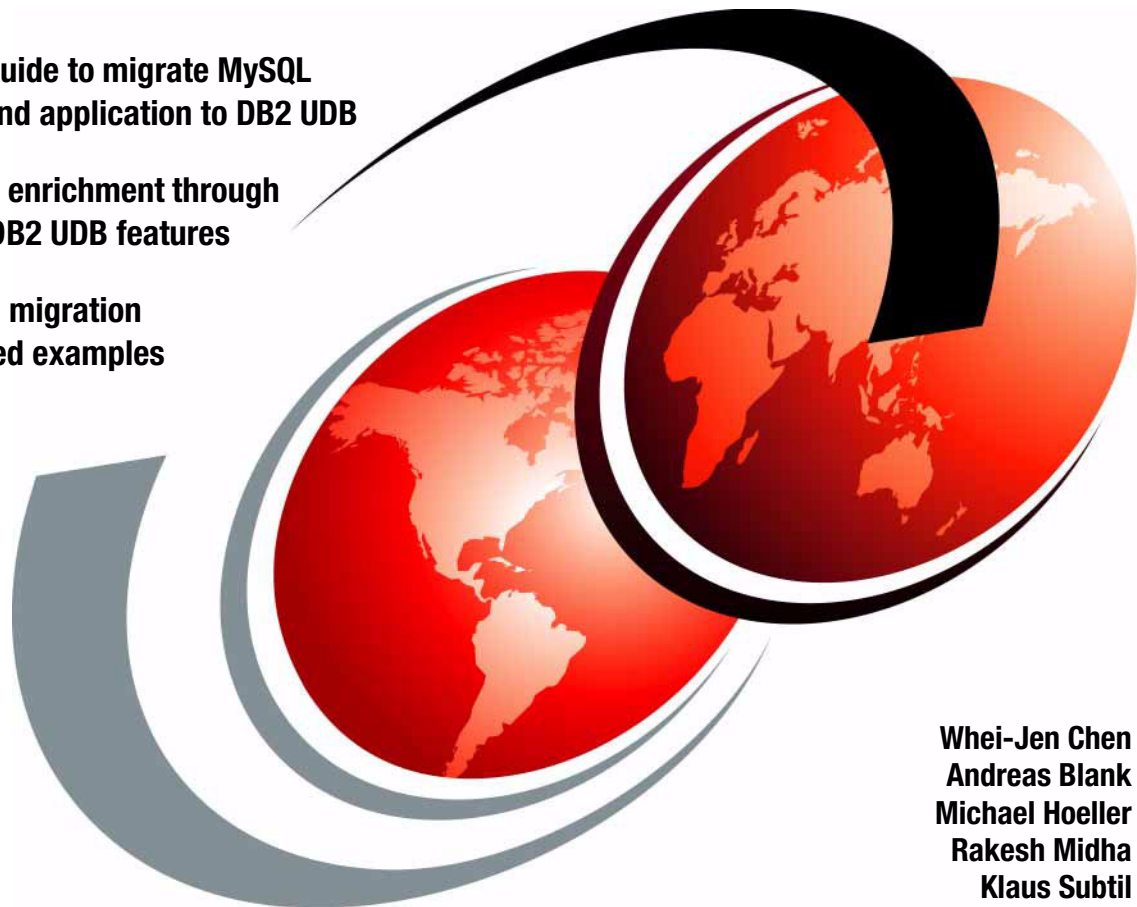


MySQL to DB2 UDB Conversion Guide

Complete guide to migrate MySQL
database and application to DB2 UDB

Application enrichment through
advanced DB2 UDB features

Application migration
with detailed examples



Whei-Jen Chen
Andreas Blank
Michael Hoeller
Rakesh Midha
Klaus Subtil



International Technical Support Organization

MySQL to DB2 UDB Conversion Guide

May 2004

Note: Before using this information and the product it supports, read the information in “Notices” on page xiii.

First Edition (May 2004)

This edition applies to IBM DB2 UDB Version 8.1 for Linux, UNIX, and Windows, MySQL Version 4.0, SuSE 8.0 and Red Hat 8.0.

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	ix
Tables	xi
Notices	xiii
Trademarks	xiv
Preface	xv
The team that wrote this redbook	xvi
Become a published author	xviii
Comments welcome	xviii
Chapter 1. Introduction	1
1.1 DB2 Universal Database	2
1.1.1 Product overview	2
1.1.2 DB2 UDB for Linux, UNIX, and Windows architecture	6
1.1.3 DB2 utilities	19
1.1.4 DB2 database access	21
1.2 MySQL database	28
1.2.1 MySQL architecture	29
1.2.2 MySQL design and SQL compliance	32
1.2.3 MySQL utilities	38
1.2.4 MySQL application programming interfaces (API)	41
Chapter 2. Planning the migration from MySQL to DB2 UDB	45
2.1 Migration project planning overview	46
2.1.1 Benefits of migrating to DB2 UDB	47
2.1.2 IBM migration offering	48
2.1.3 Education	49
2.2 Application assessment	49
2.3 System planning	51
2.3.1 Software	53
2.3.2 Hardware	53
2.3.3 Migration tools	53
2.4 The migration process	54
2.4.1 Porting preparation and installation	55
2.4.2 Database structure porting	55
2.4.3 Data porting	57
2.4.4 Application porting	59

2.4.5 Basic administration	60
2.4.6 Testing and tuning.	60
Chapter 3. Migration scenario	63
3.1 Application description	64
3.1.1 Steps using the application	64
3.1.2 Database structure	71
3.2 System environment	72
Chapter 4. Installation	75
4.1 DB2 UDB ESE V8.1.4 on Linux.	76
4.1.1 System requirements	76
4.1.2 Installation procedure	78
4.1.3 Instance creation.	81
4.1.4 Client setup on Linux.	82
4.2 Other software product	83
4.2.1 PHP adjustment for Unified ODBC with DB2 support.	83
4.3 MTK installation and usage.	87
4.3.1 MTK prerequisites.	87
4.3.2 MTK installation.	87
Chapter 5. Database porting	89
5.1 Data type mapping	90
5.2 Data Definition Language differences	95
5.2.1 Database manipulation	96
5.2.2 Table manipulation	99
5.2.3 Index manipulation	106
5.3 Other considerations	107
5.4 Porting database	111
5.4.1 Automatic conversion using porting tools	112
5.4.2 Manual porting	114
5.4.3 Metadata transport	116
5.5 Sample database migration.	117
Chapter 6. Data porting.	127
6.1 Considerations concerning data porting	128
6.1.1 Commands and tools supporting data porting	128
6.1.2 Differences in data formats	134
6.1.3 Differences in the user account management.	138
6.2 Sample project: Doing the data porting.	146
6.2.1 Export user data from MySQL.	147
6.2.2 Map MySQL user data to DB2 user data	147
6.2.3 Create DB2 user	148
6.2.4 Export MySQL application data.	149

6.2.5	Convert MySQL application data to DB2 format	149
6.2.6	Import application data into DB2 UDB	150
6.2.7	Basic data checking	150
Chapter 7. Application porting		155
7.1	Differences and similarities in Data Manipulation Language.	156
7.1.1	SELECT syntax.	156
7.1.2	JOIN syntax.	157
7.1.3	UNION Syntax.	158
7.1.4	Subquery syntax	159
7.1.5	Grouping, having, and ordering.	159
7.1.6	Strings	161
7.1.7	Implicit casting of data types	163
7.1.8	String concatenation and NULL values.	166
7.1.9	Record deletion	167
7.1.10	Built-in functions and operators.	168
7.2	Application source conversion.	173
7.2.1	Converting MySQL Perl applications to DB2 UDB	174
7.2.2	Converting MySQL PHP applications to DB2 UDB.	177
7.2.3	Converting MySQL Java applications to DB2 UDB.	188
7.2.4	Converting MySQL C/C++ applications to DB2 UDB	199
7.2.5	Converting MyODBC applications to DB2 UDB	211
7.2.6	Condition handling in DB2.	213
7.2.7	Special conversions	221
7.3	Additional application considerations	226
7.3.1	What is the purpose of locking?	226
7.3.2	Concurrency control and transaction isolation	227
7.3.3	DB2 isolation levels.	227
7.3.4	Locking	229
7.3.5	Specifying the isolation level in DB2	230
Chapter 8. Database administration		235
8.1	Database recovery	236
8.1.1	MySQL recovery	236
8.1.2	DB2 UDB database recovery	237
8.2	Database replication	240
8.3	Data movement	241
8.3.1	MySQL data movement	242
8.3.2	DB2 UDB data movement.	242
8.4	High availability	246
8.5	Automated tasks/jobs	248
8.6	Database configuration	249
8.6.1	MySQL configuration.	249

8.6.2	DB2 UDB configuration	250
8.7	Database management tools	254
8.7.1	MySQL phpMyAdmin and Control Center	256
8.7.2	DB2 UDB Control Center	258
8.7.3	DB2 UDB Web Command Center	260
Chapter 9.	Testing and tuning	263
9.1	Test planning	264
9.1.1	Principles of software tests	264
9.1.2	Test documentation.	264
9.1.3	Test phases.	267
9.1.4	Time planning and time exposure	269
9.2	Data checking techniques	270
9.2.1	IMPORT/LOAD messages	270
9.2.2	Data checking	273
9.3	Code and application testing	275
9.3.1	Application code check	275
9.3.2	Security testing	276
9.3.3	Tools for testing and problem tracking	276
9.4	Troubleshooting.	277
9.4.1	Interpreting DB2 informational messages	277
9.4.2	DB2 diagnostic logs	278
9.4.3	DB2 support information	282
9.4.4	Problem determination tools	285
9.5	Initial tuning	296
9.5.1	Table spaces.	297
9.5.2	Physical placement of database objects	298
9.5.3	Buffer pools	301
9.5.4	Large transactions.	303
9.5.5	SQL execution plan.	308
9.5.6	Configuration Advisor	311
9.5.7	Index Advisor	314
Chapter 10.	Advanced DB2 UDB features	317
10.1	Views.	318
10.2	Stored procedures.	320
10.3	Trigger.	324
10.4	User-defined data types (UDT)	324
10.5	User-defined functions	327
10.6	Materialized query tables (MQT)	328
10.7	Multidimensional clustering (MDC)	331
Appendix A.	Sample code for user defined functions	335
A.1	Sample code for BIT_AND	336

A.2 Sample code for FORMAT function	337
A.3 Sample code for RPAD and LPAD functions	339
A.4 Sample code for GREATEST function	346
A.5 Sample code for LEAST	353
A.6 Sample code for BIT_COUNT	359
A.7 Sample code for SUBSTRING_INDEX.....	360
Related publications	363
IBM Redbooks	363
Other publications	363
Online resources	364
How to get IBM Redbooks	366
Help from IBM	366
Index	367

Figures

0-1	The team, left to right: Andreas, Whei-Jen, Rakesh, Klaus, Michael . .	xvii
1-1	DB2 product overview	3
1-2	DB2 architecture overview	7
1-3	DB2 processes	8
1-4	DB2 Objects relationship	11
1-5	Relationship between instances, databases, and tables	12
1-6	Database partition groups in a database	13
1-7	Relationship between tables and views	15
1-8	Relationship between indexes and tables	16
1-9	Relationship between table space and containers	17
1-10	DB2 directory structure	18
1-11	Application connections to DB2 UDB	24
1-12	Conceptual MySQL architecture	29
1-13	MySQL directory structure	33
1-14	MySQL Application Programming Interfaces	41
2-1	Sample migration scenarios	52
2-2	Steps of the migration process	55
2-3	Database structure porting process	56
2-4	Data porting process	57
3-1	Flow diagram of the sample application	65
3-2	Start page of the Web application	66
3-3	Registration form	67
3-4	Registration information sent to the vendor	67
3-5	Shop or catalog view	68
3-6	Detail item view	69
3-7	Shopping cart view	70
3-8	Request/Order information sent to the vendor	71
3-9	ERD - Diagram	72
4-1	DB2 custom installation with Application Development tools selected .	79
4-2	Db2setup command options	80
4-3	Instance creation option	81
4-4	Graphical user interface for db2isetup	81
4-5	PHP Configuration options	85
4-6	MTK initial screen	88
5-1	MySQL data types	90
5-2	DB2 UDB data types	91
5-3	MySQL table types	100
5-4	Typical DB2 UDB setup	108

5-5	MySQL Application with multiple DBs instead of multiple schemas. . .	109
5-6	MTK startup window	118
5-7	MTK Message window.	119
5-8	MTK success message	119
6-1	Hierarchy of authorities	142
7-1	DB2 type 2 JDBC Driver	190
7-2	DB2 JDBC type 3 driver.	191
7-3	JDBC Universal Driver.	191
7-4	DB2 CLI activities.	202
7-5	DB2 Query processing.	206
7-6	ODBC application conversion from MyODBC to DB2 ODBC Driver . .	212
8-1	Incremental backup	238
8-2	DB2 UDB roll forward restore	240
8-3	DB2 replication center console	241
8-4	Export using control centre	244
8-5	Import using control center	245
8-6	The Task Center console.	249
8-7	Configuration parameter files.	252
8-8	DB2 Configuration Assistant	253
8-9	phpMyAdmin console.	257
8-10	MySQL control center console.	258
8-11	DB2 control center	259
8-12	DB2 Web command center console	260
9-1	Test phases during a migration project	269
9-2	Table definition for Example 9-1	271
9-3	Data file for Example 9-1	271
9-4	Sample db2level output	283
9-5	A Visual Explain access plan graph.	296
9-6	Explaining logical log	299
9-7	Visualizing CHNGPGS_THRESH parameter	303
9-8	Maximum number of locks available for default settings on Linux. . .	304
9-9	Explaining lock snapshot information.	306
9-10	Running RUNSTATS on multiple tables	309
9-11	Selecting tables for RUNSTATS command	310
9-12	RUNSTATS command options	311
9-13	Scheduling Configuration Advisor recommendations	312
9-14	Configuration Advisor recommendations.	313
9-15	The Design Advisor	316
10-1	Creating stored procedures	322
10-2	User defined data types.	326
10-3	One clustering index	331
10-4	MDC table and indexes	332

Tables

1-1	DB2 commands list	19
1-2	Multi-tier configuration examples	23
1-3	Permission information stored in tables	33
4-1	Currently supported Linux distributions, kernels, and libraries	76
4-2	DB2 installation methods	78
5-1	Default data type mapping	94
5-2	MySQL to DB2 UDB conversion of list information statement	110
6-1	Mapping of MySQL to DB2 privileges	145
7-1	Differences in DB2 and MySQL grouping, having and ordering	160
7-2	MySQL and DB2 UDB string related function	162
7-3	SQL 92 functions	168
7-4	ODBC 3.0 functions	169
7-5	MySQL and DB2 UDB operator comparison	170
7-6	Date and Time related functions	170
7-7	Other functions.	171
7-8	MySQL data type mapping to Java data type	197
7-9	DB2 UDB data types mapping to Java types	197
7-10	DB2 isolation level	227
7-11	MySQL and DB2 table comparison	229
8-1	MySQL options files	250
9-1	Aggregations for data migration verification.	275
9-2	List of monitor switches and related DBM parameters	287
9-3	Common snapshot table functions.	291
9-4	Parameters for the autoconfigure command	314

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 illustrates 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. 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 IBM's application programming interfaces.

Trademarks

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

ibm.com®	DB2®	Rational Rose®
iSeries™	DRDA®	Rational Suite®
pSeries®	Everyplace®	Rational®
zSeries®	Informix®	Redbooks™
AIX®	Intelligent Miner™	S/390®
CICS®	IBM®	TestStudio®
Distributed Relational Database Architecture™	IMS™	Tivoli®
DB2 Connect™	Multiprise®	VisualAge®
DB2 Extenders™	Net.Data®	WebSphere®
DB2 Universal Database™	PartnerWorld®	1-2-3®
	POWER™	Redbooks (logo)  ™

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

Preface

This IBM Redbook is an informative guide that describes how to migrate the database system from MySQL to DB2® UDB Version 8.1 on Linux, and how to convert applications to use DB2 UDB instead of MySQL.

This guide presents the best practices in migration strategy and planning, migration tools, porting steps, and practical migration examples. It is intended for technical staff involved in a MySQL to DB2 UDB conversion project.

This redbook is organized as follows:

- ▶ **Chapter 1** gives an introduction to the DB2 UDB and MySQL database systems.
- ▶ **Chapter 2** talks about the planning of a migration project from MySQL to DB2 UDB and considerations, which have to be made before migrating.
- ▶ **Chapter 3** introduces our sample project, which is used as an example for the migration steps throughout the rest of the redbook.
- ▶ **Chapter 4** guides through the installation of DB2 UDB and other software needed for the migration project.
- ▶ **Chapter 5** discusses the porting of the database structure and differences in data types in detail.
- ▶ **Chapter 6** provides information about porting the data from MySQL to DB2 UDB and how to set up the DB2 database security according to the MySQL user account management.
- ▶ **Chapter 7** gives detailed information about porting applications. SQL differences are discussed as well as porting considerations for different programming languages such as Perl, PHP, Java™, and C/C++.
- ▶ **Chapter 8** talks about DB2 database administration.
- ▶ **Chapter 9** details steps for testing and tuning of your migrated application and database system.
- ▶ **Chapter 10** introduces advanced DB2 UDB features that can be used in order to enhance ported applications.

The team that wrote this redbook

This redbook 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.

Andreas Blank is the CEO of LIS Logos InformationsSysteme GmbH (www.lis.de), an IBM Business Partner in Germany. He is trainer at LIS AG (www.lis-ag.ch), an IBM enabled Education Center (ECIS) in Switzerland, which provides classroom and on-site courses in Data Management, Tivoli®, and WebSphere. He has 12 years of IT experience in application development as well as database design, implementation, and administration of different databases on Linux, OS2, and Windows®. He holds a master's degree in Engineering from the Technical University of Darmstadt and is certified for DB2 UDB.

Michael Hoeller is an IT Specialist for DB2 UDB and WebSphere® in IBM Global Services, Austria. He has 12 years of experience in application architecture and development, and database design. He is an IBM Certified e-business Solution Technologist and has worked at IBM for 6 years. His areas of expertise include application development in Java and WebSphere Application Server, WebSphere Commerce and Business Intelligence solution implementation. He holds a master's degree in Business and Computer Science from the J. Kepler University, Linz.

Rakesh Midha is a Software Engineer with IBM Software Labs, Bangalore. He is currently working on IBM WebSphere Business Components development. He has 5 years of technical experience in Java and C++ server-side programming on multiple platforms and various relational database systems like DB2 UDB, Oracle, MySQL, and Microsoft® SQL Server. His area of expertise include designing and development of stand-alone to *n*-tier distributed applications in the field of banking, finance, catalog industry, and order and warehouse management systems. He holds a bachelor's degree in Electronics Engineering from the Punjab University, Chandigarh.

Klaus Subtil is an IT Specialist within the IBM DB2 Business Partner Technical Enablement team. He has more than 15 years experience in the IT industry focusing on relational database technology and application development. Klaus is an IBM Certified Solutions Expert on Universal Database 8.1 Database administration, and holds a master's degree in Business at the Open University

Business School, Milton Keynes, UK. In his current position at Klaus enables independent software vendors and system integrators for IBM's information management software. In addition, he also certifies students for DB2 at universities participating in IBM's Scholars Program.

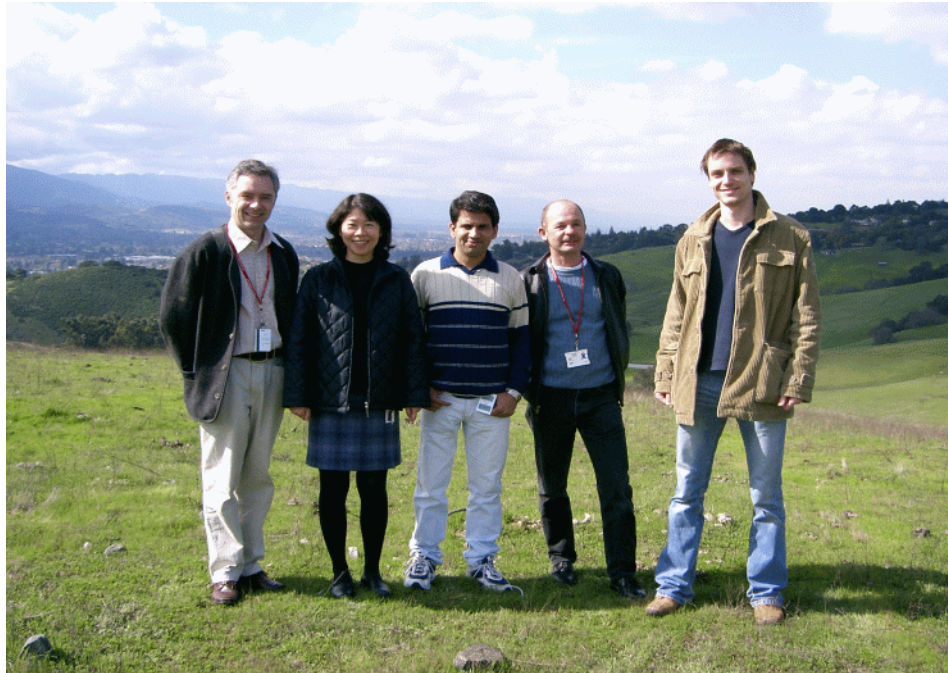


Figure 0-1 The team, left to right: Andreas, Whei-Jen, Rakesh, Klaus, Michael

The team would like to thank the following people for their contributions to this project:

Vicki Martin Petruch
IBM WW Brand Manager - DB2 on Linux

Jean-Jacques Daudenarde
IBM DB2 Migration and Everyplace® Tools, Silicon Valley Laboratory

Wilhelm Friesen
LIS Logos InformationsSysteme GmbH, Germany

Stefan Hummel
IBM Software Group Financial Services Sector Technical Sales, IBM Germany

Glen Johnson
IBM Linux Technology Center, Austin

Art Sammartino, Burt Vialpando
IBM Software Migration Project Office

Takashi Tokunaga
IBM Data Management Technical Support, IBM Japan

Albert Wong
IBM Global Service Open Source Community

Emma Jacobs
Maritza M. Dubec
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 team with IBM technical professionals, Business Partners and/or customers.

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 redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099



Introduction

In recent years the popularity of the MySQL relational database management system (RDMS) has soared due to its close association with the Linux, Apache, and PHP projects (LAMP). These open source projects have mutually benefited from the need to develop scriptable database-driven Web applications. The Apache PHP module allows the program logic of a Web application to access a MySQL backend database and render dynamic Web content. An organization that has deployed MySQL may find that due to its limited features there is a need to migrate to a more robust RDMS such as the IBM DB2 UDB. In this redbook we describe in detail how such a migration to IBM DB2 UDB can be accomplished in a facile manner.

Switching database platforms from one to another however is often a big challenge for database administrator and developer. Complexity, total cost, and the risk of downtime are reasons that often restrain IT decision makers to start such a project. The primary goal of this book therefore is to show that migration from MySQL to DB2 is feasible, and to provide help in planning and implementing data migration from MySQL to DB2 UDB. This migration guide refers to the last general available versions of MySQL and DB2 UDB, which is MySQL 4.0 and DB2 UDB 8.1.

Confronted with this demand, a certain level of knowledge in both environments is required to start with the migration process.

The goal of this chapter is to introduce the architecture and design of both database engines: MySQL and DB2 Universal Database™ (DB2 UDB). This chapter includes the following points:

- ▶ DB2 UDB 8.1
 - DB2 family
 - DB2 architecture
 - DB2 utilities
 - DB2 client access
- ▶ MySQL database
 - MySQL architecture
 - MySQL design and SQL compliance
 - MySQL utilities
 - MySQL client application programming interfaces (API)

1.1 DB2 Universal Database

DB2 UDB is a high scalable database, developed to meet even the highest demands that a most critical database application can have. DB2 UDB however is also an easy installable and manageable RDBMS. The goal of this section is to give an idea about DB2 UDB, its architecture, tools, and client connectivity.

1.1.1 Product overview

DB2 UDB on Linux, UNIX®, and Windows span the spectrum and range all the way from products on handheld devices to large clusters and mainframes (see Figure 1-1). More detail information can be found at:
<http://www-306.ibm.com/software/data/db2/udb/>

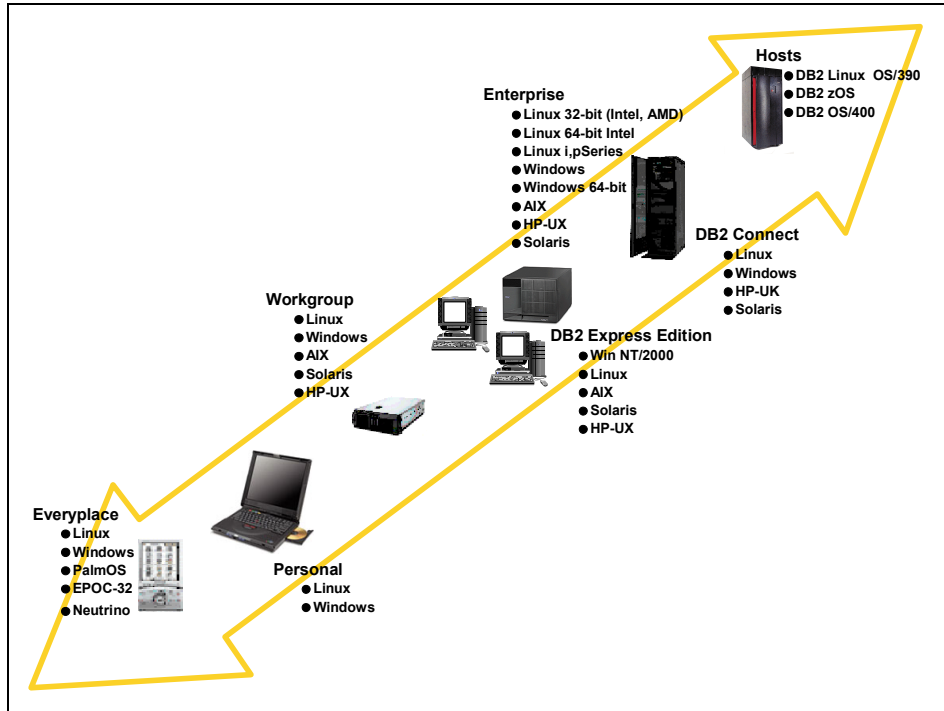


Figure 1-1 DB2 product overview

DB2 UDB editions for the production environment

DB2 provides different packages for users based on their business need. This section introduces the various DB2 packages:

► DB2 UDB Personal Edition

The DB2 UDB Personal Edition provides a single-user database engine that can be deployed on Linux or Windows based systems. It will not accept remote database requests, however, and it contains DB2 UDB client components and will serve as a remote client to a DB2 Server. The DB2 UDB Personal Edition can also be used for connecting and managing other databases servers on the network. This makes it the perfect choice for deployment in occasionally connected or remote office implementations that do not require multi-user capability.

► DB2 UDB Workgroup Server Edition

The DB2 UDB Workgroup Server Edition is the database server designed for deployment in a departmental or small business environment. It has a user based licensing model designed to provide an attractive price point for smaller installations while still providing a full function database server. DB2 UDB

Workgroup Server Edition can be deployed on systems with up to 4 CPUs on Linux, Windows, and UNIX servers.

▶ **DB2 UDB Workgroup Server Unlimited Edition**

The DB2 UDB Workgroup Server Unlimited Edition offers a simplified per processor licensing model for deployment in a departmental or small business environment that has Internet users or number of users, which makes per processor licensing more attractive than the DB2 UDB Workgroup Server Edition licensing model. The DB2 UDB Workgroup Server Unlimited Edition is also for use on Linux, UNIX, and Windows systems with up to 4 CPUs.

▶ **DB2 UDB Express**

The DB2 UDB Express is a specifically tailored database offering for small and medium businesses (SMBs). The key features include simplified deployment, autonomic management capabilities, and application development support. It is designed for independent software vendors who need an easy to install database integrated into their application software solution. It is a multi-user version that supports local and remote applications in stand alone and local area network (LAN) environments.

▶ **DB2 UDB Enterprise Server Edition**

The DB2 UDB Enterprise Server Edition meets the database server needs of mid size to large businesses. This product is the ideal foundation for building data warehouses, transaction processing, or Web-based solutions, as well as a back-end for packaged solutions like ERP, CRM, and SCM. In addition, the DB2 UDB Enterprise Server Edition offers connectivity and integration for other enterprise DB2 and Informix® data sources. This version also provides the ability to create partitioned databases or to run on cluster. The *DB2 Database Partitioning Feature (DPF)* capability provides the customer with multiple benefits including scalability to support very large databases or complex workloads, and increased parallelism for administration tasks. The DPF is a chargeable licensing option of DB2 UDB Enterprise Server Edition.

▶ **DB2 UDB Data Warehouse Standard Edition**

The DB2 Data Warehouse Standard Edition is a powerful platform building Business Intelligence (BI) solutions. It is a complete datamart infrastructure product that includes DB2 UDB Workgroup Edition and other features. It can be deployed on Linux, UNIX, and Windows servers.

▶ **DB2 UDB Data Warehouse Enterprise Edition**

The DB2 UDB Data Warehouse Enterprise Edition is IBM's complete enterprise level offering for customers building the most demanding Business Intelligence solutions, and is part of the IBM DB2 UDB framework for BI. It includes DB2 UDB Enterprise Server Edition and other features.

▶ **Other DB2 editions**

There are also DB2 UDB versions for iSeries™, pSeries®, and zSeries® available. For details, please see:

<http://www-306.ibm.com/software/data/db2/>

Products for accessing legacy and host data

With the following DB2 products, you can extend your enterprise system to access the legacy system:

▶ **DB2 Connect™ Personal Edition**

The DB2 Connect Personal Edition provides the application programming interface (API) drivers and connectivity infrastructure to enable direct connectivity from desktop applications to zSeries and iSeries database servers. This product is specifically designed and is licensed for enabling two-tier, client-server applications running on individual workstations, and as such is not appropriate for use on servers.

▶ **DB2 Connect Enterprise Edition**

The DB2 Connect Enterprise Edition addresses the needs of organizations that require robust connectivity from a variety of desktop systems to zSeries and iSeries database servers. DB2 client software is deployed on desktop systems, and provides drivers that connect client-server applications running on these desktop systems to a DB2 Connect server (gateway) that accesses host data. The licensing model for this product is user based.

▶ **DB2 Connect Application Server Edition**

The DB2 Connect Application Server Edition product is identical to the DB2 Connect Enterprise Server in its technology. However, its licensing terms and conditions are meant to address specific needs of multi-tier, client-server applications, as well as applications that utilize Web technologies. DB2 Connect Application Server Edition license charges are based on the size and number of processors available to the application servers where the application is running.

▶ **DB2 Connect Unlimited Edition**

The DB2 Connect Unlimited Edition product is ideal for organizations with extensive usage of DB2 Connect, especially where multiple applications are involved. This product provides program code of the DB2 Connect Personal Edition as well as program code identical to the DB2 Connect Application Server Edition for unlimited deployment throughout an organization.

DB2 for application developers

DB2 provides two packages for application development:

▶ **DB2 UDB Personal Developer's Edition**

DB2 UDB Personal Developer's Edition enables a developer to design and build single user desktop applications. This offering comes with Linux and Windows versions of the DB2 UDB Personal Developer's Edition products as well as the DB2 UDB Extenders.

▶ **DB2 UDB Universal Developer's Edition**

DB2 UDB Universal Developer's Edition offers a low-cost package for a single application developer to design, build, or prototype applications for deployment of any of the DB2 client or server platforms. It includes client and server DB2 editions for all Linux, UNIX, and Windows supported platforms, DB2 Connect, DB2 Extenders™, and Intelligent Miner™. The software in this package cannot be used for production systems.

DB2 for pervasive platforms

The last, but not the least, the DB2 offering is:

▶ **DB2 Everyplace**

DB2 Everyplace is a relational database and enterprise synchronization server that enables enterprise applications and enterprise data to be extended to mobile devices such as personal digital assistants (PDAs) and smart phones. DB2 Everyplace can also be embedded into devices and appliances to increase their functionality and market appeal. The product can be used as a local independent database on a mobile device, or query information on remote servers when a connection is available.

Additional DB2 UDB features

In addition to the product offerings for Linux, UNIX, Windows, and accessing legacy and host data, DB2 also offers a great variety of features, which partly are included in some DB2 UDB packages. Information about additional DB2 features and products can be found at:

<http://www-306.ibm.com/software/data/db2/relatedproducts.html>

1.1.2 DB2 UDB for Linux, UNIX, and Windows architecture

Figure 1-2 shows the DB2 UDB architecture overview. DB2 UDB implements a dedicated process architecture.

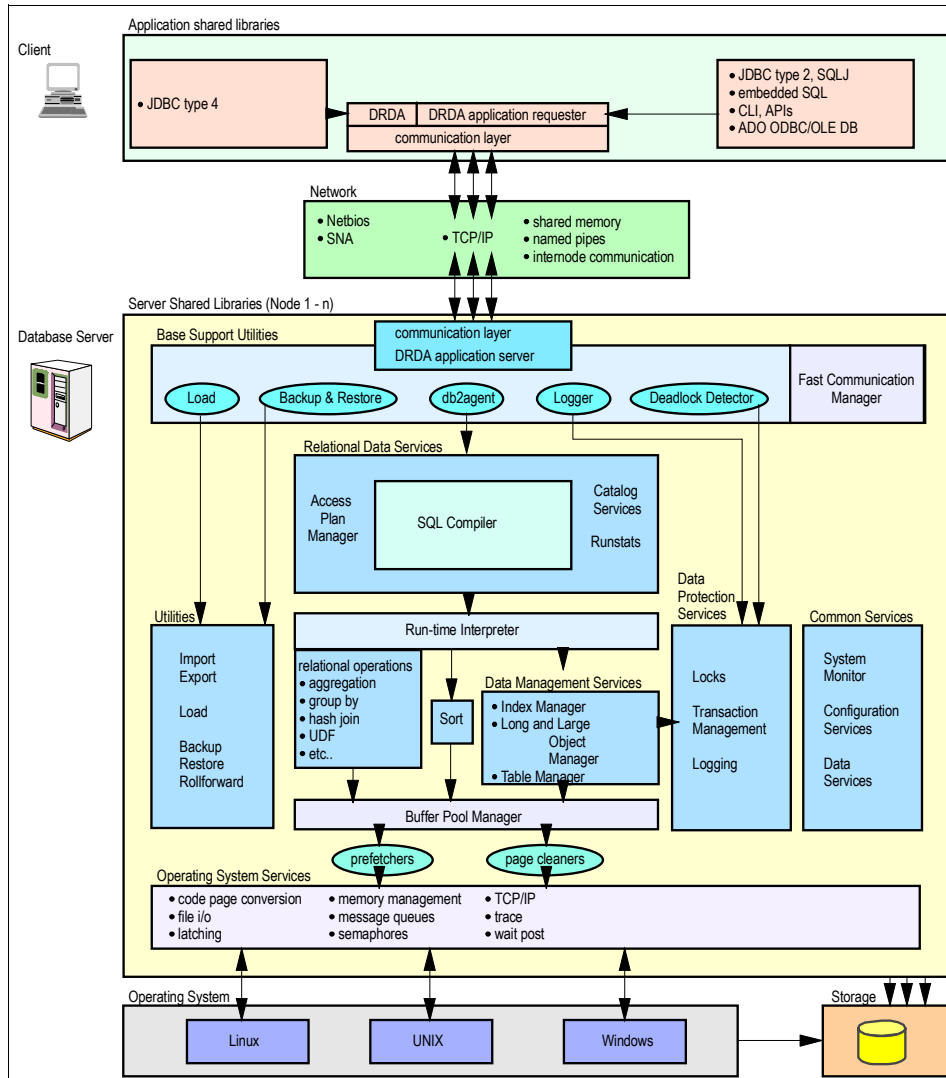


Figure 1-2 DB2 architecture overview

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 the database managers, database, and application.

To enable access to a special database, the DB2 instance process responsible for the database must be running on the DB2 server. When an instance process is started, several processes are created and interact with each other to maintain connected applications and the database. There are several background processes in DB2 that are pre-started, others start on a need-only basis. This section explains some of the important background processes.

DB2 UDB processes

The DB2 UDB server activities are performed by Engine Dispatchable Units (EDU) that are defined as background processes on Linux systems.

Some DB2 background processes are started with the instance, and others are initialized when the database is activated by a connection. Figure 1-3 shows the necessary background processors of the DB2 UDB server at the instance, application, and database level. In the following sections, we discuss some of the important processes on the respective level.

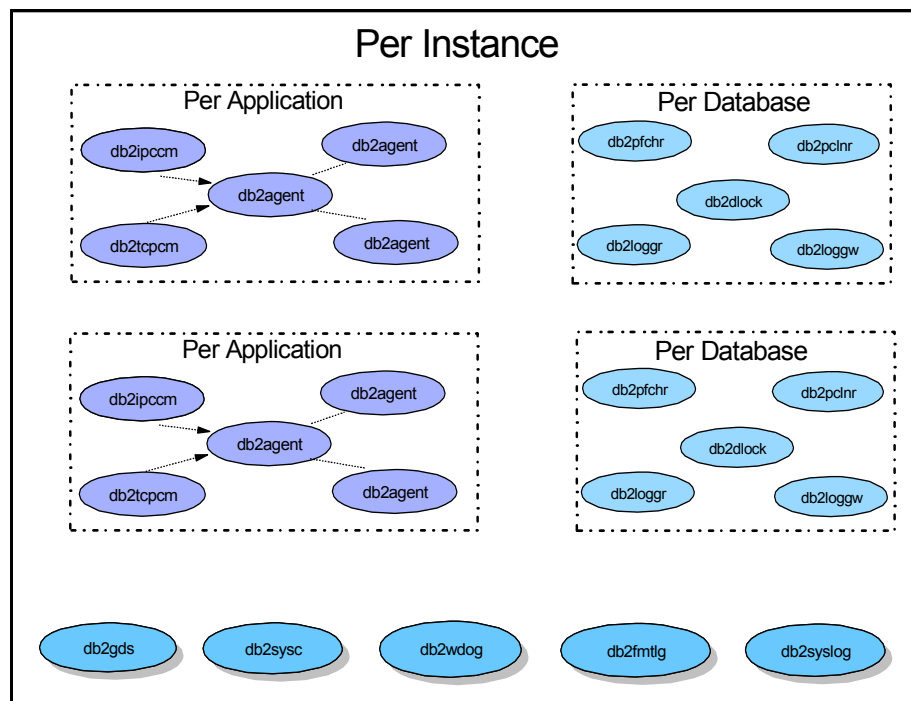


Figure 1-3 DB2 processes

Instance level processes

The following background processes will start as soon as the DB2 UDB 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 as UNIX processes.

▶ **DB2 system controller (db2sysc):**

This is the system controller processor. Without this process the instance cannot function properly.

▶ **DB2 watchdog (db2wdog):**

This process is required only in UNIX platforms. It is the parent process for all other processes.

▶ **DB2 format log (db2fmtlg):**

Pre-allocates log files in the log path when the LOGRETAIN database configuration parameter is enabled, and the USEREXIT is disabled.

▶ **DB2 system logger (db2syslog):**

This is the system logger process responsible for the writing operating system error log.

Database level processes

The following background processes are started when a connection activates the database:

▶ **DB2 log reader (db2loggr):**

This process reads the log files during transaction rollback, restart recovery, and roll forward operations.

▶ **DB2 log writer (db2logw):**

This is the log writer process that flushes the database log from the log buffer to the transaction log files on disk.

▶ **DB2 page cleaner (db2pclnr):**

Asynchronous process to make room in the buffer pool by writing changed pages to disk, before prefetchers read pages on disk storage and move into the buffer pool.

▶ **DB2 prefetcher (db2pfchr):**

This process retrieves data from disk asynchronously, and moves it into the buffer pool before the application requests the data.

▶ **DB2 deadlock detector (db2dlock):**

This is the database deadlock detector process. It scans the lock list (the lock information memory structure of DB2) and looks for deadlock situations.

Application level processes

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

▶ **DB2 communication manager (db2ipccm):**

This is the inter-process communication (IPC) process started for each application connecting locally. This process communicates with the coordinating agent to perform the database tasks.

▶ **DB2 TCP manager (db2tcpdm):**

This is the TCP communication manager process. This process is started when the remote client or applications connect to the database using TCP/IP communication. This process communicates with the coordinating agent to perform database tasks.

▶ **DB2 coordinating agent (db2agent):**

This process handles requests from applications. It 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):**

The 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, we must set the intra-parallelism database parameter to YES.

DB2 database objects

In this section, DB2 objects and their relationship to each other are introduced. Figure 1-4 shows the basic DB2 database objects.

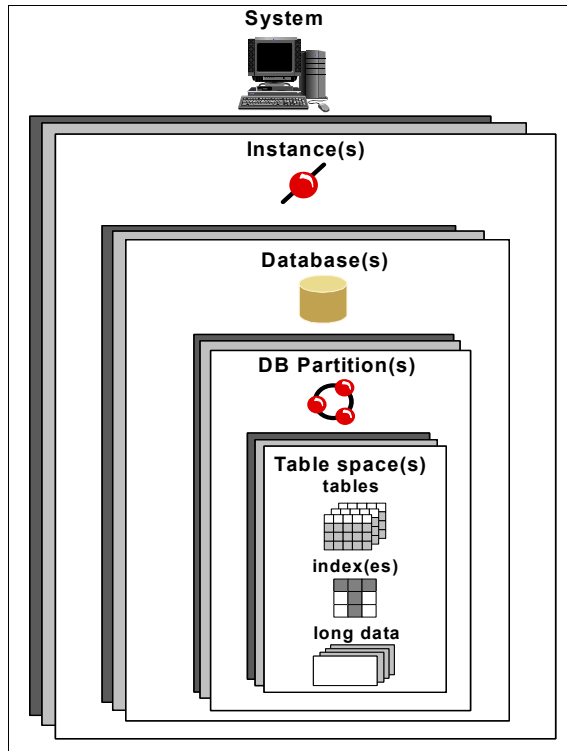


Figure 1-4 DB2 Objects relationship

► Instances

An instance is DB2 code that manages data. It controls what can be done to the data, and manages system resources assigned to it. Each instance is a complete, fairly independent environment. It contains all the database partitions defined for a given parallel database system. An instance has its own databases (which other instances cannot access directly), and all its database partitions share the same system directories. It also has separate security from other instances on the same machine (system), allowing for example both production and development environments to be run on the same machine under separate DB2 instances without interfering with each other.

► Databases

A relational database presents data as a collection of tables. Each database includes a set of system catalog tables that describe the logical and physical structure of the object in the database, a configuration file containing the parameter values configured for the database, and a recovery log. An

application or a user connects to a specified database to read or manipulate data in tables.

Figure 1-5 shows the relationship between instances, databases, and tables.

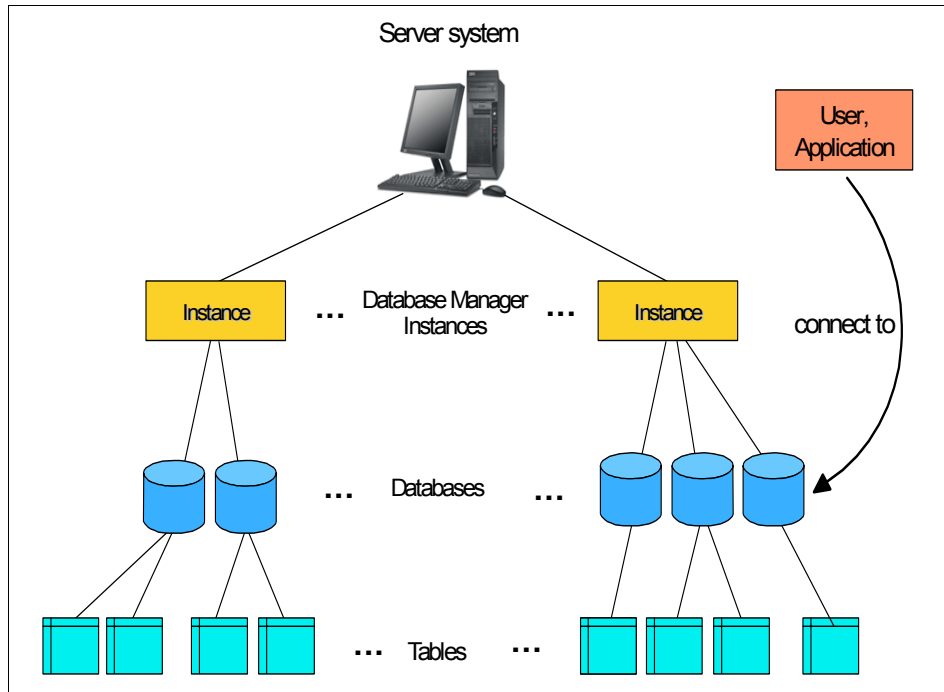


Figure 1-5 Relationship between instances, databases, and tables

► Database partition groups

A database partition group is a set of one or more database partitions (Figure 1-6). Before creating tables for the database, you first need to create the database partition group where the table spaces will be stored, and then create the table space where the tables will be stored. If a partition group is not specified, there is a default group where table spaces are allocated. In earlier versions of DB2, database partition groups were known as *nodegroups*. In a non-partitioned environment, all the data resides in a single partition, therefore it is not necessary to worry about partition groups.

