'

Backup successful. The timestamp for this backup image is : 20070226211604
```

Restore database

The RESTORE DATABASE command is used to recreate a damaged or corrupted database from a backup. You can restore a database over an existing database, or to a new database. With the restore database command, you can redirect your restore to change the physical location of the table space containers. You can restore the full database, a single table space, or a group of table spaces. You can restore an image direct from a TSM Server.

To restore to an existing database, you must have SYSADM, SYSCTRL or SYSMAINT authority. To restore to a new database, you must have SYSADM or SYSCTRL authority.

Oracle equivalent command

There is no Oracle command or utility that is directly equivalent to the DB2 RESTORE DATABASE command. You might consider the DB2 RESTORE DATABASE command as being similar to the `IMPORT` or `IMPDP` tools for logical

backups, or the RMAN RESTORE and RMAN RECOVER commands for
physical backups.

Example

Example 11-63 shows how to restore a database from a backup.

Example 11-63   Restoring a database from a backup

```
db2 => restore database sample from '/db2/backup'
SQL2539W  Warning!  Restoring to an existing database that is the same as the
backup image database.  The database files will be deleted.
Do you want to continue ? (y/n) Y
DB20000I  The RESTORE DATABASE command completed successfully.
```

Roll forward database

The ROLLFORWARD DATABASE command recovers a database to a point in
time by applying transactions recorded in the database log files after a database
or table space backup image has been restored, or if any table space has been
taken offline by the database due to a media failure. You can roll forward a
database until a specific point in time, or until the end of the database logs. You
must have SYSADM, SYSCTRL or SYSMAINT authority to roll forward a
database.

Oracle equivalent command

The `rollforward database` command can be considered as being equivalent to
the Oracle RECOVER DATABASE command in SQL*Plus in the way that both
commands apply transactions stored in the database log files (in a DB2
database) or the archived log files (in an Oracle database) to recover a database
or table space.

Example

Example 11-64 shows how to roll forward a database.

Example 11-64   Roll forward a database

```
db2 => rollforward db sample to end of logs and complete
```

Activating database log retention logging

In this section we show how to activate database log retention logging.

Oracle equivalent command

Activating the LOGRETAIN database parameter is similar to changing the Oracle
database from the NOARCHIVELOG mode to the ARCHIVELOG mode.

Example

Example 11-65 shows how to activate the database configuration LOGRETAIN parameter.

Example 11-65   Activating the LOGRETAIN database configuration parameter

```
-- Trying to perform an online backup
db2 => backup database db2_emp online to '/db2/backup'
SQL2413N  Online backup is not allowed because the database is not recoverable
or a backup pending condition is in effect.

-- It did not work because the retention logging is not enabled.
-- Checking the current value of LOGRETAIN parameter.
db2 => select name, substr(value,1,20) as running, substr(deferred_value,1,20)
as DEFERRED from sysibmadm.dbcfg where name = 'logretain'

NAME                             RUNNING              DEFERRED
-------------------------------- -------------------- -----------------
logretain                        OFF                  OFF

-- Updating the LOGRETAIN DB parameter to enable retention logging
db2 => update db cfg using logretain yes
DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
SQL1363W One or more of the parameters submitted for immediate modification
were not changed dynamically. For these configuration parameters, all
applications must disconnect from this database before the changes become
effective.

-- Checking the deferred value for logretain
db2 => select name, substr(value,1,20) as running, substr(deferred_value,1,20)
as DEFERRED from sysibmadm.dbcfg where name = 'logretain'

NAME                             RUNNING              DEFERRED
-------------------------------- -------------------- -----------------
logretain                        OFF                  RECOVERY

-- Reactivating the database to have the change take effect
db2 => connect reset
DB20000I  The SQL command completed successfully.

db2 => deactivate database db2_emp
SQL1496W  Deactivate database is successful, but the database was not
activated.

db2 => activate database db2_emp
SQL1116N  A connection to or activation of database "DB2_EMP" cannot be made
because of BACKUP PENDING.  SQLSTATE=57019
```

```
-- The database is in backup pending state because we changed the
-- database configuration parameter LOGRETAIN. After this change, a
-- database backup is required to activate the database.
-- Performing a full backup.
db2 => backup database db2_emp to '/db2/backup'

Backup successful. The timestamp for this backup image is : 20070226212604


-- Trying to activate the database again
db2 =>  activate database db2_emp
DB20000I  The ACTIVATE DATABASE command completed successfully.


-- Checking the value for logretain recovery, and try to perform an online
backup.
db2 => connect to db2_emp

   Database Connection Information

 Database server        = DB2/AIX64 9.1.1
 SQL authorization ID   = DB2INST1
 Local database alias   = DB2_EMP

db2 => select name, substr(value,1,20) as running, substr(deferred_value,1,20)
as DEFERRED from sysibmadm.dbcfg where name = 'logretain'

NAME                             RUNNING              DEFERRED
-------------------------------- -------------------- -----------------
logretain                        RECOVERY             RECOVERY

db2 => backup database db2_emp online to '/db2/backup'
Backup successful. The timestamp for this backup image is : 20070226212819
```

List history and prune history

The LIST HISTORY command is used to list the history of the recovery and
administrative events. For example, you can list all backups taken for a
database, or the creation and deletion of table spaces.

The PRUNE HISTORY command is used to delete entries from the recovery
history file, or to delete log files from the active log path. Regularly pruning the
recovery history file can prevent it from growing too large.

Oracle equivalent command

There is no direct equivalent command in Oracle to trim the alert.log file to which
the table space operations, begin backup, end backup, and backups taken with
RMAN are logged.

Example

Example 11-66 shows how to list all backups from a database. The sample output only shows the first backup log.

Example 11-66   Listing backup history

```
db2 list history backup all for db2_emp

                   List History File for db2_emp

Number of matching file entries = 5


 Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
 -- --- ------------------ ---- --- ------------ ------------ ---------
  B  D  20070226210800001   F    D  S0000000.LOG S0000000.LOG
 ------------------------------------------------------------------------
  Contains 6 tablespace(s):

  00001 SYSCATSPACE
  00002 USERSPACE1
  00003 USER_IND_TBS
  00004 USER_LOB_TBS
  00005 USER_DATA_TBS
  00006 SYSTOOLSPACE
 ------------------------------------------------------------------------
    Comment: DB2 BACKUP DB2_EMP OFFLINE
 Start Time: 20070226210800
   End Time: 20070226210807
     Status: A
 ------------------------------------------------------------------------
  EID: 6 Location:
/db2/backup
```

Example 11-67 shows how to delete the entries in the recovery log file.

Example 11-67   Deleting backup history

```
-- Verifying history records

db2 list history backup all for db2_emp

                   List History File for db2_emp

Number of matching file entries = 5
...

--Deleting old entries
```

```
db2 prune history 20070227000000
DB20000I   The PRUNE command completed successfully.

-- Verifying history records
db2 list history backup all for database db2_emp

                    List History File for db2_emp

Number of matching file entries = 1
```

11.4 Automatic database management

DB2 provides many automatic capabilities for administrative activities and
performance tuning. With regard to administrative activities, you can perform
database backup automatically, keep statistics current, and reorganize tables
and indexes as necessary. Also, you can set the automatic storage for table
space. In relation to performance, the configuration advisor is used to create an
auto-configured database and the automatic memory management can control
the memory resource for a database.

To enable the automatic maintenance features, the automatic maintenance
database configuration parameters are set to AUTOMATIC or ON. The
confirmation automatic maintenance wizard is provided for configuring the
required automatic maintenance activity. Some of the automatic features are
enabled by default when you create a database.

From the DB2 Control Center, you can check your database maintenance
settings in the database panel in the right bottom frame, as shown in Figure 11-3.
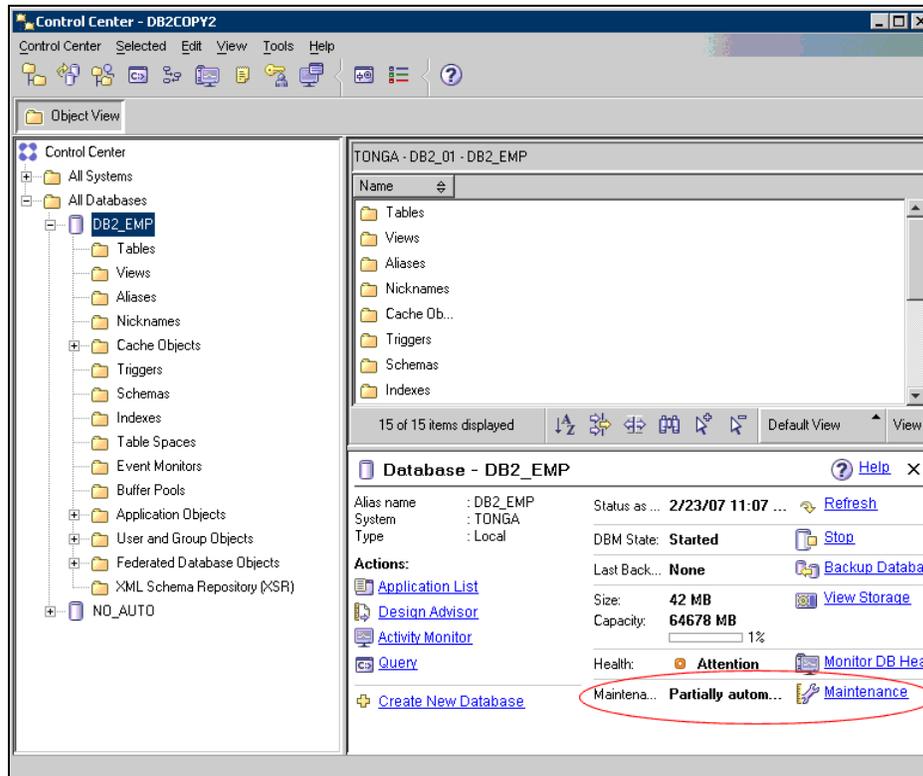
Figure 11-3   DB2 Control Center - database panel

The database panel displays brief information about the current database including the database status, date of last backup, size of database, database health status, and automatic management setting. A link to launch the wizards for the management of these activities is provided along the side. In the left side of the database panel are links to launch the most frequently accessed GUI tools such as Command Center, Activity Monitor, and Design Advisor.

You can also use the DB2 command to check or update the automatic database maintenance settings as follows:

```
/WORK # db2 get db cfg for db2_emp |grep "Automatic"
 Automatic maintenance                     (AUTO_MAINT) = ON
   Automatic database backup          (AUTO_DB_BACKUP) = OFF
   Automatic table maintenance         (AUTO_TBL_MAINT) = ON
     Automatic runstats                 (AUTO_RUNSTATS) = ON
     Automatic statistics profiling   (AUTO_STATS_PROF) = OFF
       Automatic profile updates        (AUTO_PROF_UPD) = OFF
       Automatic reorganization           (AUTO_REORG) = OFF
```

In this section, we cover how you can manage your database automatically by category.

11.4.1 Automatic database configuration

In DB2, a database is tuned by the configuration advisor automatically and the memory of the database including the bufferpool, sortheap and locklist is dynamically tuned depending on the workload. Also, DB2 introduces a few more AUTOMATIC parameters.

Create database with the configuration advisor

The DB2 Configuration Advisor helps you to tune performance and to balance memory requirements for a database by suggesting which configuration parameters to modify and by providing suggested values for them.

In DB2 9, the Configuration Advisor is automatically invoked when you create a database. To disable this feature or to explicitly enable it, use the **db2set** command before creating the database as shown:

```
/WORK # db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
/WORK # db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

The Configuration Advisor determines and sets the database configuration parameters and the size of the default buffer pool (IBMDEFAULTBP). The values are selected based on system resources and the intended use of the system.

This initial automatic tuning means that your database will have better performance than a database created with the DB2 default values. It also means that you will spend less time tuning your system after the database has been created. The Configuration Advisor can be invoked at any time (even after your databases are populated) to recommend and, optionally, apply a set of configuration parameters to optimize DB2 performance based on the current system characteristics. Figure 11-4 shows a sample output of the Configuration Advisor.
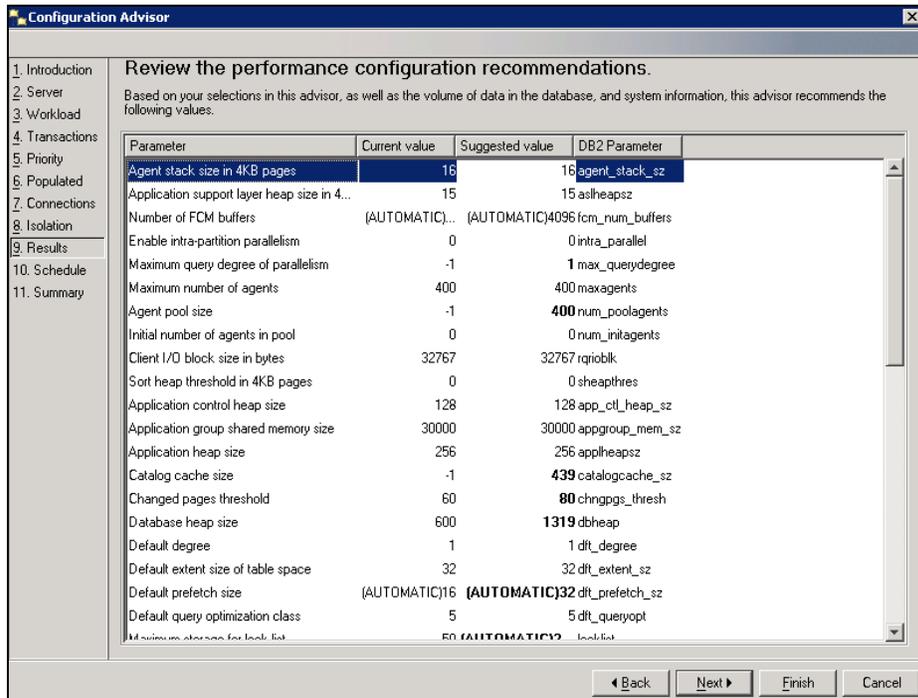
Figure 11-4   Sample output from Configuration Advisor

Self-tuning memory

The self-tuning memory feature is for automating the management of the memory resource for DB2 databases. This feature simplifies the task of memory configuration by automatically adjusting the values for several memory configuration parameters including sort, the package cache, the lock list, and the buffer pools. The memory tuner is responsive to significant changes in workload characteristics and iteratively adjusts the values of the memory configuration parameters and the sizes of the buffer pools to optimize performance.

In DB2 9, the self-tuning memory feature is enabled by default for a single partitioned database only. You can disable this feature setting the database configuration parameter SELF_TUNING_MEM to OFF.

The procedure to enable the self-tuning memory feature is as follows:

1. Enable self-tuning for database

   The DB CFG parameter SELF_TUNING_MEM should be set to ON. You can set or check the value of SELF_TUNING_MEM through the DB2 Control Center or by command.

To use the GUI tool, select **Control Center** → **Change Database Configuration Parameter.** Figure 11-5 illustrates the Database Configuration window for setting this parameter.
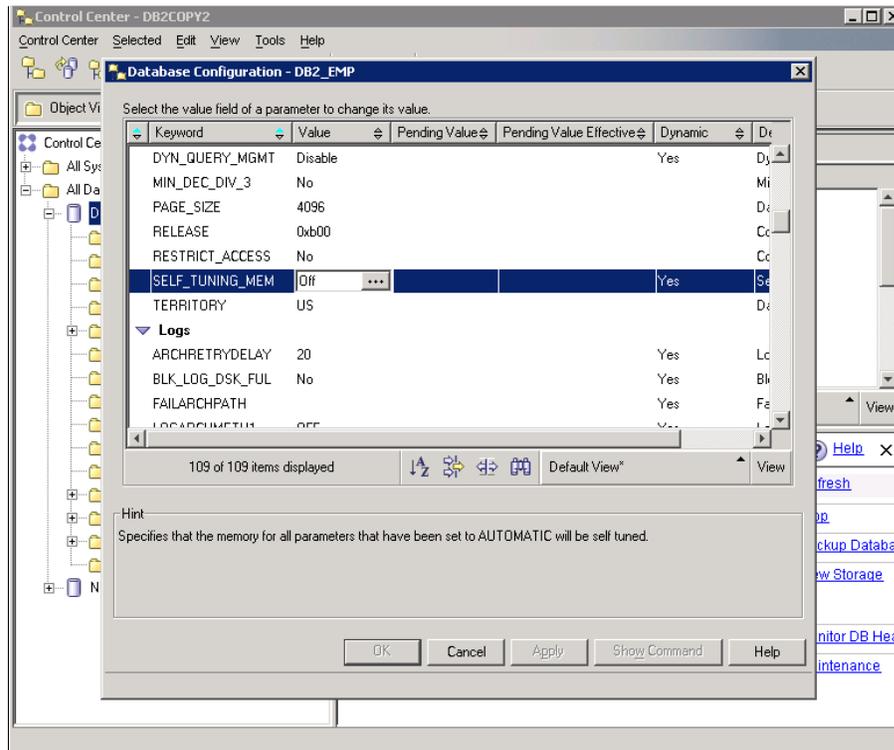
Figure 11-5   DB2 Control Center - database configuration window

To retrieve the value by command:

```
/WORK # db2 get db cfg |grep SELF_TUNING_MEM
 Self tuning memory                    (SELF_TUNING_MEM) = OFF
```

To set the value by command:

```
/WORK # db2 update db cfg using self_tuning_mem on
```

2. Enable self-tuning of the memory area

   Database configuration parameters related to database resource must be set to AUTOMATIC to have DB2 adjust the values automatically, depending on the workload. Some of these parameters have dependencies that are also required to be set. For example, if you want to enable LOCKLIST for self-tuning, you must set both LOCKLIST and MAXLOCKS to AUTOMATIC.

Table 11-5 lists the database configuration parameters for self-tuning the memory area and their dependencies.

Table 11-5   Self-tuning memory parameters and their dependencies

Parameter	Description	Dependency with other parameter
DATABASE_MEMORY	The amount of shared memory reserved for the database shared memory region	
LOCKLIST	Maximum storage allocated to the lock list	Set MAXLOCKS to AUTOMATIC
MAXLOCKS	Percentage of lock lists per application	
PCKCACHESZ	Package cache size	
SHEAPTHRES_SHR	Sort heap threshold for shared sorts	► In DB CFG: Set SORTHEAP to AUTOMATIC ► In DBM CFG: Set SHEAPTHRES to 0
SORTHEAP	Sort list heap	In DBM CFG: Set SHEAPTHRES to 0

3. Set the size of the buffer pool to AUTOMATIC to enable self-tuning for the buffer pool.

   You can use the DB2 command to enable self-tuning for an existing buffer pool as follows:

   `/WORK # `**`db2 alter bufferpool`**` IBMDEFAULTBP `**`size automatic`**

   Figure 11-6 shows how to create a buffer pool with self-tuning memory enabled through the Control Center.
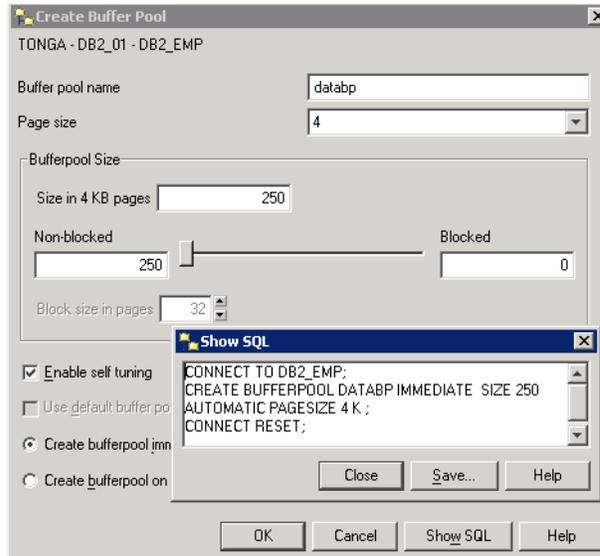
Figure 11-6   Creation of buffer pool with self-tuning memory feature

For more information on self tuning memory, refer to "Automatic configuration using self-tuning memory" in the DB2 publication *Performance Guide*, SC10-4222.

Other self-tuning configuration parameters

In addition to the tune memory area configuration parameters, DB2 also tunes the processor-related configuration parameters automatically. Table 11-6 lists these processor-related parameters and explains how DB2 sets their values.

Table 11-6   Processor-related configuration parameters

Parameter	Description	The value when set to AUTOMATIC
NUM_IOCLEANERS	Number of asynchronous page cleaners	Based on the number of CPUs configured on the current machine, as well as the number of local logical database partitions in a partitioned database environment.
NUM_IOSERVERS	Number of prefetchers	Based on the parallelism settings of the table spaces in the current database partition.

11.4.2 Automatic storage

The automatic storage feature simplifies storage management for table spaces. When you create a database, you specify the storage paths for placing the table space data. DB2 will then manage the container and space allocation for the table spaces as they are created and populated.

A database can only be enabled for automatic storage when it is first created. DB2 creates an automatic storage database by default. Here is an example to create database-enabled automatic storage:

```
CREATE DATABASE ADB ON /data/path1, /data/path2
```

This is the same as the following statement:

```
CREATE DATABASE ADB AUTOMATIC STORAGE YES ON /data/path1,
/data/path2
```

If you want to create a database without the automatic storage feature, you must create the database using the AUTOMATIC STORAGE NO clause as follows:

```
CREATE DATABASE ASNODB1 AUTOMATIC STORAGE NO
```

After you create a database-enabled automatic storage feature, you can add the storage or alter the storage with the ALTER DATABASE command as shown in Figure 11-7.
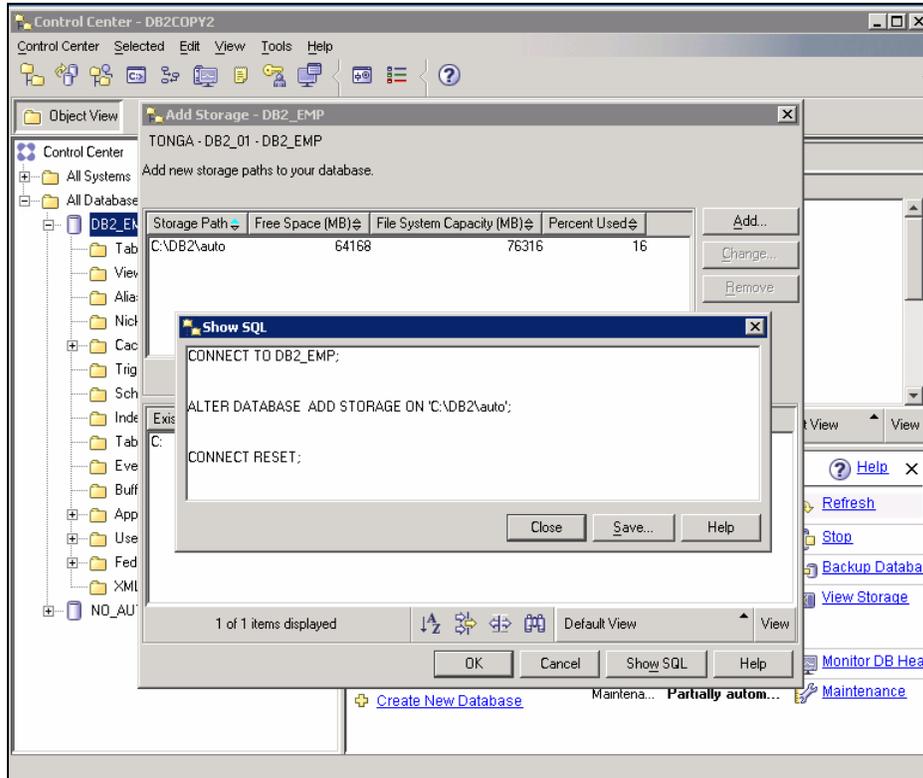
Figure 11-7   Alter storage of a database in DB2 Control Center

For more information, refer to "Creating a database" in the DB2 publication *Administration Guide: Implementation*, SC10-4221.

11.4.3 Automated REORG on tables and indexes

After frequent table data inserts and deletes, data might be fragmented and the logically sequenced data might be on non-sequential physical data pages. Therefore, the database manager must perform additional read operations to access data. You must consider reorganizing the table to match the index and to reclaim space, in order for the database manager to access your data efficiently.

When you use the REORG command, the table or the index is reorganized and space is reclaimed. This maintenance activity can be automated. Automatic reorganization manages offline table and index reorganization without users having to worry about when and how to reorganize their data. You can set up the automatic table and index reorganization by using the Configure Automatic

Maintenance wizard. Figure 11-8 shows the Defragmented Data configuration setting window.
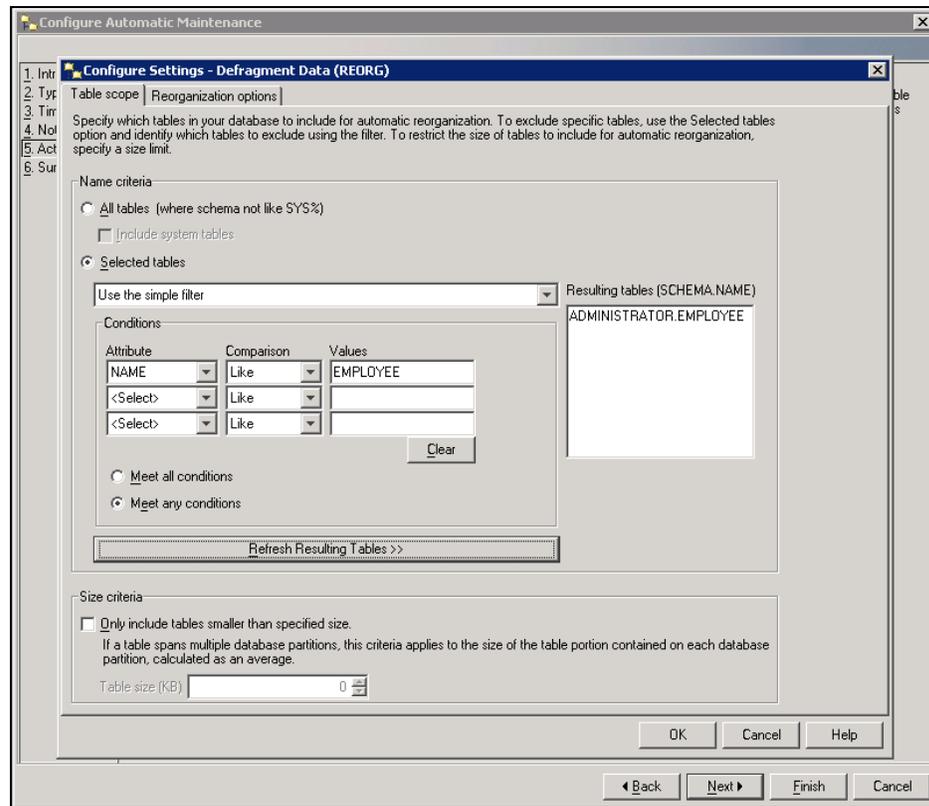
Figure 11-8   Configure Automatic Maintenance window

For more information about REORG of tables and indexes, refer to "Table and index maintenance" in DB2 *Performance Guide*, SC10-4222.