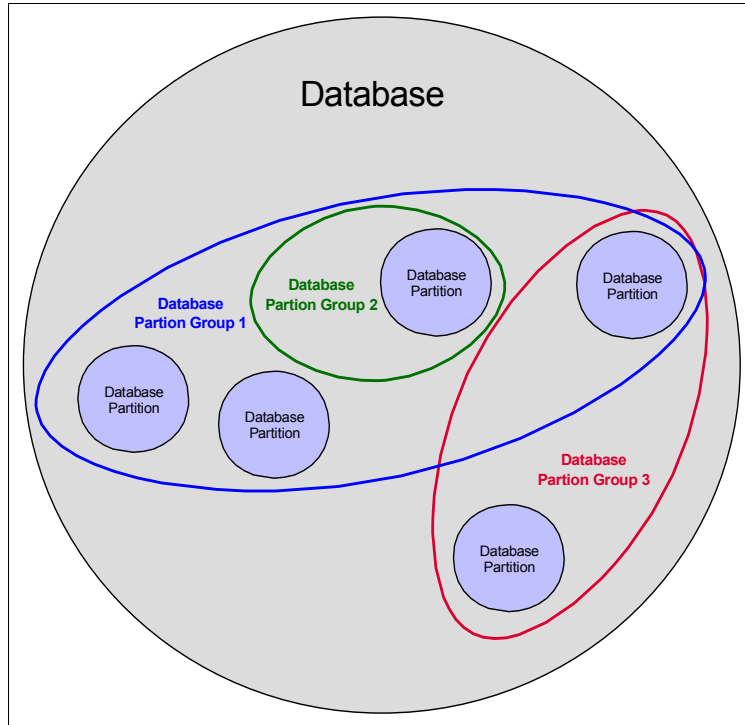


Figure 1-6 Database partition groups in a database

► **System catalog tables**

Each database includes a set of system catalog tables, which describe the logical and physical structure of the data. DB2 UDB creates and maintains an extensive set of system catalog tables for each database. These tables contain information about the definitions of database objects such as user tables, views, and indexes, as well as security information about the privilege that users have on these objects. Catalog tables are created when the database is created, and are updated during the course of normal operation. You cannot explicitly create or drop them, but you can query and view their contents using the catalog views.

► **Table spaces**

A database is organized into subdivided table spaces. A table space is a place to store data. When creating a table, you can decide to have certain objects such as indexes and large object (LOB) data kept separately from the rest of the table data. A table space can also be spread over one or more physical storage devices.

Table spaces reside in database partition groups if they were created. Table space definitions and attributes are maintained in the database system catalog. Containers are assigned to table spaces. A container is an allocation of physical storage (such as a file or a device).

A table space can be either system managed space (SMS), or database managed space (DMS). For an SMS table space, each container is a directory in the file system of the operating system, and the operating system's file manager controls the storage space. For a DMS table space, each container is either a fixed size pre-allocated file, or a physical device such as a disk, and the database manager controls the storage space.

► **Schemas**

A schema is an identifier, by default the user ID, which qualifies tables and other database objects. A schema can be owned by an individual, and the owner can control access to the data and the objects within it. A schema name is used as the first part of a two-part object name. For example, a schema named Smith might qualify a table named *SMITH.PAYROLL*.

► **Tables**

A relational database presents data as a collection of tables. Data in a table are arranged in columns and rows. The data in the table is logically related, and relationships can be defined between tables. Table data is accessed through Structured Query Language (SQL), a standardized language for defining and manipulating data in a relational database. A query is used in applications or by users to retrieve data from a database. The query uses SQL to create a statement in the form of:

```
SELECT <data_name> FROM <table_name>
```

► **Views**

A view provides a different way of looking at data in one or more tables; it is a named specification of a result table. The specification is a SELECT statement that runs whenever the view is referenced in a SQL statement. A view has columns and rows just like a base table. All views can be used just like base tables for data retrieval. Figure 1-7 shows the relationship between tables and views.

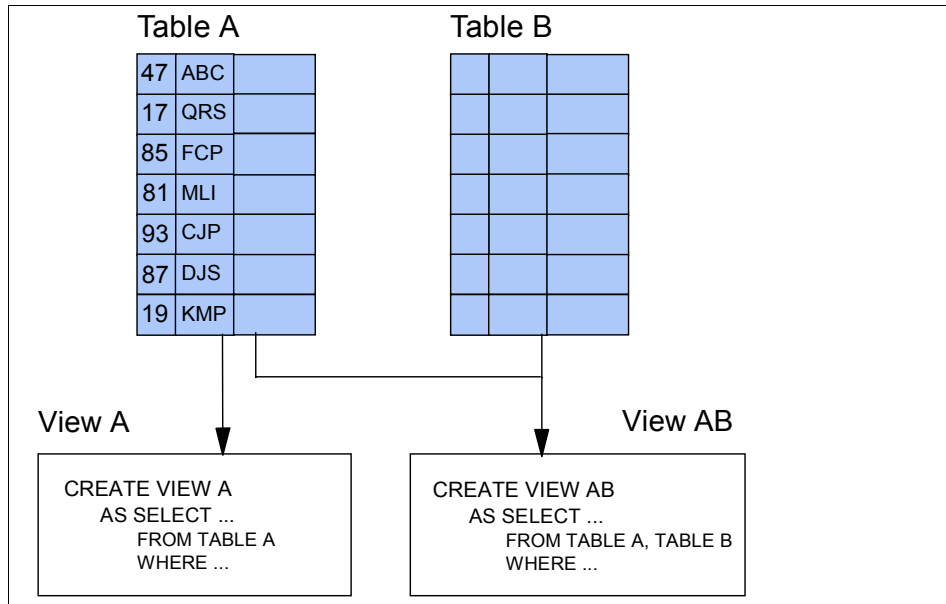


Figure 1-7 Relationship between tables and views

► Indexes

An index is a set of keys, each pointing to rows in a table. For example, table A has an index based on the first column in the table (Figure 1-8). This key value provides a pointer to the rows in the table: value *19* points to record *KMP*. An index allows efficient access when selecting a subset of rows in a table by creating a direct path to the data through pointers.

The DB2 SQL Optimizer chooses the most efficient way to access data in tables. The optimizer takes indexes into consideration when determining the fastest access path.

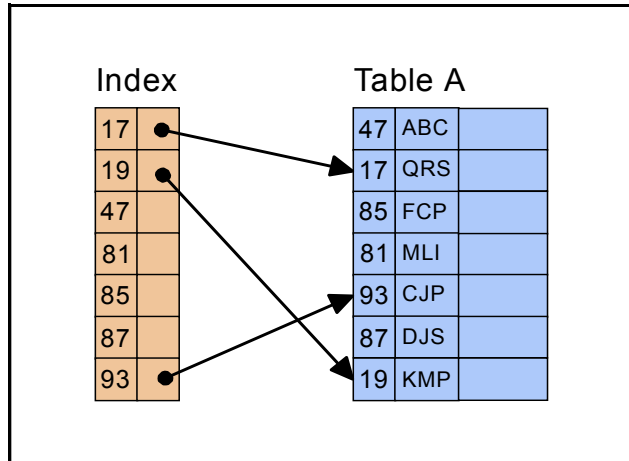


Figure 1-8 Relationship between indexes and tables

► **Containers**

A container is a physical storage device. It can be identified by a directory name, a device name, or a file name. A container is assigned to a table space. A single table space can span many containers, but each container can belong to only one table space as shown in Figure 1-9.

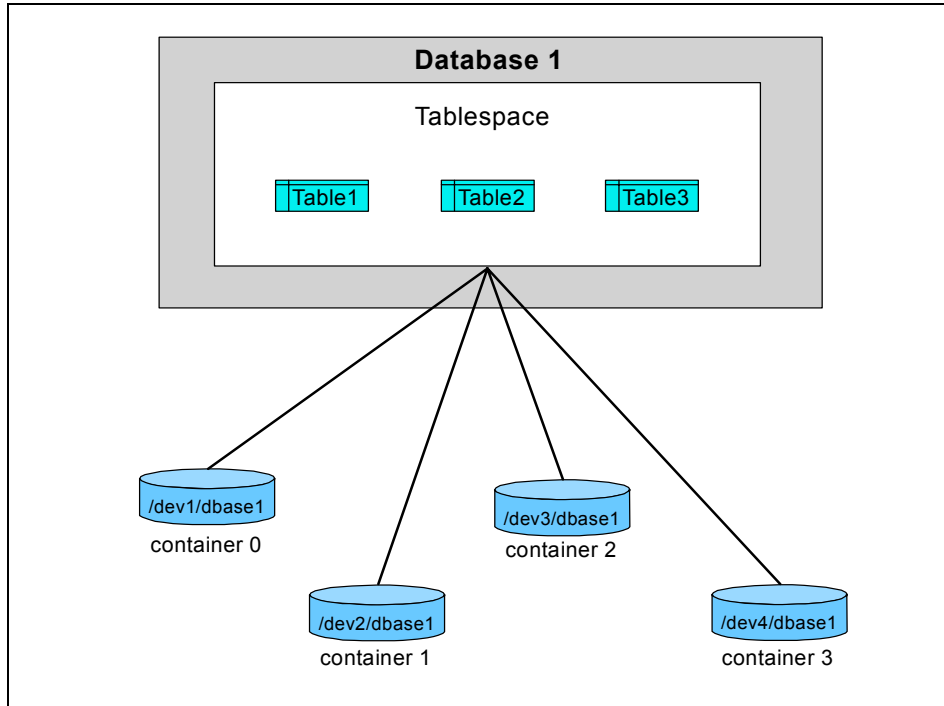


Figure 1-9 Relationship between table space and containers

► Buffer pools

A buffer pool is the amount of memory allocated to cache table and index data pages. The purpose of the buffer pool is to improve system performance. Data can be accessed much faster from memory than from disk; therefore, the fewer times the database manager needs to read from or write to a disk (I/O) synchronously, the better the performance of the application. The size of the buffer pool is the single most important performance tuning area, because you can reduce the delay caused by synchronous I/O.

DB2 directory structure

On Linux systems the default installation path for DB2UD V8.1 is `/opt/IBM/db2/V8.1`. `$DB2DIR` is the environment variable for the DB2 installation directory. Figure 1-10 shows the default directory structure for a simple **CREATE DATABASE** command with no table space options specified. By default DB2 creates SMS table space containers in the specified database directory. The three default table spaces created are system catalog, system temporary, and user table space. For the log files, DB2 creates a directory called *sqllogdir*. On a Linux system a *sqllib* directory will be created under the instance home directory, which has a symbolic link to the DB2 installation directory.

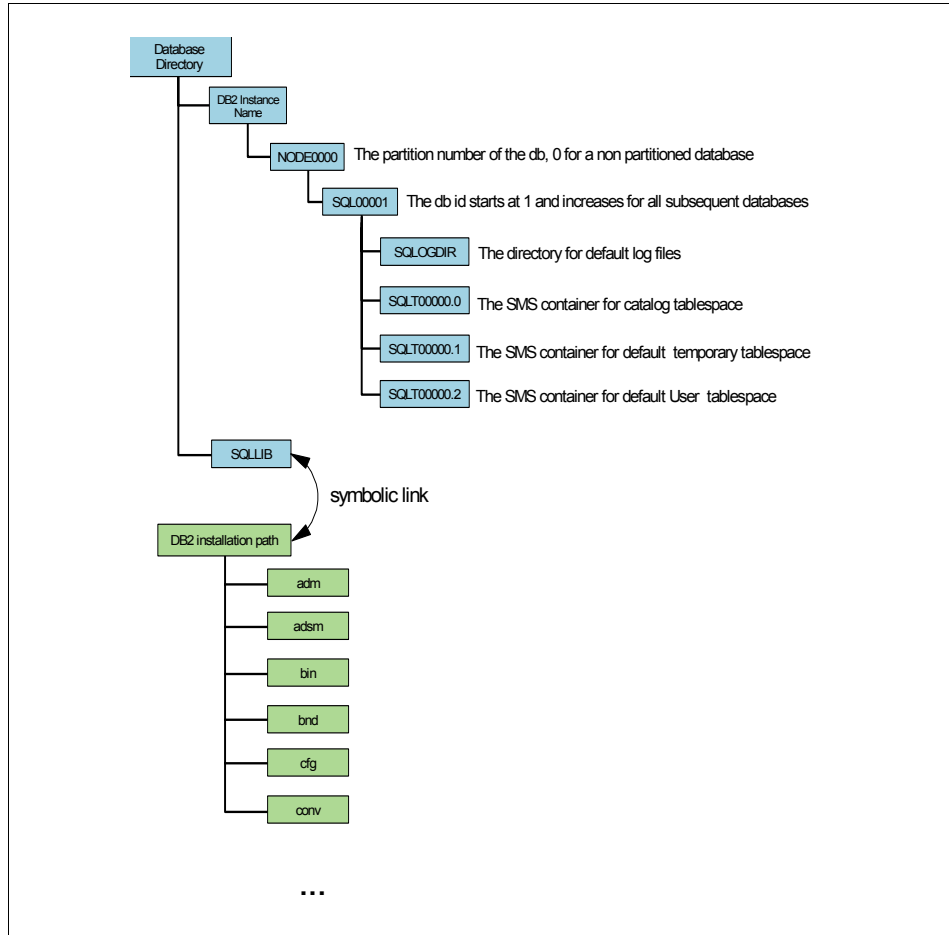


Figure 1-10 DB2 directory structure

DB2 catalog

In DB2 UDB, 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 catalog is automatically created with the database. 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 are found in the *SYSCAT* schema. *SYSSTAT* views are updateable views containing statistical information that are found in the *SYSTAT* schema. The complete DB2 UDB

catalog views can be found in *DB2 UDB SQL Reference* Volume 1 and 2, SC09-4484 and SC09-4485.

1.1.3 DB2 utilities

All DB2 system commands are installed in the *sqllib/bin* directory by the installation procedure. Some of the most important commands in DB2 UDB are listed in Table 1-1. For a more detail description of the DB2 administration GUI tools, see Chapter 8., “Database administration” on page 235.

Table 1-1 DB2 commands list

Command	Command description
dasauto	Autostart DB2 Administration Server
dascrt	Create a DB2 Administration Server
dasdrop	Remove a DB2 Administration Server
dasmigr	Migrate a DB2 Administration Server
db2	Command Line Processor invocation
db2admin	DB2 Administration Server
db2adv is	DB2 Index Advisor
db2cap	CLI/ODBC Static Package Binding Tool
db2cc	Start Control Center
db2cfexp	Connectivity configuration export tool
db2cfimp	Connectivity configuration import tool
db2cidmg	Remote database migration
db2ckbkp	Check backup
db2cli	DB2 interactive CLI
db2cmd	Open DB2 command window
db2dart	Database analysis and reporting tool
db2empfa	Enable multipage file allocation
db2eva	Event analyzer
db2evmon	Event monitor productivity tool

Command	Command description
db2evtb1	Generate event monitor target table definitions
db2exfmt	explain table-format tool
db2expln	DB2 SQL explain tool
db2gncol	Update generated column values
db2gov	DB2 governor
db2govlg	DB2 governor log query
db2hc	Start Health Center
db2icrt	Create instance
db2idrop	Remove instance
db2ilist	List instances
db2imigr	Migrate instance
db2isetaup	Start instance creation interface (UNIX)
db2iupdt	Update instances
db2jdbcbind	DB2 JDBC package binder
db2level	Show DB2 service level
db2look	DB2 statistics and DDL extraction tool
db2move	Database movement tool
db2relocatedb	Relocate database
db2saml	Create sample database
db2set	DB2 profile registry command
db2setup	Install DB2
db2sql92	SQL92 compliant SQL statement processor
db2sqljbind	DB2 SQLJ profile binder
db2sqljcustomize	DB2 SQLJ profile customizer
db2sqljprint	DB2 SQLJ profile printer
db2start	Start DB2

Command	Command description
db2stop	Stop DB2
db2updv8	Update database to Version 8 current level
sqlj	DB2 SQLJ translator
dynexpln	Explain dynamic SQL (deprecated)

1.1.4 DB2 database access

In this section the following topics are discussed:

- ▶ DB2 clients
- ▶ Application access
- ▶ DB2 application programming interfaces

DB2 clients

To access a DB2 UDB database, a DB2 client has to be installed on the client system. IBM offers three types of DB2 clients:

▶ Run-time Client

This client provides you access to DB2 UDB servers with application interfaces such as JDBC, SQLJ, ODBC, CLI, and OLE DB. This client can be used if no DB2 server administration has to be done from this client.

▶ Administration Client

The Administration Client has all features of the DB2 Run-Time Client plus tools to administer a DB2 Server.

▶ Application Development Client

This client provides a collection of graphical and non-graphical tools for developing applications. It includes all components of the DB2 Administration Client.

For client-server communication, DB2 supports several communication protocols like TCP/IP, APPC, NPIPE, NetBIOS, etc. 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, the **db2set DB2COMM** command must be executed. For TCP/IP, a unique port address has to be specified in the database manager configuration. This port is registered in the services file. To reserve port 50000 with the service name db2icdb2, for example, the entry in services file is:

```
db2icdb2 50000/tcp
```

For update this information in the database manager following command is used:

```
db2 UPDATE DBM CFG USING SVCENAME db2icdb2
```

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 databases are configured under a node, which describes the host information like protocol, port, etc. To configure a remote TCP/IP node following command is used:

```
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 used is:

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

When using the Configuration Assistant GUI tool to add a database connection, a database discovery can be started to find the desired database.

Note: DB2 Discovery method is enabled at the instance level using the *DISCOVER_INST* parameter, and at database level using *DISCOVER_DB* parameter.

Application access

When deploying applications with DB2 UDB different methods can be used:

► Single-tier

In this configuration the application and the database reside on the same system. In enterprise environments, it may be rare to see such a configuration, because remote access to a database server is typically required. Nonetheless, this is quite common for developing applications that can later be deployed transparently in a multi-tier DB2 environment without any changes or batch applications.

► Client/Server or 2-tier

The application and the database reside on separate systems. The machines where the application runs typically have a DB2 client installed, which communicates over the network to a database server. For the application, the physical location of the data is transparent. The application communicates with the DB2 client using a standard interface (for example, ODBC) and the DB2 client takes over the task of accessing the data over the network. In some cases, such as browser or Java based access, it is not necessary to have the DB2 client running on the same machine where the application executes.

DB2 provides exceptional flexibility for mixing and matching client and server platforms in a heterogeneous environment. DB2 client and server code is available for a wide variety of platforms. For example, the application can execute on a Windows based machine with a DB2 client for Windows, which can then access a DB2 database on a Linux server. Likewise, the Linux machine can act as a client and access data from UNIX servers or mainframes.

► **Multi-tier**

In a multi-tier configuration, the application, DB2 client, and the data source typically reside on separate systems. Examples of such configuration scenarios are illustrated in Table 1-2 below.

Table 1-2 Multi-tier configuration examples

Client	Middle-tier	Server
Web-browser	Web server DB2 Client	DB2 database server
Application client	Application server DB2 client	DB2 database server 1 DB2 database server 2
Application DB2 client	DB2 connect gateway	zSeries, iSeries
Application DB2 client	DB2 server	Secondary Data Sources (for example, Mainframe DB2, Non-DB2, non-relational)

IBM recognizes that in many cases there may be a need for accessing data from a variety of distributed data sources rather than one centralized database. The data sources can be from IBM, such as DB2 or Informix, or non-IBM databases, such as Oracle, or even non-relational data, such as files or spreadsheets. As illustrated in the last scenario in the Table 1-2, IBM offers the most comprehensive business integration solution by allowing federated access to a variety of distributed data sources.

DB2 application programming interfaces (APIs)

In order to access or manage DB2 objects, several different programming interfaces can be used as seen in Figure 1-11.

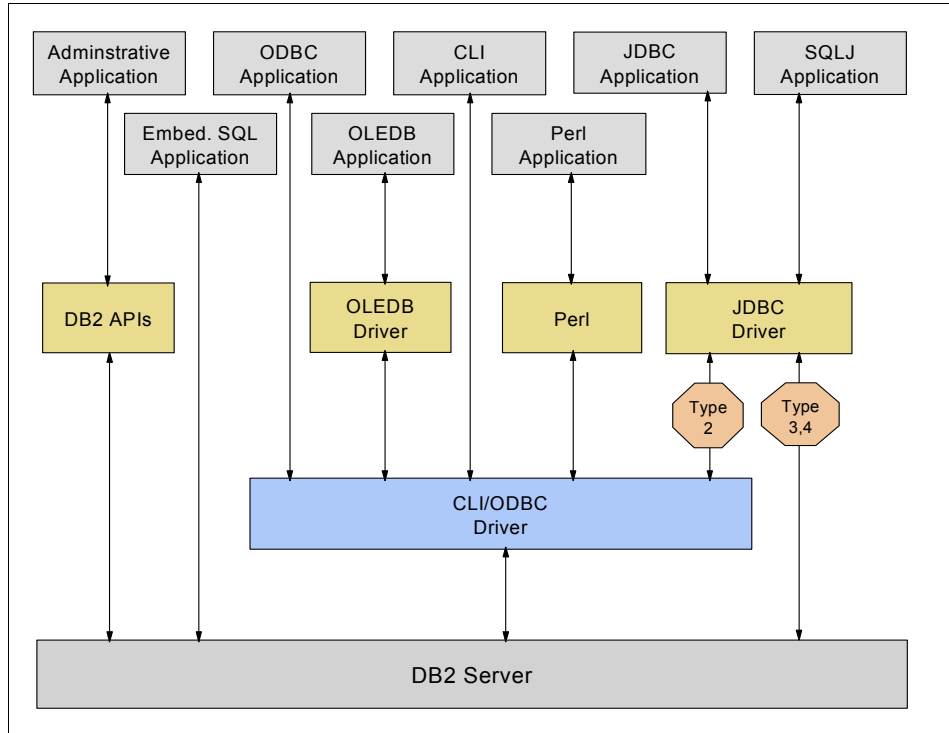


Figure 1-11 Application connections to DB2 UDB

DB2 administrative API

DB2 provides numerous administrative APIs, which allow applications perform database administration tasks available in DB2 UDB Control Center; for example, import and export data, creating, activating, backing up, or restoring a database. These calls can be included within embedded SQL and DB2 CLI applications. Examples of API programs can be found in the DB2 home directory *sqllib/sample/* for different programming languages. For additional information refer to *DB2 Administrative API Reference*, SC09-4824.

Embedded SQL statements in applications

Two different kind of SQL statements have to be distinguished:

► **Static SQL statements**

With static SQL statements, you know before compile time that the SQL statement type and the table and column names. The only unknowns are the specific data values the statement is searching for or updating. This values can be represented in host language variables.

Before compiling and linking the program, precompiling and binding of the embedded SQL statements has to be done. Precompiling converts embedded SQL statements into DB2 run-time API calls that a host compiler can process, and then creates a bind file. The **bind** command creates a package in the database. This package then contains the SQL operation and the access plan that DB2 will use to perform the operation.

► **Dynamic SQL**

Dynamic SQL statements in an application are built and executed at runtime. For a dynamically prepared SQL statement, the syntax has to be checked and an access plan has to be generated during the program execution.

Examples of embedded static and dynamic SQL can be found in the DB2 home directory: *sqliib/samples/*.

DB2 Call Level Interface (DB2 CLI)

DB2 CLI is a programming interface that can be used from C and C++ applications to access DB2 databases. DB2 CLI is based on the Microsoft Open Database Connectivity (ODBC) specification, and the ISO CLI standard. The DB2 CLI library can be loaded as an ODBC driver by an ODBC driver manager. DB2 CLI includes support for many ODBC and ISO SQL/CLI functions, as well as DB2 specific functions.

When using DB2 CLI, the application passes dynamic SQL statements as function arguments to the database manager for processing. Because of this, applications use common access packages provided with DB2, DB2 CLI applications do not need to be precompiled or bind. Only compiling and linking of the application is needed. Before DB2 CLI or ODBC applications can access DB2 databases, the DB2 CLI binds files that come with the DB2 Application Development Client to each DB2 database that will be accessed. This occurs automatically with the execution of the first statement.

Typically, when building an ODBC application, an ODBC driver manager is needed, which are normally provided by platform vendors like Microsoft or others. There is also an ODBC driver manager for Linux available, which can be found at <http://www.unixodbc.org/>. However, 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. Examples of C programs using CLI calls can be found in the DB2 home directory: *sqliib/samples/cli* For additional information regarding CLI, refer to *Call Level Interface Guide and Reference*, Volume 1 and Volume 2, SC09-4849 and SC09-4850.

Java Database Connectivity application (JDBC)

DB2's Java support includes JDBC, a vendor-neutral dynamic SQL interface that provides data access to the application through standardized Java methods.

Similar to DB2 CLI, you do not have to precompile or bind a JDBC program. As a vendor-neutral standard, JDBC applications offer increased portability. The JDBC API, which is similar to the CLI/ODBC API, provides a standard way to access databases from Java code. The Java code passes SQL statements to the DB2 JDBC driver, which handles the JDBC API calls. Java's portability enables the delivery of DB2 access to clients on multiple platforms, requiring only a Java-enabled Web browser, or a Java runtime environment.

DB2 Version 8 offers different ways of creating Java applications, either using a type 2, type 3, or type 4 JDBC driver:

► **Type 2 driver:**

With a type 2 driver, calls to the JDBC application driver are translated to Java native methods. The Java applications that use this driver must run on a DB2 client, through which JDBC requests flow to the DB2 server. This is typically how DB2 is accessed by WebSphere Application Server.

Tip: If you want to prototype CLI calls before placing them in a program, you can use the **db2cli.exe** (Windows) or **db2cli** (Linux) file in the *sqlib/samples/cli* directory. There is also a document called INTCLI.DOC, which advises you about how to use the utility.

► **Type 3 driver:**

The DB2 JDBC type 3 driver, also known as the applet or net driver, consists of a JDBC client and a JDBC server. The DB2 JDBC applet driver can be loaded by the Web browser along with the applet. Another way is to use the applet driver in standalone Java applications. When the applet requests a connection to a DB2 database, the applet driver opens a TCP/IP socket to the DB2 JDBC applet server, which is the machine where the Web server resides. After a connection is set up, the applet driver sends each of the subsequent database access requests from the applet to the JDBC server through the TCP/IP connection. The JDBC server then makes corresponding DB2 calls to perform the task. Upon completion, the JDBC server sends the results back to the JDBC client through the connection. The use of the type 3 driver is being deprecated with DB2 Version 8 because of the new type 4 driver.

► **Type 4 driver:**

The JDBC type 4 driver, which is new for Version 8, can be used to create both Java applications and applets. To run an applet that is based on the type 4 driver, only a Java enabled browser is required, which downloads the applet and the JDBC driver (db2jcc.jar). To run a DB2 application with a type 4 driver, only an entry for the JDBC driver in the *class path* and no DB2 client is required. This type 4 driver provides the initial implementation of the new

JDBC driver architecture known as the *IBM DB2 JDBC Universal Driver*. The Universal Driver is architected as an abstract JDBC processor that is independent of driver-type connectivity or target platform. Examples of JDBC calls can be found in *sqllib/samplesjava/jdbc*. For detailed information on the Java support provided by DB2 Version 8, we strongly recommend the whitepaper *Developing Enterprise Java Applications Using DB2 Version 8*:
<http://www.ibm.com/developerworks/db2/library/techarticle/0209hutchison/0209hutchison.html>

Embedded SQL for Java (SQLj)

DB2 Java embedded SQL (SQLj) support is provided by the DB2 AD Client. With DB2 SQLj support, in addition to DB2 JDBC support, SQLj applets, applications and stored procedures can be built to contain static SQL and use embedded SQL statements that are bound to a DB2 database.

SQLj applications use JDBC as a foundation for tasks such as connecting to databases and handling SQL errors, but also contain embedded static SQL statements in separate SQLj source files. Unlike the other languages that can contain embedded SQL (COBOL, C, C++) the Java code is not precompiled, instead the SQLj translator converts SQLj clauses into JDBC statements. As SQLj shares its underlying connection with that of JDBC, applications, it can connect to DB2 using either type 2, type 3, or type 4 drivers.

Examples of SQLj calls can be found in *sqllib/samplesjava/sqlj*. More detailed information can also be found in article “Developing Enterprise Java Applications Using DB2 Version at:

<http://www-106.ibm.com/developerworks/db2/library/techarticle/0209hutchison/0209hutchison.html>

ActiveX Data Objects and Remote Data Objects (Windows only)

DB2 supports ActiveX Data Object (ADO) applications that use the Microsoft OLE DB to ODBC bridge. ActiveX Data Objects (ADO) allows you to write applications to access and manipulate data in a database server through an OLEDB provider.

When installing the client version of DB2 Version 8.1 for Windows, optionally IBMDADB2, the IBM OLE DB 2.0 compliant provider for DB2 can also be installed. With this driver the DB2 database does not have to be cataloged as an ODBC data source.

Remote Data Objects (RDO) provide an information model for accessing remote data sources through ODBC. RDO offers a set of objects that make it easy to connect to a database, execute queries and stored procedures, manipulate results, and commit changes to the server. As RDO implements a thin code layer

over the ODBC API, it requires an ODBC data source to be created for the DB2 database that you are connecting to.

ADO.NET

DB2 UDB supports Microsoft's ADO.NET programming interface through a native managed provider. High-performing WinForm, WebForm, and mobile WebForm applications can be developed using the ADO.NET API. When used in conjunction with stored procedures and the federated database capabilities of DB2 UDB and DB2 Connect servers, this data access can be extended to include a wide variety of other data sources, including non-DB2 mainframe data (such as VSAM, CICS®, IMS™), Informix® Dynamic Server (IDS), Microsoft SQL Server, Sybase and Oracle databases as well as any data source that has an OLE DB Provider available. The IBM DB2 .NET Data Provider is native provider written in managed C# code to deliver high-performing, secure access to DB2 data.

Perl DBI

DB2 supports the Perl Database Interface (DBI) specification for data access through the DBD::DB2 driver. The Perl DBI module uses an interface that is similar to the CLI and JDBC interfaces, which makes it easy to port Perl prototypes to CLI and JDBC. More information according Perl DBI can be found at:

<http://www-306.ibm.com/software/data/db2/perl/>

1.2 MySQL database

MySQL is a common Open Source SQL database management system. It is developed and distributed by MySQL AB (<http://www.mysql.com>), a company, which builds its business by providing services around MySQL.

MySQL first was developed for UNIX and Linux applications. It became popular when Internet Service Provider (ISV) discovered that MySQL could be offered free of charge to their Internet customers providing all storage and retrieval functionality a dynamic Web application needs. It was also advantageous that ISVs mostly use Linux or UNIX in combination with APACHE as their favorite Web server environment. However, MySQL nowadays also is in use as integrated database or embedded database in various applications on almost every platform architecture.

1.2.1 MySQL architecture

MySQL is a client-server architecture based on TCP/IP. The MySQL server, which can be installed on a Windows, UNIX, Linux, or Mac platform, waits for connection on a specified port and responds to SQL statements from a client.

The conceptual architecture of the MySQL database is illustrated in Figure 1-12. In the next pages the functionality of the integrated components is discussed in more detail.

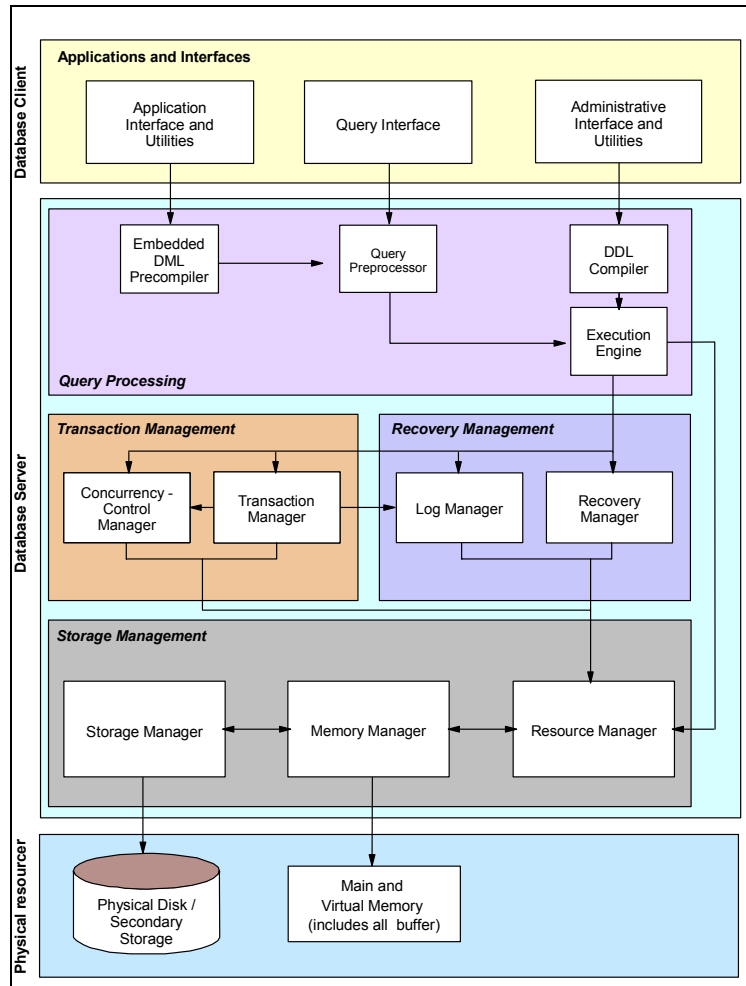


Figure 1-12 Conceptual MySQL architecture

Database client

The client layer represents the interface between the user and the database. MySQL clients include the:

- ▶ Administrative interface and utilities
- ▶ Application interface and utilities
- ▶ Query interface

This is the front end with which users interact. This component represents three kinds of users that interact with the RDBMS:

- ▶ Administrators
Use the administrative interface and utilities, such as *mysqladmin* for creating or dropping databases and shutting down the MySQL server; *isamchk*, *myisamchk* for table analysis, optimization, and crash recovery; *mysqldump* for backing up databases or copying to other server.
- ▶ Applications
Communicate with the RDBMS through MySQL APIs available for various programming languages such as C++, PHP, Java, Perl, etc.
- ▶ Query users
Interact with the RDBMS through a query interface called *mysql*, which allows the user to issue SQL statements and view the results returned from the server.

Database server

Database server represents the core functionality of the database architecture. The following four main components can be found in MySQL:

- ▶ Query processing
- ▶ Transaction management
- ▶ Recovery management
- ▶ Storage management

Query processing

The Query Processor receives all the SQL commands from MySQL application or on-line users, parses the SQL code, executes the code, and sends the user the response.

Requests received from an administrative interface are processed by the *Data Definition Language (DDL) compiler*. The DDLs are compiled for direct interaction with the database, and are passed to Execution Engine for execution.

When a request is submitted from an application, the *Embedded Data Manipulation Language (DML) Precompiler* extracts the relevant SQL

statements or translates the client commands written in programming languages like Perl or C++ into SQL statements. The embedded DML precompiler is responsible for translating requests into a format that MySQL understands.

The *Query Processor* analyses the SQL statements from the embedded DML precompiler or user interface, and creates a parse tree structure to validate the SQL query syntax.

If the query is deemed to be valid, the Query Processor then performs the security checking. It checks the access permissions and makes sure that tables and records are accessed only by the authorized users.

After security checking, the query is analyzed and optimized in order to raise performance of the query process. The MySQL optimizer uses the most restrictive index first in order to eliminate as many rows as possible to proceed with less restrictive index optimization.

The last step in query processing is the execution of the SQL statement in the *Execution Engine*. Specific tasks such as repair, backup, and recovery, which are started by the database administrator and processed from the DDL compiler are executed in the execution engine.

Transaction management

The *Transaction Manager* is responsible for making sure that transactions are logged and executed automatically in a safe and stable way. Resolving deadlock situations and issuing the commit or roll back commands to ensure database stability are done through the aid of the *Log Manager and Concurrency Manager*.

The *Concurrency-Control Manager* ensures that transactions are executed separately and independently by acquiring locks from the locking table, on appropriate data in the database from the Resource Manager. Once the lock is acquired, only operations in one transaction can manipulate the data. If a different transaction tries to manipulate the same data, the request is rejected by the Concurrency-Control Manager until the first transaction is complete.

Recovery management

The *Log Manager* is the module that is responsible for logging the operations executed in the database. The log is stored on the disk. In case of a system crash, executing each command in the log will bring the database back to its last stable state. This is done by the *Recovery Manager*.

Resource Management

The Resource Management is responsible for memory allocation and retrieving data from physical disks, and writing data to physical disks.

This component consists of three modules: the *Resource Manager*, *Memory Manager*, and the *Storage Manager*. When the Resource Manager receives and accepts a request from the Execution Engine, this request is forwarded to the Memory Manager. The Memory Manager allocates memory resources to meet the demand for the request, and returns the references of the requested data within memory to the Resource Manager, which then forwards the data to the Execution Engine. The role of the Storage Manager is to mediate between the Memory Manager and the secondary storage.

Physical resource

This layer is the bottom layer of the RDBMS architecture, and represents the main and virtual memory, and the secondary storage or physical disk. This layer is accessed through the storage management to store or retrieve data. Data stored in a RDBMS is usually:

- ▶ Data files (user data)
- ▶ Data dictionary (metadata)
- ▶ Indices
- ▶ Log information
- ▶ Statistical data

1.2.2 MySQL design and SQL compliance

In this section, the following topics are discussed:

- ▶ MySQL directory structure
- ▶ MySQL table types
- ▶ MySQL standard SQL compliance

MySQL directory structure

MySQL can be installed as one or more database within one server. For each database, MySQL creates a directory that holds data and indexes. Figure 1-13 shows the directory structure for a binary installation on SuSE Linux. Red Hat has similar directory structure.

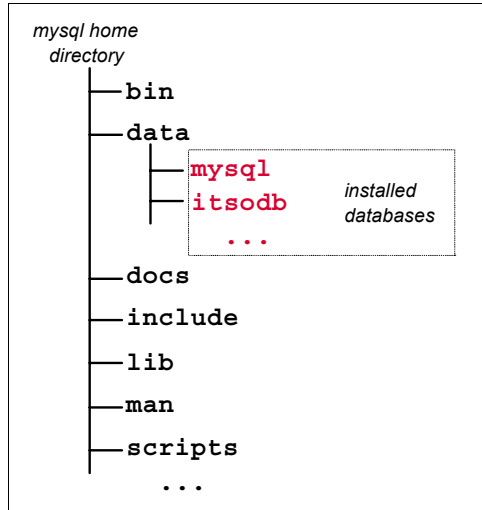


Figure 1-13 MySQL directory structure

In our example, the installed databases are *mysql*, which by default holds the security information and our sample database *itsodb*. For each database there is a directory that contains three files per table. Files with the 'MYD' extension contains the table data. Files with the 'MYI' extension contains the table's indexes. Files with the *frm* extension contain the table's structure definition known as the *schema*. All these tables are MyISAM type tables. Log files per default are created in the *data* directory. The security information data tables are in the directory */data/mysql*. Table 1-3 lists the security information data tables.

Table 1-3 Permission information stored in tables

File	Description
columns_priv.MYD	Permission for individual columns within a table
columns_priv.MYI	
columns_priv.frm	
db.MYD	Permission for individual tables
db.MYI	
db.frm	
func.MYD	Permission for user defined functions
func.MYI	
func.frm	

File	Description
columns_priv.MYD	Permission for individual columns within a table
columns_priv.MYI	
columns_priv.frm	
host.MYD	General permission by host
host.MYI	
host.frm	
tables.MYD	Permission for individual tables within a database
tables.MYI	
tables.frm	
user.MYD	General Permission by user
user.MYI	
user.frm	

MySQL table types

MySQL supports two different types of table:

- ▶ Not transaction-safe tables which are:
 - ISAM
 - MyISAM
 - HEAP
 - MERGE
- ▶ Transaction-safe tables which are:
 - InnoDB
 - BDB

Database systems that support simultaneous users must ensure that changes to the database cannot corrupt the database or leave it in an inconsistent state. Most RDBMS can bundle multiple updates (adds, changes, or deletes) together into transactions; see Example 1-1. Transactions have four important features usually referred to as the ACID properties:

- ▶ **Atomic:**
Transactions either completely succeed or completely fail. If the system crashes before the transaction completes the database's state does not change.

- ▶ **Consistent:**
Transactions preserve database consistency. A transaction transforms the database from a consistent state to another consistent state.
- ▶ **Isolated:**
A transaction's updates do not interfere with other transactions or other users of the database. Until a transaction completely succeeds, the database system conceals the individual updates from other transactions.
- ▶ **Durable:**
Once a transaction completes (commits), the updates survive in the database.

MySQL supports transactions with the InnoDB and BDB transactional storage engines. InnoDB provides full ACID compliance.

Example 1-1 Transactional Operation

```
BEGIN
INSERT INTO table1 ...
DELETE FROM table2 ...
COMMIT
```

SAM tables

This table type uses the *Index Sequential Access Method* (ISAM) with a B-tree index to “navigate” through the table. The index is stored in a file with the .ISM extension, and the data is stored in a file with the .ISD extension.

ISAM tables have the following properties:

- ▶ Compressed and fixed-length keys
- ▶ Fixed and dynamic record length
- ▶ 16 keys with 16 key parts per key
- ▶ Max key length 256
- ▶ Data is stored in machine format; therefore, machine or OS dependent

ISAM tables have some major disadvantages:

- ▶ Not binary portable across OS/platforms
- ▶ Only table sizes < 4 G supported

MyISAM tables

The MyISAM table type, based on ISAM code, was enhanced to overcome the disadvantages of the ISAM table, and to provide some more useful extensions. MyISAM is the default table type of MySQL since version 3.23. The index is stored in a file with the .MYI extension, and the data is stored in a file with the .MYD extension.

MyISAM supports three different table types:

- ▶ **Static tables**
Have a fixed length and are the default format MyISAM uses if no VARCHAR, BLOB or TEXT columns are used.
- ▶ **Dynamic tables**
Are used as default if VARCHAR, BLOB or TEXT columns are defined in the table.
- ▶ **Compressed tables**
Are read only tables. Static and dynamic tables can be compressed to use very little disk space. To write data to a compressed table, the table has to be uncompressed first.

Heap tables

Heap tables use hashed indexes and can only be held in memory. Therefore, they are mostly used as temporary tables.

InnoDB tables

In MySQL 4.0 InnoDB is enabled by default. The InnoDB engine is a complete separate database back-end produced by a Finnish company called *Innobase Oy* (<http://www.innodb.com>) and placed under MySQL. InnoDB tables are transaction safe (ACID compliant) and provide commit, rollback, and crash recovery capabilities. It also provides locking on row level and a consistent non-locking read in **SELECT** statements. Furthermore, it supports foreign key constraints. Tables and indexes are stored in a table space consisting of several files or even raw disk partitions. InnoDB uses B-tree indexes to locate data in the tables. MyISAM tables can be converted to InnoDB tables by using the **ALTER TABLE ... TYPE=INNODB** command.

BerkeleyDB (BDB) tables

In MySQL 4.0 BDB table support is provided but not activated by default. BerkeleyDB is also a transaction-safe storage engine, but has some disadvantages compared to InnoDB. BDB supports table locking and locking on page level. BDB tables are stored in files with the extension `.db`. More information regarding BerkeleyDB can be found at: <http://www.sleepycat.com>

MySQL standard SQL compliance

Up to version 4.0 MySQL meets only the entry-level SQL-92 and ODBC level 0-3.51 standard. But MySQL AB aims toward to support the full SQL-99 standard in the next versions. The most important missing compatibilities of MySQL default table MyISAM toward the ANSI-SQL-92 standard refers to:

- ▶ **Transactions:**

The MySQL default storage engine MyISAM does not support transactions. In this table type the developer of MySQL followed another paradigm for data integrity called “atomic operations”, as shown in Example 1-2. Changes produced by individual statements are committed to the database immediately as soon as they execute. In other words, a COMMIT statement implicitly follows each statement. To ensure database integrity code has to be provided in the application if required.

MySQL supports transactions with the InnoDB and BDB transactional storage engines. InnoDB provides full ACID compliance.

Example 1-2 Atomic operation