11.4.4 Automatic statistics collection

Automatic statistics collection helps to improve database performance by collecting up-to-date table statistics. DB2 determines which statistics are required and which statistics need to be updated by your workload. The RUNSTATS utility is automatically invoked in the background to ensure that the correct statistics are collected and maintained. Statistics are first collected on the tables that need statistics update the most. The DB2 optimizer can then choose an access plan based on accurate statistics.

You can enable or disable this function using the Configure Automatic Maintenance wizard.

For more information about self-tuning memory, refer to *"Automatic statistics collection"* in the DB2 publication *Performance Guide*, SC10-4222*.*

11.4.5 Automatic backup

Automatic database backup is another DB2 automatic management feature that helps ease a database administrator's job. To automate the database backup job, you must set the specifications for the backup including the backup mode, target media, the type of backup and the schedule. DB2 Backup wizard will guide you through the required automatic backup setup steps. Figure 11-9 shows a Backup wizard window.
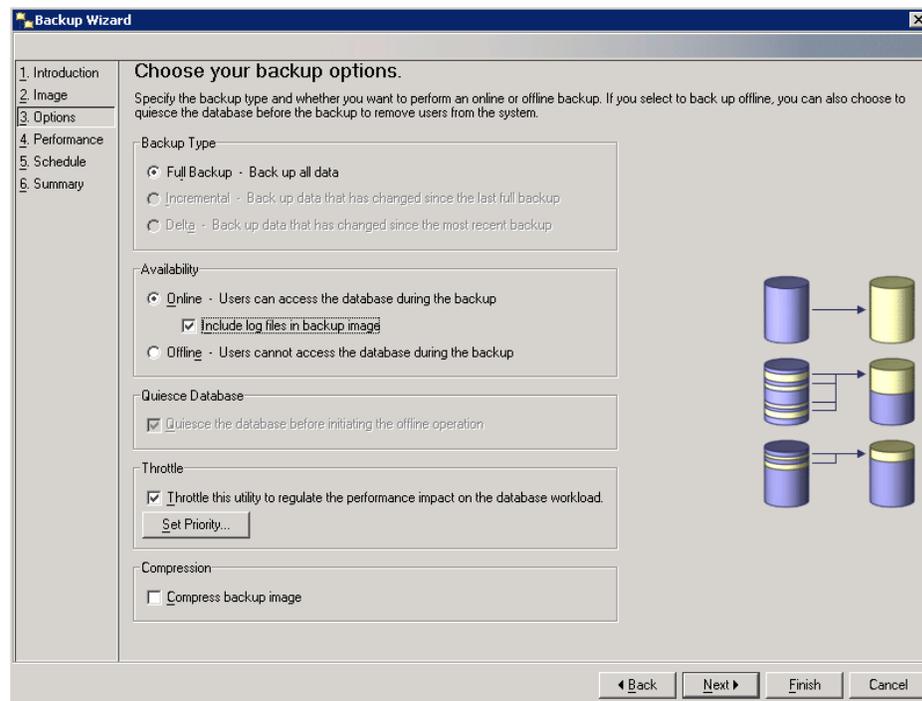
Figure 11-9   Backup wizard

11.4.6 Utility throttling

When you enable an automatic maintenance function in a database, you might have concerns about the performance impact of the maintenance activity on user transactions. The DB2 utility throttling mechanism ensures that the throttled

utilities, statistics collection, backup, and rebalancing and so on, are run as aggressively as possible without violating the policy specified by you.

The degree of impact of the throttled utilities can be set using the Utility Status Manager in the Control Center (see Figure 11-10), updating DBM CFG parameter UTIL_IMPACT_LIM directly as shown in the following example:

`/WORK # ` **`db2 update dbm cfg using UTIL_IMPACT_LIM`** 10
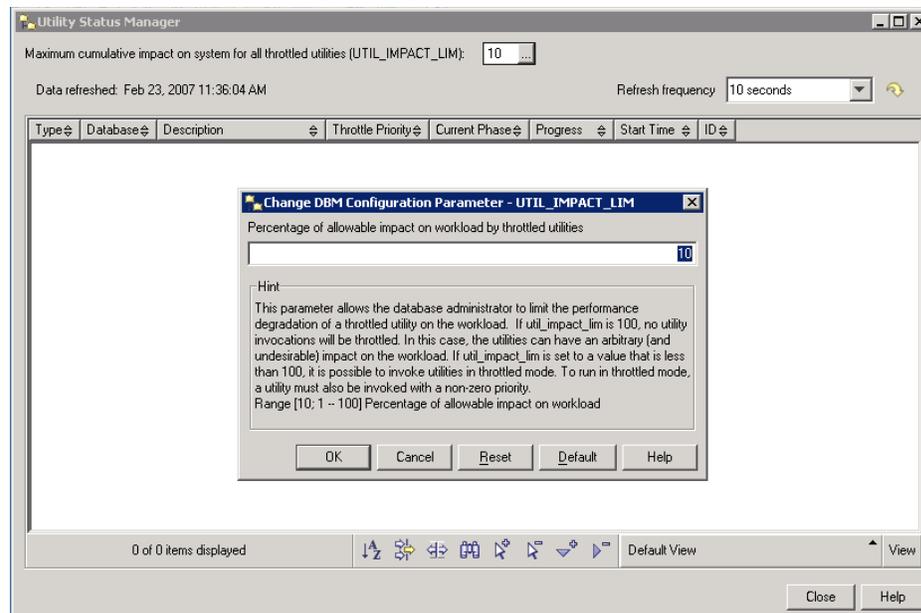
Figure 11-10   Utility Status Manager

11.4.7 Automatic diagnostics using Health Monitor

The Health Monitor is a server-side tool that proactively monitors conditions or changes in your database environment that could result in performance degradation or a potential outage. A range of health information is presented without any form of active monitoring on your part. If a health risk is encountered, DB2 will inform you and provide advise on the actions that must be taken. The Health Monitor gathers information about the system by using the snapshot monitor without imposing a performance penalty.

The health status of the database is marked as normal, warning, or alarm according to the pre-defined threshold of the health indicator. The database health information is gathered under the Health Center, As shown in Figure 11-11, the Health Center provides the details of the reason of the warning or alarm to help you pinpoint the problem. You can use the Recommendation

Advisor to obtain problem-solving recommendations such as creating indexes, resizing sort heap memory, or adding more containers to the table space.
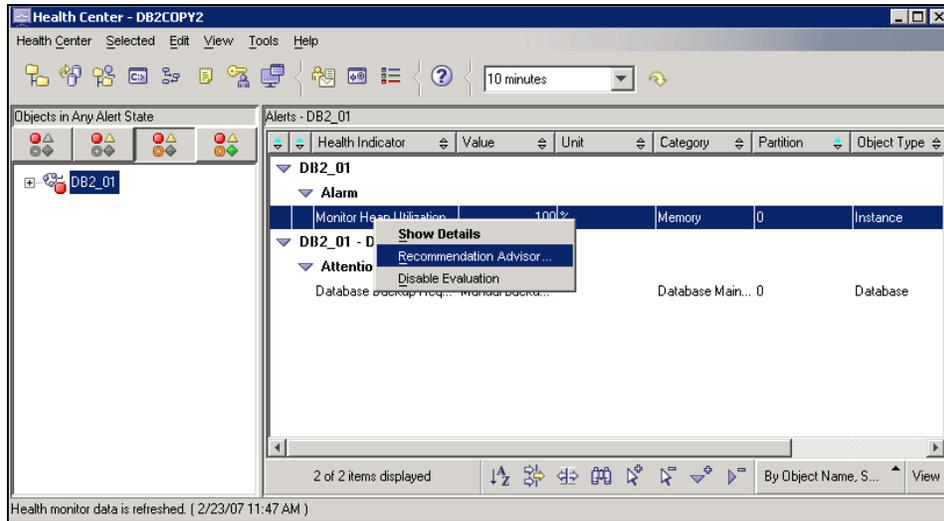
Figure 11-11   Health Center

You can also define the health indicators, modify the threshold degree of the indicators, or specify a script to run when a warning or alarm occurs from the Health Center. See Figure 11-12 on page 578.

For more information about the Health Monitor, refer to "Health Monitor" in the DB2 *System Monitor Guide and Reference,* SC10-4251*.*
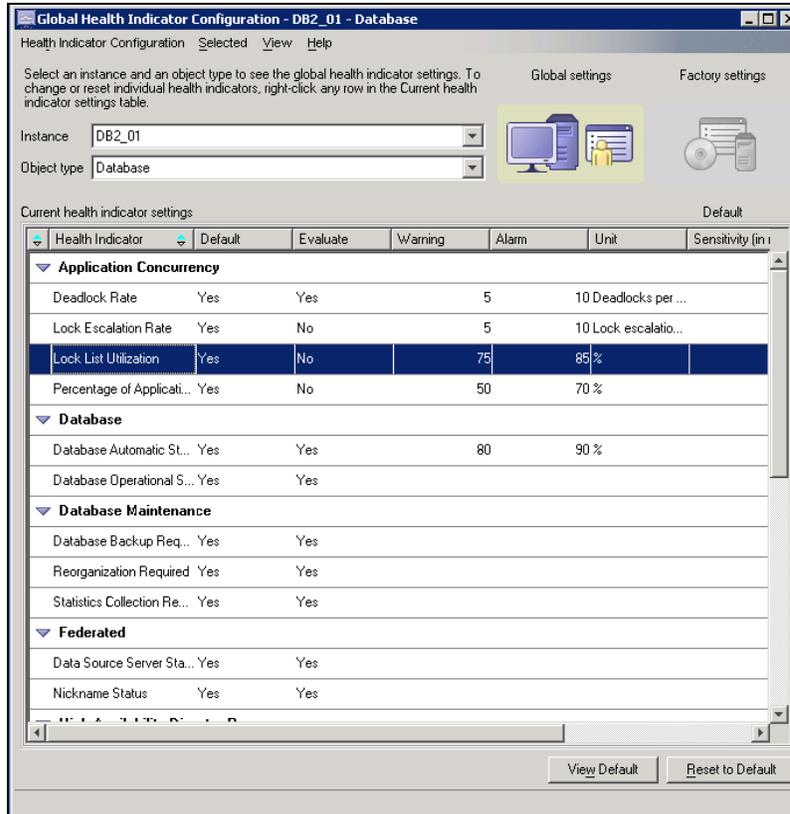
Figure 11-12   Health Center - health indicator configuration

11.5 Monitoring

In this section, we introduce the monitoring tools provided in DB2. We describe the commands, administrative views, and tools that can be used to monitor the database objects such as buffer pool, locks and SQL statements.

11.5.1 Monitoring tools

DB2 provides multiple monitoring tools for application, instances, and database monitoring. In addition to Health Monitor, you also can use the following tools for database system monitoring:

► Activity Monitor - A GUI tool for application activity monitoring including locks, log usage and performance.

- ► Snapshot monitoring - Allows you to capture information about the database and any connected applications at a specific time. Refer to Chapter 10, section , "Monitoring tools" on page 472 for more details.
- ► db2pd - A text-based monitoring tool that retrieves information from the DB2 database system memory sets.

Activity Monitor

The Activity Monitor provides a set of predefined reports based on a specific subset of monitor data to assist you in monitoring database activities and application performance, or in solving a particular problem. With Activity Monitor, you can easily determine the cause of a problem related to database performance (for example, lock waiting situations), or tune an application or a SQL statement that adversely affects performance.

You can set up an activity monitor through the Set Up Activity Monitor wizard (Figure 11-13 on page 580) in the Control Center. The Set Up Activity Monitor wizard can also be invoked using the `db2am` command.

Using the Set Up Activity Monitor, you can choose the system-defined monitoring tasks (which are the monitoring tasks predefined by DB2), or you can define your own monitoring activities. The system-defined tasks include the following categories:

- ► Resolving a general database system slowdown
- ► Resolving the performance degradation of an application
- ► Resolving an application locking situation
- ► Tuning the dynamic SQL statement cache

Examples of monitoring tasks can be to determine the following:

- ► Which application holds the largest number of locks
- ► Which application causes a lock wait
- ► Which SQL statement takes a long sort time
- ► Which SQL statements consume the most CPU time
- ► Which application performs a large number of rows read
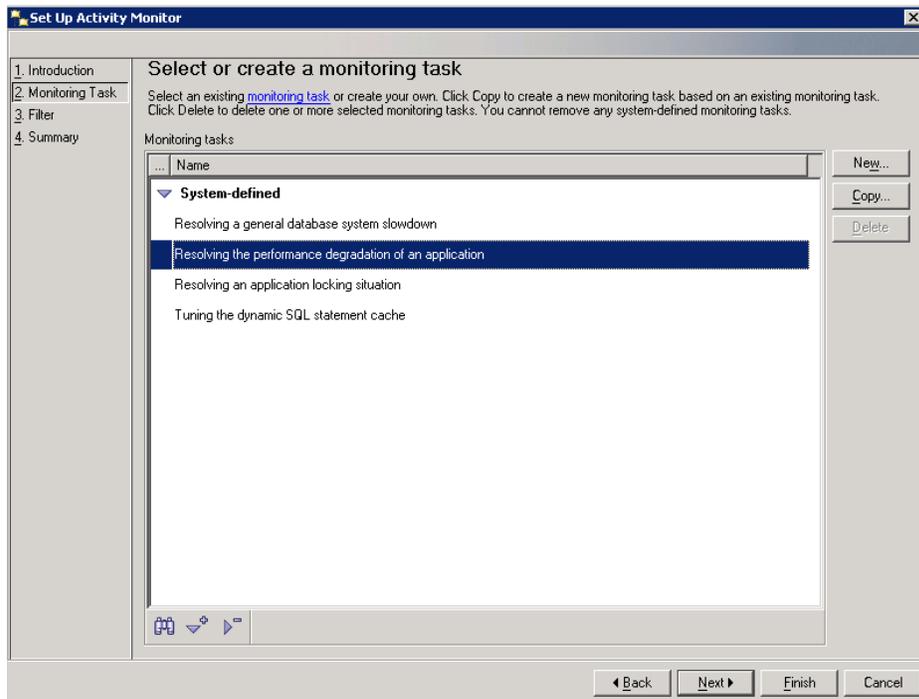
Figure 11-13   Activity Monitor - choose a category to monitor

Figure 11-14 illustrates a monitoring result window. The monitoring data is refreshed automatically if automatic refresh is set.
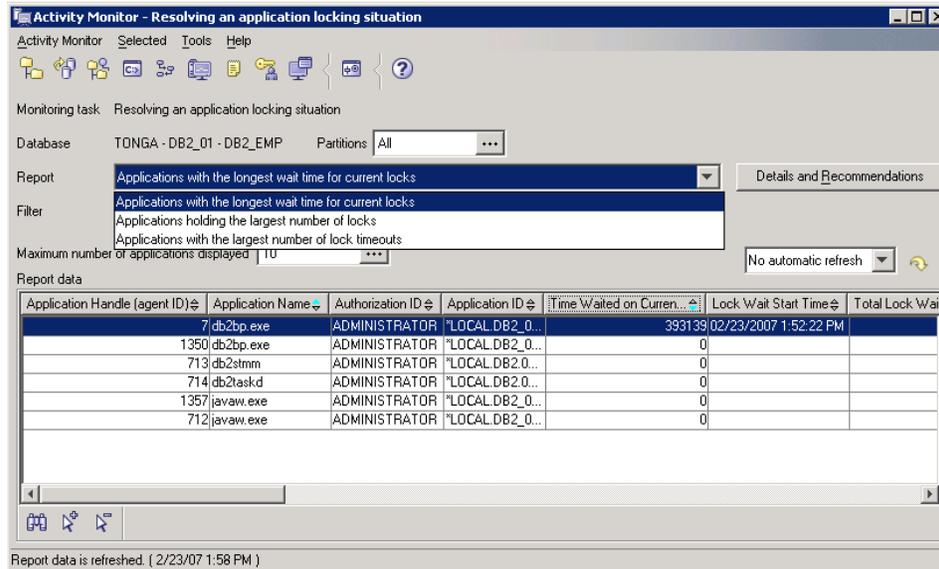
Figure 11-14   Activity Monitor - monitoring result

For more information about the Activity Monitor, refer to "Activity Monitor" in the DB2 publication *System Monitor Guide and Reference,* SC10-4251.

db2pd utility

The db2pd utility collects detailed database information such as transactions, table statistics, dynamic SQL and more, without acquiring any latches or using any engine resources. It is therefore possible (and expected) to retrieve information that is changing while db2pd is collecting information; thus, the data might not be completely accurate. Nevertheless, the tool can be useful for problem determination and for collecting information. Two benefits of collecting information without latching include faster retrieval and no competition for the engine resources.

This utility provides options for various types of information. To see the options of the db2pd utility, use the -h option as follows:

`/WORK # `**`db2pd -h`**

How to use db2pd to monitor various database objects is described in the following sections.

11.5.2 Monitoring database objects

In this section, we describe the commands for monitoring database objects.

Configuration parameter

The commands to obtain database manager and database configuration information are as follows:

► Instance (Database Manager) parameter

   – DB2 GET DBM CFG
   – db2pd -db *<dbname>* -dbmcfg

► Database configuration parameter

   – DB2 GET DB CFG
   – db2pd -db *<dbname>* -*dbcfg*

Current memory usage

The commands to obtain database memory usage information are as follows:

► Database memory usage

   – db2mtrk -i -d
   – db2pd -db *<dbname>* -mempools
   – SELECT FROM SYSIBMADM.SNAPDB_MEMORY_POOL

Example 11-68 shows a sample output of the memory monitoring command **db2mtrk**.

Example 11-68   Monitoring database memory using db2mtrk

```
/WORK #  db2mtrk -i -d
Tracking Memory on: 2007/02/27 at 15:50:34

Memory for instance

   monh      other
   448.0K    2.6M

Memory for database: DB2_EMP

   bph (3)   bph (2)   utilh      pckcacheh catcacheh bph (1)   bph (S32K)
   41.2M     203.5M    64.0K      1.3M      320.0K    5.4M      704.0K

   bph (S16K) bph (S8K) bph (S4K) shsorth   lockh     dbh       other
   448.0K     320.0K    256.0K    384.0K    2.9M      5.2M      192.0K
```

Example 11-69 shows how to obtain memory usage from the administrative view SYSIBMADM.SNAPDB_MEMORY_POOL.

Example 11-69   Query on SYSIBMADM.SNAPDB_MEMORY_POOL view

```
/WORK # cat mon_dbmem.db2
select POOL_ID, POOL_CUR_SIZE
```

```
from SYSIBMADM.SNAPDB_MEMORY_POOL
where DB_NAME='DB2_EMP' ;
/WORK # db2 -tf mon_dbmem.db2


POOL_ID        POOL_CUR_SIZE
-------------- --------------------
BP                        43253760
BP                       213385216
UTILITY                      65536
PACKAGE_CACHE              1441792
CAT_CACHE                  327680
BP                         5701632
BP                          720896
BP                          458752
BP                          327680
BP                          262144
SHARED_SORT                 393216
LOCK_MGR                   3080192
DATABASE                   5439488
OTHER                       196608

  14 record(s) selected.
```

▶ Current buffer pool list

    — db2pd -db *<dbname>* -bufferpools
    — GET SNAPSHOT FOR BUFFERPOOLS ON *<dbname>*
    — SELECT FROM SYSIBMADM.SNAPBP_PART

Example 11-70 shows how to obtain buffer pools defined from the administrative view SYSIBMADM.SNAPBP_PART.

Example 11-70   List buffer pools from SYSIBMADM.SNAPBP_PART view

```
/WORK # cat mon_bp.db2
select substr(BP_NAME,1,20) as BUFFERPOOL ,BP_CUR_BUFFSZ,BP_NEW_BUFFSZ
from SYSIBMADM.SNAPBP_PART
where DB_NAME='DB2_EMP' ;

/WORK # db2 -tf mon_bp.db2


BUFFERPOOL           BP_CUR_BUFFSZ        BP_NEW_BUFFSZ
-------------------- -------------------- --------------------
IBMDEFAULTBP                         1012                 1012
TESTEBP                             50000                50000
BP_TEST                             10000                10000
IBMSYSTEMBP4K                          16                   16
IBMSYSTEMBP8K                          16                   16
IBMSYSTEMBP16K                         16                   16
```

```
IBMSYSTEMBP32K                          16                      16

  7 record(s) selected.
```

▶ Current buffer pool usage (including statistics)

  – db2pd -db <dbname> -bufferpool

    Note that the snapshot monitor must be on to obtain the information.

  – GET SNAPSHOT FOR BUFFERPOOLS ON *<dbname>*

  – SELECT FROM SYSIBMADM.SNAPBP

  – SELECT FROM SYSIBMADM.BP_HITRATIO, SYSIBMADM.BP_READ_IO, SYSIBMADM.BP_WRITE_IO

Example 11-71 illustrates how to obtain the statistics information of the buffer pool from the administrative views.

Example 11-71   Get statistics information of buffer pool using views

```
/WORK # cat mon_bpstat.db2
select SUBSTR(BP_NAME,1,15) AS BP_NAME,
       (POOL_DATA_L_READS+POOL_INDEX_L_READS) AS TOTAL_LOGICAL_READS,
       (POOL_DATA_P_READS+POOL_INDEX_P_READS) AS TOTAL_PHYSICAL_READS
from  SYSIBMADM.SNAPBP
where DB_NAME='DB2_EMP' ;
/WORK # db2 -tf mon_bpstat.db2

BP_NAME         TOTAL_LOGICAL_READS  TOTAL_PHYSICAL_READS
--------------- -------------------- --------------------
IBMDEFAULTBP                       0                     0
TESTEBP                            0                     0
BP_TEST                            0                     0
IBMSYSTEMBP4K                      0                     0
IBMSYSTEMBP8K                      0                     0
IBMSYSTEMBP16K                     0                     0
IBMSYSTEMBP32K                     0                     0

  7 record(s) selected.

/WORK # cat mon_bphit.db2
select
       substr(BP_NAME,1,15) as BP_NAME,
       TOTAL_LOGICAL_READS as LOGICAL_READ,
       TOTAL_PHYSICAL_READS as PHYSICAL_READ,
       TOTAL_HIT_RATIO_PERCENT as HIT_RATIO
from SYSIBMADM.BP_HITRATIO
where DB_NAME='DB2_EMP' ;
/WORK # db2 -tf mon_bphit.db2
```

```
BP_NAME          LOGICAL_READ     PHYSICAL_READ     HIT_RATIO
---------------- ---------------- ----------------- ---------
IBMDEFAULTBP                    0                 0         -
TESTEBP                         0                 0         -
BP_TEST                         0                 0         -
IBMSYSTEMBP4K                   0                 0         -
IBMSYSTEMBP8K                   0                 0         -
IBMSYSTEMBP16K                  0                 0         -
IBMSYSTEMBP32K                  0                 0         -

  7 record(s) selected.
```

- ► Log usage
    - – GET SNAPSHOT FOR DATABASE ON *<dbname>*
    - – db2pd -db *<dbname>* -logs
    - – SELECT FROM SYSIBMADM.LOG_UTILIZATION

    Example 11-72 shows how to obtain the log usage information from the administrative view SYSIBMADM.LOG_UTILIZATION.

Example 11-72   Get log usage using view

```
/WORK # cat mon_log.db2
select
        LOG_UTILIZATION_PERCENT as UTIL_PERCENT,
        TOTAL_LOG_USED_KB as USED,
        TOTAL_LOG_AVAILABLE_KB as AVAILABLE,
        TOTAL_LOG_USED_TOP_KB as USED_TOP
from SYSIBMADM.LOG_UTILIZATION
where DB_NAME='DB2_EMP' ;
/WORK # db2 -tf mon_log.db2

UTIL_PERCENT USED                 AVAILABLE            USED_TOP
------------ ---------------- -------------------- --------------------
0.58                      408                68951                  777
1 record(s) selected.
```

- ► Table space list

    - – LIST TABLESPACES

    - – LIST TABLESPACES SHOW DETAIL

    - – db2pd -db <dbname> -tablespaces

    - – SELECT FROM SYSIBMADM.SNAPTBSP_PART OR SYSIBMADM.SNAPCONTAINER

    - – Storage Management wizard

        Control Center → **View Storage menu in Database panel** → **Storage Management** (see Figure 11-15 on page 586).
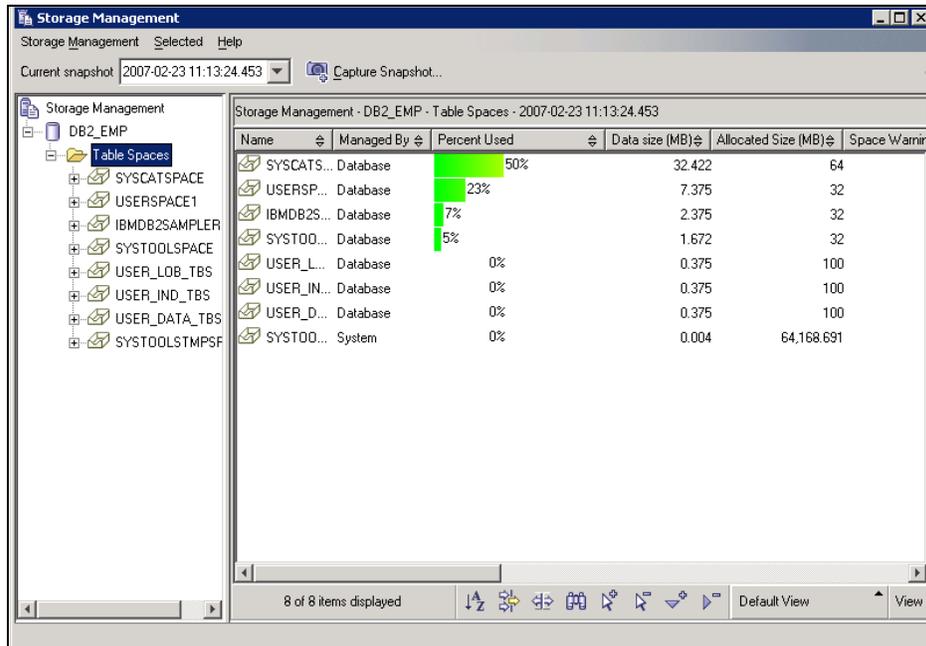
Figure 11-15   Storage Management window

Example 11-73 illustrates how to obtain the table spaces information from the administrative view SYSIBMADM.SNAPCONTAINER.