```
INSERT INTO table1 ...  
COMMIT  
DELETE FROM table2 ...  
COMMIT
```

► **Referential integrity**

Referential integrity ensures that relationships between tables remain consistent. When one table has a foreign key to another table, the concept of referential integrity states that it is not allowed to add a record to the table that contains the foreign key unless there is a corresponding record in the linked table. In MySQL Server, MyISAM tables do not support foreign key constraints. MySQL only parses the FOREIGN KEY syntax in **CREATE TABLE** commands, but does not use or store this information. Only InnoDB tables support checking of foreign key constraints including **ON DELETE CASCADE** and **ON UPDATE CASCADE**.

► **Views**

A view is a definition for a “virtual table” (virtual because there is no permanent allocation of storage space) which is assembled at reference time from selected rows and columns of one or more real tables.

Views are useful for two main reasons:

- They enable users to see data, from a generalized database design, in the form most convenient for their needs.
- They may be used to secure the database by restricting users to just that data they need to know.

MySQL does not support views.

► **Subqueries**

Subqueries are nested queries in a SQL statement (Example 1-3). They are not supported in MySQL 4.0.

Example 1-3 Subquery

```
SELECT FirstName, LastName, ZipCode
FROM customers
WHERE ZipCode =
    (SELECT ZipCode
     FROM customers
     WHERE FirstName = 'Wayne' AND LastName = 'John');
```

► **Stored procedures and Trigger**

A stored procedure is a set of SQL commands that can be compiled and stored in the server. If this is done, clients do not have to process the same set of SQL commands, but can refer to the stored procedure.

A trigger is a one or a set of SQL statements, which is invoked automatically when a particular event occurs.

Neither stored procedures nor triggers are supported in MySQL 4.0.

1.2.3 MySQL utilities

MySQL is distributed with a set of support utilities. The Internet however provides a quite larger set of third party tools to manage MySQL databases. In this section, we attempt to give a brief overview of the MySQL distributed set of supported tools.

Overview of the MySQL server-side scripts and utilities

This section introduces the distributed set of server-side tools included in the MySQL package:

► **mysqld**

Is the server SQL daemon, which has to run on the server. To use client programs, this program must be running, because clients gain access to databases by connecting the server.

► **mysqld_safe**

Is a wrapper script, which adds some more safety features to the SQL daemon such as restarting the server when an error occurs, and logging run-time information.

► **mysqld_multi**

Is a program for managing multiple MySQL server. This startup script can start or stop multiple servers installed on the system.

► **mysqld-max**

Is an extended mysqld server that includes additional features

- ▶ **isamchk**
Is a utility to describe, check, optimize, and repair ISAM tables, or to unpack a packed ISAM table
- ▶ **myisamchk**
Is a utility to describe, check, optimize, and repair MyISAM tables or to unpack a packed MyISAM table
- ▶ **mysql_install_db**
Creates the MySQL grant tables with default privileges. Normally, it is executed only once when first installing MySQL on a system.
- ▶ **mysql_fix_privilege_tables**
This script is used after an upgrade install operation to update the grant tables with any changes that were made in newer versions of MySQL.
- ▶ **make_binary_distribution**
Is a program that makes a binary release of a compiled MySQL version in order to distribute it
- ▶ **mysqlbug**
Is a bug reporting script. It can be used to send a bug reports to the MySQL list.
- ▶ **myisampack**
Is a pack utility to compress MyISAM tables
- ▶ **packisam**
Is a pack utility to compress ISAM tables

Overview of the MySQL client-side scripts and utilities

In this section, the distributed set of client-side tools provided by MySQL AB are introduced.

- ▶ **msq12mysql**
Is a script that converts mSQL programs to MySQL.
- ▶ **mysql**
Is a simple SQL shell for interactively entering queries or executing queries from file. It supports interactive and non-interactive use.
- ▶ **mysqlcc**
Is the MySQL Control Center that provides a graphical user interface (GUI) to the MySQL database server. It provides database and table management and, allows server administration.

- ▶ **mysqlaccess**
Is a script that checks the access privileges for a host, user, and database combination
- ▶ **mysqladmin**
Is a utility for performing administrative operations, such as creating and dropping databases, flushing tables to disk, reloading the grant tables, and managing log files
- ▶ **mysqlbinlog**
Is a utility to execute queries from a binary log file; this is used to recover from a crash using an old backup.
- ▶ **mysqlcheck**
Is used for table maintenance and crash recovery. In comparison to myisamchk, it can be used when mysqld is running. For mysqlchk, mysqld has to be stopped.
- ▶ **mysqldump**
Is a utility to dump a database for backup or for transferring data to another server. The dump contains SQL statements to create the tables and to populate the tables.
- ▶ **mysqlhotcopy**
Is a PERL script that is used to quickly make a backup of a database. It only can be used on UNIX to back up MyISAM and ISAM tables.
- ▶ **mysqlimport**
Provides a command-line interface to import data from text files
- ▶ **mysqlshow**
Is a utility that can be used to look at which databases exist, their tables, and also the columns in the tables
- ▶ **mysql_config**
Provides information on how to compile a MySQL client and connect it to MySQL
- ▶ **replace**
Changes strings in place in files or on the standard input
- ▶ **perror**
Can be used to display a description for a system error code

1.2.4 MySQL application programming interfaces (API)

The MySQL API is an interface by which an application program communicates with the MySQL database.

This section gives an overview of which APIs are available for MySQL.

Generally speaking there are three main approaches to connect to MySQL database as shown in Figure 1-14:

- ▶ JDBC with Java Connector
- ▶ ODBC with MyODBC
- ▶ Other APIs with C Library

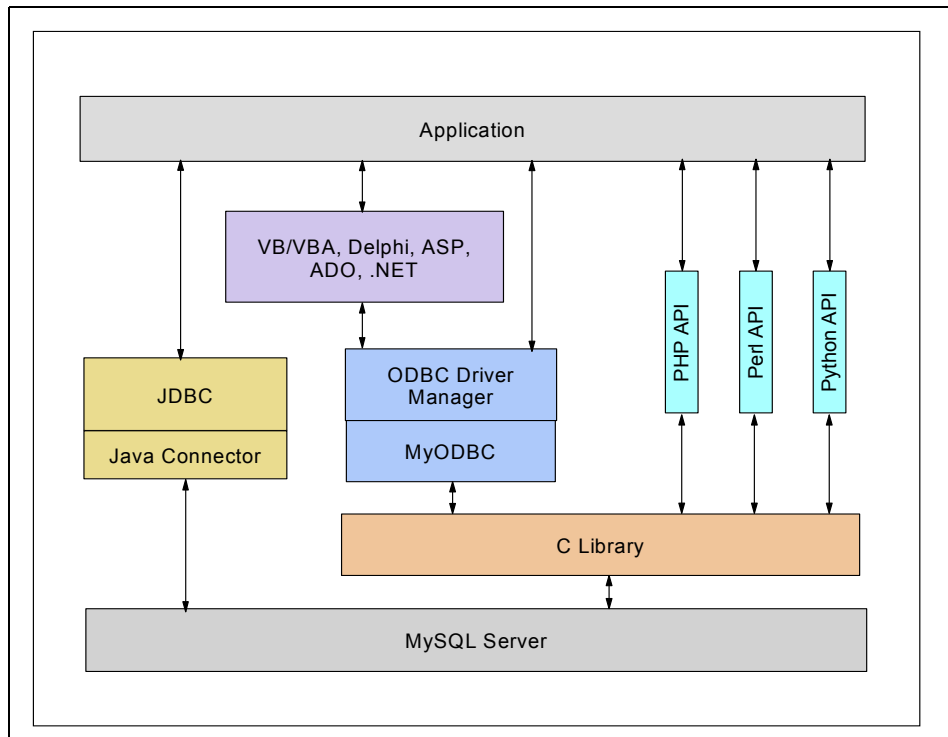


Figure 1-14 MySQL Application Programming Interfaces

The first approach is to connect the Java application using JDBC and the Connector/J, which is provided by MySQL AB.

The second approach is to either use ODBC directly from the application or utilize an application language like VB, Delphi, .NET, or ADO to access ODBC.

The third approach is to use the APIs provided by the programming languages like PHP, Perl, or Python.

There are several APIs available in MySQL:

▶ C API

The C API code is distributed with MySQL and is included in the *mysqlclient* library. It allows C programs to access a MySQL database.

▶ C++ API

The MySQL Connector/C++ is also distributed by MySQL AB. Information can be found at:

<http://www.mysql.com/products/mysql++/>

▶ JDBC API

The Connector/J is provided by MySQL AB and is used as a plugin in JDBC to allow Java applications to connect to MySQL server. Information about Connector/J can be found at:

<http://www.mysql.com/products/connector-j/>

▶ ODBC API

Open Database Connectivity (ODBC) is an API, which is based on the Call-Level Interface (CLI) specifications. The Connector/ODBC is provided by MySQL AB and is called MyODBC. Additional Information can be found:

<http://www.mysql.com/products/myodbc/>

▶ PHP API

PHP contains support for accessing several databases including MySQL. Information about MySQL access can be found in the PHP documentation, which can be downloaded at:

<http://www.php.net/download-docs.php>.

▶ PERL API

The Perl API consists of a generic Perl interface and a special database driver. The generic interface in Perl is called Database Interface (DBI) and for MySQL the driver is called DBD::mysql. This driver however does not support transactions. For transactions (see 1.2.2, “MySQL design and SQL compliance” on page 32) another driver, called DBD-mysql is needed. Information according DBI can be found at:

<http://dbi.perl.org/>

▶ Python API

The API to connect to MySQL for Python is call MySQLdb, and can be found at

<http://sourceforge.net/projects/mysql-python/>

▶ Tcl API

MySQLtcl is an API for accessing MySQL server from the Tcl programming language. It can be found at:

<http://www.xdobry.de/mysqltcl/>

► Eifel API

This interface is used from the Eiffel programming language, and can be found at:

<http://efsa.sourceforge.net/archive/ravits/mysql.htm>



Planning the migration from MySQL to DB2 UDB

As proper planning influences the success of a project significantly, this chapter discusses the overall migration project planning including the considerations before porting, assessment, and porting steps. The migration process, which are discussed in detail in the following chapters, are as follows:

- ▶ Project planning
- ▶ Application assessment and system planning
- ▶ The migration process
 - Porting preparation and installation
 - Database structure porting
 - Data porting
 - Application porting
 - Basic administration
 - Testing and tuning

This chapter informs you about how IBM migration specialists can support you in your migration project in any of the steps. The available migration tools such as the *IBM DB2 Migration Toolkit (MTK)* are also discussed.

2.1 Migration project planning overview

A migration project starts with a migration assessment to understand what needs to be done and how long it will take. A systematic and organized analysis provides a detailed picture of the full project. The assessment requires good knowledge of the application to be migrated, and the products to be used.

To assess the application you want to port, you should create an application profile that describes the application architecture, technologies used, application functions, application interface, application environment characteristics, database detail, and application size, etc.

Based on the application profile, you can then plan the software and hardware needed for the target system. The planning stage is also a good time to consider the rich DB2 functions and features of the DB2 UDB product family, which can increase your productivity and reduce the maintenance cost.

You also need to understand the skills required and the available resources for the migration project. IBM also provides a variety of DB2 courses to help IBM customers learn DB2 UDB quickly.

Migration assessment provides you the overall picture of the migration tasks and effort needed. A migration project plan can be created based on the migration assessment to manage each migration steps.

Tools can help you saving time in your migration project. Support for the typical migration steps is provided by various available tools. IBM offers the free migration tool *IBM DB2 Migration Toolkit (MTK)* for migrating from various relational database systems to DB2 UDB.

The process of database and application migration consists of the following main steps, which are discussed in detail later:

- ▶ Porting preparation and installation
- ▶ Database structure porting
- ▶ Data porting
- ▶ Application porting
- ▶ Basic administration
- ▶ Testing and tuning
- ▶ User education

Experienced IBM specialists can support you in during any phase of the migration project. Special migration offerings are provided by IBM Worldwide.

2.1.1 Benefits of migrating to DB2 UDB

DB2 UDB is number one in performance and number one in market share, with more than one million user licenses world wide. DB2 UDB offers open, industrial-strength database management for e-business, business intelligence, transaction processing, and a broad range of applications.

Some primary motivators for migration are:

- ▶ **Multi platform support**
DB2 UDB is a true cross-platform database management system (DBMS), running on a wide variety of systems including Linux, Windows 98/NT/2000/XP, Solaris, HP-UX, and AIX®.
- ▶ **Improved performance**
DB2 UDB has leading performance across a broad range of application and industry benchmarks on a range of hardware and software platforms. To have a look at some current performance measurements, please visit:
<http://www.ibm.com>
- ▶ **Industrial reliability and high availability support**
DB2 UDB provides superior availability with customer proven results.
- ▶ **Industrial scalability**
DB2 UDB has the best scalability. Only DB2 UDB does transaction processing and business intelligence scaling to 1000 nodes. For more information about scalability see:
<http://www.ibm.com>
- ▶ **Integrated support for native environments**
DB2 UDB conforms to many standards including the operating system it supports. It maps closely onto internal resources for performance and scalability. All these considerations make it more reliable and integrate it to the operating system.
- ▶ **Integrated system management tools**
DB2 UDB Version 8 introduced a number of new tools like the *Health Monitor*, the *Health Center*, the *Replication Center*, and the *Storage Management* tool. In addition it includes major improvements to existing tools like the *Configuration Assistant*, the *Control Center*, and the *Development Center*.
- ▶ **Self-managing and resource tuning capability**
DB2 UDB also has included self-managing and resource tuning database technology that lets database administrators choose to configure, tune, and manage their databases with enhanced automation. The innovative database manageability means administrators spend less time managing routine tasks, and more time focusing on tasks that help enterprises gain and maintain a sustainable competitive advantage.

- ▶ **Data replication service**
DB2 UDB includes a replication solution that ensures timely, reliable, and consistent data across an enterprise.
- ▶ **Information integration**
DB2 UDB provides a full family of information integration technologies. These products provide a robust infrastructure for storing, searching, federating, caching, transforming, and replicating diverse and distributed information. For more information about Information Integration, please visit:
<http://www.ibm.com>
- ▶ **Integrated Web access**
DB2 UDB provides Web access to enterprise data on DB2 databases through native support for Java/JDBC, embedded SQL for Java (SQLJ) and Net.Data@.
- ▶ **Web services applications**
DB2 UDB can be accessed as a Web service provider, and it is usually teamed with the *IBM WebSphere* family products to provide a complete Web services framework.
- ▶ **Basic data warehousing functionality**
DB2 UDB offers the *Data Warehouse Center*, a component that automates data warehouse processing.
- ▶ **IBM program for assistance to developers**
PartnerWorld® for Developers is an IBM program that provides business, technical, and marketing services to partners in order to help them in developing and marketing applications. Please visit <http://www.developer.ibm.com> for more information.

2.1.2 IBM migration offering

IBM migration specialists around the world have advised customers on migration projects to DB2 UDB in more than 3500 cases. Why shouldn't you use their experience in your migration project? Before starting the assessment phase, you should contact the *Software Migration Project Office (SMPO)* for *no charge*, porting estimates as well as access to a team of migration experts.

A proven methodology is used by the IBM migration team, which helps you reduce costs and risks associated with a migration while addressing its major components, the applications, the database design and the data.

IBM's consultants can advise your organization regarding a phased migration approach. This approach includes:

- ▶ Assessment of the database conversion effort
- ▶ System planning and database design

- ▶ DB2 migration assessment
- ▶ Installation of DB2 UDB and other required products
- ▶ Pilot migration
- ▶ Full migration of data and applications

IBM services are available for assistance in any of the migration phases.

If you have a migration project in mind, please contact one of the following addresses:

- ▶ In North America and Latin America: <mailto:db2mig@us.ibm.com>
- ▶ In UK, Europe, Middle East and Africa: <mailto:emeadbct@uk.ibm.com>
- ▶ In Japan, India and Asia Pacific: <mailto:APDB2@nz1.ibm.com>

You can find up to date details about current offerings, success stories, literature, and other information on the *DB2 Migrate Now!* Web site:

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

More information about the DB2 migration team can be found at the *Software Migration Project Office (SMPO)* Web site:

<http://www-306.ibm.com/software/solutions/softwaremigration/dbmigteam.html>

2.1.3 Education

DB2 UDB provides an easy to use feature rich environment. It is important that those individuals involved in the migration be appropriately trained, so that the full advantages of those features are realized.

There is a lot of training material available like self studying guides and redbooks. IBM also offers a variety of DB2 courses; a very useful one is the course for experienced database administrators that are new to DB2 UDB.

For further information regarding DB2 UDB training, please visit the DB2 Web site at:

<http://www-306.ibm.com/software/data/education/>

2.2 Application assessment

An application assessment is the first step in the migration planning. The assessment will help you understand the scope of the migration project and prepare the detail migration project plan to design the target system.

You need to understand how your application works and what resources are needed. There are probably a lot of characteristics of your application that influence your system planning and the scope of the migration effort.

The main information for collection during the application assessment includes:

- ▶ Application architecture
 - Standalone
 - Client/server
 - Tier architecture
- ▶ Database architecture
 - Number of databases
 - Number of tables
 - Number of indexes
 - Users, access rights and privileges
- ▶ Size of data stored in the database
 - Bytes stored in system tables (users)
 - Bytes stored in tables
 - Bytes stored in indexes
 - Log file size
- ▶ Source code language
 - PHP
 - Perl
 - Java
 - C/C++
 - any other programming language
- ▶ Database interface
 - Direct database access through API
 - Database access layer like ODBC/JDBC
- ▶ Operating system
 - Linux
 - AIX, UNIX
 - Windows
 - any other operating system
- ▶ Used hardware
 - CPU
 - Memory
 - Hard disk

In a client/server-environment be sure to describe the application in both the client and server environment.

2.3 System planning

Based on your application profile created during the application assessment, you should be able to plan your target system properly.

As DB2 UDB supports different hardware platforms and multiple operating systems like Linux, Windows, and AIX etc., you are not limited in one platform to migrate your databases to. You can select on which system the migrated application should run based on the application nature and future enhancement requirements.

As shown in Figure 2-1, the target system can be the same system as the source system, or a different one with a different operating system and hardware. You might even want to make your database server a separate machine from the machine your application runs on (*two-tier architecture*).

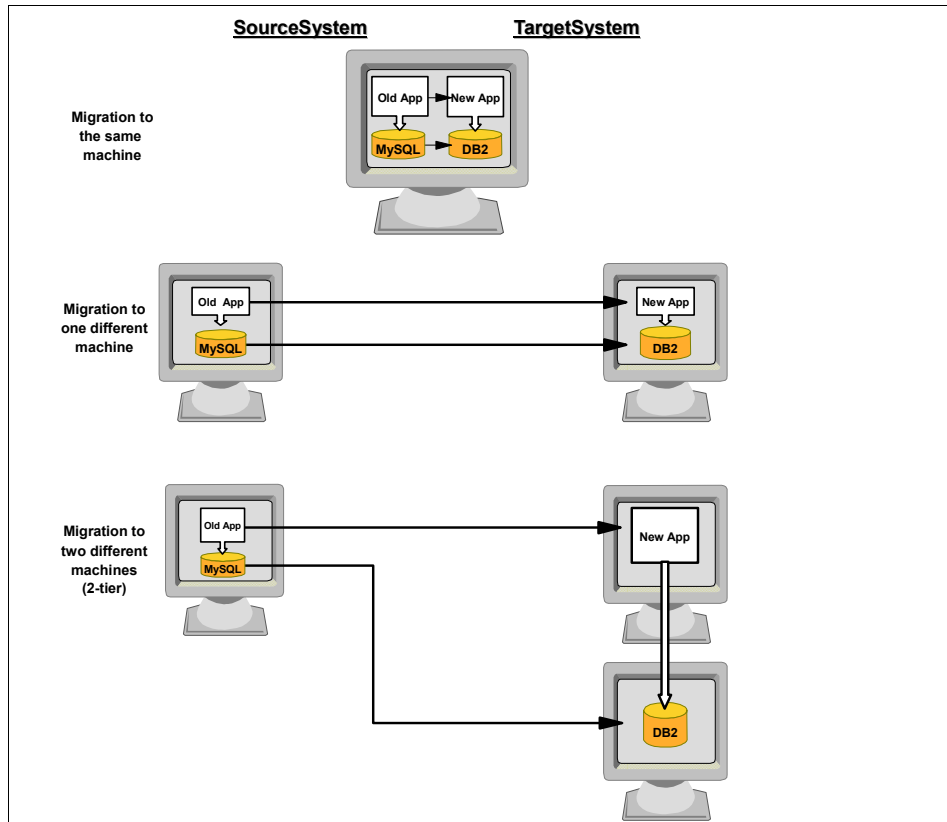


Figure 2-1 Sample migration scenarios

If you decide to use a new machine for the migrated program, you need to plan what kind of hardware you want to use, and which operating system you want to install on it.

In either case you should check if the hardware of your target system meets the minimum requirements of the following:

- ▶ Operating system
- ▶ DB2 UDB
- ▶ Application
- ▶ Data
- ▶ Migration tools (if used)

2.3.1 Software

You must determine which software must be installed on your target system. This can include the following:

- ▶ Operating system (Linux, AIX, UNIX, Windows, others)
- ▶ DB2 UDB version
- ▶ Application to be ported
- ▶ Migration tools (if used and installed on target system)
- ▶ Any software that you have on your source system, which is required by your application to run properly. This can be:
 - HTTP-server
 - Web application server
 - Development environment
 - Additional software (like LDAP or others)

Be sure to have the latest versions and fix packs of the planned products. Please check if the chosen software is supported on the chosen operating system.

2.3.2 Hardware

When starting the migration process it is important to have a target platform that meets the minimum requirements of all the software that will be installed on it. Please check the supported hardware platforms depending on the chosen software.

Your application also requires hardware resources. Be sure to have enough disk space for your application and to hold the data to be transformed.

IBM provides a variety of hardware systems especially designed to meet your business needs. For information about IBM eServers please check the IBM Web site at:

<http://www.ibm.com/eservers/>

2.3.3 Migration tools

There are free and commercial tools available to assist you migrating your application from MySQL to DB2 UDB. The tools offer a variety of functions and are available on a different set of operating system. IBM offers the free *IBM DB2 Migration Toolkit (MTK)*.

If you decide to use a tool, be sure that it fulfills the requirements you have and that you have an appropriate platform to run the tool.

IBM DB2 Migration Toolkit (MTK)

MTK helps you to migrate from relational database systems like MySQL, Oracle and Microsoft SQL Server to DB2 UDB.

The tool can be used to generate *Data Definition Language (DDL)* scripts to create a database, tables, indexes and primary keys in DB2 UDB based on an existing MySQL database.

It also supports you retrieving data out of a MySQL database and importing the data into the created DB2 UDB database.

In our sample project we used MTK on the following steps:

- ▶ Extract DDL from our MySQL sample database
- ▶ Convert DDL to DB2 UDB syntax
- ▶ Create the DB2 database
- ▶ Create the DB2 database structure
- ▶ Export MySQL data
- ▶ Import data into DB2 UDB

If you want to download MTK or get more information about it, please refer to:

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

Other available migration tools

Some other tools can be found on the Internet including the following (among others). These tools and services are not provided by IBM, nor does IBM make guarantees about the usage of these tools:

- ▶ **SQLPorter**
A commercial software by RealSoftStudio Inc. for Windows operating system. For further details see <http://www.realsoftstudio.com>
- ▶ **SQLWays**
A commercial tool by Ispirer Systems for Windows operating system. For more details see <http://www.inspirer.com>

2.4 The migration process

For accurate planning it is important to understand the steps to complete the migration project. To estimate the project effort correctly and migrate the database and application successfully, each of the steps should be planned. Figure 2-2 shows the basic migration process steps.

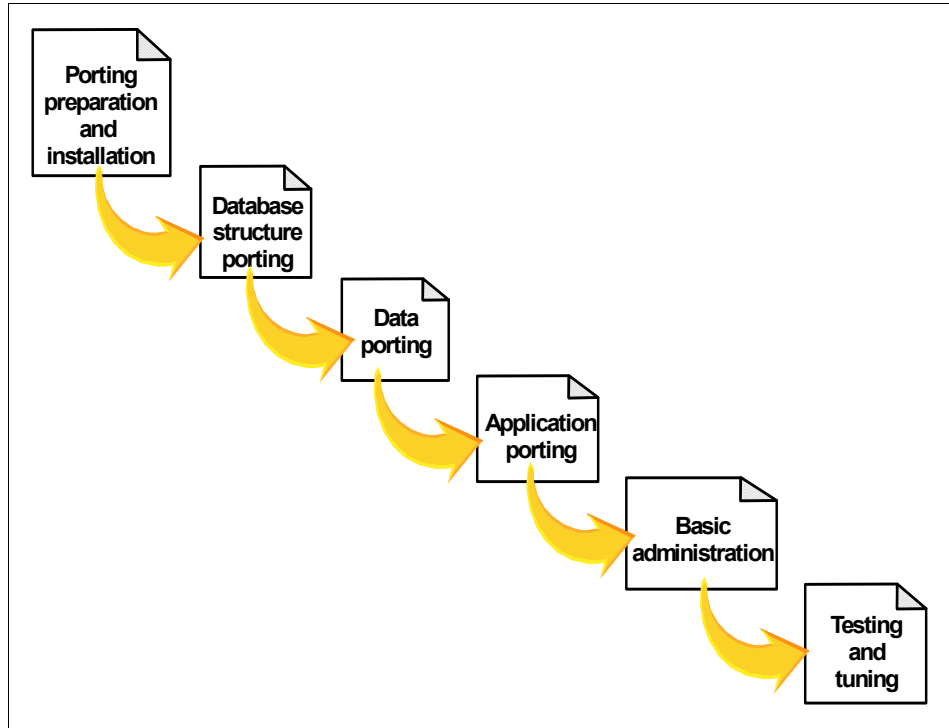


Figure 2-2 Steps of the migration process

2.4.1 Porting preparation and installation

After deciding which target system you want to use, you can set up the following: Install the hardware, the operating system with users and access rights, the network connections, any used software, and finally the DB2 UDB version of your choice.

2.4.2 Database structure porting

Once the DB2 UDB is installed in the target, you can port the database structure from the source MySQL database to DB2 UDB.

The database structure is usually described by *Database Definition Language (DDL)* statements. DDL scripts include the creation of tables, keys, and indexes. Since the DDL syntax for MySQL and DB2 UDB is different, a converting process is required. Using a tool to port database structure can save time and effort. Figure 2-3 shows the database structure porting the basic steps.

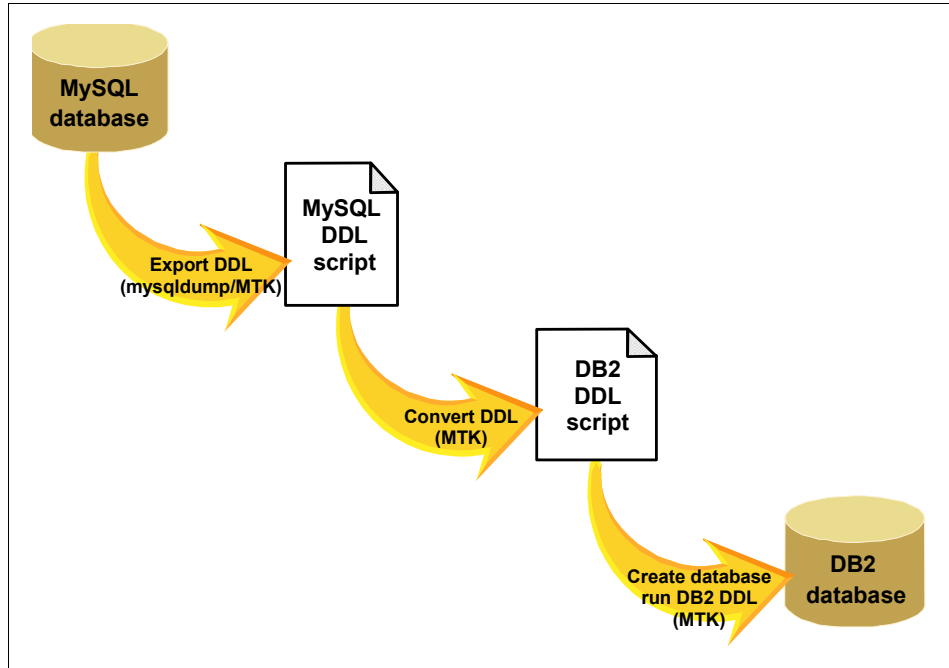


Figure 2-3 Database structure porting process

Export DDL from MySQL databases

You have to retrieve the information about tables, keys, and indexes from your source MySQL database.

Tools like MTK can do most parts of this task automatically. You should check the results of the output of the tool you use. Of course you can also do this step manually, but be aware to cover all the database objects.

Convert DDL to DB2 syntax

As the DDL used by MySQL is a little different from the DDL used by DB2 UDB, the DDL used to create MySQL database and objects have to be converted to DB2 UDB syntax.

This part is also supported by tools like MTK; again, you can do it manually. In this case be aware of the different data types that MySQL and DB2 UDB use.

If you plan to change the logical model of your database structure to enhance your application and take advantage of the DB2 UDB's functions and features, the DDL should be modified in this step.

Create DB2 database structure

Once you have the DDL scripts in DB2 syntax, you should create your DB2 UDB database and run the DDL scripts to create the database structure.

This part is again supported by tools like MTK. Be sure to check the success of the creation of all database objects like tables, keys, and indexes.

2.4.3 Data porting

With the database installed on your new system and the structure of your database created, you should migrate the data from the source to the target system.

Databases usually provide mechanisms for exporting (dumping) and importing (loading) data. We categorized the MySQL data into:

- ▶ User data: Data about users, access rights, and privileges
- ▶ Application data: Data that is created and used by an application

Figure 2-4 shows the basic steps of data porting.

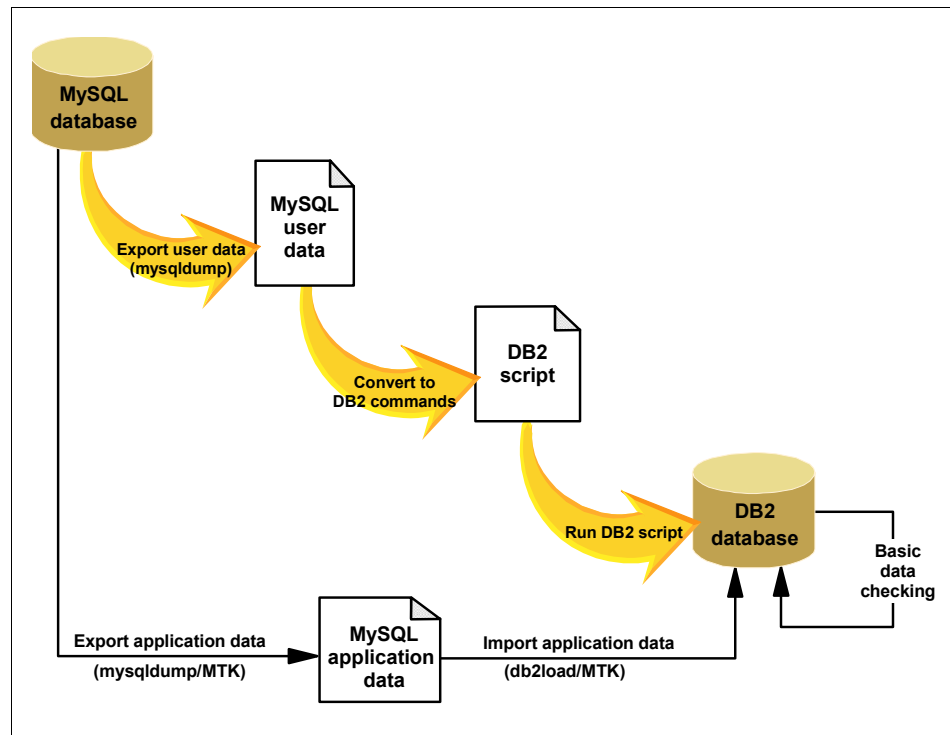


Figure 2-4 Data porting process

Export MySQL user data

MySQL and DB2 UDB use different database security mechanisms. The database object access privileges are stored in the MySQL database. You need to understand what MySQL access rights your application has and how you can map them to DB2 UDB. Depending on your application you probably have to export user data from the MySQL database.

Map MySQL user data to DB2 user data

As in DB2 UDB users, access rights and privileges are maintained in a different way, you have to convert the MySQL user data to DB2 commands for granting privileges and using operation system's user ID management functions to create user ids.

In this step, you create scripts to create users and to grant them the DB2 UDB database and objects access privileges based on the MySQL user data.

Create DB2 user data

Once you have mapped your MySQL user data to DB2 user data, your users must be created in the DB2 UDB system and the necessary privileges must be granted to them.

The scripts with the DB2 commands for creating users and granting privileges should be run.

Export MySQL application data

You have to dump the application data out of the MySQL database. This step is supported by MTK or you can do it manually. Be aware of the differences in the format of DATE, TIME and other data types.

Convert MySQL application data to DB2 format

If your MySQL application supports data types that cannot be transformed by MTK automatically (like the BLOB data type), you have to do this manually.

Import application data into DB2 UDB

The exported (and maybe converted) data must finally be loaded into the DB2 UDB tables. MTK also supports this step, and of course data can be loaded into DB2 UDB manually.

Basic data checking

Once you have loaded all data into your database, you should do basic data checking. You should check for the correct number of rows per table and the correct representation of the field-values.

Also, be sure that you have all the users created in your DB2 UDB system.

2.4.4 Application porting

The porting of the database structure, and the data can be supported by tools quite easily. The porting of the application, however, requires manual conversion. A few tools exist that can support you in some tasks such as syntax highlighting, but in most cases the main conversion must be done manually.

The extent to which you have to change the application code depends on the database interface that is used in the source application. When a database access layer is used, the adoption is not that complicated, otherwise, the effort to port the application will probably be much higher.

Application source code changes

As there are some differences in the *Data Manipulation Language (DML)* of MySQL and DB2 UDB, you may have to change SQL statements in your application code directly.

You might find things that are not natively supported in DB2 UDB that you used in MySQL, so a workaround must be established to have the application behave as before the migration.

Database interface

Whatever interface between application and database is used by your application, the access to the database must be changed because the database has been changed.

If standardized interfaces like ODBC or JDBC are used, the changes will be less than if an application uses the native API of a database product.

Condition handling

Depending on the implementation of your application, there might be some changes in the condition handling part of the application.

Additional considerations

DB2 UDB offers rich, robust functions, which you can take advantage of in your applications.

You should consider modifying your application to use some of these features, which are different to MySQL:

- ▶ Concurrency
- ▶ Locking
- ▶ Isolation level
- ▶ Transactions
- ▶ Logging
- ▶ National language support

2.4.5 Basic administration

There are regular maintenance works the database administrator performs. These administration issues should be considered during migration planning.

Every database has its own way for backup and recovery, as these are vital tasks in database administration. The database should be backed up regularly, and the data retention period should be defined based on the business requirements.

If you have backup and recovery tasks defined on the source system, you probably want to migrate these tasks as well. Be sure to port any existing scripts for backup tasks to support DB2 UDB.

Both the database backup and recovery functions should be tested to ensure a safe environment to your application.

Log files

DB2 UDB logs differently than MySQL, so database administrators should be aware of the logging level that can be set, where log information is stored, and how to read these logs.

Find more information in Chapter 8, “Database administration” on page 235.

2.4.6 Testing and tuning

Once your new system is up and running, you have to verify that data and application functionality have been ported completely, and that system behavior has not changed in a way you do not want.

If you succeed with this step, you might tune your database in order to speed up your application.

Data checking

Beside the basic data checks that should be done when exporting and importing the data, you should also check if the application handles your data correctly and that manipulations on the databases like inserts, updates, or deletes do not lead to unexpected results.

You can perform this checking manually, or you can also write some scripts to have your data checked.

Code and application testing

It is very important that the behavior of your application has not changed. Interactions between components of your application must be tested as well as each module of the application. A code review of all changed code is recommended as well.

Troubleshooting

Whenever the migration leads to a problem, like wrong data or wrong application behavior, you have to determine what the problem is in order to have it fixed.

You should understand error messages from the application as well as DB2 error messages. The troubleshooting process includes studying the DB2 log files.

See the DB2 technical support Web site for help with specific problems:

<http://www-306.ibm.com/software/data/support/>

Basic tuning

Once your new system is working perfectly, you might want to tune it for an even better performance. With the correct database configuration, and some hints from DB2 tuning tools, you can speed up your queries quite easily.

DB2 UDB provides tools like *Performance Monitor* or *Index Advisor* to support you in speeding up your DB2 UDB system.



Migration scenario

In order to describe how migration can be done from MySQL to DB2 UDB in practice, our migration scenario provides a small but real world Web application written in PHP based on a MySQL 4.0 database. When choosing an application to show the migration process, the intention was not to find an optimal written application with all the possible tricks a genius programmer would use to set up the database structure or to do the programming, but to use a real application sample as is. Although the application we use for demonstrating the migration is written in PHP, 7.2, “Application source conversion” on page 173 also provides samples in other popular programming languages like Perl, Java, and C/C++.

The following topics are discussed in detail:

- ▶ Application description
- ▶ System environment

3.1 Application description

The application we use is a Web based shop and online catalog. The customer can either request information such as availability about provided items, or order parts in the shop. In this catalog or shop the fictive vendor (BHMS) offers reproduced radiators for different cars and manufacturers.

3.1.1 Steps using the application

The flow diagram (Figure 3-1) describes the application flow and function at a high level.

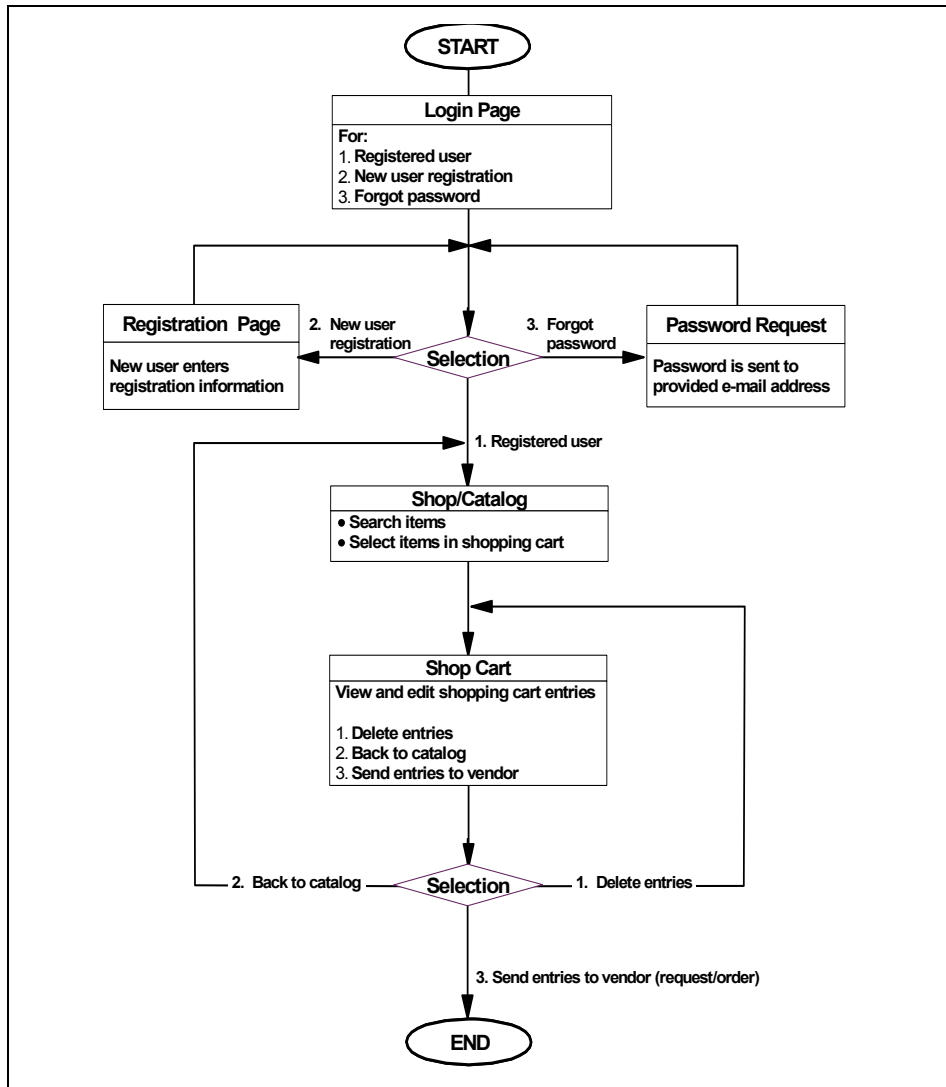


Figure 3-1 Flow diagram of the sample application

When a customer enters the Web site, the start page (Figure 3-2) provides three functions:

- ▶ Request for user access: To log in as an already registered user
- ▶ Log in to shop: To register as a new user
- ▶ You have forgotten your password?: To request the password if it was lost

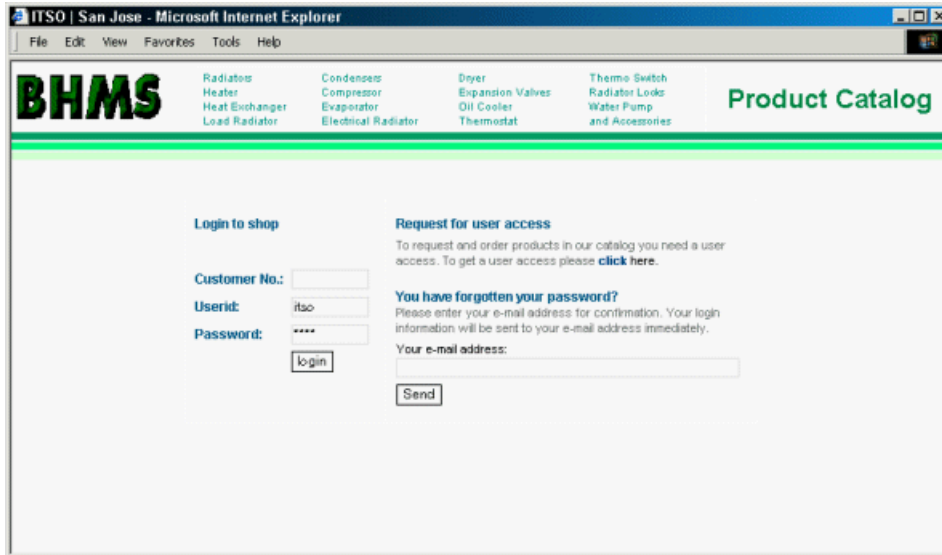


Figure 3-2 Start page of the Web application

In *Login to Shop*, the customer uses his user ID and password to log in. The customer is able not only to request information, but also to order parts in the shop. To order parts, the customer also has to enter the customer number.

From *Request for user access*, a new user can be registered. Providing a complete address information in the registration form (Figure 3-3) creates a new user in the application.

BHMS Radiators: Condensers: Dryer: Thermo Switch:
 Heater: Compressor: Expansion Valves: Radiator Locks:
 Heat Exchanger: Evaporator: Oil Cooler: Water Pump:
 Load Radiator: Electrical Radiator: Thermostat: and Accessories

Product Catalog

BHMS Product Catalog

Thank you for your interest in using our product catalog.
 Please fill out the form and click on SEND.

Userid: kathy
Password: testpw
Company: ITSO
Title: Mrs.
First name: Kathy
Last name: Smith
Address: 650 Harry Rd.
Zip code, city: 95120 San Jose CA
Country: USA
e-mail: kathy@yahoo.com
Phone: 408-927-3633
Fax:
 Send

Figure 3-3 Registration form

The application will then inform the new user to the vendor by e-mail (Figure 3-4). The entered user ID and password can be used right away to log in.

Dear Sir or Madam,

a new user has registered in the Product Catalog of your Homepage.
 Please send him his customer number as soon as possible.

Title :Mrs.
 First/last name :Kathy Smith
 Company :ITSO
 Address :650 Harry Rd.
 Zip code, city :San Jose Ca
 e-mail :kathy@yahoo.com
 Phone :408-927-3633
 Fax :
 Userid :kathy
 Password :testpw

Figure 3-4 Registration information sent to the vendor

Using the **You have forgotten your password?** link, a forgotten password can be requested and the password will be sent by the application to the provided e-mail address entered in the form.

When you successfully log in, the shop or product catalog view is presented to the customer as shown in Figure 3-5.

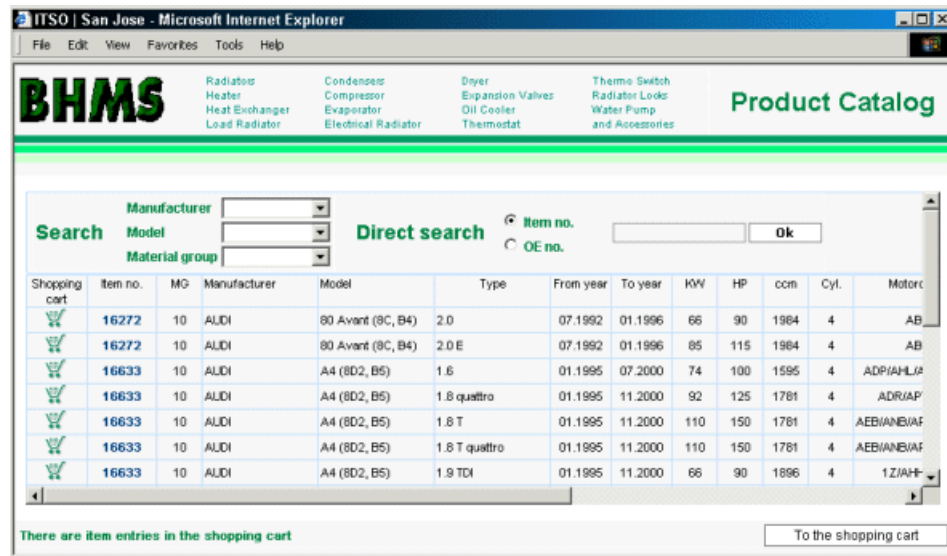


Figure 3-5 Shop or catalog view

In this provided form the user can sort, search, and select items for their shopping cart. One may also switch to a detail item view (Figure 3-6) by clicking on the highlighted item number.

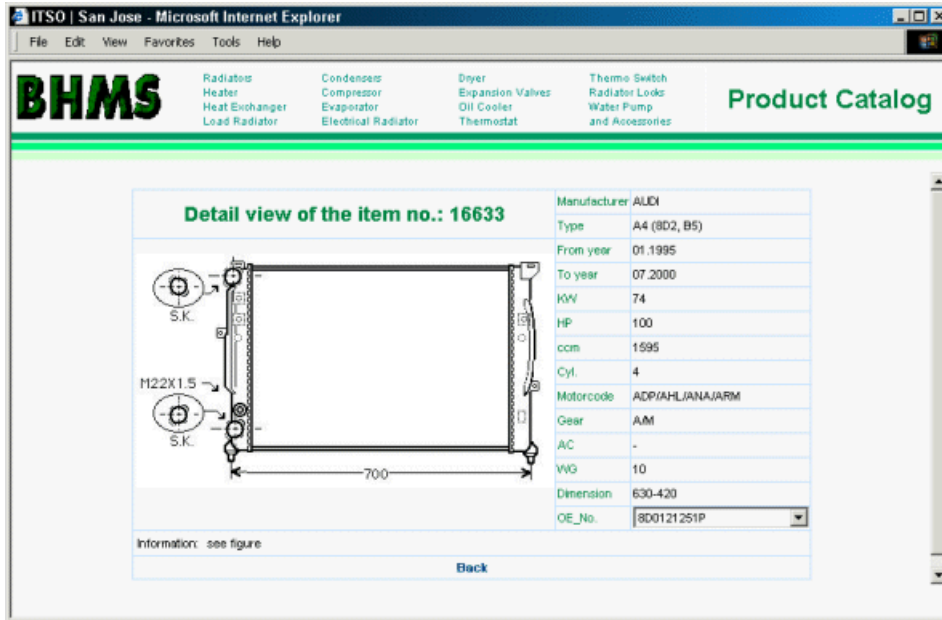


Figure 3-6 Detail item view

When selecting the items is finished, the customer will be taken to the shopping cart view (Figure 3-7). The customer can enter the quantity of items for request or order, or one can even delete entries from the shopping cart.

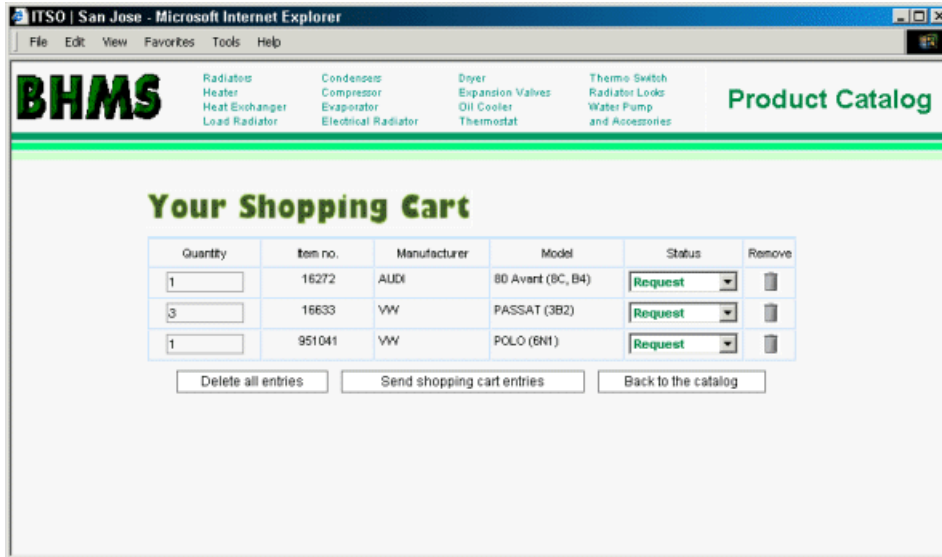


Figure 3-7 Shopping cart view

After adding the purchase order quantity to each item, the order or request is submitted by clicking **Send shopping cart entries**. In consequence, the customer information and the selected item information is sent to the vendor by e-mail (Figure 3-8).

Dear Sir or Madam,

the following Request/Order was submitted from the Product Catalog Webpage:

Title :Mrs.
 First/last name :Kathy Smith
 Company :ITS0
 Address :650 Harry Rd.
 Zip code, City :San Jose Ca
 e-mail :kathy@yahoo.com
 Phone :408-927-3633
 Fax :
 Customer No. :

Item no.	Manufacturer	Model	Quantity
16272	AUDI	80 Avant (8C, B4)	1
16633	VW	PASSAT (3B2)	3
951041	VW	POLO (6N1)	1

Figure 3-8 Request/Order information sent to the vendor

3.1.2 Database structure

The database used by the application consists of four tables as described below:

- ▶ The table USERS contains the whole customer registration information.
- ▶ The table CATALOG contains the item information.
- ▶ OENUMER is a table, which holds manufacturer item numbers.
- ▶ SHOPPING_CART is the shopping cart table, which holds the items and user information during the order transaction. After the order or request is submitted, and an e-mail is sent to the vendor, the information is deleted.

For detailed table information see Figure 3-9.

Note: These tables only use CHAR, VARCHAR, and INTEGER data types. In Chapter 5, “Database porting” on page 89, data type conversion between MySQL and DB2 UDB is discussed in detail.

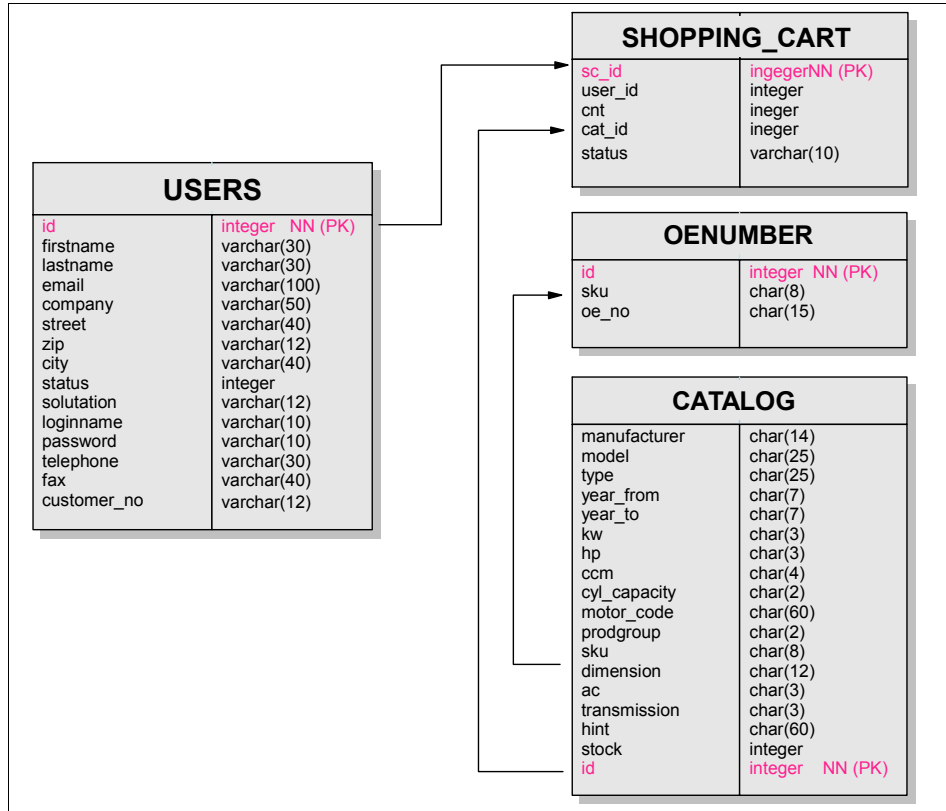


Figure 3-9 ERD - Diagram

3.2 System environment

In this section, the hardware and software environment used in our migration scenario for application porting is described.

The hardware used for porting the application sample in our scenario is an IBM Netvista Workstation:

- ▶ Intel® Pentium® 4 CPU, 2,4 GHz
- ▶ 1 GB Ram
- ▶ 39 GB IDE hard disk drive.

Note: We recommend that you allocate a minimum of 256 MB of RAM for DB2 ESE. Additional memory should be allocated for other software and communication products.

Although MySQL is becoming more and more popular with Microsoft Windows installations, it is mostly a part of LAMP (Linux, Apache, MySQL, PHP/Perl/Python), an open source enterprise software stack.

The “LAMP” software stack and additional software we use in our migration scenario is as follows:

- ▶ SuSE Linux 8.1 e
- ▶ Apache 1.3.26 (included in SuSE Linux)
- ▶ MySQL Standard 4.0.17 (MySQL AB)
- ▶ PHP 4.2.2
- ▶ DB2 UDB Enterprise Server Edition 8.1
- ▶ IBM DB2 Migration Toolkit (MTK)

DB2 for Linux can be downloaded from the Web site <http://ibm.com/db2/v8>. Further information on installing and maintaining DB2 for Linux can be found on the Web site:

<http://ibm.com/db2/linux>

IBM DB2 Migration Toolkit (MTK) is available free of charge from IBM at the following URL: <http://www.ibm.com/software/data/db2/migration/mtk> and is used to simplify and improve the migration from MySQL to DB2 Universal Database (UDB).

With MTK, the conversion of database objects such as tables and data types, and the migration of data can be done automatically into equivalent DB2 database objects.

The installation and configuration of DB2 UDB and the MTK is covered in Chapter 4, “Installation” on page 75.



Installation

This chapter discusses the target system environment setup. For the database server, we guide you through the installation process for DB2 UDB Version 8.1 for Linux including the hardware and software prerequisites. The application server has to be examined to ensure that the existing software has proper DB2 support. If this is a completed new system setup, make sure that all the required software are included in the installation list. Furthermore, we describe the download and steps to deploy the IBM DB2 Migration Toolkit for MySQL.

4.1 DB2 UDB ESE V8.1.4 on Linux

Regardless of the platform and the environment used for your migration project verification of the hardware and software requirements is the starting block.

4.1.1 System requirements

This section provides supported Linux distribution information, hardware, software, and communication requirements of DB2 UDB on Linux environment.

Linux distributions supported by DB2

Table 4-1 lists some IBM DB2 validated Linux distributions at the time of writing this book. For the most recent, all-supported Linux distributions that have gone successfully through the IBM DB2 for Linux validation program, check:

<http://www.ibm.com/software/data/db2/linux/validate/>

Table 4-1 *Currently supported Linux distributions, kernels, and libraries*

Platform	Distribution	Kernel	Library	Comment
x86	Red Hat Enterprise Linux 2.1 AS/ES/WS	2.4.9-e16	glibc 2.2.4	
	Red Hat 8.0	2.4.18-14	glibc 2.2.93-5	
	Red Hat Enterprise Linux (RHEL) 3 AS/ES/WS	2.4.21-4.EL	glibc-2.3.2-95.3	Not yet supported for production use.
	SuSE Pro 8.0	2.4.18	glibc 2.2.5	
	SuSE Pro 8.1	2.4.19	glibc 2.2.5	
	SuSE Linux Enterprise Server (SLES) 7	2.4.7	2.4.7	
	SuSE Linux Enterprise Server (SLES) 8	2.4.19	glibc 2.2.5	Validated up to SuSE Service Pack 2 level

To learn more about the IBM DB2 UDB for Linux validation program go to:

<http://www.ibm.com/software/data/db2/linux/validate/progdesc.html>

Hardware requirements

IBM hardware is supported for DB2 UDB Workgroup Server Edition and DB2 UDB Workgroup Server Unlimited Edition:

- ▶ Intel 32-bit
- ▶ IBM eServer iSeries that support Linux
- ▶ IBM eServer pSeries that support Linux

DB2 UDB Enterprise Server Edition is supported on:

- ▶ Intel 32-bit
- ▶ Intel 64-bit
- ▶ AMD 64-bit
- ▶ S/390® 9672 Generation 5 or later, Multiprise® 3000 or eServer zSeries
- ▶ IBM eServer iSeries that support Linux
- ▶ IBM eServer pSeries that support Linux

Disk requirements for DB2 UDB servers

The disk space required for DB2 UDB depends on the type of installation you choose:

- ▶ **Typical:** Requires 450 to 500 MB
With Typical installation type, DB2 is installed with most features and functionality, including graphical tools such as the Control Center and Configuration Assistant
- ▶ **Compact:** Requires 350 to 400 MB
With Compact installation type, only basic DB2 features and functions are installed, graphical tools are not included.
- ▶ **Custom:** Requires 350 to 700 MB
With Custom installation type, you can select the features you want to install. The disk space needed varies based on the selected features.

When you install DB2 Enterprise Server Edition or Workgroup Server Edition using the DB2 set up wizard, size estimates are dynamically provided by the installation program based on installation type and component selection.

If the space required for the installation type and components exceeds the space found in the path specified, the setup program issues a warning about the insufficient space. The installation can continue. If the space for the files being installed is in fact insufficient, installation will stop, and the setup program needs to be stopped if additional space cannot be provided.

Remember to include disk space allowance for required software, communication products, and documentation. In DB2 UDB Version 8, HTML, and PDF documentation is provided on separate CD-ROMs.

Memory requirements for servers

At a minimum, a DB2 UDB server installation requires 256 MB of RAM. Additional memory may be required. However, when determining memory requirements, be aware of the following:

- ▶ Additional memory may be required for non-DB2 software that may be running on your system.
- ▶ Additional memory is required to support database clients.
- ▶ Specific performance requirements may determine the amount of memory needed.
- ▶ Memory requirements will be affected by the size and complexity of your database system.
- ▶ Memory requirements will be affected by the extent of database activity and the number of clients accessing your system.

Communication requirements

When using TCP/IP as the communication protocol, no additional software is needed for connectivity. For more supported communication protocols, please refer to DB2 UDB manual *Quick Beginnings for DB2 Servers V8*, GC09-4836 or log on to IBM's Infocenter for DB2 at:

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp>

4.1.2 Installation procedure

Table 4-2 shows the four installation methods in which DB2 UDB can be installed. For completeness we have provided the information for supported Linux, UNIX, and Windows operating systems. Each of the methods listed has its own pros and cons, and depends on the environment. For a discussion on the preferred method, please refer to the IBM redbook *Up and running with DB2 for Linux*, SG24-6899.

Table 4-2 DB2 installation methods

Installation method	Linux	UNIX	Windows
DB2 setup wizard	yes	yes	yes
db2_install	yes	yes	no
Response file installation	yes	yes	yes
Native installation tool	e.g. RPM	yes	no

For our project, and for a conversion project in general, the DB2 UDB *Application Development Client* is required. It provides libraries for application

development. If the application server and the database server are to be placed at same system, you can install both DB2 UDB server and Application Development Client in one step by selecting Custom installation type.

In our project we followed the following steps for the install of DB2 UDB V8.1 on Linux:

- ▶ Log on to Linux as root user.
- ▶ Mount the CD-ROM using `mount /cdrom`
- ▶ Change to the CD-ROM directory `cd /cdrom`
- ▶ Launch DB2 Setup wizard with `d2setup`
- ▶ Choose **Install Products** once the DB2 Launchpad opens.
- ▶ On the Select installation type panel click **Custom**.
- ▶ On the Select the features panel make your choices. We make sure **Application Development tools** is selected as shown in Figure 4-1. Our sample application uses PHP, which needs the library in Application Development tools.



Figure 4-1 DB2 custom installation with Application Development tools selected

- ▶ For DAS user, we chose the default **dasusr1**.
- ▶ In Instance setup, we let DB2 create the instance for us by choosing **Create a DB2 instance**.
- ▶ We do not require the partitioning capability of DB2 in our sample application, so we chose **Single-partition instance** in **Instance use**.
- ▶ In Instance-owning user, we let DB2 create the instance owner ID and use all the default settings.
- ▶ In *F*enced user, we also let DB2 create the ID for us.

- ▶ For Tools catalog, we create a local tools catalog by selecting **Use a local database**. We also use the default local database TOOLSDB.
- ▶ For the administration contact list, we selected to use a local contact list.
- ▶ At the end, the setup wizard will provide you a summary of the installation options you have selected. Review it and click **Finish** to start the installation.

Fix pack installation

We recommend you install the latest DB2 UDB FixPak.

- ▶ Download FixPak from:
<http://ibm.com/software/data/db2/udb/support.html>
- ▶ Change to the directory in which the installation image is located.
- ▶ Enter the **installFixPak** command to launch the installation.
- ▶ Update instances to run against the new code with **db2iupd**.
- ▶ Update DB2 Administration Server (DAS) using **dasupdt**.

db2setup command options

DB2setup command provides options for you to specify the locations of the log files, trace file, or response file created during installation (see Figure 4-2). Log files are useful in verifying the installation status and tracing the problem if installation failed. By default **db2setup** starts the graphical user interface (GUI) and picks up the language flag from the operating systems settings.