Example 11-73   View table list and size using view

```
/WORK # cat mon_tbs.db2
select
        substr(TBSP_NAME,1,20) as TBS,
        count(CONTAINER_ID) as NUM_CONTAINER,
        sum(TOTAL_PAGES) as TOTAL,
        sum(USABLE_PAGES) as FREE
from SYSIBMADM.SNAPCONTAINER
group by TBSP_NAME;
/WORK # db2 -tf mon_tbs.db2

TBS                  NUM_CONTAINER TOTAL         FREE
-------------------- ------------- ------------- ---------------
SYSCATSPACE                      1         25600           25568
SYSTOOLSPACE                     1          8192            8188
SYSTOOLSTMPSPACE                 1             0               0
TEMPSPACE1                       1             0               0
USER_DATA_TBS                    1         25600           25568
USER_IND_TBS                     1         25600           25568
USER_LOB_TBS                     1         25600           25568
```

```
USERSPACE1                      1          25600         25568

  8 record(s) selected.
```

► Table space statistics

    – GET SNAPSHOT FOR TABLESPACES FOR <dbname>
    – db2pd -db <dbname> -tablespaces
    – SELECT FROM SYSIBMADM.SNAPTBSP

Example 11-74 shows how to obtain table space snapshot information from
the administrative view SYSIBMADM.SNAPTBSP.

Example 11-74   Get table space information from view

```
/WORK # cat mon_tbsstat.db2
select
        substr(TBSP_NAME,1,15),
        POOL_DATA_L_READS as POOL_DATA_L,
        POOL_DATA_P_READS as POOL_DATA_P,
        DIRECT_READS as DIRECT,
        DIRECT_READ_REQS as DIRECT_REQS
from
        SYSIBMADM.SNAPTBSP;

/WORK # db2 -tf mon_tbsstat.db2

1               POOL_DATA_L   POOL_DATA_P  DIRECT       DIRECT_REQS
--------------- ------------- ------------ ------------ -------------
SYSCATSPACE               193            0           86            10
TEMPSPACE1                  0            0            0             0
USERSPACE1                  0            0            0             0
USER_IND_TBS               24            4            0             0
USER_LOB_TBS                0            0            0             0
USER_DATA_TBS               0            0            0             0
SYSTOOLSPACE                0            0            0             0
SYSTOOLSTMPSPAC             0            0            0             0

  8 record(s) selected.
```

► Table list

    – LIST TABLES FOR ALL
    – db2pd -db <dbname> -tcbstats
    – SELECT FROM SYSIBM.TABLES

Example 11-75 shows how to obtain table information from the DB2 catalog
view.

Example 11-75   Get table

```
/WORK # cat mon_tb.db2
select
        substr(TABLE_SCHEMA,1,15),
        substr(TABLE_NAME,1,15),
        substr(TABLE_TYPE,1,10)
from
        SYSIBM.TABLES
where
        TABLE_CATALOG='DB2_EMP' and TABLE_SCHEMA='DB2INST1';

/WORK # db2 -tf mon_tb.db2

1               2               3
--------------- --------------- ----------
DB2INST1        ENCRYPTIONS     BASE TABLE
DB2INST1        LOCAL_DEPARTMEN BASE TABLE

2 record(s) selected.
```

► Table statistics (read/write, reorg)

   – GET SNAPSHOT FOR TABLES ON <dbname>
   – db2pd -db <dbname> -tcbstats
   – SELECT FROM SYSIBMADM.SNAPTAB, SYSIBMADM.SNAPTAB_REORG

   Example 11-76 illustrates how to obtain table snapshot information.

Example 11-76   Get the table statistics with get snapshot command

```
/WORK # db2 get snapshot for tables on db2_emp

            Table Snapshot

First database connect timestamp    = 02/26/2007 21:26:15.729476
Last reset timestamp                =
Snapshot timestamp                  = 02/28/2007 10:13:02.205265
Database name                       = DB2_EMP
Database path                       = /db2/data/db2inst1/NODE0000/SQL00001/
Input database alias                = DB2_EMP
Number of accessed tables           = 4

Table List
 Table Schema        = SYSIBM
 Table Name          = SYSTABLES
 Table Type          = Catalog
 Data Object Pages   = 40
 Index Object Pages  = 20
```

```
LOB Object pages    = 448
Rows Read           = 1298
Rows Written        = 0
Overflows           = 0
Page Reorgs         = 0


Table Schema        = DB2INST1
Table Name          = LOCAL_DEPARTMENT
Table Type          = User
Data Object Pages   = 1
Rows Read           = 3
Rows Written        = 0
Overflows           = 0
Page Reorgs         = 0


Table Schema        = DB2INST1
Table Name          = ENCRYPTIONS
Table Type          = User
Data Object Pages   = 1
Rows Read           = 5
Rows Written        = 0
Overflows           = 0
Page Reorgs         = 0


Table Schema        = SYSIBM
Table Name          = SYSROUTINES
Table Type          = Catalog
Data Object Pages   = 46
Index Object Pages  = 68
LOB Object pages    = 960
Rows Read           = 4
Rows Written        = 0
Overflows           = 0
Page Reorgs         = 0
```

11.5.3 Applications activity

In this section, we list the commands for monitoring applications:

► List of current application clients

  – `LIST APPLICATIONS` or `list applications show detail`

  – `db2pd -db <dbname> -applications`

  – `SELECT FROM SYSIBMADM.APPLICATIONS`

  – **DB2 Control Center** → **Database Panel** → **Application list (**see
    Figure 11-16 on page 590).

Example 11-77 shows how to list applications connected to the database using the administrative view.

Example 11-77   Get the list of current applications

```
/WORK # cat mon_app.db2
select
        AGENT_ID,
        substr(APPL_NAME,1,15),
        substr(AUTHID,1,15),
        substr(APPL_STATUS,1,10),
        CLIENT_PID
from
        SYSIBMADM.APPLICATIONS
where
        DB_NAME='DB2_EMP';

/WORK # db2 -tf mon_app.db2

AGENT_ID      2               3             4          CLIENT_PID
------------- --------------- ------------- ---------- ------------
         1266 db2bp           DB2INST1      UOWEXEC          503952
           17 db2taskd        DB2INST1      CONNECTED        544866
           16 db2stmm         DB2INST1      UOWWAIT          544866

  3 record(s) selected.
```
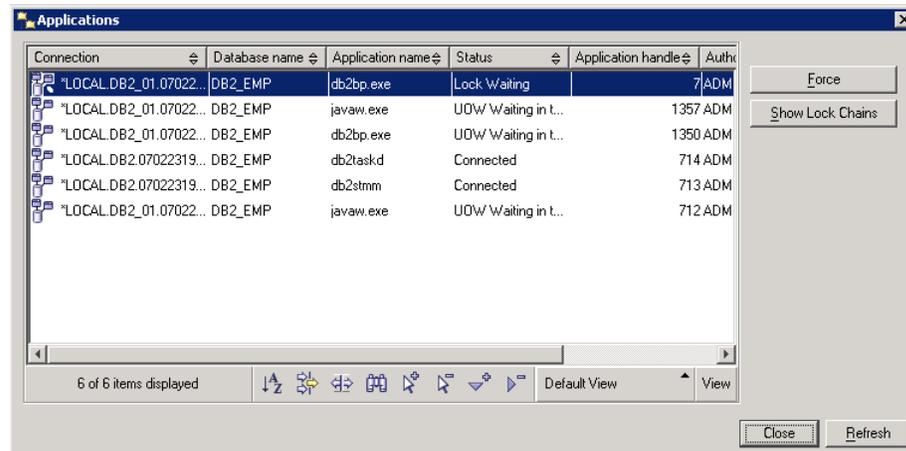

Figure 11-16   Get list of applications in Applications window from DB2 Control Center

► Find the application that holds the oldest transaction

Sometimes your log is full because the oldest transaction did not commit an opened transaction. When you receive the log full message, you can determine which transaction is the oldest by using the following command:

```
GET SNAPSHOT FOR DB ON <dbname>
```

Example 11-78 shows how to find the application that holds the oldest transaction.

Example 11-78   Find out the oldest transaction from DB snapshot

```
/WORK # db2 get snapshot for db on db2_emp |grep -i oldest
Appl id holding the oldest transaction     = 16
```

► Locks in database

- GET SNAPSHOT FOR LOCKS ON <dbname>
- db2pd -db <dbname> -locks
- SELECT FROM SYSIBMADM.SNAPLOCK,SYSIBMADM.LOCKS_HELD
- Activity Monitor

Example 11-79 shows how to obtain lock snapshot information from the administrative view.

Example 11-79   Get lock information using view

```
/WORK # cat mon_lock.db2
select
        AGENT_ID,
        LOCK_OBJECT_TYPE,
        LOCK_MODE,
        LOCK_STATUS,
        substr(TABNAME,1,15)as TABNAME
from SYSIBMADM.SNAPLOCK
;
/WORK # db2 -tf mon_lock.db2

AGENT_ID   LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS TABNAME
---------- ------------------ ---------- ----------- ----------
        66 INTERNALV_LOCK     S          GRNT        -
        66 INTERNALP_LOCK     S          GRNT        -
        66 INTERNALP_LOCK     S          GRNT        -
        41 ROW_LOCK           X          GRNT        EMPLOYEE
        41 INTERNALV_LOCK     S          GRNT        -
        41 INTERNALP_LOCK     S          GRNT        -
        41 INTERNALP_LOCK     S          GRNT        -
        41 TABLE_LOCK         IX         GRNT        EMPLOYEE
        23 ROW_LOCK           X          GRNT        DEPARTMENT
        23 INTERNALP_LOCK     S          GRNT        -
        23 TABLE_LOCK         IX         GRNT        DEPARTMENT
```

```
11 record(s) selected.
```

► Lock wait

- GET SNAPSHOT FOR LOCKS ON <dbname>

- db2pd -db <dbname> -locks

- SELECT FROM SYSIBMADM.LOCKWAITS
  SELECT from SYSIBMADM.SNAPLOCKWAIT

- Activity monitor

Figure 11-17 is a sample application locking report from Activity Monitor.

Figure 11-17   Monitoring locks from Activity Monitor

You can see the lock chain between the lock holder and the lock waiter in Figure 11-18.
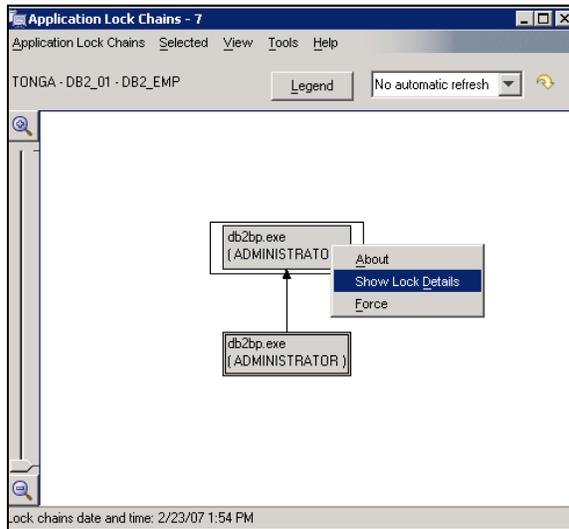
Figure 11-18   Application lock chain

From the Application Lock Chains window, you drill down further to see the lock details, as shown in Figure 11-19.
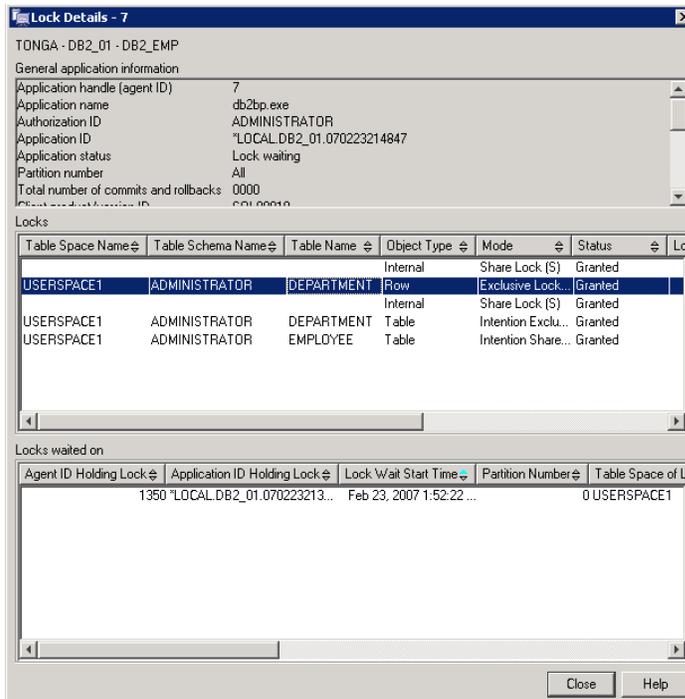
Figure 11-19   Lock details window

▶ SQL statements

  – `GET SNAPSHOT FOR DYNAMIC SQL ON <dbname>`

  – `db2pd -db <dbname> -dynamic` or `db2pd -db <dbname> -static`

  – Select information from the administrative views:

    SYSIBMADM.SNAPAPPL or SYSIBMADM.SNAPDYN_SQL or
    SYSIBMADM.SNAPSTMT or SYSIBMADM.TOP_DYNAMIC_SQL or
    SYSIBMADM.LONG_RUNNING_SQL.

  – Activity monitor

  Example 11-80 shows how to obtain application information from the
  administrative view SYSIBMADM.SNAPAPPL.

Example 11-80   Shows application activities using SYSIBMADM.SNAPAPPL view

```
/WORK # cat mon_appl.db2
select
        AGENT_ID  as ID,
        UOW_LOG_SPACE_USED  as LOG,
        ROWS_READ as  READ,
        ROWS_WRITTEN as  WRITE,
```

```
        LOCKS_HELD as   LOCKS,
        LOCK_WAITS as   LWAIT
from
        SYSIBMADM.SNAPAPPL
where
        DB_NAME='DB2_EMP' ;
/WORK # db2 -tf mon_appl.db2

ID       LOG      READ        WRITE      LOCKS    LWAIT

-------- -------- ----------- ---------- -------- -------
    1087        0         101         51        0        0
    1044        0           3          0        0        0
    1043        0           0          0        0        0
    1042        0          82          0        2        0

  4 record(s) selected.
```

Example 11-81 shows how to find out the number of times the SQL statements are executed.

Example 11-81   Find the most run SQL statement

```
/WORK # cat mon_sql.db2
select
        NUM_EXECUTIONS as NUM_EXEC,
        AVERAGE_EXECUTION_TIME_S as AVG_TIME,
        STMT_SORTS as SORT,
        SORTS_PER_EXECUTION as NUM_SORT,
        substr(STMT_TEXT,1,30)
from
        SYSIBMADM.TOP_DYNAMIC_SQL
order by 1 desc;
/WORK # db2 -tf mon_sql.db2

NUM_EXEC    AVG_TIME  SORT  NUM_SORT  5

----------- --------- ----- --------- ------------------------------
          2         0     0         0 select * from SYSIBMADM.TOP_DY
          2         0     0         0 CALL SYSINSTALLOBJECTS( 'DB2AC
          2         0     0         0 CREATE TABLE PRODUCTS ( PROD_N
          2         0     0         0 select * from products
          2         0     0         0 DECLARE GLOBAL TEMPORARY TABLE
          2         0     0         0 CREATE TABLE PRODUCT_TXS ( TX_
          2         0     0         0 LOCK TABLE SYSTOOLS.HMON_ATM_I
          1         0     0         0 values length(encrypt('IBM Red
          1         0     0         0 CALL SYSINSTALLOBJECTS( 'DB2AC
          1         0     0         0 update employee set firstnme='
          1         0     0         0 set encryption password = 'The
```

```
....

  66 record(s) selected.
```

Data types

This appendix explains data types in different environments:

- ► Supported SQL data types in C/C++
- ► Supported SQL data types in Java
- ► Mapping Oracle data types to DB2 data types

A.1 Supported SQL data types in C/C++

Table A-1 provides a complete list of SQL data types, C and C/C++ data type mapping, and a quick description of each. Note that DB2 has multiple definitions for DATE and multiple types for NUMBER.

For more information about mapping between SQL data types and C and C++ datatypes, refer to the DB2 document *Developing Embedded SQL Applications,* SC10-4232.

Table A-1   Oracle to DB2 data type mapping

	SQL data type sqltype	C/C++ type	sqllen	Description
integer	SMALLINT (500 or 501)	short short int sqlint 16	2	► 16-bit signed integer ► Range between (-32,768 and 32,767) ► Precision of 5 digits
	INTEGER INT (496 or 497)	long long int sqlint32	4	► 32-bit signed integer ► Range between (-2,147,483,648 and 2,147,483,647) ► Precision of 10 digits
	BIGINT (492 or 493)	long long long __int64 sqlint64	8	► 64-bit signed integer
floating point	REAL FLOAT (480 or 481)	float		► Single precision floating point ► 32-bit approximation of a real number ► FLOAT(n) can be synonym for REAL if 0 < n < 25
	DOUBLE (480 or 481) DOUBLE PRECISION	double	8	► Double precision floating point ► 64-bit approximation of a real number ► Range in (0, -1.79769E+308 to -2.225E-307, 2.225E-307 to 1.79769E+308) ► FLOAT(n) can be synonym for DOUBLE if 24 < n < 54

	SQL data type sqltype	C/C++ type	sqllen	Description
decimal	DECIMAL(p,s) DEC(p,s) (484 or 485) NUMERIC(p,s) NUM(p,s)	double / decimal	p/2+1	► Packed decimal ► If precision /scale not specified, default is (5,0) ► Max precision is 31 digits, and max range between (-10E31+1 ... 10E31 -1) ► Consider using char / decimal functions to manipulate packed decimal fields as char data

date / time	DATE (384 or 385)	struct {
		  short len;
		  char data[10];
		} dt;
		
		char dt[11];
			10	► Null-terminated character form (11 characters) or varchar struct form (10 characters);
				► struct can be divided as desired to obtain the individual fields
				► Example: 11/02/2000
				► Stored internally as a packed string of 4 bytes

	TIME (388 or 389)	char	8	► Null-terminated character form (9 characters) or varchar struct form (8 characters); ► struct can be divided as desired to obtain the individual fields ► Example: 19:21:39 Stored internally as a packed string of 3 bytes
	TIMESTAMP (392 or 393)	char	26	► Null-terminated character form (27 characters) or varchar struct form (26 characters); ► struct can be divided as desired to obtain the individual fields ► Example: 2003-08-04-01.02.03.000 000 ► Stored internally as a packed string of 10 bytes

	SQL data type sqltype	C/C++ type	sqllen	Description
character	CHAR (452 or 453)	char	n	► – Fixed-length character string consisting of n bytes ► Use char[n+1] where 1 <= n <= 254 ► If length not specified, defaults to 1
	VARCHAR (460 or 461)	char	n	► Null-terminated variable length character string ► Use char[n+1] where 1 <= n <=32672
	VARCHAR (448 or 449)	struct tag { short int; char[n] }	len	► Non null-terminated varying character string with 2-byte string length indicator ► Use char[n] in struct form where 1<= n <= 32672 ► Default SQL type
	LONG VARCHAR (456 or 457)	struct tag { short int; char[n] }	len	► Non null-terminated varying character string with 2-byte string length indicator ► se char[n] in struct form where 32673<= n <= 32700
	CLOB(n) (408 or 409)	clob	n	► Non null-terminated varying character string with 4-byte string length indicator ► se char[n] in struct form where 1 <= n <= 2147483647
	CLOB (964 or 965)	clob_locator		► Identifies CLOB entities residing on the server
	CLOB (920 or 921)	clob_file		► Descriptor for file containing CLOB data

	SQL data type sqltype	C/C++ type	sqllen	Description
binary	BLOB(n) (404 or 405)	blob	n	► Non null-terminated varying binary string with 4-byte string length indicator ► Use char[n] in struct form where 1 <= n <= 2147483647
	BLOB (960 or 961)	blob_locator		► Identifies BLOB entities on the server
	BLOB (916 or 917)	blob_file		► Descriptor for the file containing BLOB data
double-byte	GRAPHIC(1) GRAPHIC(n) (468 or 469)	sqldbchar	24	► sqldbchar is a single double-byte character string ► For a fixed-length graphic string of length integer which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed. ► Precompiled with WCHARTYPE NOCONVERT option

	VARGRAPHIC(n) (464 or 465)	struct {
		  short int;
		  sqldbchar[n]
		} tag;
		
		alternately:
		sqldbchar[n+ 1]
			n*2+ 4	► For a varying-length graphic string of maximum length integer, which may range from 1 to 16336.
				► Precompiled with WCHARTYPE NOCONVERT option.
				► Null terminated variable-length

	LONG VARGRAPHIC(n) (472 or 473)	struct {
		  short int;
		  sqldbchar[n]
		} tag;
				► For a varying-length graphic string with a maximum length of 16350 and a 2-byte string length indicator 16337<=n <=16350
				► Precompiled with WCHARTYPE NOCONVERT option

	SQL data type sqltype	C/C++ type	sqllen	Description
	DBCLOB(n) (412 or 413)	dbclob		► For non-null-terminated varying double-byte character large object maximum length in double-byte characters. ► 4 bytes string length indicator ► Use dbclob(n) where 1<=n <= 1073741823 double-byte characters. ► Precompiled with WCHARTYPE NOCONVERT option
	DBCLOB	dbclob_locator		► Identifies DBCLOB entities residing on the server ► Precompiled with WCHARTYPE NOCONVERT option
	DBCLOB	dbclob_file		► Descriptor for file containing DBCLOB data ► Precompiled with WCHARTYPE NOCONVERT option
external data	Datalink(n)		n+54	► The length of a DATALINK column is 200 bytes
	XML (988 or 989)	struct { sqluint32 length; char data[n]; }		► XML value

A.2 Supported SQL data types in Java

Table A-2 shows the Java equivalent of each SQL data type, based on the JDBC specification for data type mappings. The JDBC driver converts the data exchanged between the application and the database using the following mapping schema. Use these mappings in your Java applications and your PARAMETER STYLE JAVA procedures and UDFs.

For more information about mapping between SQL data types and C and C++ datatypes, refer to the DB2 document *Developing Embedded SQL Applications,* SC10-4232.

Table A-2   SQL data types mapped to Java declarations

	SQL data type sqltype	Java type	sqllen	Description
integer	SMALLINT (500 or 501)	short	2	16-bit, signed integer
	INTEGER (496 or 497)	int	4	32-bit, signed integer
	BIGINT [1] (492 or 493)	long	8	64-bit, signed integer
floating point	REAL (480 or 481)	float		Single precision floating point
	DOUBLE (480 or 481)	double	4	Single precision floating point
	DOUBLE (480 or 481)	double	8	Double precision floating point
decimal	DECIMAL(p,s) (484 or 485)	java.math. BigDecimal	n/2	Packed decimal
date / time	DATE (384 or 385)	java.sql.Date	10	10-byte character string
	TIME (388 or 389)	java.sql.Time	8	8-byte character string
	TIMESTAMP (392 or 393)	java.sql. Timestamp	26	26-byte character string
character	CHAR (452 or 453)	java.lang.String	n	Fixed-length character string of length n where n is from 1 to 254
	CHAR FOR BIT DATA	byte[]		Fixed-length character string of length n where n is from 1 to 254
	VARCHAR (448 or 449)	java.lang.String	n	Variable-length character string, n <= 32672

	SQL data type sqltype	Java type	sqllen	Description
	VARCHAR FOR BIT DATA	byte[]		Variable-length character string
	LONG VARCHAR (456 or 457)	java.lang.String	n	Long variable-length character string, n <= 32672
	CLOB(n) (408 or 409)	java.lang.Clob	n	Large object variable-length character string
binary	BLOB(n) (404 or 405)	java.lang.Blob	n	Large object variable-length binary string
double-byte	GRAPHIC(n) (468 or 469)	java.lang.String	n	Fixed-length double-byte character string
	VARGRAPHIC(n) (464 or 465)	java.lang.String	n*2+4	Non-null-terminated varying double-byte character string with 2-byte string length indicator
	LONG VARGRAPHIC(n) (472 or 473)	java.lang.String	n	Non-null-terminated varying double-byte character string with 2-byte string length indicator
	DBCLOB(n) (412 or 413)	java.lang.Clob	n	Large object variable-length double-byte character string
	CLOB(n) (408 or 409)	java.sql.Clob	n	Non null-terminated varying character string with 4-byte string length indicator
binary	BLOB(n) (404 or 405)	java.sql.Blob	n	Non null-terminated varying binary string with 4-byte string length indicator
	XML	com.ibm.db2.jcc.DB2Xml		XML value

A.3 Mapping Oracle data types to DB2 data types

Table A-3 summarizes the mapping from Oracle data types to corresponding DB2 data types. The mapping is one to many and depends on the actual usage of the data.

Table A-3   Mapping Oracle data types to DB2 data types

Oracle data type	DB2 data type	Notes
CHAR(n)	CHAR(n)	1 <= n <= 254
VARCHAR2(n)	VARCHAR(n)	n <= 32762
NCHAR(n)	CHAR(n) [a]	1 <= n <= 254
NVARCHAR2(n)	VARCHAR(n) [a]	n <= 32762
LONG	LONG VARCHAR(n)	if n <= 32700 bytes
LONG	CLOB(2GB)	if n <= 2 GB
NUMBER(p)	SMALLINT / INTEGER / BIGINT	▶ SMALLINT, if 1 <= p <= 4 ▶ INTEGER, if 5 <= p <= 9 ▶ BIGINT, if 10 <= p <= 18
NUMBER(p,s)	DECIMAL(p,s)	if s > 0
NUMBER	FLOAT / REAL / DOUBLE	
RAW(n)	CHAR(n) FOR BIT DATA / VARCHAR(n) FOR BIT DATA BLOB(n)	▶ CHAR, if n <= 254 ▶ VARCHAR, if 254 < n <= 32672 ▶ BLOB, if 32672 < n <= 2 GB
LONG RAW	LONG VARCHAR(n) FOR BIT DATA / BLOB(n)	▶ LONG, if n <= 32700 ▶ BLOB, if 32700 < n <= 2GB
BLOB	BLOB(n)	if n <= 2 GB
CLOB	CLOB(n)	if n <= 2 GB
NCLOB	DBCLOB(n)	if n <= 2 GB, use DBCLOB(n/2)

Oracle data type	DB2 data type	Notes
DATE	TIMESTAMP	► Use Oracle TO_CHAR() function to extract for subsequent DB2 load. ► Oracle default format is DD-MON-YY
DATE (only the date)	DATE (MM/DD/YYYY)	Use Oracle TO_CHAR() function to extract data for subsequent DB2 load.
DATE (only the time)	TIME (HH24:MI:SS)	Use Oracle TO_CHAR() function to extract for subsequent DB2 load.
TIMESTAMP	TIMESTAMP	
XMLType	XML	

a. You can map Oracle NCHAR and NVARCHAR2 columns to DB2 CHAR and VARCHAR columns created in an Unicode database or column created with CCSID clause to an Unicode compatible code set.

Terminology mapping

This appendix contains the terminology mapping of Oracle to DB2.

Table B-1   Oracle terminology to DB2 mapping

Oracle	DB2	Comments
Oracle EE	DB2 Enterprise 9	Enterprise product
Oracle Parallel	DB2 Enterprise DPF	Support node partitioning
Oracle Gateway	DB2 Connect	DRDA® access to hosts
PL/SQL	SQL Procedural Language	Programming language extension to SQL. DB2 stored procedures can be programmed in SQL Control Statements (subset of PSM standard), Java, C, C++, COBOL, Fortran, OLE, and REXX™. DB2 functions can be programmed in Java, C, C++, OLE, or SQL control statements.
SQL*PLUS	DB2 CLP	Command line interface to the server

Oracle	DB2	Comments
Instance	Instance	Processes and shared memory. In DB2 it also includes a permanent directory structure: an instance is usually created at install time (or can be later) and must exist before a database can be created. A DB2 instance is also known as the *database manager* (DBM). A DB2 instance can have multiple databases. But an Oracle instance can only have one database.
Database	Database	Physical structure containing data. In Oracle, multiple instances can use the same database, and an instance can connect to one and only one database. In DB2, multiple databases can be created and used concurrently in the same instance.
Control files and .ora files	DBM and database configuration files, etc.	In Oracle, files that name the locations of files making up the database and provide configuration values. In DB2, each instance (DBM) and database has its own set of configuration parameters stored in a binary file; there are also other internal files and directories: none is manually edited.
Database Link	Federated System	In Oracle, an object that describes a path from one database to another. In DB2 a federated system is used. One database is chosen as the federated database and within it wrappers, servers, nicknames, and other optional objects are created to define how to access the other databases (including Oracle databases) and objects in them. Once an application is connected to the federated database it can access all authorized objects in the federated system.
Table spaces	Table spaces	Contains actual database data
Datafiles	Containers	Entities inside the table spaces
Segments	Objects	Entities inside the containers/data files
Extents	Extents	Entities inside the objects/segments
Data blocks	Pages	Smallest storage entity in the storage model

Oracle	DB2	Comments
Clusters	N/A	Data structure that allows related data to be stored together on disk; can be table or hash clusters. The closest facility to this in DB2 is a *clustering index*, which causes rows inserted into a table to be placed physically close to the rows for which the key values of this index are in the same range.
Data dictionary	System catalog	Metadata of the database
N/A	SMS	System-managed table space
Datafiles	DMS containers	The file and raw devices under Database-managed table space.
Data cache	Buffer pools	Buffers data in the table spaces to reduce disk I/O
Statement cache	Package cache	Caches prepared dynamic SQL statements
Redo logs	Log files	Recovery logs
Rollback segments	N/A	Store the old version of data for a mutating table. In DB2 the old version of an updated row is stored in the log file along with the new version.
SGA	Database manager and database shared memory	Shared memory area(s) for the database server. In Oracle there is one, while in DB2 there is one at the database manager (instance) level any one for each active database.
UGA	Agent / application shared memory	Shared memory area to store user-specific data passed between application process and the database server.
N/A	Package	A precompiled access plan for an embedded static SQL application stored in the server.
Package	N/A	A logical grouping of PL/SQL blocks that can be invoked by other PL/SQL applications.

C
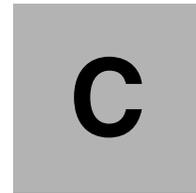
Function mapping

This appendix contains the function mapping of Oracle to DB2. The function mapping covered includes the following:

► Numeric function mapping
► Character function mapping
► Date and time function mapping
► Conversion and cast function mapping
► Aggregate function mapping
► Comparison and NULL-related function mapping
► Encoding, decoding, encryption, and decryption function mapping

For more DB2 user defined function (UDF) samples to replace Oracle functions, refer to the following Web site:

http://www.ibm.com/developerworks/db2/library/samples/db2/0205udfs/

C.1 Numeric function mapping

Table C-1 shows the numeric function mapping between Oracle and DB2.

Table C-1   Numeric functions

Oracle	DB2	Comments
ABS	ABS	Returns the absolute value.
ACOS	ACOS	Returns the arc cosine.
ASIN	ASIN	Returns the arc sine.
ATAN	ATAN	Returns the arc tangent.
ATAN2	ATAN2	Returns the arc tangent (two value).
BITAND	N/A. Implemented by UDF.	Returns the bit-wise AND of two non-negative integers. Refer to Example C-2 on page 614 for sample UDF code.
CEIL	CEIL CEILING	Returns the smallest integer greater or equal to the argument.
COS	COS	Returns the cosine.
COSH	COSH	Returns the hyperbolic cosine.
N/A	COT	Returns the cotangent.
N/A	DEGREES	Returns the number of degrees of an angle.
N/A	DIGITS	Returns a character-string representation of the absolute value of a number.
EXP	EXP	Returns the exponential function of the argument.
FLOOR	FLOOR	Returns the largest integer less or equal to the argument.
LN	LN	Returns the natural logarithm.
LOG	LOG	Returns the natural logarithm. Oracle LOG(n2,n1) is equivalent to DB2 LOG(n1)/LOG(n2). An UDF for LOG(n2,n1) is provided in Example C-1 on page 614.
LOG(10,n1)	LOG10(n1)	Returns the common logarithm (base 10).

Oracle	DB2	Comments
MOD	MOD	Returns the remainder of the first argument divided by the second argument.
N/A	MULTIPLY_ALT	Returns the product of two arguments as a decimal value.
NANVL	N/A	Returns an alternative value n1 if the input value n2 is NaN (not a number). If n2 is not NaN, then Oracle Returns n2.
POWER	POWER	Returns the result of raising the first argument to the power of the second argument.
REMAINDER	N/A	The REMAINDER function is similar to MOD function except that it uses ROUND in its formula, whereas MOD uses FLOOR.
N/A	RADIANS	Returns the number of radians for the argument that is expressed in degrees.
N/A	RAND	Returns a random number.
ROUND(arg1,arg2)	ROUND(arg1,arg2)	Rounds a value of the first argument to the number of decimal places specified by the second argument.
SIGN	SIGN	Returns: 1 when number is negative, 0 when number is zero, or 1 when number is positive.
SIN	SIN	Returns the sine.
SINH	SINH	Returns the hyperbolic sine.
SQRT	SQRT	Returns the square root.
TAN	TAN	Returns the tangent.
TANH	TANH	Returns the hyperbolic tangent.
TRUNC(n[,m])	TRUNC(n[,m]) TRUNCATE(n[,m])	Returns the truncated number n to the number of decimal places specified by m.

Oracle	DB2	Comments
WIDTH_BUCKET	N/A. Implemented by UDF.	WIDTH_BUCKET lets you construct equiwidth histograms, in which the histogram range is divided into intervals that have identical size. The UDF is avaialbe for download. See Appendix G, "Additional material" on page 701 for the download instructions.

Example C-1shows the DB2 code of LOG function.

Example: C-1   DB2 sample UDF - LOG(n2,n1)

```
CREATE FUNCTION ora_log(n1 float, n2 float)
RETURNS FLOAT
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
RETURN
    log(n2)/log(n1);
```

Example C-2 shows the DB2 code of BIT_AND function and its usage.

Example: C-2   DB2 sample UDF - BIT_AND

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created: 2004/03/06
--
-- Name of UDF: BIT_AND (N1 Integer, N2 Integer)
--
-- Used UDF: None
--
-- Description: Returns bit by bit AND of both parameters.
--             Arguments should be non negative.
--
-- Author: TOKUNAGA, Takashi
--
--------------------------------------------------------------------
CREATE FUNCTION BITAND (N1 Integer, N2 Integer)
 RETURNS Integer
 LANGUAGE SQL
 SPECIFIC BITANDOracle
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
```

```
BEGIN ATOMIC
DECLARE M1, M2, S Integer;
DECLARE RetVal Integer DEFAULT 0;

SET (M1, M2, S) = (N1, N2, 0);
WHILE M1 > 0 AND M2 > 0 AND S < 32 DO
    SET RetVal = RetVal + MOD(M1,2)*MOD(M2,2)*power(2,S);
    SET (M1, M2, S) = (M1/2, M2/2, S+1);
END WHILE;

RETURN RetVal;
END!
-------------------------------------------------------------------
--
-- Usage samples
--
--------------------- Command Entered ---------------------------

VALUES BITAND(10,8)
!
-------------------------------------------------------------------

1
-----------
          8

  1 record(s) selected.

--------------------- Command Entered ---------------------------

VALUES BITAND(1038,77)
!
-------------------------------------------------------------------

1
-----------
         12

  1 record(s) selected.
```

C.2 Character function mapping

Table C-2 shows the character function mapping between Oracle and DB2.

Table C-2   Character functions

Oracle	DB2	Comments
ASCII	ASCII	Returns the decimal representation of a character.
CHR(n)	CHR(n)	Returns an ASCII code that represents n. An UDF CHR that takes a binary number as input and return the equivalent ASCII value is provided in Example C-3 on page 619.
CHR (n USING NCHAR_CS)	N/A	Returns the character having the binary equivalent to n as a VARCHAR2 value in the national character set.
CONCAT	CONCAT	Returns the concatenation of two strings. A CONCAT(‖) UDF that supports various combination of data types is available for download. See Appendix G, "Additional material" on page 701 for the download instructions.
N/A	DIFFERENCE	Returns a value from 0 to 4 representing the difference between the sound of two strings based on applying the SOUNDEX function to the strings.
N/A	GENERATE_UNIQUE	Returns a bit data character string that is unique compared to any other execution of the same function.
INITCAP	N/A. Implemented by UDF.	Returns characters with the first letter of each word in uppercase and the reset of letters in lowercase. refer to Example C-4 on page 620 for sample code
N/A	INSERT(arg1, arg2,arg3,arg4)	Returns a string where arg3 bytes have been deleted from arg1, beginning at arg2, and where arg4 has been inserted into arg1, beginning at arg2.
INSTR	POSSTR POSITION LOCATE	Returns an integer indicating the position of the character in string that is the first character of this occurrence.

Oracle	DB2	Comments
INSTRB INSTRC INSTR2 INSTR4	N/A. Implemented by UDF.	Returns an integer indicating the position of the character in different character sets. Refer to Example C-5 on page 621 for sample code.
LENGTH	LENGTH	Returns a length of a value.
LENGTHB LENGTHC LENGTH2 LENGTH4	N/A	Returns the length of a character in different character sets.
LOWER	LOWER LCASE	Returns the lower case of a character string.
LPAD(arg1,arg2, arg3)	N/A. Implemented by UDF.	Returns arg1, left-padded to length arg2 characters with the sequence of characters in arg3. Refer to Example C-7 on page 624 for sample code. In addition, we provide three variables of LPAD that allows you to pad a character string to an intriguer or pad, by default, blanks to the first argument. See Example C-8 on page 625 and Example C-9 on page 626. LPAD(exp1,n) is equivalent to DB2 RIGHT(exp1,n)
LTRIM	LTRIM	Removes blanks from the beginning of a string expression. An UDF that can remove a set of characters is availabe for download. See Appendix G, "Additional material" on page 701 for the download instructions.
NLS_CHARSET_ DECL_LEN NLS_CHARSET_ ID NLS_CHARSET_ NAME NLS_INITCAP NLS_LOWER NLS_UPPER NLSSORT	N/A	NLS functions used in Oracle NCHAR data type. In DB2, you can use functions of the appropriate data type.

Oracle	DB2	Comments
N/A	OCTET_LENGTH	Returns the length of an expression in octets (bytes)
N/A	REPEAT(arg1,arg2)	Returns a character string composed of arg1 repeated arg2 times
REPLACE(arg1,arg2,arg3)	REPLACE(arg1,arg2,arg3)	replaces all occurrences of srg2 in arg1 with arg3
N/A	RIGHT(arg1,arg2)	Returns a string consisting of the right most arg2 bytes in arg1.
RPAD	N/A. Implemented by UDF.	Returns the first argument value, right-padded to the length specified in the 2nd argument with characters specified in the third argument. Refer to Example C-10 on page 628 for sample code. In addition, we provide two the variations of RPAD that pad blank by default or pad a string to an integer. See Example C-11 on page 629 and Example C-12 on page 630. RPAD('',n) is equivalent to DB2 SPACE(n). RPAD(exp1,n) is equivalent to DB2 LEFT(exp1,n).
RTRIM	RTRIM	Remove blanks from the end of a string expression. An UDF that can remove a set of characters is availabe for download. See Appendix G, "Additional material" on page 701 for the download instructions.
SOUNDEX	SOUNDEX	Returns a 4-character code representing the sound of the argument.
N/A	SPACE	Returns a character string that consists of a specified number of blanks.
SUBSTR	SUBSTR SUBSTRING	Returns a substring of a string.

Oracle	DB2	Comments
SUBSTRB SUBSTRC SUBSTR2 SUBSTR4	N/A	Returns a substring of a string.
TRANSLATE	TRANSLATE	Returns a string in which one or more characters in a string are converted to other characters.
TREAT	implement by CAST	Changes the declared type of an expression.
TRIM	TRIM STRIP	Removes leading or trailing blanks or other specified leading or trailing characters from a string expression.
UPPER	UPPER UCASE	Returns a string in which all the characters have been converted to uppercase characters.

Example C-3 shows the DB2 UDF for CHR function. Different from Oracle, DB2 CHR(0) returns x'20' instead of x'00'. Oracle CHR(n) accept n that is more than 255, however, DB2 CHR(n) where n>255 returns 0. To avoid changing source code that uses the CHR function, the UDF provided in Example C-3 can be used.

Example: C-3   DB2 sample UDF - CHR

```
CREATE FUNCTION MyOra.CHR (N FLOAT)
 RETURNS CHAR(1)
 SPECIFIC CHRFloat
 LANGUAGE SQL
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
RETURN COALESCE(SYSFUN.CHR(NULLIF(MOD(INT(N),256),0)),x'00')
;

CREATE FUNCTION MyOra.CHR (N INTEGER)
 RETURNS CHAR(1)
 SPECIFIC CHRInteger
 LANGUAGE SQL
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
RETURN COALESCE(SYSFUN.CHR(NULLIF(MOD(N,256),0)),x'00')
;
```

Example C-4 shows the DB2 code of the INITCAP function and its usage.

Example: C-4   DB2 sample UDF - INITCAP

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created: 2004/03/06
--
-- Name of UDF: INITCAP (C1 VarChar(4000))
--
-- Used UDF: None
--
-- Description: Convert first character of each word to uppercase
--              and other characters to lowercase.
--              Words are separated by non-alphanumeric character(s).
--
-- Author: TOKUNAGA, Takashi
----------------------------------------------------------------------
CREATE FUNCTION INITCAP (C1 VarChar(4000))
 RETURNS VarChar(4000)
 SPECIFIC INITCAPOracle
 LANGUAGE SQL
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
BEGIN ATOMIC
DECLARE AN Char(62) DEFAULT
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
DECLARE C1L Integer;
DECLARE Pos, Flag, NewFlag Integer;
DECLARE RetVal VarChar(4000);

SET C1L = LENGTH(C1);

SET (Pos, Flag, RetVal) = (1, 0, '');
WHILE Pos <= C1L DO
   SET NewFlag = SIGN(LOCATE(SUBSTR(C1,Pos,1),AN));
   SET RetVal = RetVal
                || CASE
                   WHEN Flag = 0 AND NewFlag = 1 THEN UPPER(SUBSTR(C1,Pos,1))
                   WHEN Flag = 1 AND NewFlag = 1 THEN LOWER(SUBSTR(C1,Pos,1))
                   ELSE SUBSTR(C1,Pos,1)
                   END;
   SET (Pos, Flag) = (Pos + 1, NewFlag);
END WHILE;

RETURN RetVal;
END!
```

```
--------------------------------------------------
--
-- Usage sample
--
--------------------- Command Entered --------------------------
VALUES CHAR(INITCAP('arnold talked a bear***story to maryANN, MacDonald and
jean-POLE.'),80)
!
------------------------------------------------------------------

1
------------------------------------------------------------------
Arnold Talked A Bear***Story To Maryann, Macdonald And Jean-Pole.

  1 record(s) selected.
```

Example C-5 shows the DB2 code of INSTRB function and its usage.

Example: C-5   DB2 sample UDF - INSTRB

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created: 2004/03/06
--
-- Name of UDF: INSTRB (C1 VarChar(4000), C2 VarChar(4000), N integer,
                        M integer)
--
-- Used UDF: None
--
-- Description: Return the position of char2 in char1.
--             Search will be started from position N.
--             Same function as DB2 LOCATE function.
--             Add some specifications that are not in LOCATE
               function.
--             Support for Negative value of parameter N
               (start search from end of char1)
--             Add parameter M (number of occurrence of char2)
--
-- Author: TOKUNAGA, Takashi
--
------------------------------------------------------------------
CREATE FUNCTION INSTRB (C1 VarChar(4000), C2 VarChar(4000), N integer, M
integer)
 RETURNS Integer
 SPECIFIC INSTRBOracleBase
 LANGUAGE SQL
 CONTAINS SQL
```

```
 NO EXTERNAL ACTION
 DETERMINISTIC
BEGIN ATOMIC
DECLARE Pos, R, C2L Integer;

SET C2L = LENGTH(C2);

IF N > 0 THEN
   SET (Pos, R) = (N, 0);
   WHILE R < M AND Pos > 0 DO
      SET Pos = LOCATE(C2,C1,Pos);
         IF Pos > 0 THEN
         SET (Pos, R) = (Pos + 1, R + 1);
      END IF;
   END WHILE;

   RETURN (Pos - 1)*(1-SIGN(M-R));
ELSE
   SET (Pos, R) = (LENGTH(C1)+N, 0);
   WHILE R < M AND Pos > 0 DO
      IF SUBSTR(C1,Pos,C2L) = C2 THEN
         SET R = R + 1;
      END IF;
      SET Pos = Pos - 1;
   END WHILE;

   RETURN (Pos + 1)*(1-SIGN(M-R));
END IF;

END!
----------------------------------------------------


--
-- Usage samples
--
--------------------- Command Entered ---------------------------
VALUES INSTRB('corporate floor','or',3,2)
!
----------------------------------------------------

1
-----------
        14

   1 record(s) selected.


--------------------- Command Entered ---------------------------
VALUES INSTRB('corporate floor','or',-3,2)
```

```
!
---------------------------------------------------

1
-----------
         2

    1 record(s) selected.



--------------------- Command Entered ---------------------------
VALUES INSTRB('corporate floor','or',-3,3)
!
---------------------------------------------------

1
-----------
         0

    1 record(s) selected.
```

Example C-6 shows a variation of INSTRB DB2 code that has one parameter omitted.

Example: C-6   DB2 sample UDF - INSTRB with one parameter omitted

```
-- This INSTRB code omits parameter m

----------------------------------------------------------------------
CREATE FUNCTION INSTRB (C1 VarChar(4000), C2 VarChar(4000), N integer)
 RETURNS Integer
 SPECIFIC INSTRBOracleParm3
 LANGUAGE SQL
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
RETURN
INSTRB(C1, C2, N, 1)
!
---------------------------------------------------

--
-- Usage sample
--
--------------------- Command Entered ---------------------------
```

```
VALUES INSTRB('corporate floor','or',3)
!
--------------------------------------------------------------------

1
-----------
          5

  1 record(s) selected.
```

Example C-7 shows the DB2 UDF code of LPAD function and its usage.

Example: C-7   DB2 sample UDF - LPAD.

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/08/27, 09/27, 11/06
--
-- Name of UDFs: LPAD (C1 VarChar(4000), N integer, C2 VarChar(4000))
--               LPAD (I1 Integer,       N integer, C2 Varchar(4000))
--
-- Used UDF: None
--
-- Description: Add repeatedly C2 to the left of parameter 1 (C1 or I1)
--              and return N byte.
--
-- Author: TOKUNAGA, Takashi

----------------------------------------------------------------------
CREATE FUNCTION LPAD (C1 VarChar(4000), N integer, C2 VarChar(4000))
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC LPADBase
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
 CASE
   WHEN N > length(C1) THEN
substr(repeat(C2,(N-length(C1)+length(C2))/(length(C2)+1-sign(length(C2)))),1,N
-length(C1)) || C1
   ELSE substr(C1,1,N)
 END
;
---------------------------------------------------
--
-- Usage samples
--
```

```
-------------------------------------------------------------------
SELECT char(lpad('ABCDE',15,'*.'),50) FROM SYSIBM.SYSDUMMY1;
---------------------------------------------------

1
---------------------------------------------------
*.*.*.*.*.ABCDE

    1 record(s) selected.


-------------------------------------------------------------------
SELECT char(lpad('ABCDE',3,'*.'),50) FROM SYSIBM.SYSDUMMY1;
---------------------------------------------------

1
---------------------------------------------------
ABC

    1 record(s) selected.



-------------------------------------------------------------------
SELECT char(lpad('ABCDE',15,'') || 'X',50) FROM SYSIBM.SYSDUMMY1;
---------------------------------------------------

1
---------------------------------------------------
         ABCDEX

    1 record(s) selected.
```

Example C-8 shows a simplified variation of LPAD that pads blanks to the left of the value specified in the first argument to the length specified in the second argument.

Example: C-8   DB2 sample UDF LPAD variation - pad blank to the left

```
--
LPAD - Omit the 3rd parameter. Blank is padded to the left.

-------------------------------------------------------------------
CREATE FUNCTION LPAD (C1 VarChar(4000), N integer)
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC LPADParm2
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
```

```
 RETURN
 LPAD(C1,N,' ')
;
---------------------------------------------------
--
--
-- Usage samples
--
---------------------------------------------------------------------
SELECT char(lpad('ABCDE',15),20) FROM SYSIBM.SYSDUMMY1;
---------------------------------------------------

1
-------------------
          ABCDE

    1 record(s) selected.


---------------------------------------------------------------------
SELECT char(lpad('ABCDE',3),20) FROM SYSIBM.SYSDUMMY1;
---------------------------------------------------

1
-------------------
ABC

    1 record(s) selected.
```

Example C-9 shows two variations of DB2 UDF LPAD. These functions allow you to pad a character string to the left of an integer. The difference is that the second example pads, as default, a blank to an integer.

Example: C-9   DB2 sample UDF LPAD variation - pad string to the left of an integer

```
--
-- LPAD : The 1st parameter is integer.
--
-- (1) The character string to be padded is specified in
--     the third argument.
-- ----------------------------------------------------------------
CREATE FUNCTION LPAD (I1 Integer, N integer, C2 Varchar(4000))
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC LPADIntParm3
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
```

```
    LPAD(rtrim(char(I1)),N,C2)
;
--------------------------------------------------
--
--
-- Usage samples
--

-------------------------------------------------------------------
SELECT char(lpad(9021,15,'*.'),50) FROM SYSIBM.SYSDUMMY1;
--------------------------------------------------

1
--------------------------------------------------
*.*.*.*.*.*9021

    1 record(s) selected.


-------------------------------------------------------------------
SELECT char(lpad(9021,15,''),50) FROM SYSIBM.SYSDUMMY1;
--------------------------------------------------

1
--------------------------------------------------
          9021

    1 record(s) selected.


-------------------------------------------------------------------
--
-- (2) Pad blank to the left of an integer
--
-------------------------------------------------------------------
CREATE FUNCTION LPAD (I1 Integer, N integer)
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC LPADIntParm2
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
 LPAD(rtrim(char(I1)),N,' ')
;
--------------------------------------------------

--
-- Usage sample
--
```

```
-------------------------------------------------------------------
SELECT char(lpad(9021,3),20) FROM SYSIBM.SYSDUMMY1;
-------------------------------------------------

1
-------------------
902

    1 record(s) selected.
```

Example C-10 shows the DB2 UDF code of RPAD and its usage.