```
>>-db2setup-----+-----+-----+----->
      '--i--language-'  '--l--log_file-'
>-----+-----+-----+-----><
      '--t--trace_file-'  '--r--response_file-'  +--?--+
                                           '--h-'
```

Figure 4-2 Db2setup command options

If the log file option is not specified, the db2setup.log and db2setup.err are stored in the /tmp directory of the Linux operating system. Example 4-1 shows an example of the db2setup.log.

Example 4-1 DB2setup log file

```
DB2 Setup log file started at: Thu 19 Feb 2004 01:26:00 PM PST PST
=====

Operating system information: Linux 2.4.19-4GB.#1 Fri Sep 13 13:14:56 UTC 2002
i686
```

4.1.3 Instance creation

For our project we let DB2 UDB create the DB2 instance automatically during installation.

While this is default, you can turn automatic instance creation off during installation as shown in Figure 4-3, and create instances and databases manually after the installation has completed.

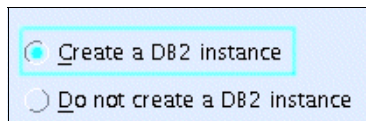


Figure 4-3 Instance creation option

DB2 UDB provides two methods to manually create instances:

- ▶ The **db2i setup** command starts a graphical tool for creating and configuring instances as shown in Figure 4-4. It allows you to specify all the required configuration parameters such as the instance owner and communication protocol in an easy and guided fashion. The command can be found in `/opt/IBM/db2/V8.1/instance`.

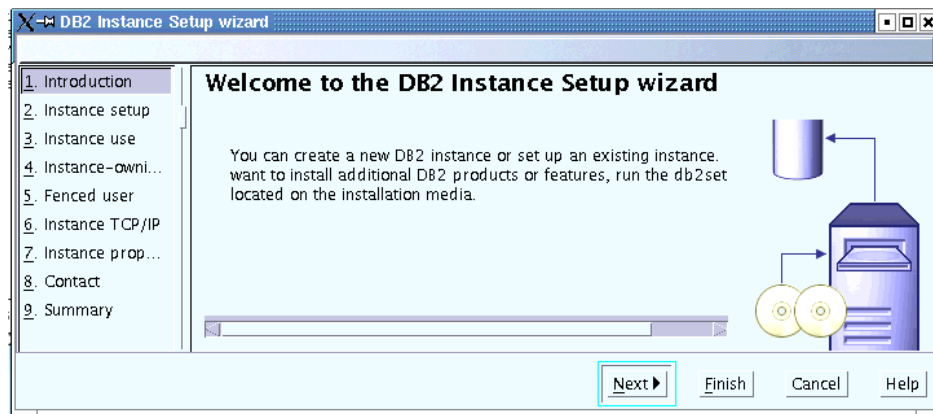


Figure 4-4 Graphical user interface for db2isetup

- ▶ The **db2icrt** command creates an instance, for example:
db2icrt u db2fenc1 db2inst1

As part of the instance creation, the installer creates three users identified as *db2inst1*, *db2fenc1*, and *dasadm1*. If you do not want to use the default user IDs, you can create the user IDs and groups ahead of time and use the IDs during creating the instance. The installer will also add the following entry to the */etc/services* file in order to allow communication from DB2 clients:

```
db2c_db2inst1 50000
```

Where *db2c_db2inst1* indicates the service name, and *50000* indicates the port number. Subsequent instances may be created on the same server simply by using one of the methods introduced above.

4.1.4 Client setup on Linux

This section discusses installation and configuration of DB2 UDB clients to access a remote DB2 UDB server. DB2 UDB provides three types of clients free of charge:

- ▶ The Run-Time Client
- ▶ The Administration Client
- ▶ The Application Development Client

All clients are supported on Linux, AIX, HP-UX, Solaris, and Windows operating systems.

Note: Typically, for a production environment, DB2 clients are installed on physically separate machines from the DB2 server machine. However, for an application development environment, it may be useful to have everything such as the DB2 database server plus the clients on the same machine.

To access a remote DB2 UDB database, you can either run the easy to use graphical tool *Configuration Assistant* or use the catalog commands to provide entries for the following three directories:

- ▶ **NODE** directory: A list of remote DB2 instances
- ▶ **ADMIN NODE** directory: A list of remote DB2 Administration servers
- ▶ **DATABASE** directory: A list of databases

To access a remote DB2 UDB database, you first catalog the DB2 UDB node, which is the server where the database resides, then catalog the database. See Example 4-2.

Example 4-2 Cataloging node and database

```
--  
-- catalog database node  
--
```



```
CATALOG TCPIP NODE lochness REMOTE lochness SERVER 50000 WITH 'Redbook system'  
--  
-- catalog database  
--  
  
CATALOG DATABASE itsodb AS itsodb AT NODE lochness WITH 'Redbook sample  
database'
```

After installing your DB2 Administration Client, you should configure it to access a remote DB2 server using the Configuration Assistant. Start the graphical interface through the DB2 Control Center or by using the command `db2cc`. For more detailed information, please refer to the IBM DB2 UDB manual *Quick Beginnings for DB2 Clients V8*, GC09-4832.

4.2 Other software product

Environment setup includes all the software used in the target system. In the migration planning, all the required software has been identified. All the identified software should be installed and configured.

The sample application used in this book is written in PHP. Therefore, we prepare the target system for the PHP application.

4.2.1 PHP adjustment for Unified ODBC with DB2 support

When migrating a PHP application from MySQL to DB2 on Linux, some preparations and verifications have to be accomplished first for unified ODBC with DB2 support. If a precompiled package of PHP is used, by default only a predefined set of functions and database support is integrated. In order to use the IBM DB2 libraries, PHP has to be recompiled using the `--with-ibm-db2` configuration option.

Note: All commands and procedure descriptions provided in this section refers to SuSE Linux 8.1. This can vary for other versions or Linux distributions.

Adjustment steps:

1. Backup `php.ini` and `httpd.conf`:

To ensure the configuration files of PHP and Apache are not lost when installing the new PHP version, we recommend backing up the `/etc/php.ini` and `/etc/httpd/httpd.conf` files.

2. Downloading PHP package:

The source code for PHP is available at: <http://www.php.net/downloads.php>

In our migration scenario we used Version 4.3.4 of PHP, and the package we downloaded was php-4.3.4.tar.gz.

3. Decompressing source package:

The following command decompresses the contents of the sources package in to a directory called php-4.3.4:

```
bash$ tar xzf php-4.3.4
```

4. Changing the working directory:

Use the **cd** command to make the newly created directory your working directory.

```
bash$ cd php-4.3.4
```

5. Specifying the configuration options for the PHP source

A list of the possible configuration options can be seen by issuing the following command:

```
bash$ configure --help
```

To see which configuration options are included in the previous installed PHP version, the PHP function `phpinfo()` can be invoked out of a Web page file. In the third section of the PHP information, the included configuration commands are listed as shown in Figure 4-5. To include all these commands, copy and paste them into the configure command and add the **--with-ibm-db2** option. Also include the **--with-apxs** option if not present yet.

System	Linux sylhester.devel.redhat.com 2.4.20-2.41smp #1 SMP Sun Feb 9 09:29:47 EST 2003 i686 athlon i386 GNU/Linux
Build Date	Feb 25 2003 09:44:31
Configure Command	./configure '--host=i386-redhat-linux' '--build=i386-redhat-linux' '--target=i386-redhat-linux-gnu' '--program-prefix=' '--prefix=/usr' '--exec-prefix=/usr' '--bindir=/usr/bin' '--sbindir=/usr/sbin' '--sysconfdir=/etc' '--datadir=/usr/share' '--includedir=/usr/include' '--libdir=/usr/lib' '--libexecdir=/usr/libexec' '--localstatedir=/var' '--sharedstatedir=/usr/com' '--mandir=/usr/share/man' '--infodir=/usr/share/info' '--cache-file= ./config.cache' '--with-config-file-path=/etc' '--with-config-file-scan-dir=/etc/php.d' '--enable-force-cgi-redirect' '--disable-debug' '--enable-pic' '--disable-rpath' '--enable-inline-optimization' '--with-bz2' '--with-db3' '--with-curl' '--with-dom=/usr' '--with-exec-dir=/usr/bin' '--with-freetype-dir=/usr' '--with-png-dir=/usr' '--with-gd' '--enable-gd-native-ttf' '--with-ttf' '--with-gdbm' '--with-gettext' '--with-ncurses' '--with-gmp' '--with-iconv' '--with-jpeg-dir=/usr' '--with-openssl' '--with-png' '--with-pspell' '--with-regex=system' '--with-xml' '--with-expat-dir=/usr' '--with-zlib' '--with-layout=GNU' '--enable-bcmath' '--enable-exif' '--enable-ftp' '--enable-magic-quotes' '--enable-safe-mode' '--enable-sockets' '--enable-sysvsem' '--enable-sysvshm' '--enable-discard-path' '--enable-track-vars' '--enable-trans-sid' '--enable-yp' '--enable-wddx' '--without-oci8' '--with-pear=/usr/share/pear' '--with-ldap=shared' '--with-imap=shared' '--with-kerberos=/usr/kerberos' '--with-ldap=shared' '--with-mysql=shared' '--with-pgsql=shared' '--with-snmp=shared' '--with-snmp=shared' '--enable-ucd-snmp-hack' '--with-unixODBC=shared' '--enable-memory-limit' '--enable-bcmath' '--enable-shmop' '--enable-versioning' '--enable-calendar' '--enable-dbx' '--enable-dio' '--enable-mcal' '--with-apxs2=/usr/sbin/apxs'
Server API	Apache 2.0 Filter
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php.ini

Figure 4-5 PHP Configuration options

For our purposes we specified only four options for the configure command:

```
bash$ configure --with-ibm-db2=/home/db2inst1/sqllib \
--with-mysql \
--with-apxs=/usr/sbin/apxs \
--with-config-file-path=/etc
```

where the `--with-ibm-db2` option specifies the location of the DB2 Application Development Client¹ instance on the machine, and the `--with-apxs` option specifies the Apache's apxs tool, which allows you to build extension modules to add to Apache's functionality.

¹ The Application Development Client or support can be installed as an option with the DB2 Server. For instructions see Section 4.1.2, "Installation procedure" on page 78.

Note: If Apache's apxs tool is not installed yet, the tool has to be installed before compiling PHP. In SuSE Linux, this package is called *apache-devel*.

6. Compiling PHP

After configuring the source files, the compile process is started using the **make** command. We piped the output to a log file in order to check for failure afterwards:

```
bash$ make > make.out
```

7. Installing PHP

Once PHP has compiled successfully, it can be installed as the root user:

```
bash$ su root
Password>
bash$ make install
```

8. Configuring Apache

As Apache already was configured for the PHP used in our case, only some changes have to be made. As part of the installation process PHP automatically modifies Apache's configuration file `/etc/httpd/httpd.conf` by inserting two lines. It has to be checked if the installer placed this two lines in the right section, and if the provided path to the modules matches the path definition of the other modules in this section (see Example 4-3).

Example 4-3 Correct section for the `libphp4.so` and `mod_php4.c`

```
# Dynamic Shared Object (DSO) Support
# Example:
# LoadModule foo_module libexec/mod_foo.so
loadModule ...
.
.
loadModule php4_module          /usr/lib/apache/libphp4.so
.
.
# Reconstruction of the complete module list from all available modules
# (static and shared ones) to achieve correct module execution order.
# [WHENEVER YOU CHANGE THE LOADMODULE SECTION ABOVE UPDATE THIS, TOO]
ClearModuleList
.
.
AddModule mod_php4.c
```

9. Checking for `php.ini`.

Either copy `php.ini-dist` in the Configuration File Path (see Figure 4-5) of PHP and rename it to `php.ini`, or if your PHP version is the same as before copy the version that was saved in step 1 back in this directory.

10. Starting Apache httpd server:

Apache has to be restarted to use the configuration changes:

```
bash$ /etc/init.d/apache restart
```

4.3 MTK installation and usage

The IBM DB2 Migration Toolkit (MTK) for MySQL is an ingenious tool that accepts as input the arguments of the MySQL database, and the results in a new DB2 database containing the corresponding database objects.

For a free download of MTK for MYSQL log on to <http://www.ibm.com/software/data/db2/migration/mtk/>

4.3.1 MTK prerequisites

The following prerequisites are required to successfully deploy MTK for MySQL:

- ▶ Java JDK 1.4.0 or higher. Java JDK 1.4.0 is included in the *Mysql2db2.tar.GZ* installation image.
- ▶ MySQL 3.23.48-Max-log or higher: Ensure MySQL is running, usually you can start the daemon with the command `safe_mysqld &` from an account with root permissions.
- ▶ DB2 v7.2 or higher. Use the command `db2start` to make sure the DB2 Server is up and running.

Furthermore, MTK for MySQL requires the Korn shell to be installed on the system. In case Korn shell is not installed, run the following command `ln -sf ksh bash` to link the Korn shell to the Bourne shell.

For the purpose of this document we have used MTK for MySQL with DB2 UDB Version 8.1 with FixPak 4 and MySQL Version 4.0.17. Although it did not make a difference for our project, we recommend to install and run the MTK for MySQL on the source machine.

4.3.2 MTK installation

Installing MTK is simple and straight forward:

- ▶ Copy the *Mysql2db2.tar.GZ* installation image into a newly created directory.

- Expand the compressed package using:

```
tar zxvf Mysq12db2.tar.GZ
```

The package includes a jar file for *Mysq12db2* Java classes, a shell script to start the program, as well as the IBM JRE 1.4.0 needed by the program.

To launch the MTK for MySQL execute command **Mysq12db2** to start the program's user interface (see Figure 4-6).

Mysq12db2 also accept the input and output database as command parameters like:

```
Mysq12db2 --indb=<MYSQL_DB_NAME> --outdb=<DB2_DB_NAME>
```

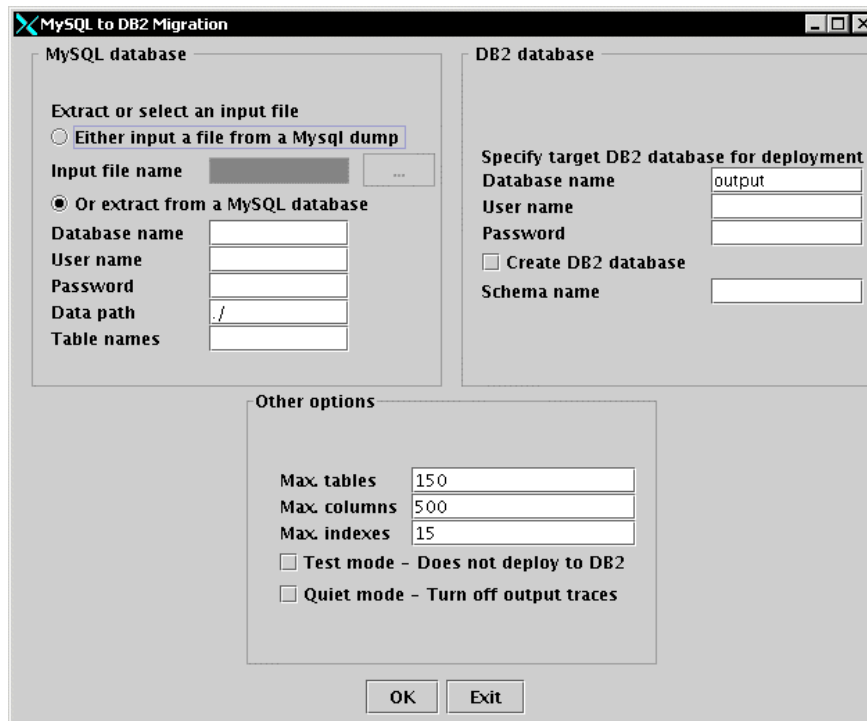


Figure 4-6 MTK initial screen



Database porting

After the migration planning, and software and hardware setup, the next main task is to migrate the source MySQL database structure onto DB2 UDB.

In this chapter we discuss process of porting the database structure from Mysql 4.0.1 server to DB2 UDB V8.1 server. Before doing this we need to look into the difference between the MySQL and DB2 UDB structure.

In the first section we discuss the data type mapping, followed by DDL differences. The DDL difference section provides a basic syntax difference between MySQL and DB2 UDB.

In the succeeding section we provide additional considerations required to keep in mind while porting the database schema from MySQL to DB2 UDB.

The last section provides detailed information of the database schema porting steps using three different approaches:

- ▶ Porting using MTK
- ▶ Manual porting
- ▶ Metadata transform

5.1 Data type mapping

This section compares MySQL and DB2 UDB data types. In general, DB2 UDB has data types equivalent to all the existing data types in MySQL.

Every column in database tables has a data type and this data type determines the value that column can contain in it. DB2 UDB has built-in data types and supports user defined data type (UDT) whereas MySQL has only built-in data types support.

Figure 5-1 shows built-in data type of MySQL.

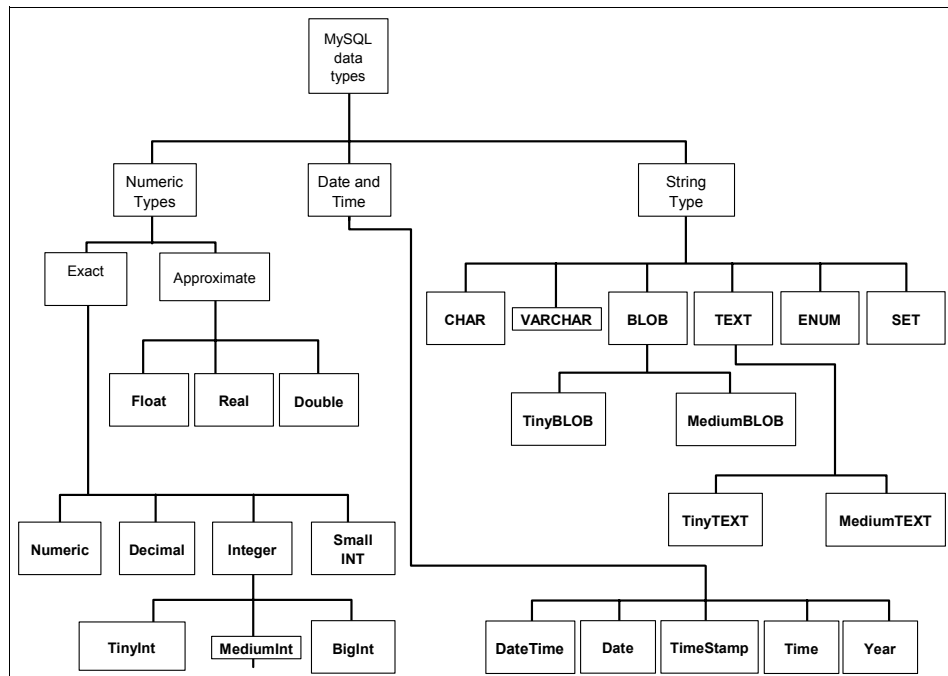


Figure 5-1 MySQL data types

Figure 5-2 shows built-in data types supported by DB2 UDB.

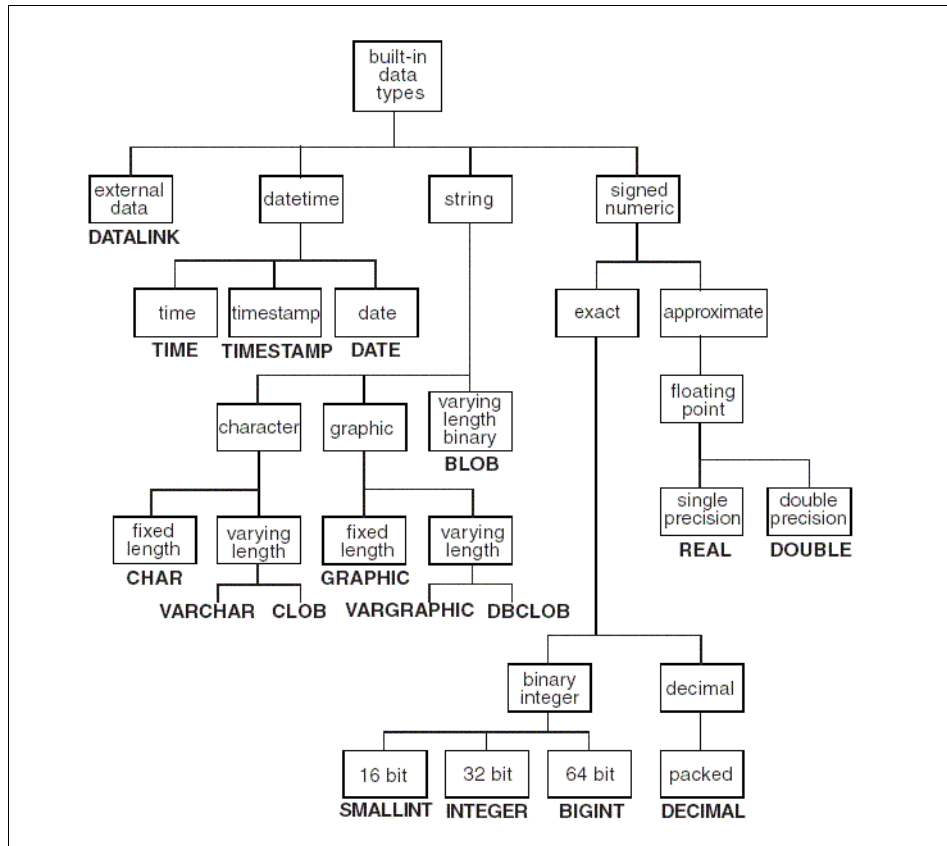


Figure 5-2 DB2 UDB data types

MySQL data types are grouped into three categories, and can be converted to DB2 UDB data types following the rules suggested below:

- ▶ Numeric type
 - TINYINT

This is a single byte integer, DB2 UDB does not have built-in single byte integer but SMALLINT is used for all the similar functionality.
 - SMALLINT

A small integer is a two-byte integer with a precision of five digits. This can be replaced by DB2 UDB SMALLINT.
 - BIT/BOOL/BOOLEAN

These are synonyms for TINYINT(1), In DB2 UDB we use SMALLINT with check constraint instead of BIT/BOOL/BOOLEAN.

- MEDIUMINT

This is a medium-size integer with a signed range of -8388608 to 8388607. In DB2 UDB we use an integer instead of this.
- INTEGER/INT

An integer is 4-byte integer for both MySQL and DB2 UDB.
- BIGINT

A big integer is an 8-byte integer for both MySQL and DB2 UDB.
- FLOAT

A float in MySQL is single precision floating point number ranging from -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. Whereas in DB2 UDB it is a double-precision floating-point number ranging from -1.79769E+308 to -2.225E-307, or from 2.225E-307 to 1.79769E+308. So, a float in MySQL can directly be mapped to real or float in DB2 UDB.
- DOUBLE

A double is a double precision floating point number for both DB2 UDB and MySQL.
- REAL

This is synonym for double in MySQL.
- DECIMAL/NUMERIC/FIXED

A decimal in MySQL is mapped to a decimal in DB2 UDB. Although MySQL and DB2 UDB implement decimals in a different way, externally they behave same.
- ▶ Data and time type
 - DATE

A date in MySQL is mapped to DB2 UDB date and both use 4 bytes (first two for year, third for month, and last for day). The range of the MySQL date is year 1000-9999, whereas DB2 UDB supports date from 0001-9999.
 - DATETIME

A date and time combination in MySQL is displayed as YYYY-MM-DD HH:MM:SS ranging from year 1000 to 9999. In DB2 UDB, the timestamp is used for a similar purpose, which is a seven part value (year, month, day, hour, minute, second, and microsecond).
 - TIMESTAMP

A timestamp in MySQL is date time combination with range of 1970-01-01 00:00:00 to the year 2037. It automatically set to the date and time of the most recent operation if you do not give it a valid value. This is same as timestamp in DB2 UDB except for its range.

- TIME

MySQL time represents a clock ranging from -838:59:59 to 838:59:59. It is mapped to DB2 UDB time, which is a 24 hour clock.

- YEAR

A year in two or four-digit format representing the year from 1901 to 2155. It is mapped to SMALLINT in DB2.

- ▶ String/character types

- CHAR

A fixed length string in MySQL is represented with the same name in DB2 UDB, It can be mapped to CHAR in DB2.

- VARCHAR

A variable-length string is also same as DB2 UDB VARCHAR for its range, which is 0-255 characters in MySQL and 0-32672 bytes in DB2.

- TINYBLOB

MySQL tinyblob is a binary large object column with a maximum length of 255. This is mapped to DB2 BLOB(255).

- TINYTEXT

MySQL tinytext is a character stream of maximum length 255. It is mapped to DB2 CLOB(255).

- BLOB

MySQL blob is a binary data column with a maximum length of 65535. This is mapped to DB2 BLOB(65K).

- TEXT

This is a text column with a maximum length of 65535 and is mapped to DB2 CLOB(65K).

- MEDIUMBLOB

MySQL mediumblob is a blob column with a maximum length of 16777215. This can be mapped to DB2 BLOB(16M).

- MEDIUMTEXT

MySQL mediumtext is a text column with a maximum length of 16777215. This can be mapped to DB2 CLOB(16M).

- LONGBLOB

MySQL longblob is very large blob column with a maximum length of 4294967295. It is generally mapped to DB2 BLOB(4G) not logged.

– LONGTEXT RAW

MySQL longtext is text column with a maximum length of 4294967295 is generally mapped to DB2 CLOB(4G) not logged.

– ENUM VARCHAR2

MySQL has a special type enumeration, which is a string object that can have only one value chosen from the list of values 'value1', 'value2',..., NULL. This is implemented in DB2 UDB using VARCHAR() with check constraints.

– SET VARCHAR2

Another MySQL special type set. It is a string object that can have zero or more values, which must be chosen from the list of values 'value1', 'value2',,... This is implemented in DB2 UDB using VARCHAR() with check constraints.

Table 5-1 shows the default data type mappings between two databases used by MTK and the sample application.

Table 5-1 Default data type mapping

MySQL 4.1	DB2 UDB 8.1
TINYINT	SMALLINT
SMALLINT	SMALLINT
MEDIUMINT	INTEGER
INT	INTEGER
INTEGER	INTEGER
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
REAL	REAL
DECIMAL	DECIMAL
NUMERIC	NUMERIC
DATE	DATE
DATETIME	TIMESTAMP

MySQL 4.1	DB2 UDB 8.1
TIMESTAMP	TIMESTAMP
TIME	TIME
YEAR	SMALLINT with check constraint
CHAR	CHAR
VARCHAR	VARCHAR
TINYBLOB	BLOB(255)
TINYTEXT	CLOB(255)
BLOB	BLOB(65K)
TEXT VARCHAR2,	CLOB(65K)
MEDIUMBLOB	BLOB(16M)
MEDIUMTEXT	CLOB(16M)
LOB	BLOB(2G) NOT LOGGED
LONGTEXT RAW,	CLOB(2G) NOT LOGGED
ENUM VARCHAR2,	VARCHAR() with check constraints
SET VARCHAR2,	VARCHAR() with check constraints

5.2 Data Definition Language differences

In this section we address the Data Definition Language (DDL) syntax difference between MySQL and DB2 DDL statements, and provide the DB2 UDB conversion. These differences can be syntactical, semantic, or functional.

Both MySQL and DB2 UDB follows structured query language (SQL), standardized language used to access databases, and is defined by the ANSI/ISO.

The data definition language is a subset of SQL, which serves to create or delete a database and its structure (tables, views, and indexes); to grant or revoke user privileges; and to define referential integrity rules.

5.2.1 Database manipulation

MySQL database objects are always stored in one directory. This gives advantage of simplicity but with a big performance bottleneck because of slow disk seek, slow search, and non-indexing of data. MySQL depends on the operating system's capabilities for distributing its data across disks. It uses symbolic links to different disks for different databases, and for database and table distributing.

On Linux machines MySQL also uses file system mounting options but most of the time MySQL uses symbolic link. This can be done by creating a directory where you have an extra space:

```
bash>cd <file system with space>
bash>mkdir mysqldata
```

then creates a symbolic link to the newly created directory using:

```
bash>cd /usr/local/mysql
bash>ln -sf <file system with space>/mysqldata data
```

Now you can create a database from MySQL mysql prompt using:

```
mysql>create database itsodb
```

MySQL users can distribute tables using symbolic linking or the data and index directory options of create table.

MySQL does not provide anything for logically dividing the database into different nodes or distributing tables onto different table spaces except for the optional InnoDB table, which supports multiple table space distributed in different files. Also, MySQL does not give any options for distributing tables into different segments according to usage or the user. We discuss this in 5.3, "Other considerations" on page 107.

DB2 UDB uses a better approach for the logical and physical distribution of the database and the database elements in different sectors. While migrating the database from MySQL to DB2 UDB, you can use these features for the performance enhancement of your application.

Instance

A DB2 UDB server can have more than one instance. One instance can have multiple databases. One instance per application database has the advantage that the application and database support do not have to coordinate with others for activities that need to take database or instance off-line. For migration purposes, you can create one instance for your database application environment. It can be created easily using the **db2icrt** command:

```
bash>db2icrt itso
```

Database partition group/node group

As the name implies it represents a container for set of partitions where the table space resides. This can be used if you want to spread your DB2 UDB database across multiple servers in a cluster or multiple nodes. There are no database partition group design considerations if you are using a non-partitioned database. The nodegroup can be created within a database using:

```
db2>create nodegroup itsospecial on all nodes
```

Database

A database represents your data as a collection of data in structured fashion. It includes a set of system catalog tables that describe the logical and the physical structure of the data, a configuration file containing environment parameter values used by the database, and a recovery log with ongoing and archivable transactions.

Creating a database

The database in DB2 UDB can be created simply by using:

```
db2>create database itsodb
```

This command initializes a new database with a default path, and table spaces. It creates three initial table spaces and the system tables, and allocates the recovery log.

You can use the create database statement with options to personalize the database as shown in Example 5-1.

Example 5-1 Create database

```
db2>create database itsodb
      catalog tablespace managed by system using
      ('/home/itsodb/database/catalog')
      extentsize 16 prefetchsize 16
      user tablespace managed by system using ('/home/itsodb/database/user4K')
      extentsize 16 prefetchsize 16
      temporary tablespace managed by system using
      ('/home/itsodb/database/temp')
```

Dropping a database

In MySQL you can drop the database using:

```
mysql>drop database [if exists] itsodb
```

This will remove all the database files (.BAK, .DAT, .HSH, .ISD,.ISM, .ISM, .MRG, .MYD,.MYI, .db, .frm) from your filesystem.

DB2 UDB has a similar command:

```
db2>drop database itsodb [at node]
```

This command deletes the database contents and all log files for the database, uncatalogs the database, and deletes the database subdirectory.

Alter database

MySQL **alter database** allows you to change the overall characteristics of a database. For example, the character set clause changes the database character set, and the collation clause changes the database collation. The basic syntax for altering the database is:

```
mysql>alter database itsodb CHARACTER SET charset_name COLLATE  
collation_name
```

In DB2 UDB you can use **UPDATE DATABASE CONFIGURATION** and **UPDATE DATABASE MANAGER CONFIGURATION** to set the database and database manager configuration parameters. Using this you can change a lot of configuration parameters like the log file size, log file path, heap size, cache size, etc.