Example: C-10   DB2 sample UDF - RPAD

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/08/27, 09/27, 11/06
--
-- Name of UDFs: RPAD (C1 VarChar(4000), N integer, C2 VarChar(4000))
--               RPAD (I1 Integer,       N integer, C2 Varchar(4000))
--
-- Used UDF: None
--
-- Description: Add repeatedly C2 to the right(RPAD) of
--              parameter 1 (C1 or I1) and return N byte.
--
-- Author: TOKUNAGA, Takashi
--


-------------------------------------------------------------------
CREATE FUNCTION RPAD (C1 VarChar(4000), N integer, C2 VarChar(4000))
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC RPADBase
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
 substr(C1 ||
repeat(C2,((sign(N-length(C1))+1)/2)*(N-length(C1)+length(C2))/(length(C2)+1-si
gn(length(C2)))),1,N)
;
-------------------------------------------------

--
-- Usage samples
--
-------------------------------------------------------------------
```

```
SELECT char(rpad('ABCDE',12,'*.'),20) FROM SYSIBM.SYSDUMMY1;
--------------------------------------------------

1
--------------------
ABCDE*.*.*.*

    1 record(s) selected.



----------------------------------------------------------------------
SELECT char(rpad('ABCDE',3,'*.'),20) FROM SYSIBM.SYSDUMMY1;
--------------------------------------------------

1
--------------------
ABC

    1 record(s) selected.



----------------------------------------------------------------------
SELECT char(rpad('ABCDE',20,'') || 'X',50) FROM SYSIBM.SYSDUMMY1;
--------------------------------------------------

1
--------------------------------------------------
ABCDE               X

    1 record(s) selected.
```

Example C-11 shows two variations of DB2 UDF LPAD. These functions allow
you to pad a character string to the left of an integer. The difference is that the
second example pads, as default, a blank to an integer.

Example: C-11   DB2 sample UDF - RPAD variation - pad with blank

```
--
-- RPAD - omits the 3rd parameter, pads blank instead.

----------------------------------------------------------------------
CREATE FUNCTION RPAD (C1 VarChar(4000), N integer)
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC RPADVarCharParm2
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
```

```
 RETURN
 RPAD(C1,N,' ')
;
----------------------------------------------------

--
-- Usage sample
--

----------------------------------------------------------------------
SELECT char(rpad('ABCDE',15) || 'X',50) FROM SYSIBM.SYSDUMMY1;
----------------------------------------------------

1
----------------------------------------------------
ABCDE          X

     1 record(s) selected.



----------------------------------------------------------------------
SELECT char(rpad('ABCDE',3) || 'X',50) FROM SYSIBM.SYSDUMMY1;
----------------------------------------------------

1
----------------------------------------------------
ABCX

     1 record(s) selected.
```

Example C-12 shows two variations of RPAD that allow you to pad a character string to the right of an integer. The first one requires three arguments. The third argument is the character string to be padded to the right of the integer specified in the first argument. The second code example is a simplified version of the first one. It takes only two arguments and will pad a blank to the right of the integer specified in the first argument. The usage samples are included.

Example: C-12   DB2 sample UDF RPAD variation - pad character string to an integer

```
--
-- RPAD variation - The 1st parameter is integer.
--
-- (1) Three arguments are required. The third argument is
--      the character string to be padded to the right of the
--      integer specified in the first argument.
--
----------------------------------------------------------------------
```

```
CREATE FUNCTION RPAD (I1 Integer, N integer, C2 Varchar(4000))
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC RPADIntParm3
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
 RPAD(rtrim(char(I1)),N,C2)
;
---------------------------------------------------
--
-- Usage samples
--

-----------------------------------------------------------------
SELECT char(rpad(927,12,'*.'),50) FROM SYSIBM.SYSDUMMY1;
---------------------------------------------------

1
--------------------------------------------------
927*.*.*.*.*

    1 record(s) selected.


-----------------------------------------------------------------
SELECT char(rpad(927,12,'') || 'X',50) FROM SYSIBM.SYSDUMMY1;
--------------------------------------------------

1
--------------------------------------------------
927         X

    1 record(s) selected.


-----------------------------------------------------------------
--
-- (2) RPAD - pad blank to the right of an integer
--
-----------------------------------------------------------------
CREATE FUNCTION RPAD (I1 Integer, N integer)
 RETURNS VARCHAR(4000)
 LANGUAGE SQL
 SPECIFIC RPADIntParm2
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
```

```
 RPAD(rtrim(char(I1)),N,' ')
;
---------------------------------------------------

--
-- Usage samples
--
-------------------------------------------------------------------
SELECT char(rpad(9021,3),20) FROM SYSIBM.SYSDUMMY1;
---------------------------------------------------

1
-------------------
902

    1 record(s) selected.
```

C.3 Date and time function mapping

Table C-3 shows the date and time function mapping between Oracle and DB2.

Table C-3   Date and time functions

Oracle	DB2	Comments
ADD_MONTHS	N/A. Implemented by UDF.	Returns the date argument plus integer months. Refer to Example C-13 on page 634 for sample code.
CURRENT_DATE	CURRENT DATE	Returns the current date.
CURRENT_TIME STAMP	CURRENT TIMESTAMP	Returns the current date and time.
N/A	DAYNAME	Returns a mixed case character string containing the name of the day.
N/A	DAYOFWEEK	Returns the day of the week from a value, where 1 is Sunday and 7 is Saturday.
N/A	DAYOFYEAR	Returns the day of the year from a value.
N/A	DAYS	Returns an integer representation of a date.
DBTIMEZONE	CURRENT TIMEZONE	Returns the value of the database time zone.

Oracle	DB2	Comments
EXTRACT(datetime)	YEAR() MOMTH() DAY() HOUR() MINUTE() SECOND() MICROSECOND()	Extracts and returns the value of a specified datetime field from a datetime expression.
FROM_TZ	N/A	Converts a timestamp value and a time zone.
N/A	JULIAN_DAY	Returns an integer value representing the number of days from Jan 1, 4712 B.B. to the date specified in the argument.
LAST_DAY	N/A. Implemented by UDF.	Returns the date of the last day of the month of the input date. We also provide code sample that accept timestamp as the input. Refer to Example C-14 on page 636 for sample code.
MONTHS_BETW EEN(arg1,arg2)	N/A. Implemented by UDF.	Returns number of months between dates specified in arg1 and arg2. Refer to Example C-15 on page 638 for sample code.
NEW_TIME(date, arg1,arg2)	N/A. Implemented by UDF.	Returns the date and time in the time zone specified in arg2 of the date specified in the first input argument. Refer to Example C-16 on page 641 for sample code.
NEXT_DAY(arg1, arg2)	N/A. Implemented by UDF.	Returns the date of the first weekday named by arg2 that is later than the date arg1. Refer to Example C-17 on page 643 for sample code.
NUMTOSDINTER VAL NUMTOYMINTER VAL	N/A	Convert a value to an interval value.
ROUND(date,fmt)	N/A. Implemented by UDF.	Returns date rounded to the unit specified by the format model fmt. Refer toExample C-18 on page 646 for sample code.

Oracle	DB2	Comments
SESSIONTIMEZONE	CURRENT TIMEZONE	Returns the time zone of the current session.
SYS_EXTRACT_ UTC	Implement by CURRENT TIMESTAMP - CURRENT TIMEZONE	Returns the UTC time from a datetime value with time zone offset or time zone.
SYSDATE	CURRENT DATE	Returns the current date.
SYSTIMESTAMP	implement by CURRENT TIMESTAMP+ CURRENT TIMEZONE	Returns the current timestamp.
N/A	TIMESTAMPDIFF	Returns an estimated number of intervals of the type defined by the first argument, based on the difference between two timestamps.
TRUNC(arg1,arg2)	N/A Implemented by UDF	Returns arg1 with the time portion of the day truncated to the unit specified by arg2. Refer to Example C-19 on page 647 for sample code.
TZ_OFFSET	N/A	Returns the time zone offset

Example C-13 shows DB2 user-defined function ADD_MONTHS and its usage.

Example: C-13   DB2 UDF ADD_MONTHS

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/08/31, 11/06
--
-- Name of UDF: ADD_MONTH (D Date, N integer)
--
-- Used UDF: None
--
--
-- Description: Add N month to the date specified in D.
--              If the date specified in D is the last day
--              of the month, the date returned will be the
--              last day of the returned month.
--
```

```
-- Author: TOKUNAGA, Takashi
--


--------------------------------------------------------------------
CREATE FUNCTION ADD_MONTHS (D Date, N integer)
 RETURNS Date
 LANGUAGE SQL
 SPECIFIC ADD_MONTHSOracle
 DETERMINISTIC
 NO EXTERNAL ACTION
 CONTAINS SQL
 RETURN
CASE
WHEN day(D + 1 day) = 1 THEN (D + N month + 4 days) - day(D + N month + 4 days)
days
ELSE D + N month
END;
----------------------------------------------------


--
-- Usage samples
--
--------------------------------------------------------------------
SELECT add_months(date('2001-02-27'),11) FROM sysibm.sysdummy1;
----------------------------------------------------

1
----------
2002-01-27

    1 record(s) selected.


--------------------------------------------------------------------
SELECT add_months(date('2001-02-28'),11) FROM sysibm.sysdummy1;
--------------------------------------------------

1
----------
2002-01-31

    1 record(s) selected.


--------------------------------------------------------------------
SELECT add_months(date('2001-01-29'),13) FROM sysibm.sysdummy1;
--------------------------------------------------

1
----------
2002-02-28
```

```
    1 record(s) selected.


--------------------------------------------------------------------
SELECT add_months(date('2003-01-31'),13) FROM sysibm.sysdummy1;
---------------------------------------------------


1
----------
2004-02-29

    1 record(s) selected.
```

Example C-14 shows the two DB2 sample code for LAST_DAY and theirs usage.
One function accepts a date as the input argument, the other accepts timestamp
as the input argument.

Example: C-14   DB2 sample UDF - LAST_DAY

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/08/29, 11/06, 2002/05/05
--
-- Name of UDF: LAST_DAY (D Date)
--              LAST_DAY (D Timestamp)
--
-- Used UDF: None
--
-- Description: Last day of month.
--
-- Author: TOKUNAGA, Takashi
--
-- (1) Argument is a date
--------------------------------------------------------------------
CREATE FUNCTION LAST_DAY (D Date)
 RETURNS Date
 LANGUAGE SQL
 SPECIFIC LAST_DAYDate
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
D + 1 month - day(D + 1 month) day
;
---------------------------------------------------

--
-- Usage sample
--
```

```
-----------------------------------------------------------------------
select hiredate, last_day(date(hiredate)) "Last_Date"
  from (values ('1980-10-17')
              ,('1980-01-31')
              ,('1981-02-22')
              ,('1981-03-02')
              ,('1987-04-19')
              ,('1981-05-01')
              ,('1981-06-09')
              ,('1981-07-08')
              ,('1981-09-28')
              ,('1981-11-17')
              ,('1981-12-01')
              ,('1981-12-31')
              ,('1982-01-23')
) q(hiredate)
;
-----------------------------------------------------

HIREDATE   Last_Date
---------- ----------
1980-10-17 1980-10-31
1980-01-31 1980-01-31
1981-02-22 1981-02-28
1981-03-02 1981-03-31
1987-04-19 1987-04-30
1981-05-01 1981-05-31
1981-06-09 1981-06-30
1981-07-08 1981-07-31
1981-09-28 1981-09-30
1981-11-17 1981-11-30
1981-12-01 1981-12-31
1981-12-31 1981-12-31
1982-01-23 1982-01-31

    13 record(s) selected.




-----------------------------------------------------------------------
select last_day(date('2001-1-1')) from sysibm.sysdummy1;
-------------------------------------------------

1
----------
2001-01-31

    1 record(s) selected.
-----------------------------------------------------------------------
```

```
--
-- (2) Argument is a timestamp
--
-------------------------------------------------------------------
CREATE FUNCTION LAST_DAY (D Timestamp)
 RETURNS Timestamp
 LANGUAGE SQL
 SPECIFIC LAST_DAYTimestamp
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
D + 1 month - day(D + 1 month) day
;
--------------------------------------------------
--
-- Usage sample
--
-------------------------------------------------------------------
select sysdate(),
   last_day(sysdate()) "Last",
   last_day(sysdate()) - sysdate() "Days Left"
   from sysibm.sysdummy1
;
-------------------------------------------------

1                          Last                       Days Left
---------------- ------------------------ ---------------------
2002-05-06-08.41.41.723001 2002-05-31-08.41.41.723001      25000000.000000

    1 record(s) selected.
```

Note that the result of subtracting two timestamps is a timestamp duration. Its data type is DECIMAL(20,6) with the format as follows:

`yyyymmddhhmiss.ssssss`

Example C-15 shows the DB2 sample code of MONTHS_BETWEEN and its usage.

Example: C-15   DB2 sample UDF - MONTHS_BETWEEN

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/09/01, 2002/04/16
--
-- Name of UDF: MONTHS_BETWEEN (D1 Date, D2 Date)
--
```

```
-- Used UDF: None
--
-- Description: Months between D1 and D2.
--              Fractions are calculated considering the month with
--              31 days.
--              If D1 and D2 are both end of month,
--                 then fractional will be 0.
--
-- Author: TOKUNAGA, Takashi
--


--------------------------------------------------------------------
CREATE FUNCTION MONTHS_BETWEEN (D1 Date, D2 Date)
 RETURNS Double
 LANGUAGE SQL
 SPECIFIC MONTHS_BETWEENdate
 DETERMINISTIC
 CONTAINS SQL
 NO EXTERNAL ACTION
 RETURN
(year(D1) - year(D2)) * 12 + month(D1) - month(D2) +
CASE
WHEN day(D1 + 1 day) = 1 and day(D2 + 1 day) = 1 THEN 0
ELSE double(day(D1) - day(D2)) / 31
END
;
---------------------------------------------------
--
-- Usage sample
--
--------------------------------------------------------------------
select months_between(date('1995-02-02'),date('1995-01-01')) from
sysibm.sysdummy1;
---------------------------------------------------

1
-----------------------
  +1.03225806451613E+000

    1 record(s) selected.



--------------------------------------------------------------------
select months_between(date('2001-07-06'),date('2001-06-06')) from
sysibm.sysdummy1;
---------------------------------------------------

1
```

```
-----------------------
  +1.00000000000000E+000

    1 record(s) selected.




--------------------------------------------------------------------
select months_between(date('2001-06-21'),date('2001-06-07')) from
sysibm.sysdummy1;
-------------------------------------------------

1
-----------------------
  +4.51612903225806E-001

    1 record(s) selected.




--------------------------------------------------------------------
select months_between(date('2001-06-21'),date('2001-06-07'))*31 from
sysibm.sysdummy1;
---------------------------------------------------

1
-----------------------
  +1.40000000000000E+001

    1 record(s) selected.




--------------------------------------------------------------------
select months_between(date('2001-06-30'),date('2001-02-28')) from
sysibm.sysdummy1;
--------------------------------------------------

1
-----------------------
  +4.00000000000000E+000

    1 record(s) selected.




--------------------------------------------------------------------
select months_between(date('2000-02-29'),date('1995-01-31')) from
sysibm.sysdummy1;
```

```
------------------------------------------------

1
-----------------------
  +6.10000000000000E+001

    1 record(s) selected.
```

Example C-16 shows the DB2 code sample of NEW_TIME function and its usage.

Example: C-16   DB2 sample UDF - NEW_TIME

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/11/05, 11/12, 2002/05/06
--
-- Name of UDF: NEW_TIME (D Timestamp, Z1 Varchar(3), Z2 Varchar(3))
--
-- Used UDF: None
--
-- Description:
--      Convert time of time zone Z1 to time zone Z2.
--      Common across the DB2 family.
--
--      Z1 and Z2 must be following strings:
--
--      AST, ADT: Atlantic standard time and Atlantic daylight time
--      BST, BDT: Bering standard time and Bering daylight time
--      CST, CDT: Central standard time and Central daylight time
--      EST, EDT: Eastern standard time and Eastern daylight time
--      GMT:      Greenwich mean time
--      HST, HDT: Hawaiian standard time and Hawaiian daylight time
--      MST, MDT: Mountain standard time and Mountain daylight time
--      NST:      Newfoundland standard time
--      PST, PDT: Pacific standard time and Pacific daylight time
--      YST, YDT: Yukon standard time and Yukon daylight time
--
-- Author: TOKUNAGA, Takashi
--
--------------------------------------------------------------------
CREATE FUNCTION NEW_TIME (D Timestamp, Z1 Varchar(3), Z2 Varchar(3))
 RETURNS Timestamp
 LANGUAGE SQL
 SPECIFIC NEW_TIMECommon
 DETERMINISTIC
 CONTAINS SQL
```

```
 NO EXTERNAL ACTION
 RETURN D
- DECIMAL(
         CASE ucase(Z1)
         WHEN 'AST' THEN -4
         WHEN 'ADT' THEN -3
         WHEN 'BST' THEN -11
         WHEN 'BDT' THEN -10
         WHEN 'CST' THEN -6
         WHEN 'CDT' THEN -5
         WHEN 'EST' THEN -3
         WHEN 'EDT' THEN -2
         WHEN 'GMT' THEN  0
         WHEN 'HST' THEN -10
         WHEN 'HDT' THEN -9
         WHEN 'MST' THEN -7
         WHEN 'MDT' THEN -6
         WHEN 'NST' THEN -3.3
         WHEN 'PST' THEN -8
         WHEN 'PDT' THEN -7
         WHEN 'YST' THEN -9
         WHEN 'YDT' THEN -8
         ELSE null
         END
      * 10000,6,0)
+ DECIMAL(
         CASE ucase(Z2)
         WHEN 'AST' THEN -4
         WHEN 'ADT' THEN -3
         WHEN 'BST' THEN -11
         WHEN 'BDT' THEN -10
         WHEN 'CST' THEN -6
         WHEN 'CDT' THEN -5
         WHEN 'EST' THEN -3
         WHEN 'EDT' THEN -2
         WHEN 'GMT' THEN  0
         WHEN 'HST' THEN -10
         WHEN 'HDT' THEN -9
         WHEN 'MST' THEN -7
         WHEN 'MDT' THEN -6
         WHEN 'NST' THEN -3.3
         WHEN 'PST' THEN -8
         WHEN 'PDT' THEN -7
         WHEN 'YST' THEN -9
         WHEN 'YDT' THEN -8
         ELSE null
         END
      * 10000,6,0)
 ;
```

```
---------------------------------------------------
--
-- Usage samples
--
--------------------------------------------------------------------
SELECT NEW_TIME(TIMESTAMP('2001-10-31-22.35.17.000000'),'AST','ADT') FROM
sysibm.sysdummy1;
---------------------------------------------------

1
------------------------
2001-10-31-23.35.17.000000

    1 record(s) selected.



--------------------------------------------------------------------
SELECT NEW_TIME(TIMESTAMP('1999-11-10-01.23.45.000000'),'AST','PST') FROM
sysibm.sysdummy1;
--------------------------------------------------

1
------------------------
1999-11-09-21.23.45.000000

    1 record(s) selected.



--------------------------------------------------------------------
SELECT NEW_TIME(TIMESTAMP('1999-11-10-01.23.45.000000'),'NST','PST') FROM
sysibm.sysdummy1;
--------------------------------------------------

1
------------------------
1999-11-09-20.53.45.000000

    1 record(s) selected.
```

Example C-17 shows the DB2 code sample of the NEXT_DAY function and its usage.

Example: C-17   DB2 sample UDF - NEXT_DAY

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
```

```
-- Created: 2002/09/05
--
-- Name of UDF: NEXT_DAY (D Date, DN Varchar(10))
--
-- Used UDF: None
--
-- Description: Returns first day with dayname specified by DN after D.
--
-- Author: TOKUNAGA, Takashi
--


---------------------------------------------------------------------
CREATE FUNCTION NEXT_DAY (D Date, DN Varchar(10))
 RETURNS Date
 SPECIFIC NEXT_DAYAnyLang
 LANGUAGE SQL
 CONTAINS SQL
 EXTERNAL ACTION
 NOT DETERMINISTIC
BEGIN ATOMIC
DECLARE N      INTEGER DEFAULT 0;
Loop1: WHILE ucase(dayname(D + N days)) <> ucase(DN) AND N < 7 DO
   SET N = N + 1;
END WHILE Loop1;
IF N = 0 THEN
   RETURN D + 7 days;
ELSEIF N < 7 THEN
   RETURN D + N days;
ELSE
   SIGNAL SQLSTATE 'U1846' SET MESSAGE_TEXT = 'Specified dayname is invalid.';
END IF;
END
!
----------------------------------------------------
--
-- Usage samples
--
---------------------------------------------------------------------
Values next_Day(date('1998-03-15'), 'Tuesday');
----------------------------------------------------

1
----------
1998-03-17

    1 record(s) selected.


---------------------------------------------------------------------
```

```
Values next_Day(date('1992-3-15'), 'SUNDAY');
--------------------------------------------------

1
----------
1992-03-22

    1 record(s) selected.



----------------------------------------------------------------------
Values next_Day(date('1992-3-15'), 'tuesday');
--------------------------------------------------

1
----------
1992-03-17

    1 record(s) selected.




----------------------------------------------------------------------
Values next_Day(date('1992-3-29'), 'Wednesday');
--------------------------------------------------

1
----------
1992-04-01

    1 record(s) selected.




----------------------------------------------------------------------
Values next_Day(date('1992-3-15'), ' Sunday');
--------------------------------------------------

1
----------
SQL0438N Application raised error with diagnostic text: "Specified dayname is
invalid.".   SQLSTATE=U1846
```

Example C-18 shows the DB2 sample code for ROUND function. We also include a simplified version which returns a rounded timestamp in a default format.

Example: C-18   DB2 sample UDF - ROUND for Timestamp

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created: 2004/03/12
--
-- Name of UDF: ROUND (inTS Timestamp, Fmt VarChar(5))
--
-- Used UDF: None
--
--
-- Description: Return rounded imestamp value according
--              to specified format.
--
-- Refer to Oracle manual for String format
-- Author: Tokunaga, Takashi
--
-------------------------------------------------------------------
CREATE FUNCTION ROUND(inTS Timestamp, Fmt VarChar(5))
 RETURNS Timestamp
 LANGUAGE SQL
 SPECIFIC ROUND_Timestamp2
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
BEGIN ATOMIC
DECLARE Jan01, Mon01 DATE;
DECLARE UCASE_Fmt VarChar(5);

SET UCASE_Fmt = UCASE(Fmt);

IF UCASE_Fmt  = 'WW' THEN
   SET Jan01 = DATE(SUBSTR(CHAR(inTS),1,4)||'001');
ELSEIF UCASE_Fmt = 'W' THEN
   SET Mon01 = DATE(inTS) - (DAY(inTS)-1) DAYS;
END IF;

RETURN
CASE
WHEN UCASE_Fmt IN ('CC', 'BCC') THEN
    TIMESTAMP(SUBSTR(DIGITS(YEAR(inTS)+100),7,2) ||
'00-01-01-00.00.00.000000')
WHEN UCASE_Fmt IN ('SYYYY', 'YYYY', 'YEAR', 'SYEAR', 'YYY', 'YY', 'Y') THEN
    TIMESTAMP(SUBSTR(CHAR(inTS + 7 MONTHS),1, 4) || '-01-01-00.00.00.000000')
WHEN UCASE_Fmt IN ('MONTH', 'MON', 'MM', 'RM') THEN
    TIMESTAMP(SUBSTR(CHAR(inTS + 1 MONTH - 15 DAYS),1, 7) ||
'-01-00.00.00.000000')
WHEN UCASE_Fmt IN ('DDD', 'DD', 'J') THEN
```

```
         TIMESTAMP(SUBSTR(CHAR(inTS),1,10) || '-00.00.00.000000')
WHEN UCASE_Fmt = 'WW' THEN
      TIMESTAMP(CHAR(Jan01 + MOD(DAYOFWEEK(inTS)-DAYOFWEEK(Jan01)+7,7) DAYS,ISO)
|| '-00.00.00.000000')
WHEN UCASE_Fmt = 'W' THEN
      TIMESTAMP(CHAR(Mon01 + MOD(DAYOFWEEK(inTS)-DAYOFWEEK(Mon01)+7,7) DAYS,ISO)
|| '-00.00.00.000000')
WHEN UCASE_Fmt IN ('DAY', 'DY', 'D') THEN
      TIMESTAMP(CHAR(DATE(inTS) - (DAYOFWEEK(inTS)-1) DAYS,ISO) ||
'-00.00.00.000000')
WHEN UCASE_Fmt IN ('HH', 'HH12', 'HH24') THEN
      TIMESTAMP(SUBSTR(CHAR(inTS),1,13) || '.00.00.000000')
WHEN UCASE_Fmt = 'MI' THEN
      TIMESTAMP(SUBSTR(CHAR(inTS),1,16) || '.00.000000')
END;

END!
-----------------------------------------------------------------
--
-- (2) This simplified ROUND version returns a timestamp with
--     the default format 00.00.00.000000
-----------------------------------------------------------------
CREATE FUNCTION ROUND(inTS Timestamp)
 RETURNS Timestamp
 LANGUAGE SQL
 SPECIFIC ROUND_Timestamp1
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
RETURN
      TIMESTAMP(SUBSTR(CHAR(inTS), 1, 10) || '-00.00.00.000000')
!
```

Example C-19 shows the DB2 sample code for TRUNC function. We also include an simplified version which returns the truncated timestamp with a default format.

Example: C-19   DB2 sample UDF - TRUNC

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created: 2004/03/12
--
-- Name of UDF: TRUNC (inTS Timestamp, Fmt VarChar(5))
--
-- Used UDF: None
--
--
```

```
-- Description: Return truncated timestamp value according
--              to specified format.
--
-- Refer to Oralce maual for string format
--
-- Author: Tokunaga, Takashi

CREATE FUNCTION TRUNC(inTS Timestamp, Fmt VarChar(5))
 RETURNS Timestamp
 LANGUAGE SQL
 SPECIFIC TRUNC_Timestamp2
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
BEGIN ATOMIC
DECLARE Jan01, Mon01 DATE;
DECLARE UCASE_Fmt VarChar(5);

SET UCASE_Fmt = UCASE(Fmt);

IF UCASE_Fmt  = 'WW' THEN
   SET Jan01 = DATE(SUBSTR(CHAR(inTS),1,4)||'001');
ELSEIF UCASE_Fmt = 'W' THEN
   SET Mon01 = DATE(inTS) - (DAY(inTS)-1) DAYS;
END IF;

RETURN
CASE
WHEN UCASE_Fmt IN ('CC', 'BCC') THEN
     TIMESTAMP(SUBSTR(DIGITS(YEAR(inTS)+100),7,2) ||
'00-01-01-00.00.00.000000')
WHEN UCASE_Fmt IN ('SYYYY', 'YYYY', 'YEAR', 'SYEAR', 'YYY', 'YY', 'Y') THEN
     TIMESTAMP(SUBSTR(CHAR(inTS),1, 4) || '-01-01-00.00.00.000000')
WHEN UCASE_Fmt IN ('MONTH', 'MON', 'MM', 'RM') THEN
     TIMESTAMP(SUBSTR(CHAR(inTS),1, 7) || '-01-00.00.00.000000')
WHEN UCASE_Fmt IN ('DDD', 'DD', 'J') THEN
     TIMESTAMP(SUBSTR(CHAR(inTS),1,10) || '-00.00.00.000000')
WHEN UCASE_Fmt = 'WW' THEN
     TIMESTAMP(CHAR(Jan01 + MOD(DAYOFWEEK(inTS)-DAYOFWEEK(Jan01)+7,7) DAYS,ISO)
|| '-00.00.00.000000')
WHEN UCASE_Fmt = 'W' THEN
     TIMESTAMP(CHAR(Mon01 + MOD(DAYOFWEEK(inTS)-DAYOFWEEK(Mon01)+7,7) DAYS,ISO)
|| '-00.00.00.000000')
WHEN UCASE_Fmt IN ('DAY', 'DY', 'D') THEN
     TIMESTAMP(CHAR(DATE(inTS) - (DAYOFWEEK(inTS)-1) DAYS,ISO) ||
'-00.00.00.000000')
WHEN UCASE_Fmt IN ('HH', 'HH12', 'HH24') THEN
     TIMESTAMP(SUBSTR(CHAR(inTS),1,13) || '.00.00.000000')
WHEN UCASE_Fmt = 'MI' THEN
```

```
     TIMESTAMP(SUBSTR(CHAR(inTS),1,16) || '.00.000000')
END;

END!
-------------------------------------------------------------------
--
-- (2) This simplified TRUNC version returns a timestamp with
--     the default format 00.00.00.000000
--
-------------------------------------------------------------------
CREATE FUNCTION TRUNC(inTS Timestamp)
 RETURNS Timestamp
 LANGUAGE SQL
 SPECIFIC TRUNC_Timestamp1
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
RETURN
     TIMESTAMP(SUBSTR(CHAR(inTS), 1, 10) || '-00.00.00.000000')
!
```

C.4 Conversion and cast function mapping

Table C-4 shows the conversion and cast function mapping between Oracle and DB2.

Table C-4   Conversion and cast functions

Oracle	DB2	Comments
ASCIISTR	N/A	Returns the ASCII string equivalent of the argument.
BIN_TO_NUM	N/A	Converts a bit vector to its equivalent number.
CAST	CAST	Converts one built-in datatype or collection-typed value into another built-in datatype or collection-typed value.
CHARTOROWID	N/A	Converts a value from CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype to ROWID Oracle datatype.
COMPOSE	N/A	Returns an Unicode string in its fully normalized form in the same character set as the input.

Oracle	DB2	Comments
CONVERT	N/A	Converts a character string from one character set to another.
DECOMPOSE	N/A	Returns an Unicode string after decomposition in the same character set as the input.
HEXTORAW	N/A. Implemented by UDF.	Converts CHAR containing hexadecimal digits in the character set to a raw value. Refer to Example C-20 on page 652 for sample code.
NUMTODSINTERVAL NUMTOYMINTERBVAL	N/A	Converts a number to an Oracle INTERVAL data type.
RAWTOHEX	HEX	Converts raw to a character value containing its hexadecimal equivalent.
RAWTONHEX	N/A	Converts raw to the Oracle NVARCHAR2 character value containing its hexadecimal equivalent.
ROWIDTOCHAR	N/A	Converts a row ID value to the Oracle VARCHAR2 data type.
ROWIDTONCHAR	N/A	Converts a row ID value to the Oracle NVARCHAR2 data type.
SCN_TO_TIMESTAMP	N/A	Returns the approximate timestamp associated with the Oracle SCN(System Change Number).
TIESTAMP_TO_SCN	N/A	Returns the approximate system change number (SCN) associated with that timestamp.
TO_BINARY_DOUBLE	DOUBLE	Returns a double-precision floating-point number.
TO_BINARY_FLOAT	FLOAT	Returns a single-precision floating-point number.
TO_CHAR(character)	N/A	Converts the Oracle NCHAR, NVARCHAR2, CLOB, or NCLOB data to the database character set.
TO_CHAR (datetime)	TO_CHAR	Converts a datetime value to a string in the format specified by the date format.

Oracle	DB2	Comments
TO_CHAR (datetime, 'DAY')	DAYNAME (expression)	Returns the name of the day in datetime.
TO_CHAR (datetime, 'D')	DAYOF WEEK (expression)	Returns the day of week of datetime.
TO_CHAR (datetime, 'DDD')	DAYOFYEAR (expression)	Returns the day of year of datetime.
TO_CHAR (datetime, 'J')	JURIAN_DAY (expression)	Returns the Jurian day of datetime.
TO_CHAR (datetime, 'MONTHD')	MONTHNAME (expression)	Returns the name of the month in datetime.
TO_CHAR(number)	N/A. Implement by CHAR function or UDF.	Converts the number using the optional format specified. Refer to Example C-21 on page 654 for sample code.
TO_CLOB	N/A	Converts a value of Oracle NCLOB type to LOB value.
TO_DATE	DATE() FORMAT_TIMESTAMP() TO_DATE()	Returns a date from a character string. Two UDFs that convert a date and time string based on the format specified are availabe for download. See See Appendix G, "Additional material" on page 701 for the download instructions.
TO_DSINTERVAL	N/A	Converts a character string of the Oracle CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to an INTERVAL DAY TO SECOND value.
TO_LOB	BLOB() CLOB()	Returns a LOB(BLOB or CLOB) representation of a string.
TO_MULTI_BYTE	N/A	Returns characters with all of its single-byte characters converted to their corresponding multibyte characters.
TO_NCHAR(character) TO_NCHAR(datetime) TO_NCHAR(number) TO_NCLOB	N/A	Converts an argument value to the Oracle national character set.

Oracle	DB2	Comments
TO_NUMBER	BIGINT() INT() FLOAT() DECIMAL() DOUBLE() REAL() SMALLINT()	Converts an argument to a value of the Oracle NUMBER datatype. In DB2 you can choose an appropriate function according to the type.
TO_SINGLE_BYTE	N/A	Returns a character string with all of its multibyte characters converted to their corresponding single-byte characters.
TO_TIMESTAMP	TO_DATE TIME TIMESTAMP TIMESTAMP_FORMAT TIMESTAMP_ISO	Returns a value of TIMESTAMP datatype converted to CHAR data type.
TO_TIMESTAMP_TZ	N/A	Converts the argument to a value of TIMESTAMP WITH TIME ZONE data type.
TO_YMINTERVAL	N/A	Converts the argument to an INTERVAL YEAR TO MONTH type.
TRANSLATE(expr, from_str,to_str)	TRANSLATE(expr,to_str,from_str)	Returns *expr* with all occurrences of each character in *from_str* replaced by its corresponding character in *to_str*.
UNISTR	N/A	Takes as its argument a text literal or an expression that resolves to character data and returns it in the national character set.

Example C-20 shows the DB2 sample of HEXTORAW function and its usage.

Example: C-20   DB2 sample UDF - HEXTORAW

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created 2004/03/06
-- Updated 2006/10/19: Remove leading and trailing blanks.
--                     If input length id odd, add leading '0' to
--                     the hex string.
--          2007/03/08: Replace IF statement with Complex expression
```

```
--                    to add leading '0'.
--                    Take steps for CHR(0) = x'20' by COALESCE and NULLIF.
--
-- Syntax:: HexToRaw (HX VarChar(4000))
--
-- Description: Convert input hexadecimal value to character string.
--              Leading and trailing blanks will be removed.
--              If input length is odd, add leading '0' to the hex string.
--
-- Author: TOKUNAGA, Takashi
--

---------------------------- Command Entered -----------------------------
CREATE FUNCTION HexToRaw (HX VarChar(4000))
 RETURNS VarChar(2000) FOR BIT DATA
 SPECIFIC HexToBitChar
 LANGUAGE SQL
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
BEGIN ATOMIC
DECLARE HXU VarChar(4000);
DECLARE HXLen Integer;
DECLARE XDigits CHAR(16) DEFAULT '123456789ABCDEF';
DECLARE RetVal VarChar(2000) FOR BIT DATA;
DECLARE Pos Integer;
DECLARE DiagText VARCHAR(70);

SET (HXU, HXLen) = (
SUBSTR('0',1,MOD(LENGTH(LTRIM(RTRIM(HX))),2))||UCASE(LTRIM(RTRIM(HX)))
                 ,LENGTH(LTRIM(RTRIM(HX)))+MOD(LENGTH(LTRIM(RTRIM(HX))),2)
);

IF TRANSLATE(HXU, '*', ' 0'||XDigits) <> '' THEN
   SET DiagText = 'Input data contains invalid character. Position = '
                 || RTRIM(CHAR(HXLen - LENGTH(LTRIM(TRANSLATE(HXU, '*', '
0'||XDigits))) + 1));
   SIGNAL SQLSTATE VALUE 'UX002' SET MESSAGE_TEXT = DiagText;
END IF;

SET (Pos, RetVal) = (1, '');
WHILE Pos < HXLen DO
   SET RetVal = RetVal || COALESCE(CHR(NULLIF( LOCATE(SUBSTR(HXU,Pos,
1),XDigits)*16

+LOCATE(SUBSTR(HXU,Pos+1,1),XDigits), 0 )),x'00');
   SET Pos = Pos + 2;
END WHILE;
```

```
        RETURN RetVal;
        END!




        ---------------------------- Commands Entered ----------------------------
        SELECT n
             , HEX(n)              AS HEX_N
             , SUBSTR(HEXTORAW(HEX(n)),1,4) AS HEXTORAW
          FROM (VALUES 63065, 65535, -65536, 0) TestData(n)!
        ---------------------------------------------------------------------------


        N           HEX_N    HEXTORAW
        ----------- -------- ------------------
              63065 59F60000 x'59F60000'
              65535 FFFF0000 x'FFFF0000'
             -65536 0000FFFF x'0000FFFF'
                  0 00000000 x'00000000'

          4 record(s) selected.
```

Example C-21 shows the DB2 sample code for the TO_CHAR function and its usage.

Example: C-21   DB2 sample UDF - TO_CHAR

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created: 2004/03/07
-- Updated: 2006/09/27 Added single parameter(Float) version.
--          2007/03/13 Tuning performance.
--
-- Name of UDF: TO_CHAR (DI DEC(31,10), FMT VARCHAR(50))
--
-- Used UDF: None
--
-- Description: Returns character string expression of DI according to FMT.
--
--     Sopported format characters:
--        ,  : put comma at this position. multiple commas are allowed.
--        .  : put period at this position. only one period can be specified.
--        9  : digit with leading zero supressed.
--        0  : starting digit without leading zero supressed.
--        S  : return '-' for negative value.
--             return '+' for positive value.
--             Note: only allowed at leftmost or rightmost position.
--
-- Author: TOKUNAGA, Takashi
--
```

```
--------------------------------------------------------------------------------
--------
DROP SPECIFIC FUNCTION TO_CHARDecimal


CREATE FUNCTION TO_CHAR (DI DEC(31,10), FMT VARCHAR(50))
 RETURNS VARCHAR(50)
 SPECIFIC TO_CHARDecimal
 LANGUAGE SQL
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC

BEGIN ATOMIC
DECLARE Retv  VARCHAR(50);
DECLARE DIdg  CHAR(31);
DECLARE Posd
      , Posf
      , SigI  SMALLINT;

IF length(rtrim(translate(FMT, '    ', ',.90S'))) > 0
OR length(ltrim(rtrim(translate(FMT, '   ', ',90S')))) > 1
OR posstr(FMT, 'S') NOT IN (0, 1, length(FMT))
THEN
   RETURN '*** Format Error. ***';
END IF;

SET DIdg  = digits(DI);
SET Posd  = 22 - length( replace( substr(FMT, 1, posstr(FMT||'.', '.') - 1),
',', '') )
                + sign(posstr(substr(FMT, 1, posstr(FMT||'.', '.') - 1),'S'));

SET (Posf, SigI, Retv) = (1, 0, '');

WHILE Posf <= length(FMT) DO
   SET SigI = CASE
               WHEN substr(FMT,Posf,1) IN ('0', '.')
                 OR substr(FMT,Posf,1) = '9' AND substr(DIdg,Posd,1) <> '0' THEN
                    1
               ELSE SigI
               END;

   SET Retv = CASE
               WHEN SigI = 1
                AND substr(FMT,Posf,1) IN ('0', '9') THEN
                    Retv || substr(DIdg,Posd,1)
               WHEN SigI = 1
                AND substr(FMT,Posf,1) IN (',', '.') THEN
```

```
                        Retv || substr(FMT,Posf,1)
                WHEN substr(FMT,Posf,1) = 'S' THEN
                        Retv || CASE
                                WHEN DI > 0 THEN '+'
                                WHEN DI < 0 THEN '-'
                                ELSE            ' '
                                END
                WHEN Posf = length(FMT) THEN
                        Retv || '0'
                ELSE Retv || ' '
                END;

    SET Posd = CASE
                WHEN substr(FMT,Posf,1) IN ('0', '9') THEN
                        Posd + 1
                ELSE Posd
                END;

    SET Posf = Posf + 1;
END WHILE;

RETURN Retv;
END!
--------------------------------------------------
DB20000I  The SQL command completed successfully.




CREATE FUNCTION TO_CHAR (InFloat FLOAT)
 RETURNS VARCHAR(50)
 SPECIFIC TO_CHARDFloatOnly
 LANGUAGE SQL
 CONTAINS SQL
 NO EXTERNAL ACTION
 DETERMINISTIC
RETURN TRANSLATE(RTRIM(LTRIM(TRANSLATE(CHAR(DEC(InFloat,31,10)),'','0'))),'0','
')


---------------------------- Commands Entered ------------------------------
VALUES TO_CHAR(1.2345E+2);
----------------------------------------------------------------------------


1
--------------------------------------------------
123.45

  1 record(s) selected.
```

```
-------------------------- Commands Entered -----------------------------
VALUES TO_CHAR(-1234567890,'9999999999S');
-------------------------------------------------------------------------

1
-------------------------------------------------
1234567890-

  1 record(s) selected.


-------------------------- Commands Entered -----------------------------
VALUES TO_CHAR(-1234567890,'99,999,999,999.S');
-------------------------------------------------------------------------

1
-------------------------------------------------
 1,234,567,890.-

  1 record(s) selected.


-------------------------- Commands Entered -----------------------------
VALUES TO_CHAR(-1234567890.123,'S99,999,999,999.99');
-------------------------------------------------------------------------

1
-------------------------------------------------
- 1,234,567,890.12

  1 record(s) selected.


-------------------------- Commands Entered -----------------------------
VALUES TO_CHAR(0,'S99,999,999,999.99');
-------------------------------------------------------------------------

1
-------------------------------------------------
             .00

  1 record(s) selected.


-------------------------- Commands Entered -----------------------------
VALUES TO_CHAR(0,'S99,999,999,900.99');
```

```
--------------------------------------------------------------------------

1
-------------------------------------------------
            00.00

  1 record(s) selected.
```

C.5 Aggregate function mapping

Table C-5 shows the aggregate function mapping between Oracle and DB2.

Table C-5   Aggregate functions

Oracle	DB2	Comments
AVG	AVG	Returns the average of a set of numbers.
CORR	CORR CORRELATION	Returns the coefficient of correlation of a set of number pairs.
CORR_*	N/A	Returns the coefficient nonparametric or rank correlation of a set of number pairs.
COUNT	COUNT	Returns the number of rows or values in a set of rows or values.
COVAR_POP	COVARIANCE	Returns the covariance of a set of number pairs.
COVAR_SAMP	N/A	Returns the sample covariance of a set of number pairs.
CUME_DIST	N/A	Returns the cumulative distribution of a value in a group of values.
DENSE_RANK	DENSE_RANK ()	Returns the value of dense rank position of a row.
FIRST_VALUE	N/A	Returns the first value in an ordered set of values
GROUP_ID	N/A	Assigns an unique integer number, beginning with 0, to duplicate groups resulting from a GROUP BY specification.
GROUPING	GROUPING	This function is used with grouping-sets and super-groups to indicate sub-total of rows generated by a grouping set.

Oracle	DB2	Comments
GROUPING_ID	N/A	Returns a number corresponding to the GROUPING bit vector associated with a row.
LAG (expr [, offset][,default]) [OVER (analytic_clause)]	N/A	Provides access to more than one row of a table at the same time without a self join.
LAST	N/A	Returns the last value in an ordered set.
MAX	MAX	Returns the maximum value of a set of numbers.
MEDIAN	N/A	Returns the median which is an inverse distribution function that assumes a continuous distribution model.
MIN	MIN	Returns the minimum value of a set of numbers.
PERCENT_RANK	N/A	Computes an interpolated value that would fall into a specified percentile value with respect to the ORDER BY CLAUSE, assuming continuous distribution.
PECENTILE_CONT	N/A	An inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into that percentile value with respect to the sort specification.
PERCENTILE_DISC	N/A	Same to PERCENTILE_CONT, but assumes a discrete distribution model.
RANK(expr, ...) WITHIN GROUP (ORDER BY ...)	N/A	Computes the rank of a row among all rows of the aggregation group.
RANK() OVER(analytic_clause)	RANK()	Computes the rank of a row among all rows of the aggregation group.

Oracle	DB2	Comments
RATIO_TO_REPORT(rr_expr) OVER(analytic_clause)	N/A	Computes the ratio of a value to the sum of a set of values.
REGR_regr_type (expr1, expr2) [OVER (analytic_clause)] e.g., REGR_SLOPE REGR_INTERCEPT REGR_COUNT REGR_R2 REGR_AVGX REGR_AVGY REGR_SXX REGR_SYY REGR_SXY	REGR_regr_type (expr1, expr2) [OVER (analytic_clause)] e.g., REGR_SLOPE REGR_INTERCEPT REGR_COUNT REGR_R2 REGR_AVGX REGR_AVGY REGR_SXX REGR_SYY REGR_SXY	Least-squares regression line to a set of number pairs returned by expr1 and expr2.
ROW_NUMBER() OVER	ROW_NUMBER() OVER	Assigns a unique sequential number, beginning with 1, to each row returned to the query.
STATS_BINOMIAL _TEST	N/A	An exact probability test used for dichotomous variables, where only two possible values exist.
STATS_CROSSTAB	N/A	Returns the result of a method called crosstabulation which is used to analyze two nominal variables.
STATS_F_TEST	N/A	Tests whether two variances are significantly different.
STATS_KS_TEST	N/A	A Kolmogorov-Smirnov function that compares two samples to test whether they are from the same population or from populations that have the same distribution.
STATS_MODE	N/A	Returns the value that occurs with the greatest frequency.

Oracle	DB2	Comments
STATS_MW_TEST	N/A	Returns the result of a Mann Whitney test which compares two independent samples to test the null hypothesis that two populations have the same distribution function against the alternative hypothesis that the two distribution functions are different.
STATS_ONE_WAY _ANOVA	N/A	Returns the result of the one-way analysis of variance function (STATS_ONE_WAY_ANOVA) which tests differences in means for statistical significance by comparing two different estimates of variance.
STATS_T_TEST	N/A	Returns the result of the t-test functions
STATS_WSR_TES T	N/A	Returns the result of a Wilcoxon Signed Ranks which test of paired samples to determine whether the median of the differences between the samples is significantly different from zero.
STDDEV	STDDEV	Returns the sample standard deviation of a set of numbers.
STDDEV_POP	N/A	Computes the population standard deviation and returns the square root of the population variance.
STDDEV_SAMP	N/A	Computes the cumulative sample standard deviation and returns the square root of the sample variance.
SUM	SUM	Returns the sum of a set of numbers.
VAR_POP	N/A	Returns the population variance of a set of numbers after discarding the nulls in this set.
VAR_SAMP	N/A	Returns the sample variance of a set of numbers after discarding the nulls in this set.
VARIANCE	VARIANCE	Returns the variance of a set of numbers.

C.6 Comparison and NULL-related function mapping

Table C-6 shows the comparison and NULL-related function mapping between Oracle and DB2.

Table C-6   Comparison and NULL-related functions

Oracle	DB2	Comments
GREATEST	N/A	Returns the greatest value of the list of values.
LEAST	N/A	Returns the least of the list of values
COALESCE	COALESCE VALUE	Returns the first argument that is not null.
LNNVL	N/A	Evaluates a condition when one or both operands of the condition may be null.
NULLIF	NULLIF	Compares two expressions. Returns a null value if the arguments are equal, otherwise it returns the value of the first argument.
NVL	N/A. Implemented by COALESCE function. A UDF that allows mixed data type input is available for download. See Appendix G, "Additional material" on page 701 for the download instructions.	Replaces null with a string.
NVL2(arg1,arg2, arg3)	N/A. Implemented by CASE clause.	Returns arg2 when arg1 is not null and Returns arg3 when arg1 is null

C.7 Encoding, decoding, encryption, and decryption function mapping

Table C-7 shows the encoding, decoding, encryption, and decryption function mapping between Oracle and DB2.

Table C-7   Encoding, decoding, encryption, and decryption functions

Oracle	DB2	Comments
DECODE(expr, srch_val1,result1,,, ,default)	N/A. Implemented by CASE clause.	Compares the *expr* to each *srch_val* one by one. If the *expr* is equal to a *srch_val*, then returns the corresponding *result*. An UDF is provided for dwonload. See Appendix G, "Additional material" on page 701for the download instructions.
DUMP	N/A Implemented by UDF.	Returns a string value containing the data type code, length in bytes, and internal representation of argument. Refer to Example C-22 on page 663 for sample code
N/A	DECRYPT_BIN DECRYPT_CHAR	Returns a value that is the result of decrypting encrypted-data.
N/A	ENCRYPT	Returns a value that is the result of encrypting data.
N/A	GETHINT	Returns the password hint if one is found in the encrypted-data.

Example C-21 shows the DB2 sample code for DUMP function and its usage.

Example: C-22   DB2 sample UDF - DUMP

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- Created: 2004/03/13
--
-- Name of UDF: DUMP (EXP VarChar(32672)/Char(254), RetFmt Integer, SPos
Integer, Len Integer)
--
-- Input data type: VarChar, Char
--
-- Used UDF: X2D, X2O
--
-- Description: Returns data type, length and internal representations.
```

```
--
--              RetFmt specify format of returned value.
--               8 --- Octal
--              10 --- Decimal
--              16 --- Hexadecimal
--              17 --- Character
--
-- Author: TOKUNAGA, Takashi
--

(1) VARCHAR, CHAR

(1-1) VARCHAR

DROP SPECIFIC FUNCTION DUMPVarChar!

------------------------- Command Entered -------------------------
CREATE FUNCTION DUMP (EXP VarChar(32672), RetFmt Integer, SPos Integer, Len
Integer)
 RETURNS VarChar(4000)
 SPECIFIC DUMPVarChar
 LANGUAGE SQL
 CONTAINS SQL
 EXTERNAL ACTION
 DETERMINISTIC
BEGIN ATOMIC
DECLARE ByteData  VarChar(1000);
DECLARE HexData   VarChar(2000) DEFAULT '';
DECLARE HexComma  VarChar(3000);
DECLARE SPosN     Integer;
DECLARE Pos       Integer;

SET SPosN = CASE WHEN SPos > 0 THEN SPos ELSE 1 END;
SET ByteData = SUBSTR(EXP,SPosN,CASE WHEN Len > 0 THEN Len ELSE
LENGTH(EXP)-SPosN+1 END);

IF RetFmt <> 17 THEN
   SET HexData =  HEX(ByteData);
END IF;

IF RetFmt = 16 THEN
   SET (Pos , HexComma)= (1, '');
   WHILE Pos < LENGTH(HexData) DO
      SET HexComma = HexComma || ',' || SUBSTR(HexData,Pos,2);
      SET Pos = Pos + 2;
   END WHILE;
END IF;

RETURN
```

```
 'Type=96 Len=' || RTRIM(CHAR(LENGTH(EXP))) || ': '
   || CASE RetFmt
      WHEN 17 THEN ByteData
      WHEN 16 THEN SUBSTR(HexComma,2)
      WHEN 10 THEN X2D(HexData)
      WHEN  8 THEN X2O(HexData)
      ELSE        X2D(HexData)
      END;

END!
-----------------------------------------------------------------------



------------------------ Command Entered -------------------------
VALUES VARCHAR(DUMP('abcde',17,2,3),50)
!
-----------------------------------------------------------------------


1
--------------------------------------------------
Type=96 Len=5: bcd

  1 record(s) selected.



------------------------ Command Entered -------------------------
VALUES VARCHAR(DUMP('abcde',16,2,3),50)
!
-----------------------------------------------------------------------


1
--------------------------------------------------
Type=96 Len=5: 62,63,64

  1 record(s) selected.