```
db2> update database manager configuration using diaglevel 3  
db2> update database configuration for itsodb2 using logfilesiz 8
```

You can also change the physical and logical portioning by allocating the table space and paging for the database.

Table space

A table space is a storage structure containing tables, indexes, large objects, and long data. Table spaces reside in database partition groups. They allow you to assign the location of database and table data directly onto containers. DB2 UDB allows for two types of table spaces: System Managed Space (SMS), which is maintained by the system, and Data Managed Space (DMS), which is maintained by the DB2 UDB administrator.

The DB2 UDB database should have at least three table spaces:

- ▶ One catalog table space, which contains system catalog tables
- ▶ One or more user table space, which contains user defined tables
- ▶ One or more temporary table space, which contains temporary tables

It can be created using:

```
db2> create regular tablespace itsodb8k pagesize 8192 managed by system  
using ('/home/itso/database/user8K') extentsize 8 prefetchsize 8 bufferpool  
itsodb8k
```


Schema

A schema is an identifier such as a user ID that helps group tables and other database objects. A schema can be owned by an individual, and the owner can control access to the data and the objects within it. A schema is also an object in the database. It may be created automatically when the first object in a schema is created. We can create a schema using:

```
db2>create schema itsoschema authorization itso
```

5.2.2 Table manipulation

Tables are logical structures maintained by the database manager. Tables are made up of columns and rows.

MySQL tables

As shown in Figure 5-3, MySQL supports two types of tables: transaction-safe table and non transaction-safe table. Transaction-safe tables (InnoDB and BDB) are crash safe and can take part in transactions; they provide the concurrency feature and allow commit and rollback. On the other hand, non transaction-safe tables (HEAP, ISAM, MERGE, and MyISAM) are much faster, and consume less space and memory.

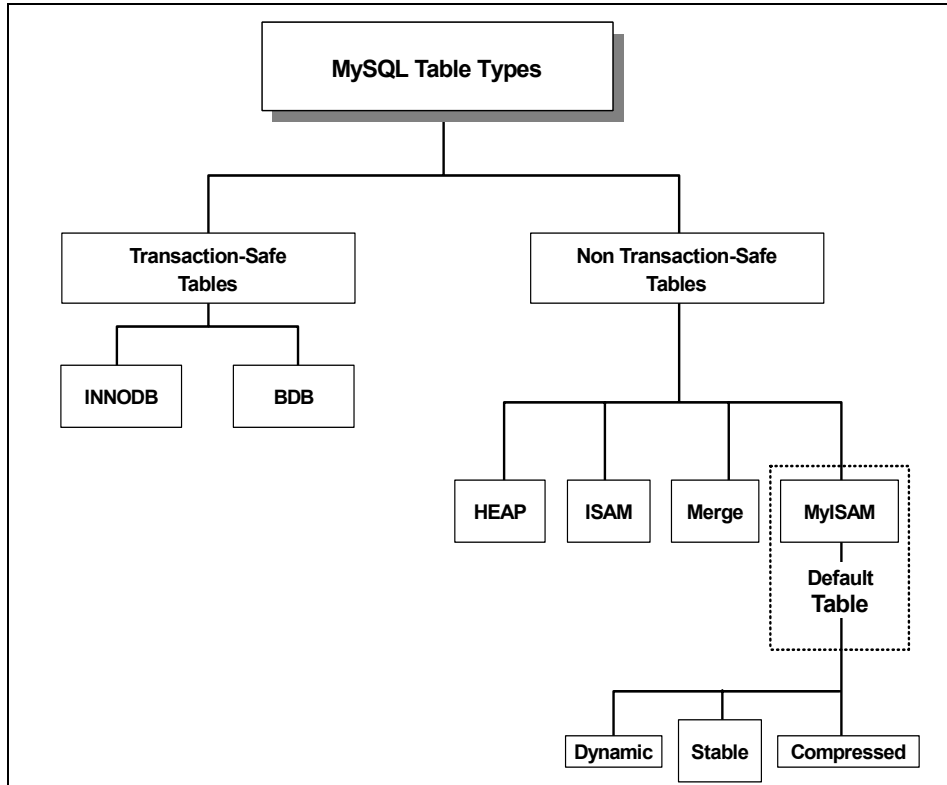


Figure 5-3 MySQL table types

The following tables are the basic storage elements in MySQL:

► InnoDB table

InnoDB is transaction safe storage engine with commit, rollback, and crash recovery capabilities. It provides locking on row level and also provides consistent non-locking read-in select statements for better concurrency and performance.

InnoDB is a complete database service placed under MySQL. It has its own buffer pool for caching data and indexes in main memory. It supports multiple table spaces. It can be created using the statement:

```
mysql>create table itsoinnodb (i int ,f float ) type=innodb;
```

► BDB table

Another type of transaction safe table in MySQL is BerkeleyDB (BDB). It supports page level locking, supports the commit and rollback features of the transaction, and is more stable while MySQL is in crash. It can be configured and created using the following commands:

```
bash>./configure --with-berkeley-db
mysql>create table itsobdb (i int ,f float ) type=bdb;
```

► **MyISAM table**

MyISAM is the default table type in MySQL. It is based on the ISAM code, and supports crash recovery, concurrent insert, big files on operating system/filesystem that support big files, better indexing, and index compression and table compression.

MyISAM files are smaller than ISAM files, but it needs more CPU time for inserting and accessing data from compressed tables and indexes.

MyISAM is a default MySQL table so it can be created either with specifying `type=myisam` or not specifying any type:

```
mysql>create table itsomyisam (i int ,f float );
```

► **ISAM table**

ISAM is an older implementation of the MySQL tables, and is quite similar to MyISAM, and it is deprecated now. It allows compressed and fixed length keys, fixed and dynamic record length, machine/OS format data and max key length of 256. It can be created with the same create table statement with `type=isam`:

```
bash>create table itsoisam (i int ,f float ) type=isam;
```

DB2 UDB conversion of MySQL table types

In DB2 UDB all the tables take part in the transaction. So MySQL InnoDB, BDB, MyISAM and ISAM tables can all be converted to a DB2 UDB regular table. A DB2 UDB regular table is a general purpose table.

A regular table is created with the CREATE TABLE statement and is used to hold persistent user data.

Create table syntax

In this section we give you a high level overview of the difference in the create table syntax of MySQL and DB2 UDB. MySQL create table statements are quite simple with few exceptions. Example 5-2 shows the creating of MySQL InnoDB. Figure 5-3 shows the DB2 conversion, no major change required.

Example 5-2 creating MySQL InnoDB table

```
mysql>create table itosinno(col1 int, col2 char(10)) type=InnoDB ;
```

Example 5-3 DB2 conversion of creating MySQL InnoDB table

```
db2>create table itsoinno(col1 int, col2 char(10))
```

Example 5-4 is a MySQL MyISAM table creation example. Example 5-5 is the DB2 conversion. The main changes include:

- ▶ Changes in the data type according to data type mapping
- ▶ Instead of `auto_increment`, *generated by default as identity* is used.

Example 5-4 Creating MySQL MyISAM table

```
mysql>create table itsotest1 (  
    wk_id int(11) unsigned NOT NULL auto_increment,  
    user_id int(11) unsigned default NULL,  
    cnt int(10) unsigned default NULL,  
    cat_id int(12) unsigned default NULL,  
    status varchar(10) default NULL,  
    PRIMARY KEY (wk_id)) type=MyISAM;
```

Example 5-5 DB2 Conversion of MySQL MyISAM table creation

```
db2>create table itsotest1(  
    wk_id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,  
    user_id INT default NULL,  
    cnt INT default NULL,  
    cat_id INT default NULL,  
    status VARCHAR(10) default NULL,  
    PRIMARY KEY (wk_id));
```

Alter table

Alter table is a statement to change one or more properties of a table. The syntax of alter table for MySQL and DB2 UDB is quite similar and is shown in Example 5-6.

Example 5-6 MySQL and DB2 UDB alter table example

```
mysql>alter table itsotest1 modify status varchar(20);
```

```
db2>alter table itsotest1 alter column status set data type varchar(20)
```

Alter table in DB2 UDB does not support the dropping of columns, but this can be easily achieved using the temporary table. Scripts for doing this are shown in Example 5-7.

Example 5-7 DB2 UDB dropping column using temporary table

```
db2>create temporary table temp as  
    (select col1, col2, col4 from itsotable1)  
    definition only not logged initially
```

```
db2>insert into temp select col1,col2, col3 from itsotable1
```

```
db2>drop table itsotable1

db2>create table itsotable1 as
      (select * from temp)
      definition only not logged initially

db2>insert into itsotable1 select * from temp

db2>drop table temp
```

Drop table

Tables can easily be deleted from database by issuing drop table statements as shown below.

For MySQL:

```
DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name [, tbl_name,...] [RESTRICT |
CASCADE]
```

For DB2 UDB:

```
DROP table tblname
```

Non-persistent storage tables

Apart from the storage engines discussed above MySQL and DB2 support, here are some temporary or intermediate table types.

► **MERGE table**

A merge table is a collection of identical MyISAM tables that can be used as one. You can only select, delete, and update from the collection of the tables, and dropping the merge table leads to drop the merge specification.

Tables can be merged only if they have identical columns, key information, and the packing. Example 5-8 shows the statements for merge table in MySQL.

Example 5-8 Usage of merge table in MySQL

```
mysql>create table itsotable1
      (col1 int not null auto_increment primary key, col2 char(20));

mysql>create table itsotable2(col1 int not null auto_increment primary key,
col2 char(20));

mysql>create table itsomerge
      (col1 int not null auto_increment, col2 char(20), key(col1));
      TYPE=MERGE UNION=(itsotable1,itsotable2) INSERT_METHOD=LAST;
```

```
mysql>insert into itsotable1 (col2) VALUES ("itso1"),("itso2");
```

```
mysql>insert into itsotable2 (col2) VALUES ("itso3"),("itso4");
```

DB2 UDB uses the updateable UNION ALL view to achieve the above feature. UNION ALL views are commonly used for logically combining different but semantically related tables. UNION ALL view is also used for unification of like tables for better performance, manageability, and integrating federated data sources:

UNION ALL Views are created using:

```
db2>create view itsounionview as select * from itsotable1 union all
select * from itsotable2
```

► HEAP table

A heap table is a hashed index and is always stored in memory. Heap tables are fast but not crash safe. Data is lost when MySQL crashes.

The MySQL internal HEAP tables use 100% dynamic hashing without overflow areas. There is no extra space needed for free lists. HEAP tables also do not have any problem with delete and inserts, which normally is common with hashed tables. It can be created using

```
mysql> create table itsoheap type=heap select * from itsotable1;
```

MySQL HEAP tables can be migrated to a temporary table, materialized query tables, or indexes depending upon your requirements.

– Temporary table

DB2 UDB temporary tables are tables used for storing data in non-persistent, in-memory, session-specific tables. Once the session is over the definition of this table is lost. When your application is using HEAP table in this fashion, temporary tables can be declared in your application by calling statements:

- Create user temporary table space if not existing using:

```
create user temporary tablespace discompose managed by system using
('usertemp1')
```

- Declare temporary table in the application:

```
declare global temporary table distemper like itsotable1 on commit
delete rows not logged in discompose
```

– Materialized query table

A materialized query table (MQT) is a table whose definition is based on the results of a query, and whose data is in the form of precomputed results. If the SQL compiler determines that a query will run more

efficiently against a materialized query table than the base table or tables, the query executes against the materialized query table:

```
create table mqtitso as (select * from itsotable1) data initially
deferred refresh deferred
```

In addition, DB2 UDB also supports tables for supporting clustering and query performance enhancement. These tables can also be used according to your requirements:

- ▶ Materialized query table (MQT)/summary table

MQT also known as summary table in previous can also be used to improve the query performance. We discuss MQT in more detail in 10.6, “Materialized query tables (MQT)” on page 328.

- ▶ Multidimensional clustering (MDC) tables

MDC table has a physical cluster on more than one key or dimension at the same time. The MDC table maintains its clustering over all dimensions automatically and continuously, thus eliminating the need to reorganize the table in order to restore the physical order of the data.

If you create multi-dimensional clustering (MDC) tables, the performance of many queries might improve because the optimizer can apply additional optimization strategies. Some advantages of MDC tables are quicker and less scanning because of dimension block, faster lookup, block level index ANDing and ORing, and faster retrieval.

It can be created using:

```
db2>create table itsomdc (col1 int, col2 int, col3 int, col4 char(10))
organize by dimensions(col1,col2,col3)
```

- ▶ Range-clustered tables (RCT)

RCT are implemented as sequential clusters of data that provide fast, direct access. It is a table layout scheme where each record in the table has a predetermined offset from the logical start of the table. Some advantages associated with RCT are direct and fast access, lesser maintenance, lesser logging, lesser locking, lesser buffer pool size, and lesser indexing. It can be created using:

```
db2> create table itsorct
(col1 int not null, col2 int not null, col3 char(10), col4 float)
organized by key sequence(col1 starting from 1 ending at 10000)
allow overflow
```

- ▶ Typed table/hierarchy table

Typed tables are tables that are defined with a user-defined structured type. With typed tables you can establish a hierarchical structure with a defined

relationship between those tables called a *table hierarchy*. The table hierarchy is made up of a single root table, supertables, and subtables.

It can be created as shown in Example 5-9.

Example 5-9 Usage of typed/hierarchy table

```
db2>create table itsotypedtable1 of itsotype(ref is 0id user generated)
```

```
db2>create table itsosubtable1 of itsotype2 under itsotypedtable1
      inherit select privileges
```

```
db2>create table itsosubtable2 of itsotype2 under itsotypedtable1
      inherit select privileges
```

► Views

Views are the named specification of a result table. This specification is a select statement that is run whenever the view is referenced in an SQL statement. It can be used just like a base table.

A simple view can be created by create statement as shown in Example 5-10.

Example 5-10 View example

```
db2>create table itsotable1(col1 int, col2 int,
                          col3 char(20),col4 float, col5 char(30))
```

```
db2>create table itsotable2(col6 int, col7 int, col8 char(20),
                          col9 float, col10 char(30))
```

```
db2>create view itsoview(col1,sum1,col4,col10) as
      select col1,col1+col6,col4,col10
      from itsotable1,itsotable2
      where col1<10 and col6>10
```

5.2.3 Index manipulation

An index is an object in the database system, which uses indexing techniques for faster retrieval of the data from tables. It is an ordered set of pointers to rows of a base table managed by the database manager directly.

MySQL supports both single column indexes and multi-column indexes. MySQL can have three types of indexes:

- Primary key
- Unique
- Index

DB2 UDB supports all the features supported by MySQL with same kind of terminologies. So, they map directly while migrating.

Create Index

The following is a MySQL create index statement:

```
CREATE [UNIQUE|FULLTEXT] INDEX index_name [index_type]
ON tbl_name (index_col_name,...)
index_col_name:
col_name [(length)] [ASC | DESC]
```

Creating an index in DB2 UDB is quite similar. It can be done using:

```
CREATE [unique] INDEX index-name on tablename (columnnames ASC|DESC)
SPECIFICATION ONLY INCLUDE(column-name)
CLUSTER/EXTEND USING index-extension-name (constant-expression)
PCTFREE 10/PCTFREE integer LEVEL PCTFREE integer MINPCTUSED integer
ALLOW/DISALLOW REVERSE SCANS
PAGE SPLIT SYMMETRIC/PAGE SPLIT HIGH/LOW
COLLECT STATISTICS DETAILED SAMPLED
```

Drop index

Drop index statement for MySQL and DB2 UDB:

```
mysql>DROP INDEX index_name ON tbl_name;
db2>DROP INDEX index_name
```

5.3 Other considerations

Up until now we discussed approaches for converting database elements, which use similar approaches in two products. In the subsequent sections we discuss conversion of the database objects, which does not map directly in MySQL and DB2 UDB. We also discuss server and database placement architecture.

Multiple servers

In some cases multiple MySQL servers are placed on the same machine. It may be for the reason of user management or testing, or differentiating applications and keeping them independent. MySQL provides an option to run multiple servers on same machine using several operating parameters.

There are several ways to configure new server; we used:

```
bash> /path/to/mysqld --socket=file_name --port=port_number
```

or you can use:

```
bash> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
```

```
bash> MYSQL_TCP_PORT=3307
bash> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
bash> mysql
```

DB2 UDB supports similar functionality using multiple database manager instances on the same machine; each database manager instance has its own configuration files, directories, and database. Figure 5-4 shows the typical scenario.

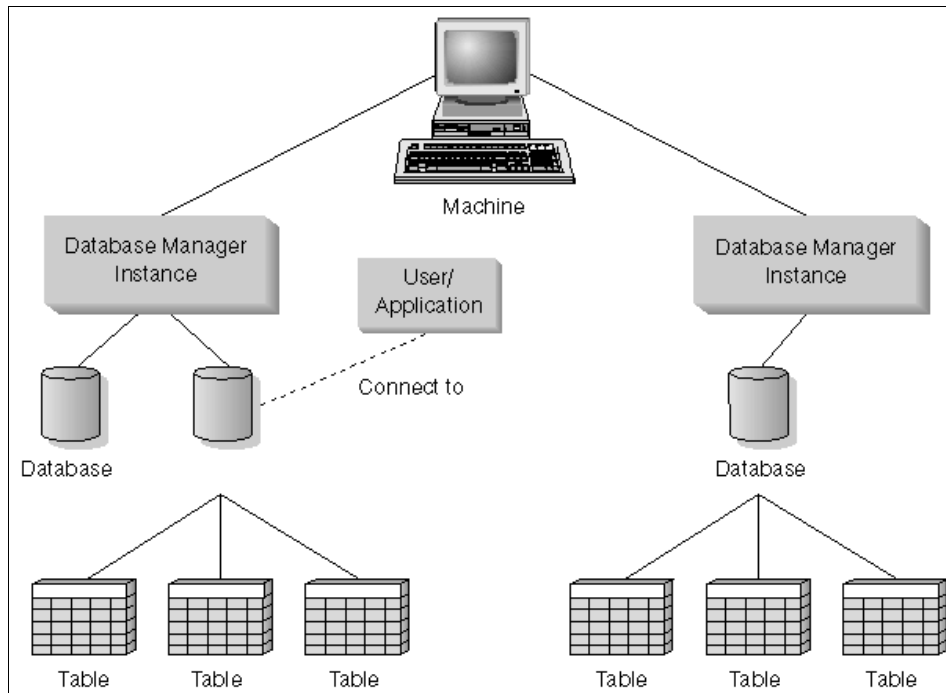


Figure 5-4 Typical DB2 UDB setup

DB2 UDB on all the UNIX systems support creation of different instances on the same machine with a different DB2 UDB version.

Multiple databases and schema conversion

In MySQL, tables within a database cannot be logically grouped by a schema or qualified name based on the application. To fulfill this requirement MySQL tables are placed in separate database. Application can then access these tables in the multiple databases.

MySQL supports accessing of the tables in multiple databases using the same connection. So an application connected to MySQL can use two tables in different databases in a single statement.

The queries in Example 5-11 shows how it works

Example 5-11 Multiple database example

```
mysql>create database itsodb_1;
mysql>create database itsodb_2;
mysql>connect itsodb_1;
mysql>create table itso1(col1 int, col2 char(10));
mysql>insert into itso1 values(1,"itso");
mysql>connect itsodb_2;
mysql>create table itso2(col1 int, col2 char(10));
mysql>insert into itso2 values(1,"itso1");
mysql>select * from itso2,itsodb_1.itso1;
```

This works in MySQL and give result

col1	col2	col1	col2
1	itso1	1	itso
1	itso1	2	itso

Figure 5-5 shows the architecture of multiple MySQL databases accessed by single application using same connection.

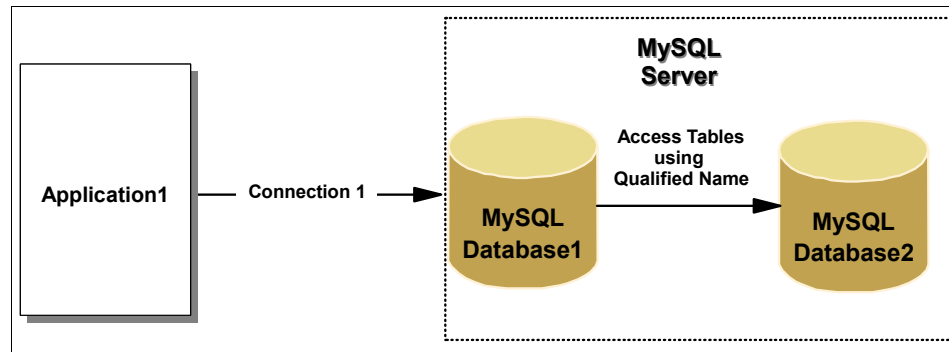


Figure 5-5 MySQL Application with multiple DBs instead of multiple schemas

DB2 UDB supports multiple database access with database links in a federated system. For a none federated system, DB2 UDB application uses more logical technique by using multiple schemas in place of multiple database used in same application. Each database can have multiple schemas and each table belongs to a particular schema.

When migrating the MySQL applications using multiple databases, all the databases used in the applications can be placed under one DB2 UDB database.

Each MySQL database is now represented by a DB2 UDB schema to hold the tables under that database. All the tables can be accessed by the application using single connection. The DBA only need to manage one database.

DB2 UDB schema can be created using

```
db2>create schema itsoschema1 authorization itsoschema1
```

The tables created in the particular schema can be accessed using the full table qualifier *table schema.table name*.

Table placement

MySQL does not support table space for managing physical location and page size or distributing tables onto the different table spaces except for optional InnoDB which supports multiple table space distributed in different files.

DB2 UDB supports table spaces to establish the relationship between the physical storage devices used by your database system and the logical containers used to store data. Table spaces reside in database partition groups. They allow you to assign the location of table data directly onto containers.

Before migration of database structure you should create a proper table space of different sizes in DB2 UDB, so that when you create tables they are placed into the table spaces which will give best performance.

List information

MySQL provides a show command to list the information about databases, tables, columns, or status information about the server.

DB2 UDB provide commands for getting the instances, database, table space, etc. information. The DB2 UDB system catalogues contain all the information about table, column, and index etc. You can use *describe* command to display table structure or use select statement to get the details of the table definition.

Table 5-2 shows examples of MySQL and corresponding DB2 UDB statements/commands to list database or table related information.

Table 5-2 MySQL to DB2 UDB conversion of list information statement

MySQL	DB2 UDB
show databases	ist database directory
show table	select tablename from syscat.tables
show columns from tablename	describe select * from tablename

MySQL	DB2 UDB
show index from tablename	select indname from syscat.indexes where tabname=tablename

Referential integrity

Referential integrity is the constraints defined on the table and its columns, which helps you to control the relationship of data in different tables. Essentially, it involves primary keys, foreign keys, and unique keys.

Primary keys and unique keys are treated similarly in MySQL and DB2 UDB whereas MySQL currently only parses foreign keys syntax in create table statement, but does not use/store the information about foreign keys except in InnoDB tables which support checking of foreign key constraints, including CASCADE, ON DELETE, and ON UPDATE.

DB2 UDB provide full support for foreign keys. With the full referential integrity functionality from DB2 UDB, your application can be released from the job of taking care of the data integrity issue. Example 5-12 shows creation and usage of foreign key constraint in DB2 UDB.

Example 5-12 Foreign key constraint usage

```
db2> alter table itsotable1 add constraint foreign1 foreign key (id)
      references itsotable2 on delete set null
```

We discuss more about foreign keys creation in Section 5.5, “Sample database migration” on page 117

5.4 Porting database

Now you have understand MySQL database structure/schema relative to different types of DB2 UDB objects, such as database, tables, views, indexes and referential integrity, in this section we give you details of the database structure/schema conversion from existing MySQL database to DB2 UDB.

Database schema conversion can be done in various ways but most common approaches are:

- ▶ Automatic conversion using porting tools
- ▶ Manual porting
- ▶ Metadata transport

In general all the above approaches take existing MySQL database as input and pass it through following functional engine:

- ▶ Capture the database schema information from MySQL.
- ▶ Modify schema information for DB2 UDB.
- ▶ Create the database in DB2 UDB with structure.

5.4.1 Automatic conversion using porting tools

Using porting tool is the most common and easiest of all the approaches. The IBM DB2 Migration Toolkit (MTK) simplify your migration to DB2 UDB. There are a number of third party migration tools available in market which can be used but IBM does not guarantee the correct functionality of these tools.

MTK for MySQL

IBM DB2 Migration Toolkit (MTK) for MySQL is a simple migration tool that takes the MySQL database to be migrated and the new DB2 UDB database to be created as input parameters. With MTK, you can automatically convert data types, tables, columns, views and indexes into equivalent DB2 UDB database elements. MTK provides database administrators (DBAs) and application programmers the functionality needed to automate the migration task. You can reduce the downtime, eliminate human error, and cut back on person hours and other resources associated with the database migration by using the following features found in the MTK:

- ▶ Extract database metadata from source DDL statements using direct source database access or imported SQL scripts.
- ▶ Automate the conversion of database object definitions.
- ▶ Generate and run data movement scripts
- ▶ Track the status of object conversions and data movement, including error messages, error location, and DDL change reports, using the migration log.

Download, installation and usage instructions for MTK are available in Chapter 4, Section 4.3, “MTK installation and usage” on page 87.

The database structure migration process using MTK is as follow:

- ▶ The MTK will first connect to MySQL database using the specified user and password (the current user if none is specified) and dump all the DDL and data using the `mysql dump` utility. It will store this DDL output in a file in the current directory with the name `<MYSQL_DATABASE_NAME>.mysql.create.<CURRENT_TIME>`. It stores MySQL load files for all the data in the tables in the current directory with the names `<TABLENAME>.txt`. `mysql:daemon` needs full write permission to the directory stores these files During migration, you can use `chmod 777 <dirname>` command to give the write access and then change it back later.

- ▶ The tool will then parse this mysqldump file to make it DB2 UDB compatible. This new file will also be stored in the current directory with the name `<DB2_DATABASE_NAME>.db2.create.<CURRENT_TIME>`
- ▶ The tool will then create the structure of the database in DB2 UDB by using this parsed file.
- ▶ A log of all DB2 UDB commands with associated output is stored in a file in the current directory by the name `<DB2_DATABASE_NAME>.db2.log.<CURRENT_TIME>`. *Examine* this log for any errors that might occur.
- ▶ When database conversion is finished, you can migrate your data, applications and other components which uses DB2 UDB. It is very important that you verify the migrated data after running the tool.

There are a few limitation in the current version of MTK

- ▶ MTK does not support the conversion of BLOB data type, so manual conversion is required for this.
- ▶ MTK may not work properly for a timestamp because of different date time formats, so manual conversion is required for this.
- ▶ MTK may not work properly for very complex databases. So extra care has to be taken while converting very complex databases.
- ▶ Data type mapping in MTK are fixed and cannot be changed, so manual change in script is required if you want to change the mapping.

Other migration tools

There are other migration tools which can be used to port MySQL database to DB2 UDB.

- ▶ SQLWays

SQLWays 3.6 is an advanced migration tool from Ispirer Systems Ltd. that ensures quick and easy transfer of database structures and data from MySQL to DB2 UDB. SQLWays runs only on Microsoft Windows 9x/NT/2000/XP
- ▶ SqlPorter

The SQLPorter Migration Toolkit is a tool from RealSoftStudio Inc. that simplifies the process of migrating data and databases across different databases on different platforms. The SQLPorter Migration Toolkit allows you to migrate an entire database (data and schema) in an integrated, visual environment. The SQLPorter Migration Toolkit employs an intuitive user interface and a series of wizards to simplify the migration process. All components of the Migration Workbench are written in C++. It is supported both on Windows and Linux system.

5.4.2 Manual porting

In the previous section we discuss migration using MTK. Sometime it is required to follow more personalized manual process to migrate database instead of the standard tool. For those few cases we describe manual conversion process, with a focus on converting MySQL objects and features which are not automatically converted by MTK.

In Section 5.1, “Data type mapping” on page 90 and Section 5.2, “Data Definition Language differences” on page 95, we demonstrate syntax and semantic differences between MySQL and DB2 UDB, which can be used for manual conversion. We already talked about creation, deletion, and altering of various database objects like database, tables, index, views etc., and how they are mapped while converting from MySQL and DB2 UDB. The manual process involves following steps:

- ▶ Capturing the database schema information from MySQL

MySQL offer a utility `mysqldump` that extract the database structure and deposit it into a text file. The structure is represented in DDL, and can be used to recreate the database elements in DB2 UDB server. Syntax for `mysqldump` is as shown below:

```
bash> mysqldump -u itso itsodb > mysqlobjects.ddl
```

If you are dumping very large database it is recommended to use `--quick` or `--opt` option. Without these options `mysqldump` loads whole result set into memory before dumping the result.

For further information on this utility refer MySQL Technical Reference.

- ▶ Modify schema information for DB2 UDB

Now that you have captured the source MySQL database structure using `mysqldump` utility, it is time to modify schema information and make it work for DB2 UDB. These manual changes can be

- DDL changes

First step in schema modification is the conversion of the create statement for various database objects like database, tables, views, indexes etc. Please refer Section 5.2, “Data Definition Language differences” on page 95 for these conversions. Also we need to write a DDL script for creating database and table space.

- Data type changes

Check the data type used in the table definition. Change the MySQL data type to DB2 UDB data type. Please refer Section 5.1, “Data type mapping” on page 90 for data type conversions.

- Reserved words conversion

There are many reserved words in MySQL and DB2 UDB which cannot be a valid name for the column and database element. Please refer MySQL and DB2 UDB Reference guide for more detail about reserved words and change the conflicting names in the DDL statements.

► Create database and database objects

Now that you have DDL statements modified, you are ready to create the DB2 UDB database and database objects. This can be done from command line processor (CLP).

It is a common practice to place the database and table space creation statements in one file and tables, views etc. creation statements in a separate file. The database and table spaces will be created first. Once the database and table spaces are in place, the tables, views etc., other objects will be created. The database creation scripts should be based on the logical and physical database design. In our example, we created database using the *create-database.sql* script as shown in Example 5-13.

Example 5-13 create-database.sql script

```
-- Create the Initial Database
create database itsodb
  catalog tablespace managed by system
  using ('/home/itsodb/database/catalog') extentsize 16 prefetchsize 16
  user tablespace managed by system
  using ('/home/itsodb/database/user4K') extentsize 16 prefetchsize 16
  temporary tablespace managed by system
  using ('/home/itsodb/database/temp')

connect to itsodb

-- Create a Bufferpool with 8K and 16K Pages
create bufferpool itsodb8k size 2000 pagesize 8192
create bufferpool itsodb16k size 1000 pagesize 16384

-- Make the Bufferpool Change Take Effect
disconnect itsodb
connect to itsodb

-- Create a Tablespace with 8K and 16K Pages
create regular tablespace itsodb8k pagesize 8192 managed by system
  using ('/home/itso/database/user8K')
  extentsize 8 prefetchsize 8 bufferpool itsodb8k

create regular tablespace itsodb8k pagesize 16384 managed by system
  using ('/home/itso/database/user16K')
  extentsize 8 prefetchsize 8 bufferpool itsodb16k
```

```
-- Create a System Temporary Tablespace with 8K and 16K Pages
create system temporary tablespace itsotemp8k pagesize 8 K managed by system
  using ('/home/itso/database/temp8K')
  extentsize 32 prefetchsize 16 bufferpool itsodb8K

create system temporary tablespace itsotemp16k pagesize 16 K managed by system
  using ('/home/itso/database/temp16K')
  extentsize 32 prefetchsize 16 bufferpool itsodb16K

disconnect itsodb
```

To create database, invoke the above SQL script from DB2 UDB Command Line Window or bash shell by using.

```
db2>@create-database.sql
bash>db2 -f create-database.sql
```

We then pick up database object creation statements from the output of mysqldump, *mysqlobjects.ddl* file and create *db2objects.ddl* script. Change the DDL statements and data types following section 5.1 and 5.2 of this chapter. You can create any additional statement required.

Now we execute the above created DDL scripts from DB2 Command Line Window or bash shell as shown in Example 5-14 and Example 5-15 below.

Example 5-14 create database objects from DB2 command line window

```
db2>connect to itsodb
db2>@db2objects.ddl
db2>disconnect itsodb
```

Example 5-15 create database objects from bash shell

```
bash>db2 connect to itsodb
bash>db2 -tf db2objects.ddl
bash>db2 disconnect itsodb
```

5.4.3 Metadata transport

In this section we discuss about using the database modelling tool for the migration of a database structure from MySQL to DB2 UDB. There are number of modelling tools existing in market which supports capturing of database model from MySQL database. In most of the tools a logical model defines the database design and physical model maps to the target database.

- ▶ IBM Rational® Rose® Professional Data Modeler Edition

The IBM Rational Rose Data Modeler Edition is a data design tool which integrates application design with database design and maps data model with object model. It allows database designers, business analysts and developers to work together through a common language. Use this tool for migration requires your MySQL model file to be available to converting to DB2 UDB physical model.

- ▶ CA AllFusion ERwin Data Modeler

ERWin is database modelling tool which support easy creation and maintenance of relational databases. It helps you create entity-relationship diagram from MySQL database, which can be converted to DB2 database structure. It can automatically generates tables, key-elements, and database design.

- ▶ Embarcadero Technologies ER/Studio

ER/Studio is a model-driven application for the logical and physical analysis, design, creation, and maintenance of database objects. It can reverse-engineer the complete schema from MySQL database.

Database structure conversion process using modeling tools

Database structure conversion using the modeling tool is a very neat technique for database conversion as lot of applications already have entity-relation diagram. The conversion can be done very easily by following the steps given below:

- ▶ Reverse-engineer database objects using a DDL script or an existing database using one of the modeling tool.
- ▶ Switch to a physical model.
- ▶ Select DB2 database as a target database and generate a DDL script for a new target.
- ▶ Create DB2 database structure using DDL scripts.

Limitation of this technique is that not everyone can afford costly modeling tools just for migration.

5.5 Sample database migration

This section demonstrate the database structure conversion for our sample BHMS Application from MySQL to DB2 UDB. BHMS is small real world Web application consisting of tables, views and indexes. Before starting the database conversion, please see the existing database structure in 3.1.2, “Database structure” on page 71.

For demonstration purposes we describe a step-by-step process using MTK and then describe manual method for additional objects, which are not converted using MTK:

► Starting MTK

<MTK Installation directory>/home/itsosj/mtk/Mysq12UDB

This will launch MTK Java application window as shown in Figure 5-6.

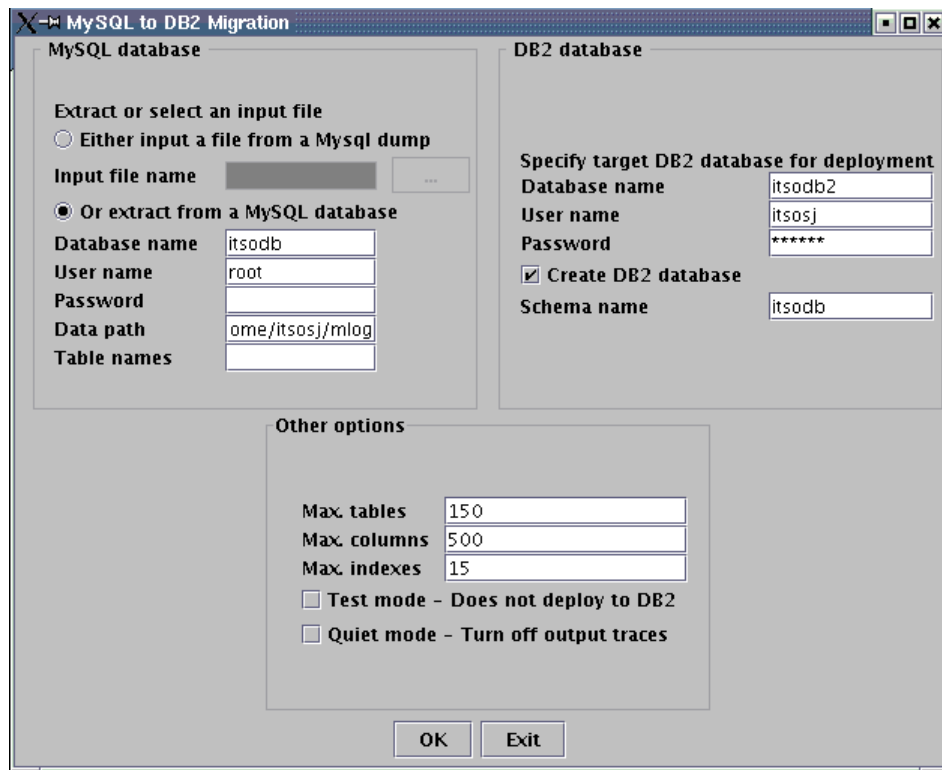


Figure 5-6 MTK startup window

- Enter Database name, User name, Password, Data path, and select the option **Test mode Does not deploy to DB2**.

With the **Test Mode Does not deploy to DB2** option selected, MTK will generate DB2 DDL in a file, and will not execute the DDL to create the DB2 database structure. This allows you to change the data type, any column names, or add any additional statements to add DB2 objects. Now click **OK**.

- The message box as shown in Figure 5-7 is displayed.

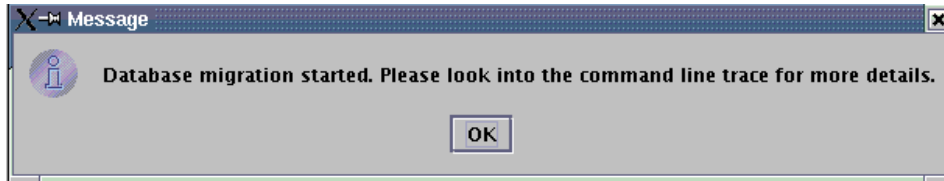


Figure 5-7 MTK Message window

- ▶ On completion of above task, you will see the successful migration window and output as shown in Example 5-16.

Example 5-16 MTK output console

```
mysql2db2:> Grabbing MySQLdump output... Done.
mysql2db2:> Parsing Dump Files...
mysql2db2:> * -> Parsed CREATE statement for Table: catalog
mysql2db2:> * -> Parsed CREATE statement for Table: oenummer
mysql2db2:> * -> Parsed CREATE statement for Table: users
mysql2db2:> * -> Parsed CREATE statement for Table: shopping_cart
mysql2db2:> * -> Parsed CREATE statement for Index: manufacturer
mysql2db2:> * -> Parsed CREATE statement for Index: sku
mysql2db2:> * -> Parsed CREATE statement for Index: name
mysql2db2:> * -> Parsed CREATE statement for Index: sc_id
mysql2db2:> Database Test Migration Successful.
```

- ▶ In Test mode option, MTK will generate a number of files:
 - `<dbname>.mysql.create.<timestamp>`
 - `output.mysql.create.<timestamp>`
 - data file corresponding to each table
- ▶ This allows you to examine the DDL generated by MTK. The DDL can be run in later time to create DB2 database and object.
- ▶ If you want MTK to create DB2 objects and move the data for you, select **Quiet Mode - turn off output trace** after lunch to MTK (see Figure 5-6). MTK will automatically create the DB2 database and load data into it. The final window as shown in Figure 5-8 will mark the completion.

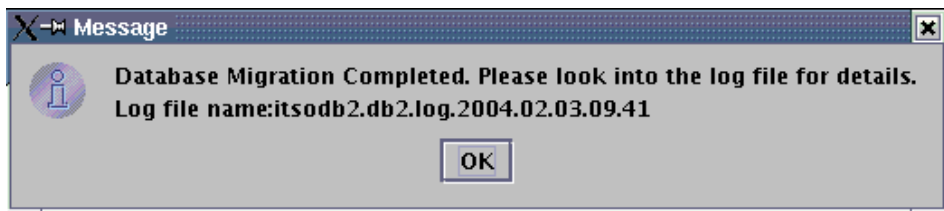


Figure 5-8 MTK success message

In this case you will get an additional line in the command window stating:

```
mq12db2:> Database Migration Completed. Please look into the log file for details.
```

```
Log file name:itsodb2.db2.log.2004.02.03.09.41
```

Now you need to go and check the `itsodb2.db2.log.2004.02.03.09.41` file for correctness of data and a table.

Modify generated DDL

In this section we concentrate on *mysql.create* and *output.mysql.create* files generated by MTK under Test Mode. The first file consists of create statements for MySQL, and the second file contains those statements converted to DB2 UDB.

Now, we have database create statements, the default table create, and index create statements, we will have a closer look at these create statements and change them according to our requirement. In our case we got MySQL statements in *itsodb.mysql.create.2004.01.06.01.38*, which is shown in Example 5-17.

Example 5-17 input database

```
--  
-- Table structure for table `catalog`  
--  
  
CREATE TABLE catalog (  
  manufacturer char(14) default '0',  
  model char(25) default NULL,  
  type char(25) default NULL,  
  year_from char(7) default NULL,  
  year_to char(7) default NULL,  
  kw char(3) default NULL,  
  hp char(3) default NULL,  
  ccm char(4) default NULL,  
  cyl_capacity char(2) default NULL,  
  motor_code char(60) default NULL,  
  prodgroup char(2) default NULL,  
  sku char(8) default NULL,  
  dimension char(12) default NULL,  
  ac char(3) default NULL,  
  transmission char(3) default NULL,  
  hint char(60) default NULL,  
  stock int(10) unsigned default NULL,  
  id int(12) unsigned NOT NULL auto_increment,  
  PRIMARY KEY (id),  
  KEY manufacturer (manufacturer,model)  
) TYPE=MyISAM;
```

```

--
-- Table structure for table `oenummer`
--

CREATE TABLE oenummer (
  id int(11) NOT NULL auto_increment,
  sku char(8) default '0',
  oe_no char(15) default NULL,
  PRIMARY KEY (id),
  KEY artnr (sku)
) TYPE=MyISAM;
--
-- Table structure for table `users`
--

CREATE TABLE users (
  id int(11) NOT NULL auto_increment,
  firstname varchar(30) default NULL,
  lastname varchar(30) NOT NULL default '',
  email varchar(100) NOT NULL default '',
  company varchar(50) default NULL,
  street varchar(40) default NULL,
  zip varchar(12) default NULL,
  city varchar(40) default NULL,
  status int(1) NOT NULL default '1',
  solutation varchar(12) default NULL,
  loginname varchar(10) default NULL,
  password varchar(10) default NULL,
  telephon varchar(30) default NULL,
  fax varchar(40) default NULL,
  customer_no varchar(12) default NULL,
  PRIMARY KEY (id),
  KEY name (lastname,firstname)
) TYPE=MyISAM;
--
-- Table structure for table `shopping_cart`
--

CREATE TABLE shopping_cart (
  sc_id int(11) unsigned NOT NULL auto_increment,
  user_id int(11) unsigned default NULL,
  cnt int(10) unsigned default NULL,
  cat_id int(12) unsigned default NULL,
  status varchar(10) default NULL,
  PRIMARY KEY (sc_id),
  UNIQUE KEY sc_id (sc_id)
)

```

```
) TYPE=MyISAM;
```

This is converted to *output.db2.create.2004.01.06.01.38*, which in the sample application case is as following (Example 5-18).

Example 5-18 Output DB2 statements

```
--
-- Table structure for table `catalog`
--
CREATE TABLE catalog (
  manufacturer CHAR(14) default '0',
  model CHAR(25) default NULL,
  type CHAR(25) default NULL,
  year_from CHAR(7) default NULL,
  year_to CHAR(7) default NULL,
  kw CHAR(3) default NULL,
  hp CHAR(3) default NULL,
  ccm CHAR(4) default NULL,
  cyl_capacity CHAR(2) default NULL,
  motor_code CHAR(60) default NULL,
  prodgroup CHAR(2) default NULL,
  sku CHAR(8) default NULL,
  dimension CHAR(12) default NULL,
  ac CHAR(3) default NULL,
  transmission CHAR(3) default NULL,
  hint CHAR(60) default NULL,
  stock INT default NULL,
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  PRIMARY KEY (id)
);

--
-- Table structure for table `oenummer`
--
CREATE TABLE oenummer (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  sku CHAR(8) default '0',
  oe_no CHAR(15) default NULL,
  PRIMARY KEY (id)
);

--
-- Table structure for table `users`
--
CREATE TABLE users (
  id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  firstname VARCHAR(30) default NULL,
  lastname VARCHAR(30) NOT NULL default '',
```



```

email VARCHAR(100) NOT NULL default '',
company VARCHAR(50) default NULL,
street VARCHAR(40) default NULL,
zip VARCHAR(12) default NULL,
city VARCHAR(40) default NULL,
status INT NOT NULL default 1,
solutation VARCHAR(12) default NULL,
loginname VARCHAR(10) default NULL,
password VARCHAR(10) default NULL,
telephone VARCHAR(30) default NULL,
fax VARCHAR(40) default NULL,
customer_no VARCHAR(12) default NULL,
PRIMARY KEY (id)
);

--
-- Table structure for table `shopping_cart`
--
CREATE TABLE shopping_cart (
  sc_id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  user_id INT default NULL,
  cnt INT default NULL,
  cat_id INT default NULL,
  status VARCHAR(10) default NULL,
  PRIMARY KEY (sc_id)
);

```

These scripts are created considering that we will go for default schema mapping from MySQL to DB2 UDB as discussed in 5.1, “Data type mapping” on page 90. This is according to the MTK recommended mapping, but you may decide to define some different mapping. If your application requires any mapping changes, this is a time to do it. For example, in our case we want the DB2 table SHOPPING_CART to have different columns names instead of above defined, additional column for the address and change in status column size.

We can change them to the one as shown in Example 5-19.

Example 5-19 Change in the default scripts

```

CREATE TABLE shopping_cart (
  sc_id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
  user_id INT default NULL,
  quantity INT default NULL,
  catalog_id INT default NULL,
  status VARCHAR(20) default NULL,
  address VARCHAR(100) default NULL,
  primary key (sc_id)
);

```

)

If you have any timestamp and a BLOB column, special attention is required for them. This is discussed in Chapter 6, “Data porting” on page 127.

Note: You should not reduce size of any field because it may cause an error while porting data.

Enhancement

This also is the time to add any enhancement to your database using the DB2 features MySQL does not have such as referential integrity. It is very essential for the database as it ensures the consistency of data values between related columns in different tables. RI is usually maintained by using the primary key, unique key, and foreign keys. Primary and unique keys are successfully migrated using MTK. As MySQL does not support foreign keys except for table type InnoDB, if you want to use foreign keys in your database you need to add them manually.

In our sample application, based on application requirements we created referential integrity between tables in the *referential.ddl* file, which looks like Example 5-20.

Example 5-20 Added foreign keys

```
alter table itsodb.shopping_cart add constraint foreign1 foreign key (user_id)
  references itsodb.users on delete set null

alter table itsodb.catalog add constraint foreign2 foreign key (id)
  references itsodb.shopping_cart on delete set null

alter table itsodb.catalog add constraint foreign1 foreign key (sku)
  references itsodb.oenummer on delete set null
```

Now we have completed the DDL modification; we executed the above changed scripts to create the DB2 database and the objects as shown in Example 5-21.

Example 5-21 creation of tables and database in DB2 UDB

```
bash>db2 create database itsodb2
The CREATE DATABASE command completed successfully
bash>db2 connect to itsodb2
```

Database Connection Information

Database server = DB2/LINUX 8.1.4

```
SQL authorization ID = ITS0SJ
Local database alias = ITS0DB2
bash>db2 -tf "itsodb2.db2.create.2004.02.03.09.41"
DB20000I The SQL command completed successfully.
DB20000I The SQL command completed successfully.
DB20000I The SQL command completed successfully.
DB20000I The SQL command completed successfully.
```

Incase you are migrating InnoDB tables having foreign keys you need to migrate them manually as MTK does not support migration of foreign keys. So after running MTK in test mode you need to change the create table statement in *output.mysql.create.<timestamp>* and remove CONSTRAINT `constraint name`. You will create the table without the foreign key. After migrating your data you then create a foreign key on this table.

Example 5-22 shows two MySQL InnoDB tables with the referential integrity defined.

Example 5-22 MySQL InnoDB using foriegn key

```
mysql> create table innoParent(id int not null,name char(20), primary key(id)
      type=INNODB;

mysql> create table innoChild(cid int not null,name char(20), pid int not null,
      primary key(cid),index(pid),foreign key(pid)
      references innoParent(id)) type=INNODB;
```

Example 5-23 is the conversion to DB2.

Example 5-23 DB2 conversion of innoDB foreign key

```
db2>create table innochild
      (cid int not null default 0, name char(20) default null,
      pid int not null default 0, primary key(cid))

db2>create table innoparent
      (id int not null default 0, name char(20) default null,
      primary key(id));

db2>alter table innochild add constraint foreign1 foreign key (pid)
      references innoParent
```

Note: If you are adding foriegn key constaints before migrating data, make sure that you load the parent table data before the child table.



Data porting

This chapter informs you about the considerations concerning data porting from a MySQL system to a DB2 UDB system.

The usage of tools and commands supporting data dumping from MySQL and loading data into DB2 UDB is described, and what you should be aware of when using the tools.

This chapter also discusses the differences in specific data formats and data types and how you can convert them from MySQL to DB2.

We also describe how the user account management (user data, access rights, and privileges) is implemented in MySQL, and how you can port it to database access information within DB2 security.

Finally, the steps for how we did the data porting in our sample project are discussed in detail.

6.1 Considerations concerning data porting

Data porting describes the steps that are necessary to get data from one database to another one. In general, you have to unload (also referred to as *dump* or *export*) the data from the source database into one or more files, and load (also referred to as *import*) the files into the target database.

Database systems provide commands and tools for unloading and loading data. In MySQL the *mysqldump* tool is for unloading a database. DB2 UDB provides the **LOAD** and **IMPORT** commands for loading data from files into the database.

You have to be aware of differences in how specific data types are represented by different database systems. For example, the representation of date and time values may be different in different databases, and it often depends on the local settings of the system.

If the source and the target database use different formats, you need to convert the data either automatically by tools or manually. Otherwise, the loading tool cannot understand the data to load due to the wrong format.

The migration of binary data stored in BLOBs should be done manually because binary data cannot be exported to files in a text format.

Porting the user account management is a very specific step in a migration project. You need to get the information about users, access rights, and privileges out of MySQL, convert it to DB2 specific security information, and create the users in DB2 UDB according to the source data. Migrating encrypted passwords is impossible in most cases.

6.1.1 Commands and tools supporting data porting

MySQL and DB2 UDB provide tools that support data porting between these two systems. In MySQL the **mysqldump** utility is used to retrieve data from the database; the **LOAD** and **IMPORT** commands can be used to get this data into DB2 UDB.

MTK automates the use of the **mysqldump** utility and the DB2 **LOAD** command.

The **mysqldump** tool

When porting application data, this tool can be used for retrieving the data from MySQL tables. It is shipped with MySQL and is usually located in the *bin* directory of the MySQL installation.

The basic usage of the **mysqldump** tool is as follows:

mysqldump [OPTIONS] **database** [tables]

For a complete description of this tool, run **mysqldump --help**. The most important command line options are:

- ▶ **--no-data**
No data information is extracted from the database, just the SQL statements for creating the tables and indexes. This option is used for extracting DDL statements only.
- ▶ **--no-create-info**
No SQL statements for creating the exported table is extracted from the database. So, this option should be used for exporting data only. The output file containing the data can be loaded into a DB2 table.
- ▶ **--tab=<outFilePath>**
This option creates a text file with the DDL (<tablename>.sql) and a tabulator separated text file with the data (<tablename>.txt) in the given path for each specified table. This option only works when the utility is run on the same machine as the MySQL daemon. If this option is not specified, INSERT statements for each row are created.

Example 6-1 shows the usage and output of the **mysqldump** command without the **--no-create-info** and the **--tab** options. The output has DDL statements for table creation, and INSERT statements to insert data into the table.

Example 6-1 Usage of mysqldump without the --no-create-info and the --tab option

```
bash>mysqldump --user root itsodb testtable
-- MySQL dump 9.10
--
-- Host: localhost    Database: itsodb
-- -----
-- Server version      4.0.17-standard
--
--
-- Table structure for table `testtable`
--
CREATE TABLE testtable (
  name char(10) default NULL,
  birthday date default NULL
) TYPE=MyISAM;
--
-- Dumping data for table `testtable`
--
INSERT INTO testtable VALUES ('Klaus','1978-07-03');
INSERT INTO testtable VALUES ('Michael','1974-03-20');
INSERT INTO testtable VALUES ('Rakesh','1977-09-26');
```

Example 6-2 shows the usage and output of the **mysqldump** command with the **--no-create-info** but without the **--tab** options. The output has only INSERT statements for each row to insert data into the table.

Example 6-2 Usage of mysqldump with the --no-create-info but without the --tab option

```
bash>mysqldump --no-create-info --user root itsodb testtable
-- MySQL dump 9.10
-- ...
--
-- Dumping data for table `testtable`
--
INSERT INTO testtable VALUES ('Klaus','1978-07-03');
INSERT INTO testtable VALUES ('Michael','1974-03-20');
INSERT INTO testtable VALUES ('Rakesh','1977-09-26');
```

Example 6-3 shows the usage and output of the **mysqldump** command with the **--no-create-info** and the **--tab** options. The output is a file with the exported MySQL data. This file can be read by the DB2 **LOAD** command.

Example 6-3 Usage of mysqldump with the --no-create-info and the --tab option

```
bash>mysqldump --no-create-info --tab=. --user root itsodb testtable
```

```
bash>cat testtable.txt
```

```
Klaus 1978-07-03
```

```
Michael 1974-03-20
```

```
Rakesh 1977-09-26
```

Example 6-4 shows the usage and output of the `mysqldump` command without the `--no-create-info` but with the `--tab` options. The output are two files: one containing the DDL statements for table creation, the other one with the exported MySQL data. The second file can be read by the DB2 `LOAD` command.

Example 6-4 Usage of mysqldump without the --no-create-info but with --tab option

```
bash>mysqldump --tab=. --user root itsodb testtable
```

```
bash>cat testtable.sql
```

```
-- MySQL dump 9.10
```

```
-- ...
```

```
--
```

```
-- Table structure for table `testtable`
```

```
--
```

```
CREATE TABLE testtable (  
  name char(10) default NULL,  
  birthday date default NULL  
) TYPE=MyISAM;
```

```
bash>cat testtable.txt
```

```
Klaus 1978-07-03
```

```
Michael 1974-03-20
```

```
Rakesh 1977-09-26
```

For porting the application data only, MTK generates the following call of `mysqldump`:

```
mysqldump --no-create-info --tab=<outFilePath>  
          --user=<username> [--password=<pwd>] <dbname> [<tablenames>]
```

DB2 loading tools

DB2 UDB provides two utilities for loading data into a database: the `LOAD` and the `IMPORT` command.

In general, the `LOAD` utility is faster than the `IMPORT` utility, because it writes formatted pages directly into the database, while the `IMPORT` utility performs SQL

insert statements. The **LOAD** utility validates the uniqueness of the indexes, but it does not fire triggers, and does not perform referential or table constraints checking.

DB2 LOAD command

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 contains four phases:

1. Load data.
2. Build indexes.
3. Delete rows with a unique key violation or a datalink violation.
4. Copy index data from the system temporary table space to the original table space.

See Example 6-5 for a simplified syntax diagram for the **LOAD** command. For a complete syntax description please visit:

<http://www-306.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8document.d2w/report?fn=r0008305.htm>

Example 6-5 Simplified syntax of the DB2 LOAD command

```
>>-LOAD--+-+-----+--FROM----+filename---+--+OF--filetype----->
      '-CLIENT-'           +-pipename---+
                          +-device-----+
                          '-cursorname-'

>--+-----+-----+-----+-----+-----+-----+-----+----->
|          .-----|
|          V         |
|'-MODIFIED BY---filetype-mod-+-'|
>--+-----+-----+-----+-----+-----+-----+-----+----->
|'-SAVECOUNT--n-' '|-ROWCOUNT--n-' '|-WARNINGCOUNT--n-'
>--+-----+-----+-----+-----+-----+-----+-----+----->
|'-MESSAGES--message-file-'
>--+-----+-----+-----+-----+-----+-----+-----+----->
|'-TEMPFILES PATH--temp-pathname-' +-REPLACE---+
|                                     +-RESTART---+
|                                     '-TERMINATE-'
>--+-----+-----+-----+-----+-----+-----+-----+----->
|'-INTO--table-name-+-|
|          .-----|
|          V         |
|'-((---insert-column-+-)-'|
|
|.-ALLOW NO ACCESS-----|
>--+-----+-----+-----+-----+-----+-----+-----+----->
|'-ALLOW READ ACCESS--+-|
|                                     '-USE--tablespace-name-'
>--+-----+-----+-----+-----+-----+-----+-----+----->
|'-CHECK PENDING CASCADE--+-IMMEDIATE-+-|
|                                     '-DEFERRED--|
>--+-----+-----+-----+-----+-----+-----+-----+----->
|'-LOCK WITH FORCE-'
```

For loading the application data into the DB2 UDB, MTK generates the following call of the **LOAD** command:

```
db2 LOAD from "<outFilePath>/<tablename>.dat"
of DEL
modified by\
coldel0x09
timestampformat=\YYYY-MM-DD HH:MM:SS\
insert into <schemaname>.<tablename>
```

DB2 IMPORT command

The **IMPORT** utility inserts data from an input file into a table or updateable view. If the table or view receiving the imported data already contains data, you can either replace or append to the existing data.

MTK by default uses a `TIMESTAMP` format that is 'YYYY-MM-DD hh:mm:ss', so this matches the MySQL `DATETIME` values. If you want to import MySQL `TIMESTAMP` values, you have to change the `LOAD` command in the `deploy.sh` script according to the following:

```
db2 LOAD from "'<outFilePath>/<tablename>.dat'" |
of DEL
modified by
coldel0x09
timestampformat=\''YYYYMMDDHHMMSS\''
insert into <schemaname>.<tablename>
```

If you have MySQL `DATETIME` values and MySQL `TIMESTAMP` values in the *same* table, you have to convert the format of the `TIMESTAMP` values to the `DATETIME` format in the data file. Then the data file can be processed by the `LOAD` command correctly.

Migration of BLOBs

BLOBs usually contain binary data. As binary data cannot be exported to text files, a different way has to be found to migrate them.

Note: If your BLOBs contain only ASCII data and no binary data, you can dump them out of MySQL and load them into DB2 UDB in the same way as all other data.

MTK does not support the automatic migration of BLOB data in the current version. Porting BLOB data from MySQL to DB2 UDB needs to be done manually.

MTK converts the DDL correctly, it exports the data correctly in case of text-only BLOBs, but the data files are *not* loaded automatically into DB2 UDB. In the case of text-only BLOBs, you just need to run the `LOAD` command for the MySQL export file generated by MTK.

If your BLOBs contain binary data (such as bitmaps), you have to write a program to migrates your data.

We provide a simple Java program `MigBlob.java` (Example 6-7) that copies data from a specified MySQL BLOB to a DB2 BLOB. This program can be easily integrated into migration scripts.

Example 6-7 Source code of the MigBlob utility

```
import java.lang.*;
import java.io.*;
import java.sql.*;

public class MigBlob {
    private static final String MYHOST = "localhost";
    private static final String MYDB = "temp";
    private static final String MYUSR = "root";
    private static final String MYPWD = "";

    private static final String DB2DB = "sample";
    private static final String DB2USR = "itsosj";
    private static final String DB2PWD = "itsosj";

    private static Connection mysqlConnection, db2Connection;

    public static void main(String[] args) throws SQLException, Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();

        mysqlConnection = DriverManager.getConnection(
            "jdbc:mysql://" + MYHOST + "/" + MYDB +
            "?user=" + MYUSR + "&password=" + MYPWD);

        db2Connection = DriverManager.getConnection(
            "jdbc:db2:" + DB2DB, DB2USR, DB2PWD);

        PreparedStatement ps1 = mysqlConnection.prepareStatement(
            "select * from blobdata");
        ResultSet rs = ps1.executeQuery();

        if ( ! rs.next() ) {
            System.out.println("No rows in blobdata...");
            return;
        }

        InputStream is = rs.getBinaryStream("b");

        PreparedStatement ps2 = db2Connection.prepareStatement(
            "insert into tempdb2.blobdata values(?)");
        ps2.setBinaryStream(1,is,is.available());
        ps2.executeUpdate();
    }
}
```

To use this program for migrating BLOBs, please follow these steps:

1. Install MySQL JDBC driver (if not installed yet)

In order to run the Java program you need to download and install the MySQL JDBC driver, if it is not already installed on your system. Make sure that both the MySQL JDBC driver and the DB2 JDBC driver, which is installed during DB2 Installation, are in your CLASSPATH environment variable. Your CLASSPATH value could look like Example 6-8.

Example 6-8 Sample CLASSPATH including MySQL and DB2 JDBC drivers

```
bash>echo $CLASSPATH
/home/db2inst1/sqllib/java/db2java.zip:/home/db2inst1/sqllib/java/db2jcc.jar:/home/db2inst1/sqllib/function:/home/db2inst1/sqllib/java/db2jcc_license_cisuz.jar:/home/db2inst1/sqllib/java/db2jcc_license_cu.jar::/usr/local/mysql/jdbc/mysql-connector-java-3.0.10-stable-bin.jar
```

2. Edit the *MigBlob* Java source code and insert your database connectivity information (user ID and password for both MySQL and DB2 UDB). Compile the Java source code:

```
bash>javac MigBlob.java
```

3. Connect to MySQL and create the temporary MySQL table for the *MigBlob* utility:

```
mysql>create table temp.blobdata (b longblob);
```

4. Connect to DB2 UDB and create the temporary DB2 table for the *MigBlob* utility:

```
db2>create table tempdb2.blobdata (b blob(2g) not logged)
```

5. Insert the BLOB you want to migrate into the temporary MySQL table:

```
mysql>insert into temp.blobdata select <blobcol> from <sourcetab>
where <keyfields>=<keyvalue>
```

6. Run the *MigBlob* utility to copy the BLOB from MySQL to DB2 UDB:

```
bash>java MigBlob
```

Please note that just *one* BLOB is copied by the *MigBlob* utility, so you have to run the utility for each BLOB you want to migrate, or you enhance the *MigBlob* utility.

7. Insert the BLOB from the temporary DB2 table into the designated field (be sure to use the key fields corresponding to the MySQL table):

```
db2>insert into <desttable> (<destcol>) (select b from
tempdb2.blobdata) where <keyfields>=<keyvalue>
```

8. Drop the temporary MySQL table:

```
mysql>drop table temp.blobdata;
```

9. Drop the temporary DB2 table:

```
db2>drop table tempdb2.blobdata
```

10.. For each BLOB that you want to migrate, proceed from step 3.

6.1.3 Differences in the user account management

The way the user information's access rights and privileges are stored in MySQL is completely different from DB2 UDB.

MySQL user account management

Migrating the user account management from MySQL to the DB2 security system requires knowledge about how the user account management data affects your application, and how user data, passwords, access rights, and privileges are stored in MySQL.

User data

When assessing your application, be sure to distinguish between the following user types:

► **Database users**

These users connect directly to the MySQL database to retrieve and manipulate data. At least one database user must exist for applications to connect to the database. These users are typically stored in the *mysql.user* table and must be migrated in data porting step. Access rights and privileges for these users are stored in the *mysql.db*, *mysql.host*, *mysql.tables_priv*, and *mysql.columns_priv* tables.

► **Application users**

These users log on to the application, but do not exist on the database level. Database access is through the application with the application's database user ID. As the information about application users is usually stored in custom application tables, the migration of application users is done when migrating the MySQL application data.

Passwords

Database users typically have associated passwords, which are stored encrypted in the *mysql.user* table. Encrypted passwords cannot be migrated and must be reset on the new system. The password of the database user that is used by an application to access the database is typically stored in a profile with restricted rights.

Note: The migration of encrypted passwords is impossible, because it is the intended purpose of encryption functions to make password unencryptable!

Access rights

You can connect to a MySQL database if you provide a user name and the associated password. Furthermore, the machine where you connect from must be associated with this user to allow a connection. This is based on the assumption that a user with a specific user name from one host is different from a user with the same user name from a different host.

This access information is stored in the *mysql.user* table in the fields *user*, *password*, and *host*. The MySQL wildcard % is often used in the host field to specify that this user can connect from any host. The wildcard ‘_’ is also sometimes used for single characters.

In Example 6-9 the user Klaus can connect from any host, the user Michael can connect from just the host lochness.almaden.ibm.com, and from localhost any user can connect.

Example 6-9 Sample user data for connection verification

```
mysql> select user, password, host from user;
```

user	password	host
Michael	2f4fba967c46f214	lochness.almaden.ibm.com
Klaus	c46f2142f4fba967	%

Please refer to the *MySQL Reference Manual* (at <http://dev.mysql.com/doc/>) for a complete description of the MySQL connection verification. You can also find information on how the entries in the *mysql.user* table are ordered when the provided connection data meets more than one connection criteria. In the above example, this would be when user Klaus connects from localhost.

Privileges

Once a connection to the MySQL database is established, each time a command is run against the database, MySQL checks if the connected user has sufficient privileges to run this command.

Privileges exist for selecting, inserting, updating, and deleting data for creating, altering, and dropping tables and other operations performed at the database level.

All privileges are stored in either the *mysql.user*, *mysql.db*, *mysql.host*, *mysql.tables_priv* or *mysql.columns_priv* tables.

Privileges in MySQL can be granted on the following levels:

▶ **Global level**

Global privileges are used on all databases on a given server and are stored in the *mysql.user* table.

▶ **Database level**

Database privileges are used on all tables in a given database and are stored in the *mysql.db* (database privileges) and *mysql.host* (host privileges) tables.

▶ **Table level**

Table privileges are used on all columns in a given table and are stored in the *mysql.tables_priv* table.

▶ **Column level**

Column privileges are used on single columns in a given table and are stored in the *mysql.columns_priv* table.

Privileges can be granted to users with the MySQL **GRANT** command; they can be revoked with the **REVOKE** command.

The privilege for a specific command is determined like this:

```
global privileges  
OR (database privileges AND host privileges)  
OR table privileges  
OR column privileges
```

To retrieve privilege information from a MySQL database, the **mysqlaccess** tool can be used.

For more information about MySQL privileges please, see the *MySQL Reference Manual*.

DB2 security system

DB2 UDB has no separate user account management, it uses the user accounts from the operating system where the database is installed on and grants the users specific database privileges needed.

User data

Creating a user for DB2 UDB means to create a user in the server's operating system, assign the user to a group, and grant specific database privileges to the user or group.

On Linux systems you must have root access to the system to create groups and users. Group information is stored in the file */etc/group*, user information in the file */etc/passwd*.

For example, if you want to create a new group db2app1 with one user db2usr1 to access a specific DB2 table, the necessary steps are:

1. Log on to the Linux system with *root* privileges.
2. Create the group. Make sure that the provided group name does not already exist. Group names should not be longer than eight characters:

```
groupadd [-g 995] db2app1
```

3. Create the user and assign it to the previously created group. Make sure that the ID for the user does not already exist. User names should not be longer than eight characters:

```
useradd [-u 1001] -g db2app1 -m -d /home/db2usr1 db2usr1 [-p passwd1]
```

If the user is going to access the DB2 database locally, then continue with the next two steps:

4. Edit the profile of the created user:

```
vi /home/db2usr1/.profile
```

5. Add the following line to the profile. Be sure to specify the path of your DB2 instance owner's home directory and to specify a blank between the dot and the command:

```
. /home/db2inst1/sql1lib/db2profile
```

Passwords

The passwords that are used for DB2 UDB are the system passwords of the user. To set a password in Linux use the **passwd <username>** command as *root* user.

Access rights

Access to DB2 databases is restricted to users that exist on the DB2 system. When connecting to a DB2 database you have to provide a valid user name and password of the server's system. The information from where a user connects (the hostname or IP address) is not required when connecting to the DB2 database. The information is in the DB2 UDB directory when the server and database are cataloged.

If you use host name feature in MySQL, you have to implement a work around (e.g. different users for each system). See Chapter 7.2.7, "Special conversions" on page 221 for more information about a workaround and sample code for a host authentication mechanism.

Authorities and privileges

Privileges enable users to create or access database objects. Authority levels provide a method of grouping privileges and control over higher-level database manager maintenance and utility operations. Together, these act to control

access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization, that is, the required privilege or authority.

Figure 6-1 illustrates the relationship between authorities and their span of control (database, database manager).

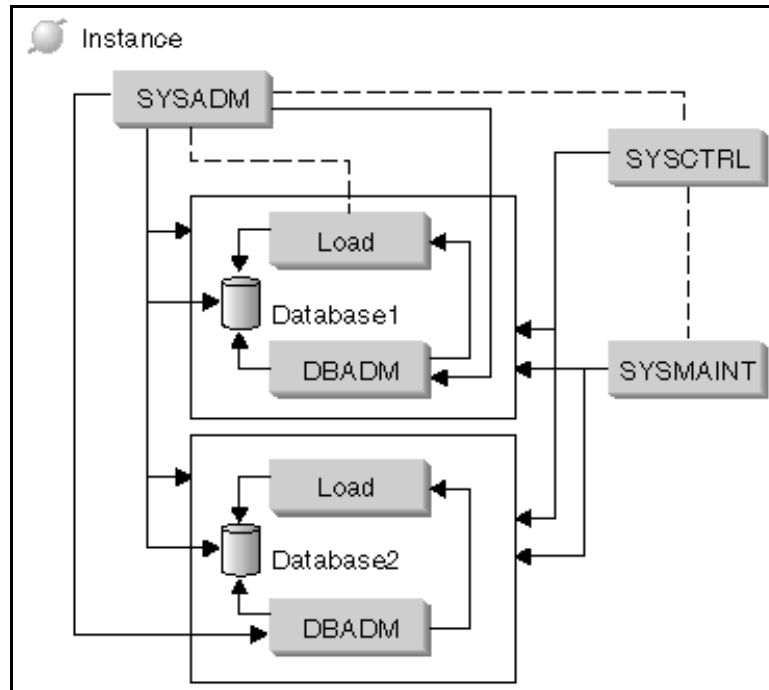


Figure 6-1 Hierarchy of authorities

A user or group can have one or more of the following levels of authorization:

- ▶ Administrative authority (SYSADM or DBADM) gives full privileges for a set of objects.
- ▶ System authority (SYSCTRL or SYSMAINT) gives full privileges for managing the system, but does not allow access to the data.
- ▶ Load authority (LOAD) gives LOAD utility or AutoLoader utility privileges to load data into tables.
- ▶ Ownership privilege (also called CONTROL privilege in some cases) gives full privileges for a specific object.
- ▶ Individual privileges may be granted to allow a user to carry out specific functions on specific objects.

- ▶ Implicit privileges may be granted to a user who has the privilege to execute a package. While users can run the application, they do not necessarily require explicit privileges on the data objects used within the package.

Users with administrative authority (SYSADM or DBADM) or ownership privileges (CONTROL) can grant and revoke privileges to and from others, using the GRANT and REVOKE statements. It is also possible to grant a table, view, or schema privilege to another user if that privilege is held WITH GRANT OPTION. However, the WITH GRANT OPTION does not allow the person granting the privilege to revoke the privilege once granted. You must have SYSADM authority, DBADM authority, or CONTROL privilege to revoke the privilege.

A user or group can be authorized for any combination of individual privileges or authorities. When a privilege is associated with a resource, that resource must exist. For example, a user cannot be given the SELECT privilege on a table unless that table has previously been created.

Note: Care must be taken to give authorities and privileges to a user name that does not exist in the system yet. At some later time, this user name can be created and will automatically receive all of the authorities and privileges associated.

Privileges in DB2 UDB can be granted on the following levels:

- ▶ **Database level**
 - CONNECT privilege
 - CREATETAB privilege
 - LOAD privilege
 - IMPLICIT_SCHEMA privilege
 - BINDADD privilege
 - others
- ▶ **Schema level**
 - CREATEIN privilege
 - ALTERIN privilege
 - DROPIN privilege
- ▶ **Table space level**
 - USE privilege

- ▶ **Table and view level**
 - CONTROL privilege
 - SELECT privilege
 - INSERT privilege
 - UPDATE privilege
 - DELETE privilege
 - INDEX privilege
 - ALTER privilege
 - REFERENCES privilege
- ▶ **Other privileges**
 - Package privileges
 - Index privileges
 - Procedure, function and method privileges
 - Sequence privileges

Privileges can be granted to users or groups with the **GRANT** command, they can be revoked using the **REVOKE** command.

When a database is created, the following privileges are automatically granted to PUBLIC (all users):

- ▶ CREATETAB
- ▶ BINDADD
- ▶ CONNECT
- ▶ IMPLICIT_SCHEMA
- ▶ USE privilege on USERSPACE1 table space
- ▶ SELECT privilege on the system catalog views.

Mapping the user information from MySQL to DB2 UDB

When migrating from MySQL to DB2 UDB, you have to migrate the privileges of the users as well.

Table 6-1 shows the mapping from MySQL privileges to DB2 privileges, assuming that different MySQL *databases* are mapped to different DB2 *schemas*.

For example, an INSERT privilege granted in MySQL on the *global* level means that you have to grant the INSERT privilege on all existing tables in the DB2 database to the specified user. If you create a new table in DB2 UDB, you have to grant the INSERT privilege on this table to the user.

Table 6-1 Mapping of MySQL to DB2 privileges

MySQL privilege	MySQL scope	DB2 privilege or authority	DB2 scope
Select	global	Select	All tables in the database
Insert	global	Insert	All tables in the database
Update	global	Update	All tables in the database
Delete	global	Delete	All tables in the database
Create	global	Createtab	Database
Drop	global	Dropin	All schemas
Reload	global	not available	
Shutdown	global	“SYSADM, SYSCTRL, or SYSMANT authority”	Instance
Process	global	“SYSADM, SYSCTRL, or SYSMANT authority”	Instance
File	global	LOAD authority	Database
Grant	global	Control	All tables in the database
Index	global	Createin	All schemas
Alter	global	Alterin	All schemas
Show_db	global	Select	SYSCAT.TABLES
Super	global	not available	
Create_tmp_table	global	Createtab	Database
Lock_tables	global	Select	All tables in the database
Repl_slave	global	not available	
Repl_client	global	not available	
Select	database	Select	All tables in the schema
Insert	database	Insert	All tables in the schema
Update	database	Update	All tables in the schema
Delete	database	Delete	All tables in the schema
Create	database	Createin	Schema

MySQL privilege	MySQL scope	DB2 privilege or authority	DB2 scope
Drop	database	Dropin	Schema
Grant	database	Control	All tables in the schema
Index	database	Creatin	Schema
Alter	database	Alterin	Schema
Create_tmp_table	database	Createin	Schema
Lock_tables	database	Select	All tables in the schema
Select	table	Select	Table
Insert	table	Insert	Table
Update	table	Update	Table
Delete	table	Delete	Table
Create	table	not available	
Drop	table	Control	Table
Grant	table	Control	Table
Index	table	Index	Table
Alter	table	Alter	Table
Select	column	Select	Table/view
Insert	column	Insert	Table/view
Update	column	Update	Table/view

Note: Privileges on the column level are not natively supported in DB2 UDB. The workaround for maintaining privileges on that scope is by defining views for specified columns in a table, and maintaining the privileges on table level of the view.

6.2 Sample project: Doing the data porting

This chapter describes the steps that we use to migrate the data in our sample project.

6.2.1 Export user data from MySQL

We use the `mysqlaccess` utility to get the user names that have access to the database `itsodb` out of the MySQL databases. See Example 6-10.

Example 6-10 Retrieve users with access to the sample project database

```
bash>mysqlaccess % % itsodb -b -U root
mysqlaccess Version 2.06, 20 Dec 2000
...

Sele Inse Upda Dele Crea Drop Relo Shut Proc File Gran Refe Inde Alte Show Supe Crea
Lock Exec Repl Repl Ssl_ Ssl_ X509 X509 Max_ Max_ Max_ | Host,User,DB
-----+-----
Y Y Y Y Y Y N N N N N Y Y Y N N Y Y
N N N ? ? ? ? 0 0 0 | %,itsosj,itsodb
N N N N N N N N N N N N N N N N N N
N N N N N N N N N N N N N N N N N N
N N N N N N N N N N N N N N N N N N
N N N ? ? ? ? 0 0 0 | %,Klaus,itsodb
N N N N N N N N N N N N N N N N N N
N N N N N N N N N N N N N N N N N N
Y Y Y Y Y Y N N N N N Y Y Y N N Y Y
N N N ? ? ? ? 0 0 0 | localhost,itsosj,itsodb
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Y Y Y ? ? ? ? 0 0 0 | localhost,root,itsodb
N N N N N N N N N N N N N N N N N N
N N N ? ? ? ? 0 0 0 | localhost,Michael,itsodb
N N N N N N N N N N N N N N N N N N
N N N ? ? ? ? 0 0 0 | localhost,Klaus,itsodb
N N N N N N N N N N N N N N N N N N
N N N ? ? ? ? 0 0 0 | localhost,ANY_NEW_USER,itsodb
```

6.2.2 Map MySQL user data to DB2 user data

The `root` user is the database administrator and is not used by our sample application; we do not want to migrate this user. The MySQL `root` user is similar to the instance owner in DB2 UDB.

The users `Michael`, `Klaus` and `ANY_NEW_USER` (which is similar to the logical DB2 group `PUBLIC`) do not have any privileges on our sample database, so we do not need to migrate these users either.

The only remaining user is the user `itsosj`. This one is the user our application uses to connect to the MySQL database and to manipulate data.

We want to add a new group for this user called `db2app1`.

The user's privileges are set for SELECT, INSERT, UPDATE, DELETE, CREATE and DROP on database level for itsodb, so we map these privileges to SELECT, INSERT, UPDATE and DELETE for all tables in the DB2 schema itsodb and CREATEIN and DROPIN for the schema.

6.2.3 Create DB2 user

For user and group creation we use the script in Example 6-11.

Example 6-11 Sample script to create DB2 users and groups

```
bash> cat db2addusr.sh
export DB2DIR='/home/db2inst1'
export HOMEDIR='/home'

groupadd $2
useradd -g $2 -m -d $HOMEDIR/$1 $1
passwd $1
echo '. '${DB2DIR}.'/sql1lib/db2profile' >> $HOMEDIR/$1/.profile
```

The creation of our user and group was done by the *root* user as shown in Example 6-12.

Example 6-12 Creation of the user and group

```
bash> ./db2addusr.sh itsosj db2app1
Changing password for itsosj.
New password:
Re-enter new password:
Password changed
```

The granting of the privileges was done by the instance owner db2inst1 with the DB2 command shown in Example 6-13.

Example 6-13 Granting of privileges

```
db2 => connect to itsodb2
```

```
Database Connection Information
```

```
Database server      = DB2/LINUX 8.1.4
SQL authorization ID = DB2INST1
Local database alias = ITSODB2
```

```
db2 => grant createin, dropin on schema itsodb to group db2app1
DB20000I The SQL command completed successfully.
db2 => grant select, insert, update, delete on table itsodb.shopping_cart to user itsodb
DB20000I The SQL command completed successfully.
db2 => grant select, insert, update, delete on table itsodb.users to user itsodb
DB20000I The SQL command completed successfully.
db2 => grant select, insert, update, delete on table itsodb.catalog to user itsodb
DB20000I The SQL command completed successfully.
db2 => grant select, insert, update, delete on table itsodb.oenumber to user itsodb
DB20000I The SQL command completed successfully.
```

6.2.4 Export MySQL application data

The export of the MySQL application data was done automatically when exporting the MySQL data with MTK as described in Chapter 5.5, “Sample database migration” on page 117.

In Example 6-14 the created export files are listed.

Example 6-14 Automatically created export files

```
bash>ls -l *.dat
-rw-r--r--  1 itsosj  db2app1    2166 2004-02-03 09:46 catalog.dat
-rw-r--r--  1 itsosj  db2app1   186660 2004-02-03 09:46 oenumber.dat
-rw-r--r--  1 itsosj  db2app1    791 2004-02-03 09:46 users.dat
-rw-r--r--  1 itsosj  db2app1    37 2004-02-03 09:46 shopping_cart.dat
```

Each of the files is tabulator delimited containing the data from the corresponding MySQL table. This format can be read by the DB2 **LOAD** command.

6.2.5 Convert MySQL application data to DB2 format

As no special data types (TIMESTAMP or BLOB) are used in our sample application, no manual conversion has to be done in our export files.

6.2.6 Import application data into DB2 UDB

Importing data into DB2 UDB can be done automatically by using MTK. MTK creates a script containing the DB2 **LOAD** commands and then executes the script. In case of errors the script can be edited and run again.

Example 6-15 shows the DB2 **LOAD** commands that were generated by MTK for our sample project.

Example 6-15 DB2 LOAD commands for loading the data into DB2 UDB

```
db2 LOAD from "/home/itsosj/mlog/catalog.dat" of DEL modified by colde10x09
timestampformat="\YYYY-MM-DD HH:MM:SS\" insert into itsodb.catalog
db2 LOAD from "/home/itsosj/mlog/oenumber.dat" of DEL modified by colde10x09
timestampformat="\YYYY-MM-DD HH:MM:SS\" insert into itsodb.oenumber
db2 LOAD from "/home/itsosj/mlog/users.dat" of DEL modified by colde10x09
timestampformat="\YYYY-MM-DD HH:MM:SS\" insert into itsodb.users
db2 LOAD from "/home/.../shopping_cart.dat" of DEL modified by colde10x09
timestampformat="\YYYY-MM-DD HH:MM:SS\" insert into itsodb.shopping_cart
```

After importing the data into the DB2 tables, the **RUNSTATS** command should be executed in order to recreate the statistic information about indexes. The statistics information is used by the query optimizer. See 9.5.5, “SQL execution plan” on page 308 for more information about the **RUNSTATS** command.

See Example 6-16 for the creation of the statistics information in our sample project.

Example 6-16 DB2 RUNSTATS commands for recreating the statistics information

```
db2 RUNSTATS on table itsodb.catalog
DB20000I The RUNSTATS command completed successfully.
db2 RUNSTATS on table itsodb.oenumber
DB20000I The RUNSTATS command completed successfully.
db2 RUNSTATS on table itsodb.users
DB20000I The RUNSTATS command completed successfully.
db2 RUNSTATS on table itsodb.shopping_cart
DB20000I The RUNSTATS command completed successfully.
```

6.2.7 Basic data checking

Once the script is executed, a log file is created where messages are stored. Please check the log file for the success of the DB2 **LOAD** commands. You can find this information in the log file as shown in Example 6-17.

Example 6-17 Log file information about the DB2 LOAD command

```
...
Deploying LOAD Scripts...
SQL3501W The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.