------------------------ Command Entered -------------------------
VALUES VARCHAR(DUMP('abcde',10,2,3),50)
!
-----------------------------------------------------------------------


1
--------------------------------------------------
Type=96 Len=5: 98,99,100

  1 record(s) selected.
```

```
------------------------- Command Entered -------------------------
VALUES VARCHAR(DUMP('abcde',8,2,3),50)
!
-----------------------------------------------------------------------

1
-------------------------------------------------
Type=96 Len=5: 142,143,144

  1 record(s) selected.
```

D

Oracle Call Interface (OCI) mapping

This appendix provides a mapping of the most frequently used Oracle 10g OCI calls to the closest DB2 Call Level Interface (CLI) equivalents. Refer to the *Call Level Interface Guide and Reference, Volume 1*, SC10-4224, and *Volume 2*, SC10-4225 for details about the CLI calls. Numbers in parentheses refer to the corresponding notes below the tables. N/A means that the OCI call has no equivalent in CLI.

Table D-1   Connect/initialize/authorize

OCI command	CLI command
OCIInitialize	N/A
OCIEnvInit	SQLAllocHandle (1)
OCIServerAttach	SQLConnect (2), (3)
OCIServerDetach	SQLDisconnect
OCISessionBegin	N/A (3)
OCISessionEnd	N/A
OCILogon	SQLConnect (2)

OCI command	CLI command
OCILogoff	SQLDisconnect

Table D-2   Handles/descriptors

OCI command	CLI command
OCIHandleAlloc	SQLAllocHandle (1)
OCIHandleFree	SQLFreeHandle
OCIAttrGet	SQLGet__Attr (4)
OCIParamGet	N/A (5)
OCIParamSet	N/A (6)
OCIAttrSet	SQLSet__Attr (7)
OCIDescriptorAlloc	SQLSetStmtAttr (5)
OCIDescriptorFree	SQLFreeHandle (8)

Table D-3   Transaction management

OCI command	CLI command
OCITransCommit	SQLEndTran (9)
OCITransDetach	N/A
OCITransRollback	SQLEndTran (9)
OCITransStart	N/A
OCITransPrepare	N/A (10)
OCITransForget	N/A (10)

Table D-4   Bind/define/describe

OCI command	CLI command
OCIBindDynamic	SQL___Data (11)
OCIBindByName	SQLBindParameter
OCIBindByPos	SQLBindParameter
OCIBindObject	N/A (12)
OCIBindArrayOfStruct	SQLBindParameter (13)

OCI command	CLI command
OCIStmtGetBindInfo	N/A (14)
OCIDefineArrayOfStruct	N/A (13)
OCIDefineDynamic	N/A
OCIDefineByPos	SQLBindCol (15)
OCIDefineObject	N/A (12)
OCIDescribeAny	(many) (16)

Table D-5   Prepare/execute/fetch

OCI command	CLI command
OCIStmtPrepare	SQLPrepare
OCIStmtExecute	SQLExecute
OCIStmtFetch2	SQLFetch (17)

Table D-6   Miscellaneous

OCI command	CLI command
OCIBreak	SQLCancel
OCIServerVersion	SQLGetInfo (18)
OCIPasswordChange	N/A
OCIErrorGet	SQLGetDiagRec
OCIStmtGetPieceInfo	SQLGetData (11)
OCIStmtSetPieceInfo	SQL___Data (11)
OCILdaToSvcCtx	N/A
OCISvcCtxToLda	N/A

Notes for the tables:

1. SQLAllocHandle is passed the desired handle type: environment, connection, statement, or descriptor.

2. SQLDriverConnect is an alternative to SQLConnect, providing additional parameters.

3. OCISessionBegin has no CLI equivalent. To establish multiple database connections in CLI, multiple connection handles must be allocated and OCIServerAttach calls replaced by SQLConnect calls.

4. OCIAttrGet can be replaced by SQLGetConnectAttr, SQLGetEnvAttr, or SQLGetStmtAttr, depending on the type of handle that the attribute value is wanted for.

5. 5. SQLSetStmtAttr must be called with an Attribute value of SQL_ATTR_APP_PARAM_DESC or SQL_ATTR_APP_ROW_DESC. However, descriptors can be allocated implicitly instead. The function of OCIParamGet is performed by SQLSetStmtAttr (or implicitly).

6. CLI does not have complex object retrieval (COR) descriptors or handles.

7. OCIAttrSet can be replaced by SQLSetConnectAttr, SQLSetEnvAttr, or SQLSetStmtAttr, depending on the type of handle for which an attribute value is to be set.

8. SQLFreeHandle must be called with a HandleType of SQL_HANDLE_DESC (or the connection can be freed).

9. SQLEndTran does either a commit or a rollback, depending on the completion type parameter value.

10. There are no CLI calls specifically for two-phase commit, but it is supported (refer to *Call Level Interface Guide and Reference, Volume 1*, SC10-4224, and *Volume 2*, SC10-4225).

11. For piecewise operations there is no direct replacement for OCIBindDynamic, but for inserts, SQLParamData and SQLPutData must be called, and for selects, SQLGetData.

12. CLI has no special support for user-defined types. CLI treats one as it does the underlying built-in type. In SQL statements the CAST function must be used to convert a parameter with a user-defined type to the corresponding built-in type (or vice-versa). For more information, refer to Chapter 15, "User-defined types (UDT)" in *Call Level Interface Guide and Reference, Volume 1*, SC10-4224.

13. To use array inserts, SQLBindParameter must be called. In addition, calls to SQLSetStmtAttr are needed to set attributes SQL_ATTR_PARAMSET_SIZE (array size) and SQL_ATTR_PARAM_BIND_TYPE (row-wise or column-wise binding of parameters). OCIDefineArrayOfStruct calls can be ignored.

14. No direct equivalent, but SQLDescribeParam is the closest.

15. To use array fetches, SQLBindCol must be called. In addition, calls to SQLSetStmtAttr are needed to set attributes SQL_ATTR_ROW_ARRAY_SIZE (array size) and SQL_ATTR_ROW_BIND_TYPE (row-wise or column-wise array retrieval).

16. An OCIDescribeAny call should be replaced by the appropriate call from among SQLColAttribute, SQLColumns, SQLDescribeCol, SQLForeignKeys, SQLGetFunctions, SQLPrimaryKeys, SQLProcedures, SQLProcedureColumns, SQLSpecialColumns, SQLStatistics, SQLTablePrivileges, and SQLTables.

17. SQLFetch fetches a single row. Use SQLFetchScroll to return a rowset; the simplest type of usage is a basic array fetch.

18. SQLGetInfo provides much more than the server version. One call is needed per type of information wanted.

E

Converter for SQL*Loader

This appendix contains the converter and generator scripts for the samples in Chapter 9, "Script conversion" on page 423. The scripts are referred to in the text of the book, and in the comments section of each listing contain a short description of their functionalities.

This appendix provides the following converters for the Oracle control files:

► conv_ctl.pl
► gen_load_update.pl

You can download the source of the scripts as well at the additional materials link on the IBM Redbooks Internet site. For more information, see Appendix G, "Additional material" on page 701.

E.1 Converting control files for Oracle SQL*Loader

This Perl script generates a DB2 load command based on an Oracle control file. The script is tested with:

▶ GNU Perl v5.8 on Windows 2000
▶ GNU Perl v5.8.2 on AIX 5.3

GNU Perl v5.8.0 on Linux has a known bug with the split function used in the function parse_field_list().

Example: E-1 Conversion of Oracle control file to the DB2 load command

```perl
#!/usr/bin/perl
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
#  Script:      conv_ctl.pl
#  Author:      Stefan Hummel
#  Date:        01/08/2002
#
#  Syntax:      perl conv_ctl.pl -c <controlfile> [options]
#               -t LOAD | IMPORT
#               -m INSERT | REPLACE
#               -f ASC | IXF
#
#  Description: Conversion of Oracle control file (*.ctl) to DB2 load file
#
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

use Text::ParseWords;

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# initialization
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
$infile_name = '';
$table_name  = '';
$mode        = '';
$i           = 0;
$column      = 0;
$delimiter   = ';';
$method      = 'L';

$type     = 'LOAD';
$mode     = 'INSERT';
$filetype = 'ASC';

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# functions
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
sub usage {
```

```perl
        printf("USAGE: perl conv_ctl.pl -c controlfile [options]\n");
        printf("\t-t LOAD | IMPORT\n");
        printf("\t-m INSERT | REPLACE\n");
        printf("\t-f IXF | ASC\n");
        exit;
}

sub read_controlfile {
    $delim = '\s+';
    $i     = 1;
    open(CTLFILE, $_[0]);
    @ctl_lines = <CTLFILE>;
    foreach (@ctl_lines) {
        @words = &parse_line($delim, 1, $_);
        foreach (@words) {
            if ($_ ne '","') {
                @parts = split (/([,])/, $_);
                $n = 0;
                foreach (@parts) {
                    $array[$i++] = $parts[$n++];
                }
            }
            else {
                $array[$i++] = $_ ;
            }
        }
    }
    $max_line = $i-1;
    close(CTLFILE);
}

sub parse_load {
}

sub parse_data {
}

sub parse_into {
    $i++;
    $i++;
    $table_name = $array[$i];
}

sub parse_characterset {
}

sub parse_infile {
    $i++;
    $infile_name = $array[$i];
```

```perl
}

sub parse_delimiter {
   $i++; $i++; $i++;
   $delimiter = substr($array[$i],1,1);
   $method = 'P';
}

sub set_mode {
   if ('' eq $mode) {
      $mode = $_[0];
   }
}

sub parse_field_list {
   while (")" ne $array[$i]) {
      $column++;
      $i++;
      $field_list[$column][1] = $array[$i];          # 1. column name
      $i++;
      @position = split (/([:()])/, $array[$i]);
      $field_list[$column][2] = $position[2];        # 2. from column
      $field_list[$column][3] = $position[4];        # 3. up to column
      while ((" ," ne $array[$i]) && (")" ne $array[$i])) {
         $i++;
      }
   }
}

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# main
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

# read command parameters
while ($parameter=shift(@ARGV)) {
   if ("-c" eq $parameter) {
      $filename = shift @ARGV;
   }
   elsif ("-t" eq $parameter) {
      $type = shift @ARGV
   }
   elsif ("-m" eq $parameter) {
      $mode = shift @ARGV
   }
   elsif ("-f" eq $parameter) {
      $filetype = shift @ARGV
   };
};
if ('' eq $filename) {
```

```perl
      &usage;
}

# read the control file, save each word in array
&read_controlfile($filename);

# read array and parse commands
$i = 1;
while ($i <= $max_line) {
   if ("LOAD" eq uc($array[$i])) {
      &parse_load($i);
   }
   elsif ("DATA" eq uc($array[$i])) {
      &parse_data;
   }
   elsif ("INTO" eq uc($array[$i])) {
      &parse_into;
   }
   elsif ("CHARACTERSET" eq uc($array[$i])) {
      &parse_characterset;
   }
   elsif ("INFILE" eq uc($array[$i])) {
      &parse_infile;
   }
   elsif ("INSERT" eq uc($array[$i])) {
      &set_mode('INSERT');
   }
   elsif ("APPEND" eq uc($array[$i])) {
      &set_mode('APPEND');
   }
   elsif ("REPLACE" eq uc($array[$i])) {
      &set_mode('REPLACE');
   }
   elsif ("TRUNCATE" eq uc($array[$i])) {
      &set_mode('TRUNCATE');
   }
   elsif ("FIELDS" eq uc($array[$i])) {
      &parse_delimiter;
   }
   elsif ("(" eq uc($array[$i])) {
      &parse_field_list;
   }
   ;
   $i++;
};

# generate DB2 Load File
printf("%s FROM %s of %s\n", $type, $infile_name, $filetype);
if ($method eq 'L') {
```

```
                    printf("METHOD %s \n\t( ", $method);
                    for ($c=1;$c<$column;$c++) {
                        if (1 < $c) {
                            printf("\n\t ,");
                        }
                        printf("%s %s", $field_list[$c][2]
                                    , $field_list[$c][3]);
                    }
                    printf(" )\n");
                };
                if ($method eq 'P') {
                    printf("MODIFIED BY COLDEL%s \n", $delimiter);
                    printf("METHOD %s (", $method);
                    for ($c=1;$c<$column;$c++) {
                        if (1 < $c) {
                            printf(", ");
                        }
                        printf("%d", $c);
                    }
                    printf(" )\n");
                };
                printf("%s INTO %s \n\t( ", $mode, $table_name);
                for ($c=1;$c<$column;$c++) {
                    if (1 < $c) {
                        printf("\n\t ,");
                    }
                    printf("%s", $field_list[$c][1]);
                }
                printf(" )!\n");
```

E.2 Generation of additional DB2 commands

Example E-2 is a sample script to generate DB2 UPDATE commands from
Oracle control files.

Example: E-2   Generation of additional DB2 update commands

```
#!/usr/bin/perl
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# Script:      gen_load_update.pl
# Author:      Stefan Hummel
# Date:        07/31/2003
#
# Syntax:      perl gen_load_update.pl -c <controlfile>
#
# Description: Generation of additional UPDATE commands for DB2 UDB
```

```perl
#                   regarding to an Oracle control file
#
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

use Text::ParseWords;

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# initialization
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
$tablename   = '<missing>';

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# functions
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
sub usage {
  printf("USAGE: perl gen_load_update.pl -c controlfile\n\n");
  exit;
};

sub read_controlfile {
  $delim = '\s+';
  $i     = 1;
  open(CTLFILE, $_[0]);
  @ctl_lines = <CTLFILE>;
  foreach (@ctl_lines) {
     @words = &parse_line($delim, 1, $_);
     foreach (@words) {
        @parts = split (/([,])/, $_);
        $n = 0;
        foreach (@parts) {
           $array[$i++] = $parts[$n++]
        }
     }
  }
  $max_line = $i-1;
  close(CTLFILE);
};

sub current_columnname {
   $x = $_[0] - 1;
   while (('(' ne $array[$x]) && (',' ne $array[$x]) && ($x > 0)) {
      $x--;
   }
   return $array[$x + 1];
};

sub parse_nullif {
   $i++;
   $condition[$c][0] = $tablename;                # tablename
```

```perl
    $condition[$c][1] = &current_columnname($i);  # tablename
    $condition[$c][2] = 'NULL';                    # set condition
    $condition[$c][3] = $array[$i];                # where clause
    $c++;
};

sub parse_defaultif {
    $i++;
    $condition[$c][0] = $tablename;                # tablename
    $condition[$c][1] = &current_columnname($i);  # tablename
    $condition[$c][2] = 'DEFAULT';                 # set condition
    $condition[$c][3] = $array[$i];                # where clause
    $c++;
};

sub parse_into {
    $i++;
    $i++;
    $tablename = $array[$i];
};

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# main
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

# read the parameters
while ($parameter=shift(@ARGV)) {
    if ("-c" eq $parameter) {
        $filename = shift @ARGV;
    };
};
if ('' eq $filename) {
    &usage;
}

# read the control file, save each word in array
&read_controlfile($filename);

# read array and parse commands
$i = 1;
$c = 1;
while ($i <= $max_line) {
    if ("NULLIF" eq uc($array[$i])) {
        # print "NULLIF !!!"
        &parse_nullif;
    }
    elsif ("DEFAULTIF" eq uc($array[$i])) {
        # print "NULLIF !!!"
        &parse_defaultif;
```

```
    }
    elsif ("INTO" eq uc($array[$i])) {
        &parse_into;
    };
    $i++;
};
$max_condition = $c;

# generate DB2 update commend
for ($n=1;$n<$max_condition;$n++) {
    printf("UPDATE %s\n"    ,$condition[$n][0]);
    printf("SET %s=%s\n"     ,$condition[$n][1]
                             ,$condition[$n][2]);
    printf("WHERE %s\);\n\n" ,$condition[$n][3]);
};
```

Example Oracle database

This appendix contains the definitions of tables, views, triggers, and procedures of the Oracle database ORA_EMP used in this book. The complete source code can be downloaded from the IBM Redbooks Web site. Refer to Appendix G, "Additional material" on page 701 for the details.

F.1 Table definition

The table, index, and referential integrity definition of the ORA_EMP database are as follows:

```
CREATE TABLE "ACCOUNTS" (
    "ACCT_ID" NUMBER(3) NOT NULL,
    "DEPT_CODE" CHAR(3) NOT NULL,
    "ACCT_DESC" VARCHAR2(2000),
    "MAX_EMPLOYEES" NUMBER(3),
    "CURRENT_EMPLOYEES" NUMBER(3),
    "NUM_PROJECTS" NUMBER(1))
TABLESPACE USER_DATA_TBS;
ALTER TABLE ACCOUNTS ADD
    ( CONSTRAINT ACCOUNTS_DEPT_CODE_ACCT_ID
      PRIMARY KEY ( "DEPT_CODE", "ACCT_ID" ));


CREATE TABLE "DEPARTMENTS" (
    "DEPT_CODE" CHAR(3) NOT NULL,
```

```
                "DEPT_NAME" VARCHAR2(30),
                "TOTAL_PROJECTS" NUMBER,
                "TOTAL_EMPLOYEES" NUMBER)
            TABLESPACE USER_DATA_TBS;
            ALTER TABLE DEPARTMENTS ADD
                ( CONSTRAINT PK_DEPT_CODE PRIMARY KEY ( "DEPT_CODE" ));


            CREATE TABLE "DESTINATION" (
                "KEY" NUMBER(5),
                "VALUE" NUMBER)
            TABLESPACE USER_DATA_TBS;


            CREATE TABLE "EMPLOYEES" (
                "EMP_ID" NUMBER(5) NOT NULL,
                "FIRST_NAME" VARCHAR2(20),
                "LAST_NAME" VARCHAR2(20),
                "DEPARTMENT" VARCHAR2(30),
                "CURRENT_PROJECTS" NUMBER(3),
                "EMP_MGR_ID" NUMBER(5),
                "DEPT_CODE" CHAR(3) NOT NULL,
                "ACCT_ID" NUMBER(3) NOT NULL,
                "OFFICE_ID" NUMBER(5),
                "BAND" CHAR(1))
            TABLESPACE USER_DATA_TBS;
            ALTER TABLE EMPLOYEES ADD
                ( CONSTRAINT SYS_C0012108 PRIMARY KEY ( "EMP_ID" ));


            CREATE TABLE "EMP_PHOTO" (
                "EMP_ID" NUMBER(5) NOT NULL,
                "PHOTO_FORMAT" VARCHAR2(10) NOT NULL,
                "PICTURE" BLOB)
            TABLESPACE USER_DATA_TBS;
            ALTER TABLE EMP_PHOTO ADD
                ( CONSTRAINT EMP_PHOTO_PK11058611148823 PRIMARY KEY ( "EMP_ID" ));


            CREATE TABLE "EMP_PHOTO2" (
                "EMP_ID" NUMBER(5),
                "PHOTO_FORMAT" VARCHAR2(10),
                "PICTURE" BLOB)
            TABLESPACE USER_DATA_TBS;


            CREATE TABLE "EMP_RESUME" (
                "EMP_ID" NUMBER(5) NOT NULL,
                "RESUME_FORMAT" VARCHAR2(10),
```

```
        "RESUME" CLOB)
TABLESPACE USER_DATA_TBS;
ALTER TABLE EMP_RESUME ADD
    ( CONSTRAINT EMP_RESUME_UK110058551798461 UNIQUE ( "EMP_ID" ));


CREATE TABLE "LOG_TABLE" (
    "CODE" NUMBER,
    "MESSAGE" VARCHAR2(200),
    "INFO" VARCHAR2(100))
TABLESPACE USER_DATA_TBS;


CREATE TABLE "MANAGER_AUDIT" (
    "CHANGE_TYPE" CHAR(1) NOT NULL,
    "CHANGED_BY" VARCHAR2(8) NOT NULL,
    "TIMESTAMP" DATE NOT NULL,
    "OLD_EMPLOYEE_ID" NUMBER(5),
    "OLD_DEPT_CODE" CHAR(3),
    "OLD_ACCT_ID" NUMBER(3),
    "OLD_BAND" CHAR(1),
    "NEW_EMPLOYEE_ID" NUMBER(5),
    "NEW_DEPT_CODE" CHAR(3),
    "NEW_ACCT_ID" NUMBER(3),
    "NEW_BAND" CHAR(1))
TABLESPACE USER_DATA_TBS;


CREATE TABLE "OFFICES" (
    "OFFICE_ID" NUMBER(5) NOT NULL,
    "BUILDING" VARCHAR2(15),
    "OFFICE_NUMBER" NUMBER(4),
    "NUMBER_SEATS" NUMBER(4),
    "DESCRIPTION" VARCHAR2(50))
TABLESPACE USER_DATA_TBS;
ALTER TABLE OFFICES ADD
    ( CONSTRAINT SYS_C0012109 PRIMARY KEY ( "OFFICE_ID" ));


CREATE TABLE "TEMP_TABLE" (
    "NUM_COL" NUMBER,
    "CHAR_COL" VARCHAR2(60))
TABLESPACE USER_DATA_TBS;


ALTER TABLE ACCOUNTS ADD
    ( CONSTRAINT FK_ACC_DEPT_CODE FOREIGN KEY ( "DEPT_CODE" )
      REFERENCES DEPARTMENTS ( "DEPT_CODE" ) );
```

```
ALTER TABLE EMPLOYEES ADD
    ( CONSTRAINT FK_EMP_MGR_ID FOREIGN KEY ( "EMP_MGR_ID" )
      REFERENCES EMPLOYEES ( "EMP_ID" ) );


ALTER TABLE EMPLOYEES ADD
    ( CONSTRAINT FK_EMP_OFFICE_ID FOREIGN KEY ( "OFFICE_ID" )
      REFERENCES OFFICES ( "OFFICE_ID" ) );


ALTER TABLE EMPLOYEES ADD
    ( CONSTRAINT M_DEPT_CODE_ACCT_ID FOREIGN KEY ( "DEPT_CODE", "ACCT_ID" )
      REFERENCES ACCOUNTS ( "DEPT_CODE", "ACCT_ID" ) );


ALTER TABLE EMP_PHOTO ADD
    ( CONSTRAINT FK_EMP_PHOTO_ID FOREIGN KEY ( "EMP_ID" )
      REFERENCES EMPLOYEES ( "EMP_ID" ) ON DELETE CASCADE );


ALTER TABLE EMP_RESUME ADD
    ( CONSTRAINT FK_EMP_RESUME_ID FOREIGN KEY ( "EMP_ID" )
      REFERENCES EMPLOYEES ( "EMP_ID" ) ON DELETE CASCADE );



CREATE INDEX IND_ACCT_ID
 ON ACCOUNTS ( "ACCT_ID" ) TABLESPACE USER_IND_TBS;

CREATE INDEX IND_DEPT_NAME
 ON DEPARTMENTS ( "DEPT_NAME" ) TABLESPACE USER_IND_TBS;

CREATE INDEX IND_EMP_NAME
 ON EMPLOYEES ( "LAST_NAME" ) TABLESPACE USER_IND_TBS;


CREATE INDEX IND_LOG_CODE
 ON LOG_TABLE ( "CODE" ) TABLESPACE USER_IND_TBS;

CREATE INDEX IND_OFFICE_BLD
 ON OFFICES ( "BUILDING" ) TABLESPACE USER_IND_TBS;
```

F.2 View definition

There are two views in the ORA_EMP database:

```
CREATE VIEW employees_offices("DEPT_CODE", "ACCT_ID", "BUILDING", "OFFICE_ID")
AS
SELECT dept_code,
       acct_id,
       building,
       offices.office_id
FROM   offices, employees
WHERE offices.office_id = employees.office_id;


CREATE VIEW office_summary("BUILDING", "TOTAL_SEATS") AS
SELECT building,
       sum(number_seats) total_seats
FROM offices
   GROUP BY building;
```

F.3 Procedure and functions

The stored procedures and user defined function in the ORA_EMP database are
as follows:

```
CREATE OR REPLACE FUNCTION AccountFull (
  p_dept_code accounts.dept_code%TYPE,
  p_acct_id    accounts.acct_id%TYPE)
RETURN BOOLEAN IS
  v_CurrentEmployees NUMBER;
  v_MaxEmployees     NUMBER;
  v_ReturnValue      BOOLEAN;
  v_FullPercent      CONSTANT NUMBER := 90;
BEGIN
  -- Get the current and maximum employees for the requested
  -- acct_id.
  SELECT current_employees, max_employees
  INTO v_CurrentEmployees, v_MaxEmployees
  FROM accounts
  WHERE dept_code = p_dept_code
    AND acct_id = p_acct_id;
  -- If the account is more full than the percentage given by
  -- v_FullPercent, return TRUE. Otherwise, return FALSE.
  IF (v_CurrentEmployees / v_MaxEmployees * 100) > v_FullPercent
  THEN
    v_ReturnValue := TRUE;
  ELSE
    v_ReturnValue := FALSE;
  END IF;
  RETURN v_ReturnValue;
END AccountFull;
```

```
              /


CREATE OR REPLACE FUNCTION AVERAGEBAND (
  p_Department IN employees.department%TYPE,
  p_ACCT_ID IN employees.ACCT_ID%TYPE)
RETURN CHAR AS
  v_AverageBAND CHAR(1);
  v_NumericBAND NUMBER;
  v_NumberEmployees NUMBER;
BEGIN
/* First we need to see how many employees there are for
   this account. If there aren't any, we need to raise an error. */
  SELECT COUNT(*)
    INTO v_NumberEmployees
    FROM employees
    WHERE department = p_Department
      AND acct_id = p_ACCT_ID;

  IF v_NumberEmployees = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'No employees exist for ' ||
      p_Department || ' ' || p_ACCT_ID);
  END IF;

  /* Since bands are stored as letters, we can't use the AVG
     function directly on them. Instead, we can use the DECODE
     function to convert the bands to numeric values,
     and take the average of those. */

  SELECT AVG(DECODE(band, '1', 1,
                          '2', 2,
                          '3', 3,
                          '4', 4,
                          '5', 5))
    INTO v_NumericBAND
    FROM employees
    WHERE department = p_Department
    AND acct_id = p_ACCT_ID;

  /* v_NumericBAND now contains the average band, as a number from
     1 to 5. We need to convert this back into a letter. The DECODE
     function can be used here as well. Note that we are SELECTing
     the result into v_AverageBand rather than assigning to it,
     because the DECODE function is only legal in an SQL statement. */

  SELECT DECODE(ROUND(v_NumericBAND), 5, 'A',
                                      4, 'B',
                                      3, 'C',
```

```
                                              2, 'D',
                                              1, 'E')
     INTO v_AverageBand FROM dual;
   RETURN v_AverageBand;
END AverageBand;
/



CREATE OR REPLACE FUNCTION COUNTPROJECTS    (
  /* Returns the number of projects in which the employee
     identified by p_emp_ID is currently engaged */
  p_empID IN employees.emp_ID%TYPE)
RETURN NUMBER AS
  v_TotalProjects NUMBER;
  -- Total number of projects
  v_AccountProjects NUMBER;
  -- projects for one account
  CURSOR c_DeptAccts IS
    SELECT dept_code, acct_id
      FROM employees
      WHERE emp_id = p_empID;
BEGIN
  FOR v_AccountRec IN c_DeptAccts LOOP
    -- Determine the projects for this account.
    SELECT num_projects
      INTO v_AccountProjects
      FROM accounts
      WHERE dept_code = v_AccountRec.dept_code
      AND acct_id = v_AccountRec.acct_id;
    -- Add it to the total so far.
    v_Totalprojects := v_Totalprojects + v_AccountProjects;
  END LOOP;

  RETURN v_Totalprojects;
END CountProjects;
/



CREATE OR REPLACE FUNCTION MAXPROJECTS
     (p_dept_code IN employees.dept_code%TYPE)
  RETURN NUMBER
IS
  CURSOR projcur IS
    SELECT MAX(current_projects) maxprojects
     FROM employees
     WHERE p_dept_code = dept_code;
```

```
      projrec projcur%ROWTYPE;

  BEGIN

    OPEN projcur;
    FETCH projcur INTO projrec;
    RETURN projrec.maxprojects;

  END maxprojects;
  /



  CREATE OR REPLACE PROCEDURE AddNewEmployee (
    p_FirstName  employees.first_name%TYPE,
    p_LastName   employees.last_name%TYPE,
    p_Department    employees.department%TYPE) AS
  BEGIN

    -- Insert a new row in the employees table. Use
    -- employee_sequence to generate the new employee ID, and
    -- 0 for current_projects.