SQL3109N The utility is beginning to load data from file
"/home/itsosj/mlog/catalog.dat".
...
SQL3110N The utility has completed processing. "21" rows were read from the
input file.
...
Number of rows read          = 21
Number of rows skipped       = 0
Number of rows loaded        = 21
Number of rows rejected      = 0
Number of rows deleted       = 0
Number of rows committed    = 21
...
SQL3109N The utility is beginning to load data from file
"/home/itsosj/mlog/oenumber.dat".
...
Number of rows read          = 8771
Number of rows skipped       = 0
Number of rows loaded        = 8771
Number of rows rejected      = 0
Number of rows deleted       = 0
Number of rows committed    = 8771
...
SQL3109N The utility is beginning to load data from file
"/home/itsosj/mlog/users.dat".
...
Number of rows read          = 8
Number of rows skipped       = 0
Number of rows loaded        = 8
Number of rows rejected      = 0
Number of rows deleted       = 0
Number of rows committed    = 8
...
SQL3109N The utility is beginning to load data from file
"/home/itsosj/mlog/shopping_cart.dat".
...
Number of rows read          = 3
Number of rows skipped       = 0
Number of rows loaded        = 3
Number of rows rejected      = 0
Number of rows deleted       = 0
Number of rows committed    = 3
...

```

Please make sure that the number of rows read equals the number of rows committed. This should also equal to the number of records in the MySQL source table. Example 6-18 shows the MySQL command for the record count.

Example 6-18 Retrieving the number of records from MySQL

```
mysql> select count(*) from catalog;
+-----+
| count(*) |
+-----+
|      21 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from oenumber;
+-----+
| count(*) |
+-----+
|     8771 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from users;
+-----+
| count(*) |
+-----+
|         8 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from shopping_cart;
+-----+
| count(*) |
+-----+
|         3 |
+-----+
1 row in set (0.04 sec)
```

After you have checked that all the records were loaded into DB2 UDB, you should check sample data in each migrated table if the values are correct, especially if you have any-time values or decimal values. Example 6-19 shows the table content checking.

Example 6-19 Sample data of MySQL data

```
mysql> select * from catalog limit 1;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|manufacturer| model      | type | year_from | year_to | kw  | hp  | ccm | cyl | motorcode |
| pg  | sku  | dimension | ac  | transmiss | hint  | stock | id |  |  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| VW      | PASSAT (3B2) | 1.6 | 10.1996 | 11.2000 | 74  | 101 | 1595 | 4  | ADP/AHL/ANA/ARM |
| 10  | 16633 | 630-420 | +/- | M      | see figure | 10 | 2 |  |  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.05 sec)
```

```
db2 => select * from itsodb.catalog where manufacturer='VW' and model='PASSAT (3B2)' and type='1.6'
```

MANUFACTURER	MODEL	TYPE	YEAR_FR	YEAR_TO	KW	HP	CCM	CYL	
MOTORCODE			PR	SKU	DIMENSION	AC	TRANSMIS		
HINT			STOCK	ID					
VW	PASSAT (3B2)	1.6		10.1996	11.2000	74	101	1595	4
ADP/AHL/ANA/ARM			10	16633	630-420		+/-	M	
see figure				10		2			

1 record(s) selected.



Application porting

The task of migrating an application, its databases, and the associated data most often requires significant resources and commitments. Simultaneously to the meticulous planning of the porting project as a whole, it is paramount to assess the issues that may be high up to the highest level of resources. In many porting projects this is the part of application porting. This chapter attempts to give the reader “food for thought” in the following areas:

- ▶ Differences in SQL Data Manipulation Language (DML), built-in functions, and SQL semantics
- ▶ Converting application source, application programming interfaces (APIs), and condition handling
- ▶ Internals of the Database Management System that may impact the conversion, such as locking, isolation levels, transaction logging, and national language support

7.1 Differences and similarities in Data Manipulation Language

Modifying the application to work with DB2 UDB can be a significant task in the conversion process. While a large portion of this step may encompass changing code to work with a different development environment, it is likely that additional time will be spent on testing the resulting code.

7.1.1 SELECT syntax

This section focuses on the SELECT statement syntax as it is supported in MySQL, and attempts to show how MySQL extensions to the SQL standard might be implemented in DB2 Version 8. Example 7-1 shows the MySQL syntax for the SELECT statement. A discussion of individual keywords follows.

Example 7-1 MySQL SELECT syntax

```
SELECT [STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
[DISTINCT | DISTINCTROW | ALL]
select_expression,...
[INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
[FROM table_references]
[WHERE where_definition]
[GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
[WITH ROLLUP]
[HAVING where_definition]
[ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
[LIMIT [offset,] row_count | row_count OFFSET offset]
[PROCEDURE procedure_name(argument_list)]
[FOR UPDATE | LOCK IN SHARE MODE]
```

The **STRAIGHT_JOIN** keyword forces the MySQL optimizer to join tables in the specified order. In DB2 the join order is always determined by the optimizer. The optimizer choices can be limited by changing the default query optimization class using SET CURRENT QUERY OPTIMIZATION to a lower level. This may still not force the optimizer to evaluate the join order as given in the SQL statement. However, the DB2 cost-based optimizer usually chooses the best access path for a given query. For additional information see 9.5.5, “SQL execution plan” on page 308.

The options prefixed with **SQL_** are MySQL specific and do not require a DB2 equivalent.

The **DISTINCTROW** keyword is as a synonym for **DISTINCT** which is supported by DB2.

The **INTO {OUTFILE | DUMPFILe}** 'file_name' export_options selection allows you to write data to an outfile quickly without invoking the mysqldump utility. The DB2 command line processor allows you to direct the output of any **SELECT** statement to an operating system file.

The **LIMIT [offset,] row_count | row_count OFFSET offset** keyword translates to **FETCH FIRST n ROWS ONLY** in DB2. An offset to retrieve rows needs to be implemented through the **WHERE** clause if possible.

With the **SQL_SMALL_RESULT** or **SQL_BIG_RESULT** the query developer can hint to the SQL optimizer the size of the expected result set, and therefore influences the optimizer access strategy. Example 7-2 shows how the optimizer hint works.

Example 7-2 MySQL SELECT with optimizer hint

```
mysql>
mysql> select sql_small_result * from t1;
mysql>
```

DB2 UDB has a similar operator to guide SQL optimizer decisions with a different syntax as shown in Example 7-3. Please note that the number of rows is exemplary.

Example 7-3 DB2 SELECT with optimizer hit

```
db2 => select * from t1 optimize for 2 rows
DB20000I The SQL command completed successfully.
```

7.1.2 JOIN syntax

The join capabilities of a commercial and industrial strength database management system are one of the most significant functions. MySQL supports the linguistic elements for **JOIN** as shown in Example 7-4. Various aspects of **JOINS** are discussed below.

Example 7-4 MySQL JOIN Syntax

```
table_reference, table_reference
table_reference [INNER | CROSS] JOIN table_reference [join_condition]
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference [join_condition]
table_reference NATURAL [LEFT [OUTER]] JOIN table_reference
{OJ table_reference LEFT OUTER JOIN table_reference ON conditional_expr }
```

```
table_reference RIGHT [OUTER] JOIN table_reference [join_condition]
table_reference NATURAL [RIGHT [OUTER]] JOIN table_reference
```

Where `table_reference` is defined as:

```
table_name [[AS] alias] [[USE INDEX (key_list)] | [IGNORE INDEX (key_list)] | [FORCE
IN and join_condition is defined as:
ON conditional_expr |
USING (column_list)
```

The **STRAIGHT_JOIN** keyword forces the MySQL optimizer to join tables in the specified order. In DB2 the join order is always determined by the optimizer. The optimizer choices can be limited by changing the default query optimization class using **SET CURRENT QUERY OPTIMIZATION**.

A **NATURAL** join, as its name implies, can be invoked when two or more tables share exactly the same columns needed for a successful equijoin. It is semantically equivalent to DB2 **INNER JOIN** or **LEFT OUTER JOIN** with the respective join criteria specified in the **ON** clause.

According to the SQL ANSI standard when you need to join tables that share more than one column naturally, the **JOIN ... USING** syntax needs to be used. An equivalent join can be composed using the DB2 supported join syntax in the **ON** clause.

Everybody in application and database query development spends a lot of time trying to avoid them; however, cartesian products do happen from time to time, usually as the result of an equijoin condition that has been missed in a query using DB2 syntax. However, one of the advantages of the **CROSS JOIN** syntax is that a specific keyword is required to create a Cartesian product. Therefore, when the **CROSS JOIN** syntax is used in your migration project, just code a regular join in DB2 with a no join condition in the **WHERE** clause.

7.1.3 UNION Syntax

In MySQL Version 4.0.0 the newly implemented **UNION** feature (shown in Example 7-5) is very similar to the DB2 syntax, and therefore does not require additional discussion.

Example 7-5 UNION syntax in MySQL and DB2

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
[UNION [ALL | DISTINCT]
SELECT ...]
```

7.1.4 Subquery syntax

A subquery is a SELECT inside another query or statement. Therefore, subqueries can be found in other SELECT either in the column list, the WHERE clause, or the HAVING clause, and in addition in INSERT, UPDATE, and DELETE statements. In Example 7-6 a sequence of subqueries is shown in a DELETE statement.

Example 7-6 Example for subqueries in a DELETE statement

```
DELETE FROM t1
WHERE col1 > ANY
  (SELECT COUNT(*) FROM t2
   WHERE NOT EXISTS
     (SELECT col3 FROM t3
      WHERE col3 =
        (SELECT col4 FROM t4 UNION SELECT 1 FROM
         (SELECT col5 FROM t5) AS t5 )));
```

Our project uses MySQL Version 4.0.17 and subqueries are supported in MySQL from Version 4.1, and it therefore has not been possible to do a comprehensive testing of subqueries. However, after studying the reference manual for MySQL, the subquery implementation seems quite similar to the DB2 implementation. Nothing has been found causing serious issues when moving from MySQL to DB2.

7.1.5 Grouping, having, and ordering

All ANSI SQL 92 standard grouping functions available in MySQL Version 4.0 are also available in DB2. In general no significant differences were found in the area of grouping, having, and ordering. However, beyond the SQL 92 standard DB2 provides some interesting functionality that may enhance the ported application significantly. Please refer to DB2 UDB manual *SQL Reference, Volume 2*, SC09-4845. Table 7-1 lists the differences in two databases and provides conversion examples.

Table 7-1 Differences in DB2 and MySQL grouping, having and ordering

MySQL	DB2 UDB	Example
BIT_AND	Not available. Example provided	<p>MySQL: select bit_and(a),a from t1 group by a</p> <p>DB2 UDB: Refer to UDF example BITAND in Appendix A.1, "Sample code for BIT_AND" on page 336</p>
BIT_OR	Not available	<p>MySQL: select bit_or(a),a from t1 group by a</p> <p>DB 2UDB: Compare to UDF BitAnd in Appendix A.1, "Sample code for BIT_AND" on page 336</p>
COUNT(DISTINCT expr,expr,...)	DB2 allows only one expression: COUNT(DISTINCT expr). Use CONCAT for character data type or CHAR and CONCAT on numeric data types	<p>MySQL: select count(distinct a,b),a from t1 group by a"</p> <p>DB2 UDB: select count(distinct concat(a,b)), a from t1 group by a</p>
STD	STDDEV	<p>MySQL: select std(a),a from t1 group by a</p> <p>DB2 UDB: select stddev(a), a from t1 group by a</p>
STDDEV	STDDEV	<p>MySQL: select stddev(a),a from t1 group by a</p> <p>DB2 UDB: select stddev(a),a from t1 group by a</p>

MySQL	DB2 UDB	Example
not available	VARIANCE	DB2 UDB: select variance(a), a from t1 group by a
GROUP BY on alias	use column name for grouping	MySQL: select a as ab from t1 group by ab DB2 UDB: select a from t1 group by a
GROUP BY on position	use column name for grouping	MySQL: select a from t1 group by 1 DB2 UDB: select a from t1 group by a
HAVING on alias	use column name in having clause	MySQL: select a as ab from t1 group by a having ab > 0 DB2 UDB: select a from t1 group by a having a > 0

7.1.6 Strings

Unless MySQL is started in ANSI mode using `mysqld --ansi` it will behave differently from DB2 Version 8.1. As Example 7-7 illustrates, MySQL accepts single as well as double quotes as a string delimiter when started in default mode.

Example 7-7 MySQL string handling

```
mysql> select 'redbook', "redbook", "'redbook'", 'red'book';
+-----+-----+-----+-----+
| redbook | "redbook" | "'redbook'" | red'book |
+-----+-----+-----+-----+
| redbook | "redbook" | "'redbook'" | red'book |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> select "redbook", "'redbook'", "'redbook'", "red"book";
+-----+-----+-----+-----+
| redbook | 'redbook' | "'redbook'" | red"book |
+-----+-----+-----+-----+
```

```
| redbook | 'redbook' | ''redbook'' | red"book |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Since DB2 UDB has been designed and implemented according to the ANSI standard, it accepts single quotes as a string delimiter. Double quotes are used in DB2 for delimiting SQL identifiers; for example, in a migration project you may want to create table or column names in lower or mixed character notation. Example 7-8 shows how DB2 handles strings. Similar results will be achieved when MySQL runs in ANSI mode.

Example 7-8 DB2 string handling

```
db2 => select 'redbook', '"redbook"', ''"redbook"'' , 'red'book' from t1

1      2      3      4
-----
redbook "redbook" ""redbook"" red'book
```

Table 7-2 provides an overview of MySQL string related functions, and how these can be converted to DB2 UDB.

Table 7-2 MySQL and DB2 UDB string related function

MySQL	DB2 UDB	Comment
ASCII in string cast: select ascii('a') from t1	ASCII in string cast: select ascii('a') from t1	Returns ASCII code value, for example values(ascii('i')) returns 105
automatic num->string convert automatic string->num convert	Require explicit casting	Please refer to: "Implicit casting of data types" on page 163
concatenation with +	CONCAT or	Concatenation with +.
CONCAT('a','b','c')	('a' 'b' 'c')	Use to implement CONCAT(list)
ELT(n,str1,str2,str3,...)	CASE	Returns the <i>n</i> th string or NULL. Use CASE expression or UDF
FORMAT: select format(1234.5555.2) returns 1,234.56	no equivalent, implementing use UDF	Refer to UDF in Appendix A.2, "Sample code for FORMAT function" on page 337

MySQL	DB2 UDB	Comment
LPAD	no equivalent. Implement using UDF	Refer to UDF LPAD in Appendix A.3, "Sample code for RPAD and LPAD functions" on page 339
MID	SUBSTR	Function MID is synonym for SUBSTR
REVERSE	<pre>SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR)); WHILE LEN > 0 DO SET (REVSTR, RESTSTR, LEN)= (SUBSTR(RESTSTR, 1, 1) REVSTR,SUBSTR(RESTSTR, 2, LEN - 1),LEN - 1); END WHILE;</pre>	Implement UDF. For the complete code of the example refer to IBM DB2 UDB SQL Reference, Volume 1, V8, SC09-4844
RPAD	no equivalent. Implement using UDF	Refer to UDF RPAD in Appendix A.3, "Sample code for RPAD and LPAD functions" on page 339
STRCMP	CASE	Implement using CASE expression and VALUES statement
SUBSTRING_INDEX	no equivalent. Implement using UDF	Refer to UDF SUBSTRING_INDEX in Appendix A.7, "Sample code for SUBSTRING_INDEX" on page 360
TRIM (1 arg)	LTRIM,RTRIM	Requires string manipulation
TRIM; Many char extension	LTRIM,RTRIM	Requires string manipulation
TRIM; Substring extension	LTRIM,RTRIM	Requires string manipulation

7.1.7 Implicit casting of data types

We have found significant differences in the way MySQL and DB2 UDB handle the casting of data types. Unless explicitly using the CAST function, DB2 does not convert strings representing numeric values. Example 7-9 shows how MySQL implicitly casts the character value 5 to an integer value to resolve the query. Implicit casting is not only performed in unambiguous cases as shown in

the example, but also for mixed strings such as 6x. For further information on the conversion of strings and numbers, you may want to consult the *MySQL Reference Manual*.

Example 7-9 MySQL performs implicit data type casting

```
mysql> create table t1 (c1 int);
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> insert into t1 values(5);
Query OK, 1 row affected (0.02 sec)
```

```
mysql> select * from t1 where c1='5';
+-----+
| c1    |
+-----+
| 5     |
+-----+
1 row in set (0.00 sec)
```

An implicit casting of incompatible data types is not support by DB2 UDB. An explicit casting of the character value to an integer value is required as illustrated in Example 7-10.

Example 7-10 DB2 UDB requires explicit casting

```
db2 => create table t1 (c1 int)
DB20000I The SQL command completed successfully.
db2 => insert into t1 values(5)
DB20000I The SQL command completed successfully.
db2 => commit
DB20000I The SQL command completed successfully.
db2 => select * from t1 where c1 = '5'
SQL0401N The data types of the operands for the operation "=" are not
compatible. SQLSTATE=42818
```

```
db2 => select * from t1 where c1 = cast ('5' as int)
```

```
C1
-----
          5

1 record(s) selected.
```

Example 7-11 illustrates how MySQL implicitly casts numeric values and DATA, TIME, or TIMESTAMP values to strings when concatenated.

Example 7-11 MySQL implicit casting using concatenation for strings and DATE

```
mysql> select concat('ITSOSJ',1234) from t1;
+-----+
| stringcol |
+-----+
| ITSOSJ1234|
+-----+
1 row in set (0.02 sec)

mysql> select concat('ITSOSJ',current_date) as stringdate from t1;
+-----+
| stringdate |
+-----+
| ITSOSJ2004-01-23 |
+-----+
1 row in set (0.01 sec)
```

DB2 requires compatible arguments for the concatenation built-in functions as shown in Example 7-12. If the arguments are not compatible (for example, one argument is of character data type and the second argument is of numeric data type) the concatenation will fail with the error message SQL0440.

Example 7-12 DB2 UDB casting character strings and DATE explicitly

```
db2 => select concat('ITSOSJ',1234) from t1
SQL0440N No authorized routine named "CONCAT" of type "FUNCTION" having
compatible arguments was found.  SQLSTATE=42884

db2 => select concat('ITSOSJ','1234') as stringcol from t1

STRINGCOL
-----
ITSOSJ1234

1 record(s) selected.

db2 => select concat('ITSOSJ', current date) as stringdate from t1
SQL0440N No authorized routine named "CONCAT" of type "FUNCTION" having
compatible arguments was found.  SQLSTATE=42884

db2 => select concat('ITSOSJ',CAST(current date as char(20))) as stringdate
from t1

STRINGDATE
-----
ITSOSJ01/23/2004
```

1 record(s) selected.

7.1.8 String concatenation and NULL values

The ANSI92 standard states that if you concatenate a NULL value onto an existing string, the result set is NULL. Example 7-13 shows you the behavior of MySQL.

Example 7-13 How MySQL concatenates strings and NULL values

```
mysql> create table t2 (col1 char(2));
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> insert into t2 values(NULL);
Query OK, 1 row affected (0.07 sec)
```

```
mysql> select concat('abc',col1) as nullstring from t2;
+-----+
| nullstring |
+-----+
| NULL      |
+-----+
1 row in set (0.05 sec)
```

```
mysql> select concat('abc', coalesce(col1,'')) as nullstring from t2;
+-----+
| nullstring |
+-----+
| abc       |
+-----+
1 row in set (0.00 sec)
```

As shown, MySQL behaves as ANSI-92 compliant, and therefore gives you the same result sets as Example 7-14 for DB2.

Example 7-14 DB2 handles string and NULL concatenation similar

```
db2 => create table t2 (col1 char(2))
DB20000I The SQL command completed successfully.
db2 => insert into t2 values(NULL)
DB20000I The SQL command completed successfully.
db2 => select concat('abc', col1) as nullstring from t2
```

```
NULLSTRING
-----
-
```

```
1 record(s) selected.
```

```
db2 => select concat('abc', coalesce(col1,'')) as nullstring from t2
```

```
NULLSTRING
```

```
-----
```

```
abc
```

```
1 record(s) selected.
```

7.1.9 Record deletion

As some other competitive relational database management systems, MySQL introduced in Version V4.0 the TRUNCATE statement. TRUNCATE is used to delete all rows from a table when there is no need to recover the deleted records. Therefore, there is no ROLLBACK after a TRUNCATE is possible. Example 7-15 shows how TRUNCATE is used to remove all rows, and does not allow a rollback of the deleted rows.

Example 7-15 Using MySQL TRUNCATE to delete all records in a file

```
mysql> select * from t1;
```

```
+-----+
```

```
| col1 |
```

```
+-----+
```

```
| 5 |
```

```
| 10 |
```

```
+-----+
```

```
2 rows in set (0.02 sec)
```

```
mysql> truncate table t1;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from t1;
```

```
Empty set (0.00 sec)
```

```
mysql> rollback
```

```
-> ;
```

```
ERROR 1196: Warning: Some non-transactional changed tables couldn't be rolled back
```

```
mysql> select * from t1;
```

```
Empty set (0.00 sec)
```

The TRUNCATE option is primarily used to quickly delete all records from a table when no recovery of the deleted rows is required. To achieve a similar behavior

in DB2 V8.1, you may want to turn off logging with the following ALTER TABLE statement.

```
ALTER TABLE <tablename> ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE
```

7.1.10 Built-in functions and operators

A function is an operation that is denoted by a function name followed by a pair of parentheses enclosing the specification of arguments if arguments are required.

The following group of tables do not claim to be a complete reference of differences in built-in functions and operators between MySQL Version 4.0 and DB2 UDB Version 8.1. The listed built-in functions attempt to highlight those differences and map functionality between the two database management systems. Grouping the built-in functions and operators into ANSI standard, ODBC standard and other groups seems somehow arbitrary; however, this is one way of dividing the sum of all built-in functions into something smaller and easier to conquer.

Functions following the ANSI SQL standard

Table 7-3 lists functions specific to the ANSI SQL92 standard.

Table 7-3 SQL 92 functions

MySQL	DB2 equivalent	Description
select bit_length('abc')	LENGTH('abc')*8	Result depends on the encoding scheme used for character data: singly byte, double byte, UTF-8
select char_length(b) from t1; select character_length('abcd')	LENGTH(expr)	Returns the number of bytes for expression. For double byte character set (DBCS) the number of DBCS characters is returned
select extract(minute from timestamp '2000-02-23 18:43:12.987')	VALUES MINUTE('2000-02-23 18:43:12.987'), HOUR, YEAR, MONTH, DAY	Extracts part of date/time

MySQL	DB2 equivalent	Description
LOCALTIME, LOCALTIME(), LOCALTIMESTAMP, LOCALTIMESTAMP()	CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP	Are all synonyms for NOW() which returns date and time in YYYY-MM-DD HH:MM:SS or YYYYMMDDHHMMSS format.
select octet_length('abc')	LENGTH	OCTET_LENGTH is in MySQL a synonym for LENGTH
select positional in 'hello')	VALUES POSSTR('hello', 'll')	Returns the position of the search string
select substring('abcd' from 2 for 2)	SUBSTR('abcd', 2, 2)	ANSI SQL SUBSTRING
select trim(trailing from trim(LEADING FROM 'abc'))	Use RTRIM and LTRIM	Combination of LTRIM and RTRIM

Functions according to ODBC declaration

ODBC 3.0 requires a wide range of built-in function and it is amazing to find only minor differences between the two DBMS. Please note that not all functions have been tested for their results. Specifically when performing date and time arithmetic using the WEEK function and the DATE function listed in Table 7-4, the results between MySQL and DB2 maybe different. However, these may not be the only ones returning different results.

Table 7-4 ODBC 3.0 functions

MySQL	DB2	Description
select dayofmonth('2004-02-23') from t1	VALUES DAY('2004-02-23')	DAY returns the day portion of the argument
select hour(TIME '12:13:14')	VALUES HOUR(TIME('12:13:14'))	Slightly different syntax
WEEK	WEEK	DB2 starts at week 1, MySQL starts at week 0. DB2 also provides WEEK_ISO

Comparing operators

There is a plethora of function as listed in Table 7-5, Table 7-6, and Table 7-7 where syntax or implementation can be quite different. The tables attempt to summarize these differences and show code snippets for some selected functions on how to convert a MySQL function to DB2. In many cases you will find the DB2 CASE expression is really helpful when function mapping is required.

Table 7-5 MySQL and DB2 UDB operator comparison

MySQL	DB2	Comments
logical NOT as '!' in SELECT list	VALUES CASE WHEN 1!=1 THEN 0 ELSE 1 END	Implement using CASE expression and VALUES statement
%	MOD	In MySQL % is a synonym for modulo
& (bitwise and)	not available. Implement using UDF	Refer to UDF BIT_AND in Appendix A.1, "Sample code for BIT_AND" on page 336
logical AND as '&&' in SELECT list	CASE	Implement using CASE expression and VALUES statement
not equal, <> or != in SELECT list: select 1<>1	SELECT CASE WHEN 1<> 1 THEN x ELSE y END	Implement using CASE expression
Function = in SELECT list: select (1=1)	CASE	Implement using CASE expression and VALUES statement

Functions converting date and time

Working with date and time functions and operators uncovers in many cases the implementation differences for the various DBMS. Table 7-6 lists some date and time related functions. You may also find such functions in the SQL92 and ODBC standards.

Table 7-6 Date and Time related functions

MySQL	DB2 UDB	Comments
DATE_FORMAT	various options such as DAYNAME, MONTHNAME, etc.	Formatting DATE, TIME data types

MySQL	DB2 UDB	Comments
FROM_DAYS(n)	DATE(n-365)	DB2 and MySQL use different base year
FROM_UNIXTIME	TIMESTAMP_FORMAT	May deliver slightly different result
PERIOD_ADD: select period_add(9602,-12)	SELECT DATE('2004-02-23') - 12 MONTH FROM T1	Adds a number of month
PERIOD_DIFF	YEAR and MONTH	Requires some date arithmetic
WEEKDAY	DAYOFWEEK	DB2 range is 1 (Sunday) to 7, MySQL ranges 0 (Monday™) to 6."
SEC_TO_TIME	HOUR, MINUTE, SECOND	Requires some time arithmetic
TIME_TO_SEC	HOUR, MINUTE, SECOND	Requires some time arithmetic
TO_DAYS	DAYS	Requires some date arithmetic

More functions

There is always a set of items left that do not seem to fit in any of the categories defined when the categorization has been decided. Those you find in Table 7-7.

Table 7-7 Other functions

MySQL	DB2 UDB	Comments
BETWEEN in SELECT	CASE	Implement using CASE expression and VALUES statement
<< and >> (bitwise shifts)	no equivalent	Implement using <i>power</i> function: MySQL: SELECT (x>>y) SELECT(x<<y) DB2 UDB: SELECT(x/power(2,y)) SELECT(x*power(2,y)):

MySQL	DB2 UDB	Comments
BIT_COUNT	no equivalent, implement using UDF	Refer to BIT_CNT UDF in Appendix A.6, "Sample code for BIT_COUNT" on page 359
ENCRYPT	ENCRYPT	DB2 requires encryption password
FIELD	CASE	Implement using CASE expression and VALUES statement
GREATEST	FnGratst	See UDF example in Appendix A.4, "Sample code for GREATEST function" on page 346
IF	CASE	Implement using CASE expression and VALUES statement
IN on numbers in SELECT	CASE	Implement using CASE expression and VALUES statement
IN on strings in SELECT	CASE	Implement using CASE expression and VALUES statement
LOCATE as INSTR	LOCATE	Arguments are swapped
INTERVAL	CASE	Implement using CASE expression and VALUES statement
LAST_INSERT_ID	IDENTITY_VAL_LOCAL	DB2 returns NULL, MySQL return zero if not set
LEAST	FnLeastN	See UDF example in Appendix A.5, "Sample code for LEAST" on page 353
LIKE in SELECT	CASE with LIKE	Implement using CASE expression and VALUES statement

MySQL	DB2 UDB	Comments
LIKE ESCAPE in SELECT	CASE with LIKE and ESCAPE	Implement using CASE expression and VALUES statement
LOG(m,n)	LOG(m)/LOG(n)	To convert logarithm to an arbitrary base
NOT in SELECT	CASE	Implement using CASE expression and VALUES statement
NOT BETWEEN in SELECT	CASE	Implement using CASE expression and VALUES statement
NOT LIKE in SELECT	CASE	Implement using CASE expression and VALUES statement
PASSWORD	ENCRYPT	For encryption
POW	POWER™	Returns value of arg1 to the power of arg2
REGEXP in SELECT	no equivalent in DB2 UDB 8.1	see article on developer works for work-around http://www-106.ibm.com/developerworks/db2/library/techarticle/0301stolze/0301stolze.html
ROUND(1 arg)	INT(ROUND(arg1,0))	Round to integer value
VERSION	db2level	To retrieve installed version of DBMS

7.2 Application source conversion

Every programming language provides a huge amount of commands and possibilities to program a special functionality. It is almost impossible to give a migration example for every possible programming approach. Therefore, in this section we discuss the most important commands used to access a database, which are:

- ▶ **connecting database**
- ▶ **query statements**
- ▶ **disconnecting database**

7.2.1 Converting MySQL Perl applications to DB2 UDB

Two interfaces between Perl and the MySQL database have to be considered when discussing application porting from MySQL to DB2. First is *Mysql.pm*, a custom interface that works only with MySQL. The other newer interface is a plug-in for the Database Independent (DBI) set of modules. DBI provides a common Perl API for all database accesses and enables greater portability. The DBI interface is the most robust and standard. However, many legacy systems still use the *Mysql.pm* interface to connect to MySQL. In fact, the MySQL interface *Mysql.pm* is currently implemented as an emulation on top of the DBI drivers. The interface for accessing DB2 is the DBI interface with the `DBD::DB2` driver. Information about and how to install the DBI interface and the `DBD::DB2` driver can be found at the following Web sites:

<http://www.ibm.com/software/data/db2/perl/>
<http://www.perl.com/CPAN/modules/by-module/DBD/>
<http://aspn.activestate.com/ASPN/Modules/>

As applications using the DBI interface to connect to MySQL generally can be adapted to DB2 by simply changing the database driver from `DBD::Mysql` to `DBD::DB2`; this section only discusses the application conversion from *Mysql.pm* to DBI interface.

Converting *Mysql.pm* to DBI interface supported code

The interface documentation is normally installed with the appropriate module in the *perl* directory. More detailed information about the interface and the provided functions can be found there.

Connecting database

With *Mysql.pm*, five different connect statements can be used to connect to a database in a MySQL server:

```
$dbh = Mysql->Connect;  
$dbh = Mysql->Connect($host);  
$dbh = Mysql->Connect($host,$database);  
$dbh = Mysql->Connect($host,$database,$password);  
$dbh = Mysql->Connect($host,$database,$password,$user);
```

The connect statement needs the host name, database name, user ID, and password. The first four statements assume some or all connection information from the environment. If no *\$database* parameter is given, a `SelectDB` statement has to be provided to connect to a database. In Example 7-16 the connect syntax to a MySQL database is shown. For simplicity reasons the error handling is not included in following examples.

Example 7-16 MySQL database connection with the Mysql.pm interface

```
use Mysql;  
my $host="localhost";  
my $database="itsodb";  
my $user="itsosj";  
my $password="itsosj";  
  
$dbh = Mysql->connect($host,$database,$password,$user);
```

The corresponding DBI connection to a DB2 database is quite similar. Instead of the `use Mysql`, the DBI interface and the `DBD::DB2` driver have to be defined. If DB2 constants are used (e.g. `SQL_MODE_READ_ONLY` in the connection attributes, etc.) this has to be announced to the Perl interpreter by including `'use DBD::DB2::Constants'` in the Perl program (see Example 7-17).

Example 7-17 DB2 database connection with DBI interface

```
use DBI;  
use DBD::DB2::Constants;  
use DBD::DB2;  
my $user="itsosj";  
my $password="itsosj";  
  
$dbh = DBI->connect("dbi:DB2:itsodb2", $user, $password);
```

As DB2 is more powerful than MySQL, the connect statement may require a fourth argument `\%attr` which contains the connection attributes as shown in Example 7-18.

```
$dbh=DBI->connect($data_source, $user,$password, \%attr);
```

Example 7-18 DB2 specific connection attributes

<code>db2_access_mode</code>	SQL_MODE_READ_ONLY or SQL_MODE_READ_WRITE
<code>db2_clischema</code>	Character string
<code>db2_close_behavior</code>	SQL_CC_NO_RELEASE or SQL_CC_RELEASE
<code>db2_connect_node</code>	Integer (must be set in DBI->connect method; it cannot be modified afterwards)
<code>db2_current_schema</code>	Character string
<code>db2_db2estimate</code>	Integer
<code>db2_db2explain</code>	One of: SQL_DB2EXPLAIN_OFF SQL_DB2EXPLAIN_SNAPSHOT_ON SQL_DB2EXPLAIN_MODE_ON SQL_DB2EXPLAIN_SNAPSHOT_MODE_ON
<code>db2_info_acctstr</code>	Character string
<code>db2_info_applname</code>	Character string
<code>db2_info_userid</code>	Character string

db2_info_wrkstnname	Character string
db2_longdata_compat	Boolean
db2_quiet_mode	Integer
db2_sqlerrp	Character string (read only)
db2_txn_isolation	One of the following: SQL_TXN_READ_UNCOMMITTED SQL_TXN_READ_COMMITTED SQL_TXN_REPEATABLE_READ SQL_TXN_SERIALIZABLE SQL_TXN_NOCOMMIT

As in the `Mysql.pm` connect example, the DBI connect statement returns a database handle if the connection has been successfully established. Otherwise, the value `undef` is returned. All further communication with the database server takes places through this object.

SELECT query statements

Example 7-19 and Example 7-20 show the differences between the two interfaces in using the SELECT statement.

Example 7-19 Select statement used with the Mysql.pm interface

```
use Mysql;
my $host="localhost";
my $database="itsodb";
my $user="itsosj";
my $password="itsosj";
$dbh = Mysql->connect($host,$database,$password,$user);

$sql_statement = "SELECT * FROM catalog WHERE id='$id'";
$sth = $dbh->query($sql_statement);
@arr = $sth->fetchrow;
```

A DB2 conversion using the DBI interface is shown in Example 7-20.

Example 7-20 Select statement used with the DBI interface

```
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
my $user="itsosj";
my $password="itsosj";
$dbh = DBI->connect("dbi:DB2:itsodb2", $username, $password);

$sql_statement = "SELECT * FROM katalog WHERE id='$id'";
$sth = $dbh->prepare($sql_statement);
$sth->execute();
```

```
@arr = $sth->fetchrow_array;
```

The biggest difference is that in case of DBI, the statement first has to be prepared before it is executed. Two statements are needed for this functionality.

The MySQL code `fetchrow` and the respective DB2 code `fetchrow_array` functions return the next row of data from the statement handle generated by the query or execute commands.

INSERT, UPDATE, and DELETE statements

Generally, the same functions can be used for the INSERT, UPDATE and DELETE statements as for the SELECT statement discussed above. For the non-SELECT statement, DBI provides a faster substitute for `DBI::prepare/DBI::execute` pair with the `DBI::do` function:

```
$rows_affected = $dbh->do($sql_statement);
```

For further information, please refer to the DBI documentation previously.

Disconnecting database

The `Mysql.pm` interface does not provide a disconnect function, because MySQL does not support transactions yet. Whenever the database handle loses its value, `Mysql.pm` closes the database connection. However, databases that do support transactions need to be explicitly disconnected. The DBI interface supports the disconnect function for the database handle.

Example 7-21 Disconnecting DB2 database using DBI

```
use DBI;
use DBD::DB2::Constants;
use DBD::DB2;
my $user="itsosj";
my $password="itsosj";
$dbh = DBI->connect("dbi:DB2:itsodb2", $username, $password);
...
$dbh->disconnect;
```

7.2.2 Converting MySQL PHP applications to DB2 UDB

With PHP applications, different approaches can be used to access DB2:

- ▶ iODBC with ODBC-compliant drivers

iODBC is provided by OpenLink Software. iODBC is the acronym for Independent Open DataBase Connectivity, an Open Source platform

independent implementation of both the ODBC and X/Open specifications. Information about this library can be found at: <http://www.iodbc.org>

▶ unixODBC

unixODBC (<http://www.unixodbc.org>) is an open source project whose goals are to develop and promote unixODBC to be the definitive standard for ODBC on the Linux platform.

▶ Unified ODBC

Unified ODBC consists of a single set of ODBC functions provided by PHP to access different databases that have borrowed the semantics of ODBC API to implement their own API. It is more efficient to use the database native database driver because there is no ODBC involved in the communication or access path.

▶ ADOdb

ADOdb library provides a wrapper around MySQL API for supporting MySQL database, and wrapper around DB2 ODBC Driver for DB2 UDB. So you need the Unified ODBC support for using ADOdb libraries.

▶ PEAR

PEAR DB is the default database abstraction library included in PHP4, and it is quite similar to the more popular ADOdb library. As PEAR DB is also a wrapper around the DB2 ODBC driver, the Unified ODBC DB2 support must be provided for using PEAR DB libraries.

When we talk about ODBC in this section (always Unified ODBC), respectively the native DB2 driver is meant. Because of the wide similarities in the syntax between Unified ODBC and other ODBC types, and the performance advantage when using the Unified ODBC support, the application conversion is discussed with the Unified ODBC support.

Converting from native MySQL library to Unified ODBC

Using the Unified ODBC support in PHP applications does not require a special load of library files as the support already is integrated during the compilation process of PHP. A complete overview about the MySQL and the Unified ODBC functions can be found in the *PHP Manual*, which can be downloaded at: <http://www.php.net/docs.php>

Connecting database

Connecting a MySQL database consists of two parts: First a connection to the MySQL server has to be established and after that a database can be chosen.

The function specified to connect the MySQL server is given in the following declaration:


```
resource mysql_connect ( [string server [, string username [, string password  
[, bool new_link [, int client_flags]]]])
```

Another way to connect to a MySQL server is to use the `mysql_pconnect()` function, which acts very much like `mysql_connect()` with two major differences:

- ▶ When connecting the first time, the function would try to find a (persistent) link that is already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.
- ▶ The connection to the MySQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use.

The *server* variable in the `mysql_connect()` function contains the hostname or the IP address of the server that provides the MySQL database.

Choosing the MySQL database is done by the `mysql_select_db()` function:

```
bool mysql_select_db ( string database_name [, resource link_identifier])
```

The connection part of our sample application using MySQL database is shown in Example 7-22.

Example 7-22 Connecting a MySQL database

```
$db_host = "localhost";  
$db_user = "itsosj";  
$db_pass = "itsosj";  
$datab = "itsodb";  
  
$db = mysql_connect($db_host,$db_user,$db_pass)  
      or die("Could not connect ".mysql_error());  
mysql_select_db($datab,$db) or die(mysql_error());
```

When connecting a DB2 database with ODBC, the connection is done in a single ODBC command (`odbc_connect()`).

```
resource odbc_connect ( string dsn, string user, string password [, int  
cursor_type])
```

Note: As we use the IBM DB2 native driver, the DSN is the database name which is registered in the DB2 catalog. Therefore, no server address has to be declared in the connect statement.

The migrated connection part of the sample application is shown in Example 7-23. In our connection script snippet, the statement after connection statement sets the current schema, which will be used when querying the

database. The background for this approach is that compared to DB2, MySQL does not have schemas, nor instances.

In MySQL, a table can be referenced by *database.tablename*. If no database name is provide in the table name, the default is the currently used database. In DB2, table name is defined as *schema.tablename*. When reference to a DB2 table, if no schema is provided, the default schema is the user ID which is connected to the database. In an application, the `set schema` statement can be used to provide a global schema name for the tables which do not have a full table name specified. To simplify the application conversion, during database porting the tables were created with the schema of the MySQL database name (in our case *itsodb*). By using the `set schema` after database connection, we do not need to change every table name in the application.

Example 7-23 Connecting a DB2 database

```
$db_user = "itsosj";
$db_pass = "itsosj";
$datab   = "itsodb";
```

```
$db=odbc_connect($datab,$db_user,$db_pass) or die("Could not connect ".
odbc_errormsg());
$query="SET CURRENT SCHEMA='ITSODB'";
odbc_exec($db,$query) or die(odbc_errormsg($db));
```

A complement function to the persistent connect function `mysql_pconnect()` is the `odbc_pconnect()`.

SELECT query statements

Querying a table in MySQL using the PHP MySQL functions is shown in Example 7-24.

Example 7-24 MySQL select example

```
$sql = mysql_query("SELECT * FROM catalog WHERE id='$id'");
$rowsql = mysql_fetch_row($sql);
$graphic = "../graphic/".$rowsql[11].".jpg";
```

The correspondent ODBC DB2 query is shown in Example 7-25.

Example 7-25 DB2 select example

```
$sql = odbc_exec($db,"SELECT * FROM catalog WHERE id=$id");
odbc_fetch_into($sql,$rowsql);
$graphic = "../graphics/".$rowsql[11].".jpg";
```

The `odbc_exec()` function nearly corresponds to the `mysql_query()` function. The only difference is that the `odbc_exec()` function needs two parameters where the first parameter is the *connection id* returned by the `odbc_connect` statement:

```
resource odbc_exec ( resource connection_id, string query_string)
```

To get each row by row in an array equivalent to the `mysql_fetch_row()` function the `odbc_fetch_into()` function can be used without great modifications. There are only syntax differences in both statements in our case.

A problem can occur when converting SQL statements with variables to ODBC. In our MySQL example above, the *\$id* variable is in single quote (') whereas in the DB2 example the single quotes are removed. The background for the change is that the table column ID is of type integer, and ODBC does not convert strings to integer automatically whereas in the MySQL function it does.

Attention: When using ODBC functions, variable contents in a SQL statement are *not* converted to the correspondent type of the database column. ODBC expects that the variable in the SQL statement has the correct data type defined.

INSERT, UPDATE, and DELETE statements

In fact for all three SQL commands, INSERT, UPDATE, and DELETE the same MySQL and ODBC functions are used as discussed for the SELECT statement:

```
mysql_query($sql_statement)
odbc_exec($db,$sql_statement)
```

In Example 7-26 and Example 7-27, the difference between the MySQL and ODBC functions for the INSERT statement is shown.

Example 7-26 MySQL insert statement

```
$ret=mysql_query("INSERT INTO warenkorb (user_id, ktg_id) VALUES
('$user_id','$ktg_id')");
if ($ret){
    $text="Item ".$artnr." was inserted in the shopping cart";
}else{
    $text="Insert failure of item ".$artnr." into the shopping cart";
}
echo $text;
```

Example 7-27 DB2 insert statement

```
$ret=odbc_exec($db,"INSERT INTO warenkorb (user_id, ktg_id) VALUES
('$user_id','$ktg_id')");
if ($ret){
```

```
    $text="Item ".$artnr." was inserted in the shopping cart";
}else{
    $text="Insert failure of item ".$artnr." into the shopping cart";
}
echo $text;
```

There is a minor difference between these functions in returning values:

- ▶ `mysql_query()` returns a resource identifier only for `SELECT`, `SHOW`, `EXPLAIN` or `DESCRIBE` statements, and returns a `false` if the query was not executed correctly. For other type of SQL statements, `mysql_query()` returns `true` on success and `false` on error.
- ▶ `odbc_exec()` returns a result identifier if the SQL command was executed successfully and `false` if an error occurs.

Disconnecting database

Example 7-28 and Example 7-29 show the disconnect functions using the MySQL library and the ODBC library.

Example 7-28 Disconnecting MySQL database

```
$db = mysql_connect($db_host, $db_user, $db_pass)
or die("Could not connect: " . mysql_error());
echo "Connected successfully";
mysql_close($db);
```

Example 7-29 Disconnecting DB2 database

```
$db=odbc_connect($datab,$db_user,$db_pass) or die("Could not connect ".
odbc_errormsg());
echo "Connected successfully";
odbc_close($db);
```

Note: Using `mysql_close()` or `odbc_close()` is usually not necessary when databases are used that do not support transactions as non-persistent open links are automatically closed, and the associated memory is freed at the end of the script's execution.

There is a difference between both functions in returning values. The function `mysql_close()` returns `true` on success and `false` on failure. The function `odbc_close()` does not return any value. In rare cases the return value of the `mysql_close()` function is not used at all (see note above), therefore the conversion if necessary at all can mostly be done by simply replacing the function or inserting the new function at the end of program execution.

Converting MySQL ADOdb PHP applications to DB2 UDB

Active Data Object Database (ADOdb) for PHP is a database abstraction library modeled on Microsoft's ADO for building dynamic Web pages with database support. A PHP Web application can use this database class library to access any supported database. ADOdb supports MySQL, PostgreSQL, Oracle, Microsoft SQL Server, Sybase, Informix, Access, FoxPro, Interbase, ODBC, and ADO for true cross-platform database independence, which means a PHP application written for one database can be easily converted to another database.

As both MySQL and DB2 UDB are supported by ADOdb for the PHP library, the application conversion will be much easier than other incompatible APIs. ADOdb architecture is built in such a way that all the database dependant code are stored in stub functions, which means you can easily support your database specific functionality by changing the stub file.

So, for using DB2 UDB from your PHP application you need *adodb-db2.inc.php* instead of *adodb-mysql.inc.php*. As ADOdb function calls will remain the same for both the databases, the only change you may need is changing some database specific formats.

The ADOdb library provides a wrapper around MySQL API for supporting the MySQL database and wrapper around DB2 ODBC Driver for DB2 UDB. So, you need DB2 ODBC Driver for using ADOdb libraries.

The MySQL ADOdb PHP application can be ported to DB2 UDB by doing the following changes at various levels of the application:

► Loading ADOdb driver

Before using the ADOdb library you need to include PHP ADOdb libraries and initialize ADOdb variables in your program. ADOdb requires loading of at least two files:

- *adodb.inc.php*, which contains all the functions used by all database classes
- *adodb-<dbname>.inc.php*, which contains database specific code

For instance, MySQL code requires the loading of *adodb-mysql.inc.php*, and now you need to change it to a *adodb-db2.inc.php* file. This can be done by changing `ADOLoadCode('mysql')` to `ADOLoadCode('db2')`; this code actually makes sure that you load only the required driver. Example 7-30 shows the code for initialization and the changes required for DB2 UDB.

Example 7-30 Loading ADOdb driver for MySQL and DB2 UDB

```
include('/home/itso/adodb/adodb.inc.php');
$ADODB_CACHE_DIR = '/home/itso/adodb/cache';
```

```
// Loads adodb-mysql.inc.php for mysql
ADOLoadCode('mysql');
```

```
// Loads adodb-db2.inc.php for db2
ADOLoadCode('db2');
```

► Connecting to database

ADODB uses an object-oriented approach to manage the complexity of handling multiple databases. This is done by passing an input parameter for the database driver name while creating the connection object. The connection instance for DB2 can be created using `ADONewConnection('db2')` or `NewADODConnection('db2')`.

After creating the connection instance, you need to connect to the database using:

– PConnect()

It creates a persistent connection, which is faster because this connection is never closed.

– Connect()

It creates a non-persistent connection, which takes up much fewer resources though, therefore reducing the risk of your database and your Web-server becoming overloaded.

– NConnect()

It forces the creation of a new connection.

Example 7-31 and Example 7-32 show the changes in connection code in the ADODB PHP program. All the above connect methods take four parameters (systemname, username, password, databasename) in MySQL, whereas in DB2 UDB ODBC, it only requires three parameters (dbname, username, password). Also, we can set the cursor mode to `SQL_CUR_USE_ODBC` for optimization purposes.

Example 7-31 MySQL creating connection in ADODB

```
// create connection instance for MySQL
$db = &ADONewConnection('mysql');
$db->Connect('localhost', 'itso', 'itso', 'itsodb');
```

Example 7-32 DB2 creating connection in ADODB

```
// create connection instance for DB2 UDB
$conn = &ADONewConnection('db2');
$conn->curMode = SQL_CUR_USE_ODBC;
$conn->Connect("itsodb2","itso","itso");
```