    INSERT INTO employees (emp_id, first_name, last_name,
                           department, current_projects)
      VALUES (employee_sequence.nextval, p_FirstName, p_LastName,
              p_Department, 0);
    COMMIT;
  END AddNewEmployee;
  /



  CREATE OR REPLACE PROCEDURE Assign (
    /* Promotes the employee identified by the p_EmployeeID parameter in
       the account identified by the p_Dept_code and p_AcctId parameters.
       Before calling AccountPackage.AddEmployee, which actually adds
       the employee to the account, this procedure verifies that there is
       room in the account, and that the account exists. */
    p_EmployeeID IN employees.emp_id%TYPE,
    p_Dept_code IN accounts.dept_code%TYPE,
    p_AcctId IN accounts.acct_id%TYPE) AS

    v_CurrentEmployees NUMBER;
    -- Current number of empoloyees in the account
    v_MaxEmployees NUMBER;
    -- Maximum number of employees in the account

  BEGIN
```

```
      /* Determine the current number of employees registered, and
         the maximum number of employees allowed to be hired for
         this dept. */
   SELECT current_employees, max_employees
     INTO v_CurrentEmployees, v_MaxEmployees
     FROM accounts
     WHERE acct_id = p_AcctId
     AND dept_code = p_Dept_code;

   /* Make sure there is enough room for this additional employees. */
   IF v_CurrentEmployees + 1 > v_MaxEmployees THEN
     RAISE_APPLICATION_ERROR(-20000, 'Can''t assign more employees to ' ||
        p_Dept_code || ' ' || p_AcctId);
   END IF;

   /* Add the employee to the account. */

   AccountPackage.AddEmployee(p_EmployeeID, p_Dept_code, p_AcctId);

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    /* Account information passed to this procedure doesn't exist.
       Raise an error to let the calling program know of this. */
    RAISE_APPLICATION_ERROR(-20001, p_Dept_code || ' ' || p_AcctId ||
       ' doesn''t exist!');
END Assign;
/



CREATE OR REPLACE PROCEDURE  EMPLOYEEDYNAMICQUERY  (
  /* Uses DBMS_SQL to query the employees table, and puts the
   * results in temp_table. The first names, last names, and
   * majors are inserted for up to two majors inputted.
   */

  p_department1 IN employees.department%TYPE DEFAULT NULL,

  p_department2 IN employees.department%TYPE DEFAULT NULL) AS

  v_CursorID   INTEGER;
  v_SelectStmt VARCHAR2(500);
  v_FirstName  employees.first_name%TYPE;
  v_LastName   employees.last_name%TYPE;
  v_Department employees.department%TYPE;
  v_Dummy      INTEGER;


BEGIN
```

```
                    -- Open the cursor for processing.

                    v_CursorID := DBMS_SQL.OPEN_CURSOR;



                    -- Create the query string.
                    v_SelectStmt := 'SELECT first_name, last_name, department
                                        FROM employees
                                        WHERE department IN (:d1, :d2)
                                        ORDER BY v_Department, last_name';



                    -- Parse the query.

                    DBMS_SQL.PARSE(v_CursorID, v_SelectStmt, DBMS_SQL.NATIVE);



                    -- Bind the input variables.
                    DBMS_SQL.BIND_VARIABLE(v_CursorID, ':d1', p_department1);
                    DBMS_SQL.BIND_VARIABLE(v_CursorID, ':d2', p_department2);



                    -- Define the select list items.

                    DBMS_SQL.DEFINE_COLUMN(v_CursorID, 1, v_FirstName, 20);
                    DBMS_SQL.DEFINE_COLUMN(v_CursorID, 2, v_LastName, 20);
                    DBMS_SQL.DEFINE_COLUMN(v_CursorID, 3, v_Department, 30);



                    -- Execute the statement. We don't care about the return
                    -- value, but we do need to declare a variable for it.

                    v_Dummy := DBMS_SQL.EXECUTE(v_CursorID);
                    -- This is the fetch loop.
                    LOOP

                      -- Fetch the rows into the buffer, and also check for the exit
                      -- condition from the loop.

                      IF DBMS_SQL.FETCH_ROWS(v_CursorID) = 0 THEN

                        EXIT;

                      END IF;

                      -- Retrieve the rows from the buffer into PL/SQL variables.

                      DBMS_SQL.COLUMN_VALUE(v_CursorID, 1, v_FirstName);
```

```
          DBMS_SQL.COLUMN_VALUE(v_CursorID, 2, v_LastName);
          DBMS_SQL.COLUMN_VALUE(v_CursorID, 3, v_Department);

        -- Insert the fetched data into temp_table.

        INSERT INTO temp_table (char_col)
          VALUES (v_FirstName || ' ' || v_LastName || ' is a ' ||
                  v_Department || ' department.');
    END LOOP;


    -- Close the cursor.
    DBMS_SQL.CLOSE_CURSOR(v_CursorID);


    -- Commit our work.

    COMMIT;

EXCEPTION
  WHEN OTHERS THEN

    -- Close the cursor, then raise the error again.

     DBMS_SQL.CLOSE_CURSOR(v_CursorID);
     RAISE;

END EmployeeDynamicQuery;
/



CREATE OR REPLACE PROCEDURE SELECTROW
    (

        pEmp_ID   IN   EMPLOYEES.EMP_ID%TYPE,
        pRow      OUT REFPKG.RCT1
    )

    IS
    BEGIN

       OPEN pRow FOR

        SELECT FIRST_NAME, LAST_NAME, DEPARTMENT, BAND
        FROM EMPLOYEES
        WHERE Emp_ID = Emp_ID;

    END;
```

```
CREATE OR REPLACE PROCEDURE ShowFullAccounts AS
  CURSOR c_Accounts IS
    SELECT dept_code, acct_id FROM Accounts;

BEGIN

  FOR v_AccountRecord IN c_Accounts  LOOP
    -- Record all Accounts which don't have very much room left
    -- in temp_table.

    IF AccountFull(v_AccountRecord.dept_code, v_AccountRecord.acct_id) THEN
      INSERT INTO temp_table (char_col) VALUES
        (v_AccountRecord.dept_code || ' ' || v_AccountRecord.acct_id ||
        ' is almost full!');
    END IF;

  END LOOP;

END ShowFullAccounts;
/
```

F.4 Packages

The package procedures in the ORA_EMP database are as follows:

```
CREATE OR REPLACE PACKAGE AccountPackage AS

  -- Add a new Employee into the specified Account.

  PROCEDURE AddEmployee(p_EmployeeID  IN Employees.emp_id%TYPE,
                        p_dept_code   IN accounts.dept_code%TYPE,
                        p_acct_id     IN accounts.acct_id%TYPE);

  -- Removes the specified Employee from the specified Account.

  PROCEDURE RemoveEmployee(p_EmployeeID  IN Employees.emp_id%TYPE,
                           p_dept_code   IN accounts.dept_code%TYPE,
                           p_acct_id     IN accounts.acct_id%TYPE);


  PROCEDURE AccountList(p_dept_code   IN  accounts.dept_code%TYPE,
                        p_acct_id     IN  accounts.acct_id%TYPE,
                        p_NumEmployees OUT NUMBER);
```

```
      END AccountPackage;



CREATE OR REPLACE PACKAGE BODY AccountPackage AS

  -- Add a new Employee for the specified account.

  PROCEDURE AddEmployee(p_EmployeeID  IN Employees.emp_id%TYPE,
                        p_dept_code IN accounts.dept_code%TYPE,
                        p_acct_id    IN accounts.acct_id%TYPE) IS

  BEGIN

    INSERT INTO Employees (Emp_id, dept_code, acct_id)
      VALUES (p_EmployeeID, p_dept_code, p_acct_id);
    COMMIT;

  END AddEmployee;

  -- Removes the specified Employee from the specified account.

  PROCEDURE RemoveEmployee(p_EmployeeID  IN Employees.emp_id%TYPE,
                           p_dept_code IN accounts.dept_code%TYPE,
                           p_acct_id    IN accounts.acct_id%TYPE) IS
e_EmployeeNotRegistered EXCEPTION;

  BEGIN

    DELETE FROM Employees
          WHERE Emp_id = p_EmployeeID
          AND dept_code = p_dept_code
          AND acct_id = p_acct_id;

    -- Check to see if the DELETE operation was successful.
    -- If it didn't match any rows, raise an error.

    IF SQL%NOTFOUND THEN
      RAISE e_EmployeeNotRegistered;
    END IF;

    COMMIT;

  END RemoveEmployee;


  PROCEDURE AccountList(p_dept_code  IN  accounts.dept_code%TYPE,
                        p_acct_id     IN  accounts.acct_id%TYPE,
                        p_NumEmployees OUT NUMBER) IS
```

```
                 v_EmployeeID  Employees.Emp_id%TYPE;
                 -- Local cursor to fetch the registered Employees.
                 CURSOR c_RegisteredEmployees IS
                   SELECT Emp_id
                     FROM Employees
                    WHERE dept_code = p_dept_code
                    AND acct_id = p_acct_id;
             BEGIN
               /* p_NumEmployees will be the table index. It will start at
                  0, and be incremented each time through the fetch loop.
                  At the end of the loop, it will have the number of rows
                  fetched, and therefore the number of rows returned in
                  p_IDs. */
               p_NumEmployees := 0;
               OPEN c_RegisteredEmployees;
               LOOP
                 FETCH c_RegisteredEmployees INTO v_EmployeeID;
                 EXIT WHEN c_RegisteredEmployees%NOTFOUND;
                 p_NumEmployees := p_NumEmployees + 1;
               END LOOP;
           END AccountList;
     END AccountPackage;



CREATE OR REPLACE PACKAGE
               "REFPKG" AS
     TYPE RCT1 IS REF CURSOR;


END REFPKG;
/
```

F.5 Triggers

The triggers in the ORA_EMP database are as follows:

```
CREATE TRIGGER CreateEmployeeID
   BEFORE INSERT ON employees
   FOR EACH ROW


BEGIN
   /* Fill in the emp_id field of employees with the next value from
      employee_sequence. Since emp_id is a column in employees, :new.emp_id
      is a valid reference. */
   SELECT employee_sequence.nextval
```

```
      INTO :new.emp_id
      FROM dual;
END CreateEmployeeID;
/


CREATE TRIGGER EmployeesOfficesInsert
  INSTEAD OF INSERT ON employees_offices


DECLARE
  v_office_ID offices.office_id%TYPE;
BEGIN
  -- First determine the office ID
  SELECT office_id
    INTO v_office_ID
    FROM offices
    WHERE building = :new.building
    AND office_id = :new.office_id;
  -- And now update the account
  UPDATE employees
    SET office_id = v_office_ID
    WHERE dept_code = :new.dept_code
    AND acct_id = :new.acct_id;
END EmployeesOfficesInsert;
/


CREATE TRIGGER InsertEmployee
  BEFORE INSERT ON employees
  FOR EACH ROW

DECLARE
  v_num_projects accounts.num_projects%TYPE;
BEGIN
  SELECT num_projects
    INTO v_num_projects
    FROM accounts
    WHERE dept_code = :new.dept_code
    AND acct_id = :new.acct_id;

  :new.current_projects := v_num_projects;

  UPDATE accounts
    SET current_employees = current_employees + 1
    WHERE dept_code = :new.dept_code
    AND acct_id = :new.acct_id;
END InsertEmployees;
/
```

```
CREATE TRIGGER ManagersChange
  BEFORE INSERT OR DELETE OR UPDATE ON Employees
  FOR EACH ROW


DECLARE
  v_ChangeType CHAR(1);
BEGIN
  /* Use 'I' for an INSERT, 'D' for DELETE, and 'U' for UPDATE. */
  IF INSERTING THEN
    v_ChangeType := 'I';
  ELSIF UPDATING THEN
    v_ChangeType := 'U';
  ELSE
    v_ChangeType := 'D';
  END IF;
  /* Record all the changes made to managers in
     managers_audit. Use SYSDATE to generate the timestamp, and
     USER to return the userid of the current user. */
  INSERT INTO manager_audit
    (change_type, changed_by, timestamp,
     old_employee_id, old_dept_code, old_acct_id, old_band,
     new_employee_id, new_dept_code, new_acct_id, new_band)
  VALUES
    (v_ChangeType, USER, SYSDATE,
     :old.emp_mgr_id, :old.dept_code, :old.acct_id, :old.band,
     :new.emp_mgr_id, :new.dept_code, :new.acct_id, :new.band);
END ManagersChange;
/


CREATE TRIGGER office_summary_delete
  INSTEAD OF DELETE ON office_summary
  FOR EACH ROW


BEGIN
  -- Delete all of the rows in rooms which match this single row
  -- in room_summary
  DELETE FROM offices
    WHERE building = :old.building;
END office_summary_delete;
/


CREATE TRIGGER UpdateDepartments
AFTER INSERT OR DELETE OR UPDATE ON employees
```

```
DECLARE
  CURSOR c_projects IS
    SELECT  dept_code
            ,COUNT(*) AS total_employees
            ,SUM(current_projects) AS total_projects
      FROM employees
      GROUP BY dept_code;
BEGIN
  FOR v_project_rec in c_projects LOOP
     UPDATE departments
     SET  total_projects  = v_project_rec.total_projects
          ,total_employees = v_project_rec.total_employees
     WHERE dept_code = v_project_rec.dept_code;
  END LOOP;
END UpdateDepartments;
/


CREATE TRIGGER UpdateEmployees
  BEFORE INSERT OR UPDATE ON employees
  FOR EACH ROW

DECLARE
   v_dept_name  departments.dept_name%TYPE;
   v_dept_code  departments.dept_code%TYPE;
BEGIN
   v_dept_code := :NEW.dept_code;
   SELECT dept_name
     INTO v_dept_name
     FROM departments
     WHERE dept_code = v_dept_code;
   :NEW.department := v_dept_name;
END UpdateEmployees;
/
```

G

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG247048`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com``/redbooks`

Select **Additional materials** and open the directory that corresponds with the book form number, SG247048.

Using the Web material

The additional Web material that accompanies this book includes the following files:

convert_load.zip   Perl scripts to convert Oracle control files. Includes: conv_ctl.pl

	gen_load_update.pl
data_unload.zip	Shell scripts to unload data from Oracle tables. Includes: data_unload.sh count.awk desc.awk
db2_emp.zip	Scrips to create DB2 example database. Includes: crtdb.sql - create database crttbs.sql - create tablespaces
db2_xml.zip	Scrips to create DB2 XML tables. Includes: crtab.sql customer.xsd po_xml_tab.ddl po_xml_tab-insert.sql
db2bkp.sh	Script to backup DB2 example database
exp.sh	Script to export Oracle logical DB
export_table.zip	Sample stored procedure to unload Oracle data. Includes: export_table.sql
hierarch_query.zip	DB2 functions for recursive SQL. Includes: hierachical_query.fnc
put_line.zip	Zipped user-define function samples to enable file output from pure SQL for AIX, HP, Linux, Sun Solaris and Windows. Includes: put_lin_readme.pdf put_line_aix.db2v72.tar.Z put_line_aix.db2v81.tar.Z put_line_hpux.db2v72.tar.Z put_line_solaris.db2v72.tar.Z put_line_linux.db2v72.tar.Z put_lin_w2k.ZIP
UDF.zip	Zipped user-define function samples Includes: CONCAT.txt HexToRaw.txt NVL.txt Ora_CHR.txt Ora_LTRIM_RTRIM.txt TO_CHAR_N.txt TO_CHAR_TX.txt TO_DATE_S1.txt TO_DATE_S2.txt MigrateDECODE_proc.txt
ora_emp.zip	Scrips to create Oracle example database. Includes: packages.src procedures_functions.src sequences.src tables.src

```
triggers.src
views.srcl
```

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 5 MB minimum
Operating System: Windows 2000
Processor: Intel® 386 or higher
Memory: 16 MB

To download to AIX Linux or UNIX, use the equivalent or higher system capacity as specified for Windows

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see "How to get IBM Redbooks" on page 708. Note that some of the documents referenced here may be available in softcopy only.

► *DB2 UDB Exploitation of the Windows Environment,* SG24-6893

► *Up and Running with DB2 for Linux,* SG24-6899

► *DB2 UDB Evaluation Guide for Linux and Windows*, SG24-6934

► *Scaling DB2 UDB on Windows Server 2003,* SG24-7019

Other publications

These publications are also relevant as further information sources:

► *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS,* ISBN 0131477005

IBM - DB2 9
► *What's New*, SC10-4253

► *Administration Guide: Implementation*, SC10-4221

► *Administration Guide: Planning*, SC10-4223

► *Administrative API Reference*, SC10-4231

► *Administrative SQL Routines and Views*, SC10-4293

► *Administration Guide for Federated Systems,* SC19-1020

► *Call Level Interface Guide and Reference, Volume 1*, SC10-4224

► *Call Level Interface Guide and Reference, Volume 2*, SC10-4225

► *Command Reference,* SC10-4226

► *Data Movement Utilities Guide and Reference*, SC10-4227

- ► *Data Recovery and High Availability Guide and Reference*, SC10-4228
- ► *Developing ADO.NET and OLE DB Applications*, SC10-4230
- ► *Developing Embedded SQL Applications*, SC10-4232
- ► *Developing Java Applications*, SC10-4233
- ► *Developing Perl and PHP Applications*, SC10-4234
- ► *Developing SQL and External Routines*, SC10-4373
- ► *Getting Started with Database Application Development*, SC10-4252
- ► *Getting started with DB2 installation and administration on Linux and Windows*, GC10-4247
- ► *Message Reference Volume 1*, SC10-4238
- ► *Message Reference Volume 2*, SC10-4239
- ► *Migration Guide*, GC10-4237
- ► *Performance Guide*, SC10-4222
- ► *Query Patroller Administration and User's Guide*, GC10-4241
- ► *Quick Beginnings for DB2 Clients*, GC10-4242
- ► *Quick Beginnings for DB2 Servers*, GC10-4246
- ► *Spatial Extender and Geodetic Data Management Feature User's Guide and Reference*, SC18-9749
- ► *SQL Guide*, SC10-4248
- ► *SQL Reference, Volume 1,* SC10-4249
- ► *SQL Reference, Volume 2,* SC10-4250
- ► *System Monitor Guide and Reference,* SC10-4251
- ► *Troubleshooting Guide*, GC10-4240
- ► *Visual Explain Tutorial*, SC10-4319
- ► *XML Extender Administration and Programming*, SC18-9750
- ► *XML Guide*, SC10-4254
- ► *XQuery Reference*, SC18-9796
- ► *DB2 Connect User's Guide*, SC10-4229
- ► *DB2 9 PureXML Guide*, SG24-7315
- ► *Quick Beginnings for DB2 Connect Personal Edition*, GC10-4244
- ► *Quick Beginnings for DB2 Connect Servers*, GC10-4243

IBM - DB2 8.2

- *What's New V8,* SC09-4848-01

- *Administration Guide: Implementation V8*, SC09-4820-01

- *Administration Guide: Performance V8*, SC09-4821-01

- *Administration Guide: Planning V8*, SC09-4822-01

- *Application Development Guide: Building and Running Applications V8*, SC09-4825-01

- *Application Development Guide: Programming Client Applications V8*, SC09-4826-01

- *Application Development Guide: Programming Server Applications V8*, SC09-4827-01

- *Call Level Interface Guide and Reference, Volume 1, V8*, SC09-4849-01

- *Call Level Interface Guide and Reference, Volume 2, V8*, SC09-4850-01

- *Command Reference V8*, SC09-4828-01

- *Data Movement Utilities Guide and Reference V8*, SC09-4830-01

- *Data Recovery and High Availability Guide and Reference V8*, SC09-4831-01

- *Guide to GUI Tools for Administration and Development*, SC09-4851-01

- *Installation and Configuration Supplement V8*, GC09-4837-01

- *Quick Beginnings for DB2 Clients V8*, GC09-4832-01

- *Quick Beginnings for DB2 Servers V8*, GC09-4836-01

- *Replication and Event Publishing Guide and Reference*, SC18-7568

- *SQL Reference, Volume 1, V8,* SC09-4844-01

- *SQL Reference, Volume 2, V8,* SC09-4845-01

- *System Monitor Guide and Reference V8,* SC09-4847-01

- *Data Warehouse Center Application Integration Guide Version 8 Release 1*, SC27-1124-01

- *DB2 XML Extender Administration and Programming Guide Version 8 Release 1*, SC27-1234-01

- *Federated Systems PIC Guide Version 8 Release 1*, GC27-1224-01

Online resources

These Web sites are also relevant as further information sources:

- DB2 Information Center

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

► Database and Information Management home page

http://www.ibm.com/software/data/

► DB2 Universal Database home page

http://www.ibm.com/software/data/db2/udb/

► DB2 Technical Support

http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/index.d2w/report

► DB2 online library

http://www.ibm.com/db2/library

Migration to DB2

► IBM Software Migration Project Office

http://www.ibm.com/software/solutions/softwaremigration/contacts.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

! 157
$INSTHOME 83
%FOUND 192
%ISOPEN 189
%NOTFOUND 190
%ROWNCOUNT 190
%TYPE 161
.c 259
.exp 260
.rpt 72
.sqc 259
@ 157

Numerics

7x24 267
8i 91

A

acceptance tests 452
ACCOUNTS table 426
accounts.dat datafile 425–426
active subagent 20
ADMIN_COPY_SCHEMA 430
ADMIN_COPY_SCHEMA procedure 432
administration scripts 437
    conversion 437
administrator 57
AFTER trigger 154
AIX 76
alert log file 20
ALL clause 555
ALL PRIVILEGES clause 555
annotate 420
annotation 338
ANSI/IEEE 448
API 287, 294
applets 298
Application Shared Memory 15
applications 420
applications activity 589
    application with oldest transaction 590

current application clients 589
    lock wait 592
    SQL statements 594
architectural assessment 36
archive logging 558
archive logging recovery mode 559
ARCHIVELOG mode 559, 561
arrays 291
assessment 39
attribute 231
AUTOMATIC 569
automatic database configuration 567
automatic database management 565
AUTOMATIC parameters 567
automatic storage 572
    automatic BACKUP 575
    automatic statistics collection 574
    REORG on tables and indexes 573
    utility throttling 575
AUTOMATIC STORAGE NO 572
auto-start instance 524
    db2iauto command 524
    example 525
    Linux operating system 524
    Oracle equivalent command 524
    UNIX-based system 524
    Windows operating system 524
availability 266, 286
awk 272

B

background process 17, 521
backup 441
backup scripts 441
backup scripts, conversion 441
Backup Wizard 443
BEFORE trigger 154
BEGIN BACKUP command 560
BIGINT 290
bldmapp.bat 296
bldrtn 259
BP_TEST buffer pool 553
buffer pools 14

IBM

Redbooks

Oracle to DB2 Conversion Guide for Linux, UNIX, and Windows

Oracle to DB2 Conversion Guide
for Linux, UNIX, and Windows

Step-by-step guide to migrating from Oracle to DB2 9.1

Conversion examples - including XML conversion

Step-by-step guide to MTK tool usage

IBM DB2 has long been known for its technology leadership. This IBM Redbooks publication, intended for technical staff who are involved in an Oracle to DB2 conversion project, is an informative guide that describes how to migrate the database system from Oracle to DB2 Version 9 on Linux, UNIX, and Microsoft Windows platforms.

This book provides conversion methodology and step-by-step instructions for installing and using IBM Migration Toolkit (MTK) to port the database objects and data from Oracle to DB2. It illustrates, with examples, how to convert the stored procedures, functions, and triggers. Application programming and conversion considerations are discussed, along with the differences in features and functionality of the two products.

In addition, you can find script conversion samples for data loading, database administration, and reports that are useful for DBAs. The testing section provides procedures and tips for conversion testing and database tuning. The laboratory examples are performed under Oracle 10g and DB2 Version 9. However, the migration process and examples can be applied to Oracle 7, 8, and 9i.