```
);
```

► Processing query

Now the connection to the database is established and this connection can be used to execute the query. This can be done by calling the `ADOConnection.Execute()` function. The function calls for DB2 UDB and MySQL remains the same, so they do not require any changes. Whereas the SQL statements may require change due to the SQL difference in MySQL and DB2 UDB.

– Executing the SELECT query and fetching results

The query execution and result fetching code remains the same for MySQL and DB2 UDB. The Example 7-33 shows the code for query execution, and the result fetching using the ADOdb library for PHP.

Example 7-33 Select query execution

```
$recordSet = $conn->Execute('select * from katalog');
if (!$recordSet)
    print $conn->ErrorMsg();
else{
    while (!$recordSet->EOF) {
        print 'HERSTELLER= '.$recordSet->fields[0]. ' Model=
'.$recordSet->fields[1];
        $recordSet->MoveNext();
    }
    $recordSet->Close();
}
$conn->Close();
```

– Executing UPDATE/INSERT statement

The code for executing UPDATE and INSERT remains the same. The code for inserting the data is shown in Example 7-34.

Example 7-34 Executing update/insert statement

```
$sql = "INSERT INTO katalog (id, model,type) values (1,'1001','new)";
$conn->Execute($sql);
print "Number of rows affected =" . $conn->Affected_Rows() ;
$conn.Close();
```

Converting MySQL PEAR DB PHP applications to DB2 UDB

Another means to provide a database abstraction in the PHP application is using the PHP extension and add-on repository for the database (PEAR DB). PEAR DB is the default database abstraction library included in PHP4. It is quite similar to more the popular library ADOdb functionally, but the syntax for both is quite different.

Just like ADOdb, it enables the PHP application to be written once and run with minor changes for various database engines such as MySQL, Oracle, PostgreSQL, or DB2 UDB.

To provide database abstraction features, PEAR DB uses an object-oriented methodology such as classes and objects. For doing so it uses three files:

- ▶ `DB.php`
This class contains implementation of the DB class, which creates database connection objects.
- ▶ `common.php`
This class contains generic code for all the database specific implementation of the database access methods. All the drivers inherit and overload this code.
- ▶ `<drivername>.php`
This file contains database specific implementation of the access code. This class is also called the driver class and is used by PHP applications. In case of MySQL, the driver file is *mysql.php*, whereas in the case of DB2 it is *odbc.php*.

The way PEAR DB provides its services for a DB2 UDB application and MySQL application is quite different. In MySQL PEAR DB uses the currently existing extension for MySQL database, whereas PEAR DB does not provide any extension for connecting to DB2 UDB, so the ODBC extension is used for the DB2 UDB connection.

The PHP application that uses PEAR DB for database connection generally performs the following steps:

- ▶ Include PEAR DB objects
As discussed before PEAR DB requires three files *DB.php*, *common.php* and *<drivername>.php* for proper functioning of the PHP application. But you need not explicitly include all these files, you just need to include *DB.php* and its functions will include the required driver depending upon your data source name.
So for both MySQL and DB2 UDB the include statements will be same as:

```
include('/home/itso/php/PEAR/DB.php');
```
- ▶ Connect to database
PEAR DB database connection can be established using connect method of declared in *DB.php*. This method requires data source name (DSN) containing in connection parameters as input. The DSN is a URL-style string that consists of the database driver name, the user name, the password, the

hostname (for non-ODBC), and a database name. So complete DSN syntax would be

```
driver name://username:password@hostname/databasename
```

PEAR DB has a database driver for MySQL called *mysql*, it can be used to create a connection. So DSN for connecting to our database would be

```
mysql://itso:itso@localhost/itsodb
```

PEAR DB does not provide a driver for DB2 UDB. But it supports DB2 UDB using ODBC driver. SO DSN for DB2 UDB would be

```
odbc(db2)://itso:itso/itsodb2
```

You can optionally provide second parameter to `connect()` which specifies whether you want your connection to be persistent or non-persistent.

Example 7-35 shows the PEAR DB database connect for MySQL and the DB2 conversion code.

Example 7-35 PEAR DB database connection for DB2 UDB and MySQL

```
include('/usr/local/lib/php/PEAR/DB.php');

\\ MySQL database connection with DSN='mysql://itso:itso@localhost/itsodb2'
$conn = DB::connect('mysql://itso:itso@itsodb2');

\\ DB2 UDB database connection with DSN='odbc(db2)//itso:itso/itsodb2'
$conn = DB::connect('odbc(db2)://itso:itso@itsodb2');

if(DB::isError($conn))
    print "connection error : ".$conn->getMessage();
```

► **Process query**

The query can be executed by calling `query()` method on the connection object. It takes the SQL statement as an input parameter and returns the results. The results can be used in three modes depending on the SQL statement and statement's final status:

– **Error case**

In case an error occurs while executing the SQL statement you can use return object for getting error message using the method `getMessage()`. This code does not change with the change in database, so MySQL and DB2 UDB code is same. Example 7-36 shows the usage of the result object for getting the error message.

– **Result set**

If your SQL statement was a query statement, the return object can be used to fetch the result set using the `fetchRow` variable of the return object. Again, MySQL and DB2 UDB database differences do not cause any

change in the PEAR DB code. So this code require only changes in the SQL statements. Example 7-36 shows an PHP query statement execution with PEAR DB.

Example 7-36 PEAR DB PHP query statement

```
$result = $conn->query ("SELECT * FROM itsodb.katalog");
if (DB::isError ($result))
    die ("SELECT failed: " . $result->getMessage () . "\n");
while ($row = $result->fetchRow ())
    print "HERSTELLER= ".$row[0]. "Model=".$row[1];
$result->free
```

– Update report

In the cases of an update and insert statement, return objects will be *DB_ok*. In this case the connection object can be used to get the affected row using the method `affectedRows()`. This code also remains the same for MySQL and DB2 UDB. Example 7-37 shows data insertion using PEAR DB.

Example 7-37 PEAR DB PHP data manipulation statement

```
$result = $conn->query ("INSERT INTO katalog (id, model,type) values
(1,'1001','new')");
print "Number of rows affected=".$conn->affectedRows();
```

– Disconnect from the database

Finally, as a good programming practice you should release all the `ResultSets` and connections. This can be done using `free()` method of the `result` object of `query()` in cases of SQL query statements. Connection can be released using `disconnect()` method of the connection object.

7.2.3 Converting MySQL Java applications to DB2 UDB

DB2 UDB supports the usage of Java programming at the following levels:

- ▶ DB2 server side programming
 - Java stored procedure on DB2 server
 - Java user-defined function (UDF) on DB2 server
- ▶ Java applications
 - Java enabled Web browser accessing DB2 UDB using JDBC
 - Java standalone application using JDBC and SQLj
- ▶ J2EE application server
 - JavaServer Pages (JSP) using a JDBC connection

- Servlets using JDBC or SQLj
- Enterprise JavaBeans (EJB) using JDBC or SQLj

DB2 UDB provides an implementation of the two standard-based Java programming interface (APIs), Java Database Connectivity (JDBC), and embedded SQL for Java (SQLj). This section provides an overview of JDBC, SQLj, and the conversion of existing MySQL Java applications to DB2 UDB.

MySQL has an optional package MySQL Connector/J, which is a type 4 JDBC driver. The latest Version of MySQL Connector/J implements SUN's JDBC 3.0 API for relational database access.

Java database connectivity (JDBC)

JDBC is a vendor-neutral dynamic SQL interface that provides data access for your application through standardized Java methods. JDBC drivers provide the mechanics to the JDBC API to allow Java applications to access databases. Currently JDBC API is in its third revision, but most of the drivers available in market comply to JDBC1.2 or JDBC2.0 specification. IBM DB2 V8.1 supports JDBC2.1 and some methods support JDBC3.0 forms.

A JDBC application can establish a connection to a data source using the JDBC DriverManager interface. In the following sections we discuss what are the changes required in Java application code from MySQL to DB2.

IBM JDBC driver for DB2

IBM DB2 UDB provides support for various types of JDBC technology-enabled drivers. The drivers available in DB2 UDB V8.1 are:

► **DB2 JDBC type 2 driver**

The DB2 JDBC type 2 driver, also called the *native-API/partly Java driver*, lets Java applications make JDBC calls that are translated to Java native methods. The Java applications that use this driver must run a DB2 client, which is used to communicate the JDBC requests to the DB2 server. Figure 7-1 shows a call transfer for DB2 JDBC type 2 driver. As shown in Figure 7-1, this driver can be used only by Java applications. This driver is implemented using the DB2 CLI interface to communicate with DB2 UDB servers.

For using DB2 JDBC type 2 driver you need following properties:

```
drivername="COM.ibm.db2.jdbc.app.DB2Driver"  
URL="java:db2:dbname"
```

The user ID and password are implicitly picked from the DB2 client setup.

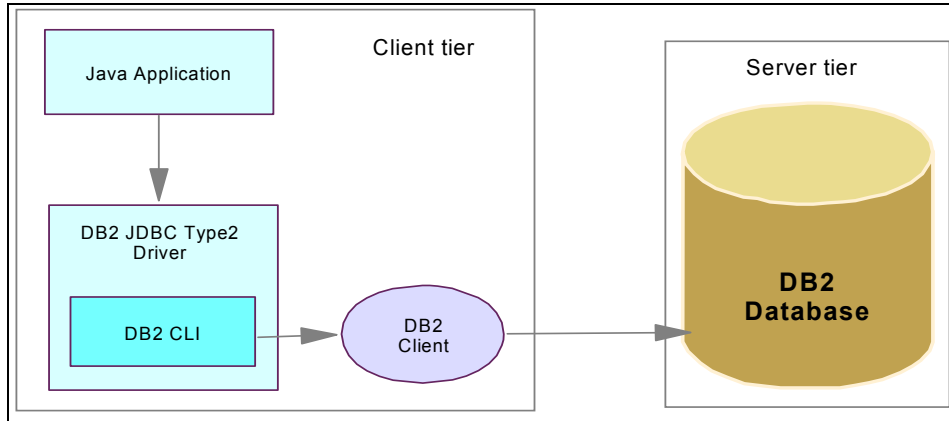


Figure 7-1 DB2 type 2 JDBC Driver

Note: The DB2 JDBC type 2 driver is J2EE certified for use with IBM WebSphere Application Server, which means it conforms to J2EE specifications.

► DB2 JDBC type 3 driver

The DB2 JDBC type 3 driver also called the *applet or net-protocol/all-Java driver* and follows a three-tiered approach. In the DB2 JDBC type 3 driver, the Java standalone application or Java applet passes the JDBC database requests through the network to the middle-tier server where the DB2 applet server is running. The DB2 applet server then translates the request (directly or indirectly) to the database-specific native-connectivity interface to pass the request to the database server. Figure 7-2 shows the various tiers involved in DB2 JDBC type 3 driver. The DB2 JDBC server process is *db2jd* and the driver is available in the *db2java.zip* in directory *sllib/java*.

For using DB2 JDBC type 3 driver you need following properties:

```

drivername="COM.ibm.db2.jdbc.net.DB2Driver"
URL="java:db2://servername:serverport/dbname"
  
```

Note: Here servername and serverport are the applet server port.

Note: The JDBC type 3 driver is deprecated for DB2 UDB Version 8.

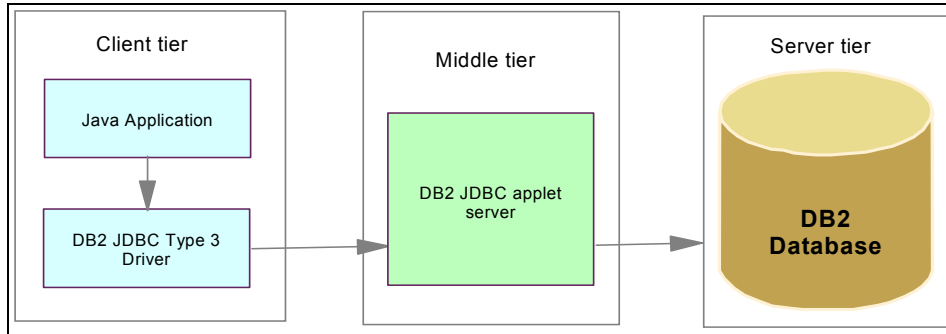


Figure 7-2 DB2 JDBC type 3 driver

► DB2 Universal JDBC driver

The DB2 JDBC Universal Driver is a single driver that include both type 2 and type 4 behavior. This type 4/ Native-protocol all-Java driver is implemented in Java and it uses the Distributed Relational Database Architecture™ (DRDA®) protocol for client/server communications. Figure 7-3 shows the JDBC Universal Driver usage in the Java application. DB2 Universal JDBC Driver does not need any service on client side, this driver is available in the *db2jcc.jar*.

For using the DB2 Universal JDBC driver, you need the following properties:

```
drivername="com.ibm.db2.jcc.DB2Driver"
URL="java:db2://servername:serverport/dbname"
```

Note: Servername and serverport refer to the database server.

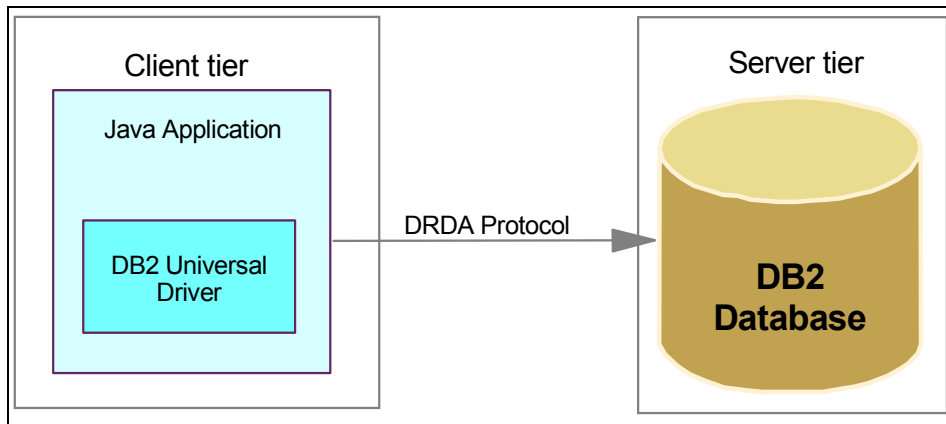


Figure 7-3 JDBC Universal Driver

The JDBC type 1 driver /JDBC-ODBC bridge provided by Sun Microsystem can be used for DB2 UDB. IBM does not guarantee or recommend this driver.

You can use any type of driver according to your requirement, but we suggest that you directly convert your MySQL Java application to DB2 Java application by using DB2 Universal JDBC driver instead of MySQL JDBC Driver.

For more information on Java application development and the JDBC specification, see the DB2 Universal Database Java Web site at:
<http://www.ibm.com/software/data/db2/udb/ad/v8/java/>

Embedded SQL for Java

DB2 UDB provides embedded SQL (both static and dynamic) access to Java applications through SQLj APIs. DB2 SQLj support allows you to create, build, and run embedded SQL for Java applications, applets, stored procedures, and user-defined functions (UDFs). SQLj requires *sqllib/java/sqlj.zip* and *sqllib/java/runtime.zip*.

Conversion of JDBC application

As both MySQL and DB2 UDB comply to JDBC specification, the Java application does not require a lot of code changes. It might be required to change SQL statements as discussed in DDL differences in Chapter 5, “Database porting” on page 89.

In this section we provide you information on the Java program conversion from MySQL to DB2 UDB.

Loading JDBC Driver

First step in the Java program is loading of the appropriate JDBC driver. This is done by calling *Class.forName(drivername)* in your Java program. Appropriate values for the driver name are already discussed in “IBM JDBC driver for DB2” on page 189. Example 7-38 shows loading of the driver for MySQL, and Example 7-39 shows the DB2 UDB conversion using Universal driver.

Connecting to database

In this part the Java program tries to establish a connection to the given database. This is done by calling *DriverManager.getConnection* with the proper URL values, which was discussed in the driver description in “IBM JDBC driver for DB2” on page 189. After this call DriverManager selects the appropriate driver from set of registered drivers, which actually connects to the database. Example 7-38 and Example 7-39 show these steps for MySQL and DB2 UDB respectively.

Example 7-38 MySQL JDBC driver loading and connection

```
import java.sql.*;

public class MySQLClient
{
    public static void main (String[] args) throws Exception
    {

        // load the driver
        Class.forName ("com.mysql.jdbc.Driver");

        // connect to database
        Connection conn = DriverManager.getConnection
("jdbc:mysql://localhost/itsodb", "userid", "passwd");

        //...

    }
}
```

Example 7-39 DB2 JDBC driver loading and connection

```
import java.sql.*;

public class DB2Client
{
    public static void main (String[] args) throws Exception
    {

        // load the driver
        Class.forName ("COM.ibm.db2.jdbc.net.DB2Driver");

        // connect to database
        Connection conn = DriverManager.getConnection
("jdbc:db2://localhost/itsodb2", "userid", "passwd");

        //...

    }
}
```

Calling query statement

As both DB2 and MySQL follow the SQL standard and the JDBC specification, the query execution code will not change much. Once a connection is

established, most of the time the only code which change is either SQL statements or return data types.

The JDBC API does not put any restrictions on the kinds of SQL statements that can be executed though JDBC, so it is the application's responsibility to pass SQL statements compatible to the database used. The connection obtained in Example 7-38 and Example 7-39 can be used for one of the following three types of statements depending upon the requirements:

► **Statement** - Simple single SQL statement

The statement can be created by using the *createStatement* method of the Connection. Example 7-40 shows the usage of *executeQuery* with a change for MySQL and DB2 UDB. As we can see in the example the only change in this code is the SQL statement.

Example 7-40 Query statement changes from MySQL to DB2 UDB

```
Statement s = conn.createStatement();

//MySQL statement
s.executeQuery ("SELECT warenkorb.wk_id, warenkorb.user_id,warenkorb.status
FROM warenkorb");

//DB2 UDB statement
s.executeQuery ("SELECT wk_id, user_id,status FROM itsodb.warenkorb");

ResultSet rs = s.getResultSet ();
while (rs.next ())
{
    int id = rs.getInt ("wk_id");
    int uid = rs.getInt("user_id");
    String status = rs.getString ("status");
    System.out.println (
        "id = " + id
        + ", user = " + uid
        + ", status = " + status);
}
rs.close ();
s.close ();
```

► **PreparedStatement** - Precompiled SQL statements

PreparedStatements are used to effectively execute the statement multiple times. Example 7-41 shows the conversion of the prepared statements. Setting appropriate values of host variables is very important while using prepared statements. In the section titled “Java, JDBC, and SQL data type conversions” on page 197 we discuss more about it.

► **callableStatement** - Statement to call stored procedures (only in DB2 UDB)

Calling insert/update/delete statement

Any statement that updates the database or inserts and deletes a value in a database can be executed using the *executeUpdate* or *execute* method of the statement. Example 7-41 shows the update of the record using the prepared statement for MySQL and conversion for DB2 UDB. The change is due to the multiple MySQL database that have been merged into one DB2 database and grouped by the table schema. If you use **set schema= schema** for a connection, then nothing needs to be changed.

Example 7-41 MySQL Prepared statement and executeUpdate

```
// MySQL prepared statement
PreparedStatement s = conn.prepareStatement("update warenkorb set anzahl=?,
,status=? where wk_id=65;");

//DB2 UDB changes for prepared statement
PreparedStatement s = conn.prepareStatement("update itsodb.warenkorb set
anzahl=?, ,status=? where wk_id=65;");

s.setInt (1, 10);
s.setString (2, "working");
int count = s.executeUpdate ();
System.out.println (count + " rows were inserted");
```

Using connection pooling and data source

The JDBC 2.0 Standard Extension API provides the data source interface as an alternative to the DriverManager for establishing a connection. When a DataSource class has been implemented appropriately, a DataSource object can be used to produce Connection objects that participate in connection pooling. You can either create a data source in your program or you can get an existing data source from the context.

MySQL provides MysqlDataSource, which is a data source implementation of MySQL data source. The MySQL data source can be created by calling a default constructor com.mysql.jdbc.jdbc2.optional.MysqlDataSource as shown in Example 7-42.

Example 7-42 MySQL datasource creation and binding

```
com.mysql.jdbc.jdbc2.optional.MysqlDataSource mysqlDs = new
com.mysql.jdbc.jdbc2.optional.MysqlDataSource();
mysqlDs.setServerName("localhost");
mysqlDs.setDatabaseName("itsodb");
mysqlDs.setUser("itso");
mysqlDs.setPassword("itso");
mysqlDs.setPortNumber(3306);
Context ctx = new InitialContext(env);
```

```
ctx.bind("jdbc/itsodb", mysqlDs);
```

On the other hand, the DB2 UDB Universal JDBC Driver and DB2 UDB type 2 JDBC driver provide a number of data source implementations. Depending on your application requirement, you can use one of the following data sources with DB2 UDB Universal JDBC Driver:

- ▶ com.ibm.db2.jcc.DB2SimpleDataSource
- ▶ com.ibm.db2.jcc.DB2DataSource
- ▶ COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
- ▶ COM.ibm.db2.jdbc.DB2XADataSource

DB2 UDB type 2 driver provides the following data source implementations:

- ▶ COM.ibm.db2.jdbc.DB2DataSource
- ▶ COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
- ▶ COM.ibm.db2.jdbc.DB2XADataSource

The DB2 UDB data source can be created in a similar fashion by initializing one of the above data sources as shown in Example 7-43. Also, the same example shows deployment of the data source in context for further use.

Example 7-43 DB2 UDB datasource creation and deployment

```
DB2SimpleDataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();

db2ds.setDatabaseName("itsodb2");
db2ds.setDescription("Itso Sample Database");
db2ds.setUser("itso");
db2ds.setPassword("itso");

Context ctx=new InitialContext();
Ctx.bind("jdbc/itsodb",db2ds);
```

Once the data source is bound, subsequent JDBC programs can use the same data source for creating the JDBC connection. The MySQL and DB2 UDB programs for creating the connection remain the same. Example 7-44 shows how to get a data source from InitialContext and how to get a connection from the data source.

Example 7-44 MySQL and DB2 UDB datasource from context

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/itsodb");// Only logical
datasource name can change in this program.
Connection con = ds.getConnection();
```

Java, JDBC, and SQL data type conversions

In this section we discuss about the way MySQL and DB2 UDB handles column type to Java data type conversions. As we saw in 5.1, “Data type mapping” on page 90, MySQL and DB2 UDB have different data types. As you migrate your Java application you should check the Java data types used to fetch your data using JDBC. The JDBC driver converts the data exchanged between the application and the database using the specified schema mapping, which is defined by both DB2 UDB and MySQL for their data types.

Table 7-8, shows MySQL to Java data types. Since MySQL does not enforce a strict type conversions, the Java programmer has to take care of data loss because of round off, overflow, or precision loss.

Table 7-8 MySQL data type mapping to Java data type

MySQL column type	Java type	Description
CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET	java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Blob, java.sql.Clob	All these columns types can be converted to any of these java data types.
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	java.lang.String, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.math.BigDecimal	round-off, overflow or loss of precision may occur if you choose a Java numeric data type that has less precision or capacity than the MySQL data type you are converting to/from.
DATE, TIME, DATETIME, TIMESTAMP	java.lang.String, java.sql.Date, java.sql.Timestamp	Locale specific data should be handled with care.

On the other hand, DB2 UDB sticks to JDBC specification, and provides a default and recommended data type mapping as shown in Table 7-9.

Table 7-9 DB2 UDB data types mapping to Java types

DB2 UDB data type	Java type	Data type description
SMALLINT	short	16-bit signed integer
INTEGER	int	32-bit signed integer
BIGINT	long	64-bit signed integer
REAL	float	Single precision floating point

DB2 UDB data type	Java type	Data type description
DOUBLE	double	Double precision floating point
DECIMAL	java.math.BigDecimal	Packed decimal
CHAR	java.lang.String	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	java.lang.String	Variable-length character string
VARCHAR FOR BIT DATA	byte[]	Variable-length character string
LONG VARCHAR	java.lang.String	Long variable-length character string
LONG VARCHAR FOR BIT DATA	byte[]	Long variable-length character string
BLOB	java.sql.Blob	Large object variable-length binary string
CLOB	java.sql.Clob	Large object variable-length character string
DBCLOB(n)	java.sql.Clob	Large object variable-length double-byte character string
GRAPHIC	java.lang.String	Fixed length double-byte character string
VARGRAPHIC	java.lang.String	Non-null-terminating varying double-byte character string with 2 byte string length indicator
LONG VARGRAPHIC	java.lang.String	Non-null-terminating varying double-byte character string with 2 byte string length indicator

DB2 UDB data type	Java type	Data type description
DATE	java.sql.Date	10-byte character string
TIME	java.sql.Time	8-byte character string
TIMESTAMP	java.sql.Timestamp	26-byte character string

JDBC methods conversion

Some of the methods specified by JDBC specification are not implemented in MySQL, so the functionality of these methods would have been implemented by you in your application code. Now you can use those methods with DB2 UDB. Some of these methods are listed here:

- ▶ Blob.truncate()
- ▶ PreparedStatement.setArray(int, Array)
- ▶ PreparedStatement.setRef()
- ▶ PreparedStatement.getParameterMetaData()
- ▶ ResultSet.getArray(int)
- ▶ ResultSet.getArray(colName)
- ▶ ResultSet.getRef(int)
- ▶ ResultSet.getRef(String)
- ▶ ResultSet.rowDeleted()
- ▶ ResultSet.rowInserted()
- ▶ ResultSet.rowUpdated()
- ▶ ResultSet.updateArray(int, Array)
- ▶ ResultSet.updateArray(String, Array)
- ▶ ResultSet.updateRef(int, Ref)
- ▶ ResultSet.updateRef(String, Ref)

7.2.4 Converting MySQL C/C++ applications to DB2 UDB

DB2 UDB provides the following programming interfaces for developing an application in C/C++:

- ▶ Embedded SQL
- ▶ DB2 Call Level Interface (CLI)

Apart from this, DB2 UDB uses C/C++ for server side programming for creating:

- ▶ Stored procedures on DB2 server
- ▶ User-defined functions (UDF) on DB2 server.

DB2 UDB provides precompilers for C, C++, COBOL, Fortran, REXX, and Java to support embedded SQL applications. Embedded SQL applications support both static and dynamic SQL statements. Static SQL statements require information of all the SQL statements, tables, and data types used at compile

time. The application needs precompile, bind, and compile before execution. In contrast, dynamic SQL statements can be built and executed at runtime. For further details on embedded SQL. For more information refer to *IBM DB2 UDB Application Development Guide: Building and Running Applications V8*, SC09-4825.

Another interface provided by DB2 UDB is DB2 Call Level Interface (CLI). It is a standard based API following the Microsoft's Open Database Connectivity (ODBC) specification and the ISO SQL/CLI standard. Both embedded SQL and DB2 CLI support database administration as well as database manipulation from C/C++ applications.

MySQL provides a client library for accessing a MySQL database from C applications. MySQL C is included in the *mysqlclient* library. It provides features for:

- ▶ Connection mechanism
- ▶ Creation and execution of SQL queries
- ▶ Status and error reporting

MySQL Connector/C++(or MySQL ++) is an additional library for accessing MySQL databases from C++ applications. It is another layer of abstraction on top of the *mysqlclient* library.

Converting applications

MySQL C API and DB2 CLI are quite similar in functionality and mechanisms to access databases. Both use the function call to pass dynamic SQL statements and do not need to precompile. We recommend that you convert MySQL C applications to DB2 CLI. This section describes conversion changes for various levels of the application:

- ▶ Connecting to the server

The first step in converting MySQL C applications is to change the include information, initialize variables, and to replace the MySQL connection with a DB2 connection. Example 7-45 shows a typical MySQL C program to initiate MySQL variables, create a connection, and terminate the connection.

Example 7-45 MySQL C application, initialize MySQL and create connection

```
#include <mysql/mysql.h> /* Include MySQL variable and function definition
*/
MYSQL *connection; /* pointer to connection handler */
int main()
{
    if(mysql_init(MYSQL *mysql)==NULL) /* initiate the mysql variable */
    {
        /* handle error */
```

```

        return 1;
    }
    if(mysql_real_connect (
        connection, /* pointer to connection handler */
        NULL, /* host to connect, default localhost*/
        NULL, /* user name, default local user*/
        NULL, /* password, default none*/
        "itsodb", /* database name*/
        0, /* port */
        NULL, /* socket */
        0 /* flags*/
    ) == NULL)
    {
        /* handle error */
        return 1;
    }
    if(mysql_close (connection)==NULL)
    {
        /* handle error */
        return 1;
    }
    exit(0);
}

```

Figure 7-4 shows a similar task using DB2 CLI. It shows the initialization tasks, which consists of: the allocation and initialization of the environment and connection handlers; creation of the connection; transaction processing; and finally termination of the connection and deallocation of the handlers.

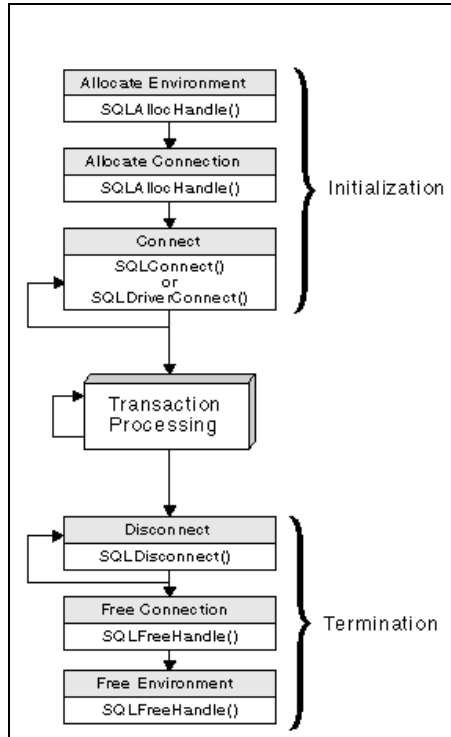


Figure 7-4 DB2 CLI activities

The Example 7-46 shows the implementation of the task defined by the figure.

Example 7-46 DB2 CLI application, connecting to database

```

#include <sqlcli.h> /* Include DB2 CLI variable and function definition */
int main()
{
    SQLRETURN ret = SQL_SUCCESS;
    int rc = 0;
    SQLHANDLE henv; /* environment handle */
    SQLHANDLE hdbc; /* connection handle */

    /* Allocate an environment handle */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    if (ret != SQL_SUCCESS)
    {
        /* handle error */
        return 1;
    }

    /* Allocate a connection handle */
  
```



```

ret = SQLAllocHandle(SQL_HANDLE_DBC, *henv, *hdbc);
if (ret != SQL_SUCCESS)
{
    /* handle error */
    return 1;
}

/* connect to the database */
ret = SQLConnect(*hdbc
                (SQLCHAR *)"itsodb2",
                SQL_NTS,
                (SQLCHAR *)NULL /*user*/,
                SQL_NTS,
                (SQLCHAR *)NULL /* password*/,
                SQL_NTS);
if (ret != SQL_SUCCESS)
{
    /* handle error */
    return 1;
}

/* disconnect from the database */
ret = SQLDisconnect(hdbc);
if (ret != SQL_SUCCESS)
{
    /* handle error */
    return 1;
}

/* free connection handle */
ret = SQLFreeHandle(SQL_HANDLE_DBC, *hdbc);
if (ret != SQL_SUCCESS)
{
    /* handle error */
    return 1;
}

/* free environment handle */
ret = SQLFreeHandle(SQL_HANDLE_ENV, *henv);
if (ret != SQL_SUCCESS)
{
    /* handle error */
    return 1;
}
exit(0);
}

```

► Processing query

A typical MySQL C API program involves three steps in query processing:

- Query construction

Depending upon your requirement you can construct a null terminated string or counted length string for the query:

```
char *query;
```

- Execute the query

For executing the query you can use `mysql_real_query()` for a counted length query string or `mysql_query()` for a null terminated query string. Example 7-47 shows the processing of a query with both `mysql_real_query()` and `mysql_query()` method calls.

- Processing of the returned results

After executing the query, the final step is to process the results. All the statements except select, show, describe, and explain do not return any results; optionally in those query mysql provides `mysql_affected_rows()` for accessing the number of rows effected.

If your query returns a result set, follow these steps for the result processing:

- Generate the result set using `mysql_store_result()` or `mysql_use_result()`.
- Fetch each row using `mysql_fetch_row()`
- Deallocate the result set using `mysql_free_result()`

Example 7-47 shows an example for both MySQL query, which returns results as well as queries that do not return results.

Example 7-47 MySQL query processing

```
MYSQL_RES *result;
if (mysql_query(conn, "SELECT wk_id, user_id,status FROM warenkorb") != 0){
    /* handle error */
    return 1;
}
else
{
    result = mysql_store_result (conn); /* generate result set */
    if (result == NULL){
        /* handle error */
        return 1;
    }else
    {
        /* process result set, then deallocate it */
        MYSQL_ROW row;
        MYSQL_FIELD* fd ;
```

```

while ((row = mysql_fetch_row (res_set)) != NULL)
{
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        if (i > 0)fputc ('\t', stdout);
        if( i == 0 || i== 1)printf ("%i", row[i]);
        if( i == 2)printf ("%s", row[i]);
    }
}
mysql_free_result (res_set);
}

```

On the other hand DB2 CLI provides a more comprehensive set of APIs for doing similar tasks. One of the essential parts of DB2 CLI is transaction processing, which is supported by all the tables in DB2 UDB. Figure 7-5 shows the typical order of function calls of query processing.

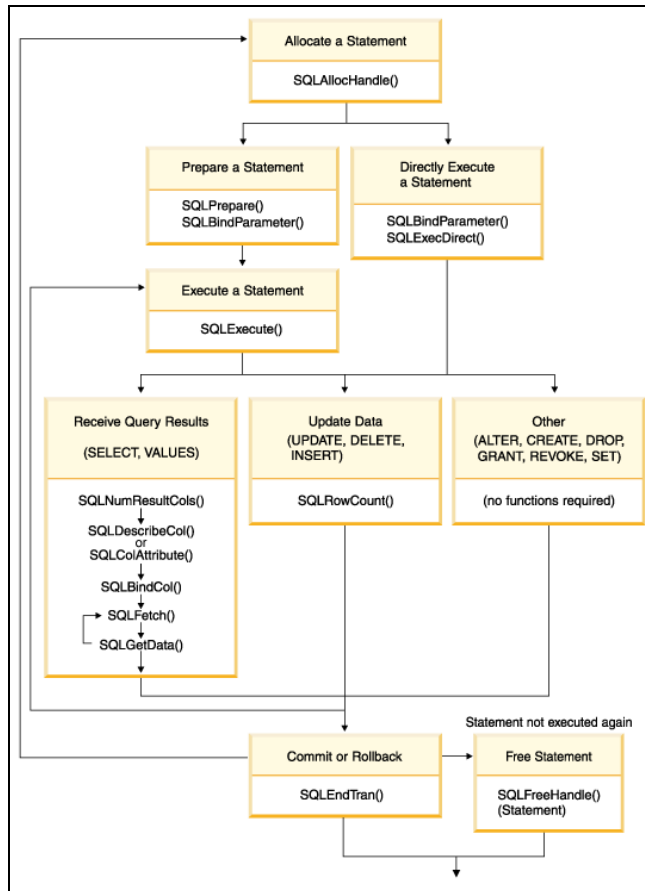


Figure 7-5 DB2 Query processing

DB2 CLI query processing involves the following steps:

a. Allocating statement handle

A statement handle tracks execution of the query for a particular connection. This can be allocated using `SQLAllocHandle()` with a `HandleType` of `SQL_HANDLE_STMT`.

b. Preparing and executing SQL statements

DB2 UDB provides two ways for preparing and executing the query:

- Prepare and execute as separate steps

If you plan to execute the same query multiple times with different parameters, you can use this technique. This involves the following steps: Preparation of query using `SQLPrepare()`, binding of the

parameters using `SQLBindParameter()`, and finally execution of the query using `SQLExecute()`. Example 7-48 shows an example for prepared statements.

- Prepare and execute in a single step

If your query is executed only once, then you can use `SQLExecDirect()` to directly call, prepare, and execute in a single step. Example 7-49 shows the usage of this method.

c. Processing results

Processing query results involves binding application variables to columns of a result set, and then fetching the rows of data into the application variables. This is done by calling `SQLBindCol()` followed by `SQLFetch()` as shown in Example 7-48.

Another way to get data without binding the column is by calling `SQLFetch()` and `SQLGetData()`, this technique is used in Example 7-49.

d. Committing or rolling back

DB2 UDB supports two commit modes: auto-commit and manual commit. This can be set using `SQLSetConnectAttr()` with the parameter `SQL_AUTOCOMMIT_ON` or `SQL_AUTOCOMMIT_OFF`. If a transaction is set to `SQL_AUTOCOMMIT_OFF` it is the programmer's responsibility to end the transaction. This can be done using `SQLEndTran()` to either rollback or commit the transaction using parameter `SQL_COMMIT` or `SQL_ROLLBACK`.

e. Deallocating statement handle

This requires unbinding of the variables, columns, or cursors (if allocated) using `SQLFreeStmt()` with the option of `SQL_CLOSE`, `SQL_UNBIND` or `SQL_RESET_PARAMS`, and then finally calling `SQLFreeHandle()` to deallocate the statement handle.

Example 7-48 DB2 CLI prepared statement with column binding, auto commit on

```
SQLHANDLE hstmt; /* statement handle */

SQLINTEGER wk_id=10;
SQLINTEGER user_id;
SQLCHAR val[15];

/* SQL statements to execute */
SQLCHAR *stmt1 = (SQLCHAR *)"SELECT user_id,status FROM warenkorb where
wk_id=?";

/* set AUTOCOMMIT on */
ret = SQLSetConnectAttr(hdbc,
                        SQL_ATTR_AUTOCOMMIT,
```

```

                                SQLPOINTER)SQL_AUTOCOMMIT_ON,
                                SQL_NTS);
if (ret != SQL_SUCCESS)
{
    /* handle error */
}

/* allocate a statement handle */
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
if (ret != SQL_SUCCESS)
{
    /* handle error */
}

/* prepare the statement */
ret = SQLPrepare(hstmt, stmt, SQL_NTS);
if (ret != SQL_SUCCESS)
{
    /* handle error */
}

/* bind parameter1 to the statement */
ret = SQLBindParameter(hstmt,
                        1,
                        SQL_PARAM_INPUT,
                        SQL_C_UBIGINT,
                        SQL_INT,
                        0,
                        0,
                        &wk_id,
                        0,
                        NULL);

if (ret != SQL_SUCCESS)
{
    /* handle error */
}

ret = SQLBindCol(hstmt,
                 1,
                 SQL_C_UBIGINT,
                 &user_id,
                 0,
                 4);

/* execute the statement */
ret = SQLExecute(hstmt);
if (ret != SQL_SUCCESS)
{

```

```

        /* handle error */
    }

    /* fetch each row and display */
    ret= SQLFetch(hstmt);
    if(ret == SQL_NO_DATA_FOUND)
    {
        printf("No data found");
    }
    while(ret != SQL_NO_DATA_FOUND)
    {
        printf("%i%",user_id);
        ret=SQLFetch(hstmt);
    }

    ret = SQLFreeStmt(hstmt, SQL_UNBIND);
    if (ret != SQL_SUCCESS)
    {
        /* handle error */
    }

    /* free the statement handle */
    ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    if (ret != SQL_SUCCESS)
    {
        /* handle error */
        return 1;
    }
}

```

Example 7-49 DB2 CLI prepare/execute in one step with SQLGetData & manual commit

```

SQLHANDLE hstmt; /* statement handle */

SQLINTEGER wk_id;
SQLINTEGER user_id;
SQLCHAR val[15];

/* SQL statements to execute */
SQLCHAR *stmt1 = (SQLCHAR *)"SELECT wk_id, user_id,status FROM warenkorb";

/* set AUTOCOMMIT on */
ret = SQLSetConnectAttr(hdbc,
                        SQL_ATTR_AUTOCOMMIT,
                        (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
                        SQL_NTS);
if (ret != SQL_SUCCESS)

```

```

{
    /* handle error */
}

/* allocate a statement handle */
ret = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
if (ret != SQL_SUCCESS)
{
    /* handle error */
}

/* execute statement 1 directly */
ret = SQLExecDirect(hstmt, stmt1, SQL_NTS);
if (ret != SQL_SUCCESS)
{
    /* handle error */
}

/* fetch each row and display */
ret= SQLFetch(hstmt);
if(ret == SQL_NO_DATA_FOUND)
{
    printf("No data found");
}
while(ret != SQL_NO_DATA_FOUND)
{
    /* get data from column 1 */
    ret = SQLGetData(hstmt,
                    1,
                    SQL_C_UBIGINT,
                    wk_id,
                    0,
                    4);
    if (ret != SQL_SUCCESS)
    {
        /* handle error */
    }
    /* get data from column 2 */
    ret = SQLGetData(hstmt,
                    1,
                    SQL_C_UBIGINT,
                    user_id,
                    15,
                    4);
    if (ret != SQL_SUCCESS)
    {
        /* handle error */
    }
    ret=SQLFetch(hstmt);
}

```



```

    }
    ret = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK );
    if (ret != SQL_SUCCESS)
    {
        /* handle error */
        return 1;
    }

    /* free the statement handle */
    ret = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    if (ret != SQL_SUCCESS)
    {
        /* handle error */
        return 1;
    }
}

```

For all the applications using the MySQL Connector/C++, you may want to consider converting them to DB2 CLI. The typical conversion process would remain the same as both MySQL C and MySQL C++ use the same flow of the program.

7.2.5 Converting MyODBC applications to DB2 UDB

MySQL supports the ODBC database API to connect to a MySQL database server using the optional product called MySQL Connector/ODBC (also known as MyODBC). MyODBC supports ODBC at two levels: MyODBC 2.5 supports ODBC 2.5x and MyODBC 3.5x is a 32-bit ODBC Driver supporting ODBC 3.51 specification.

As DB2 CLI is also based on the ODBC specification, and you can build ODBC applications without using any ODBC driver manager, so the application conversion is quite easy. All you need to do is use DB2's ODBC driver by linking your application with *libdb2*. The DB2 CLI driver also acts as an ODBC driver when loaded by an ODBC driver manager. DB2 CLI conforms to ODBC 3.51.

Figure 7-6 shows the MySQL driver and DB2 ODBC driver in the ODBC scenario; it shows that an application written for one ODBC driver can easily be converted to another driver. It also shows various components involved in the ODBC application and how they are mapped from MyODBC to DB2 ODBC.

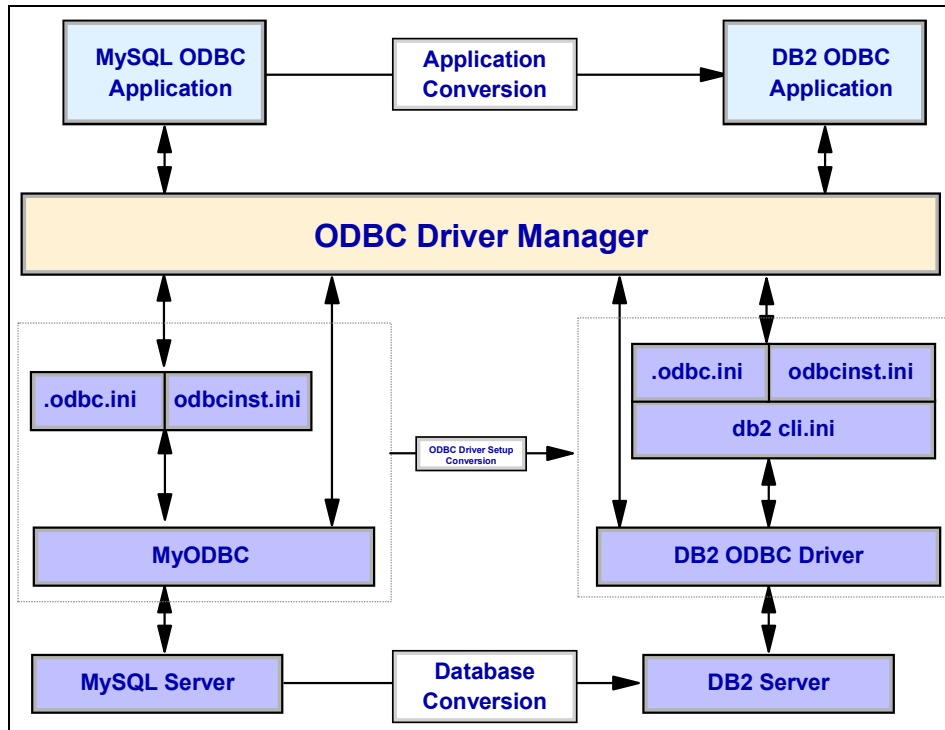


Figure 7-6 ODBC application conversion from MyODBC to DB2 ODBC Driver

Typically, the MyODBC connector has five components:

► Application

As both MyODBC and DB2 CLI are based on ODBC specification, applications do not require many changes. Though you have to perform some tasks such as:

- Changing SQL queries
- Changes in transaction management
- Proprietary methods changes, deviation from ODBC or additional methods.

► ODBC Driver Manager

DB2 CLI/ODBC Driver does not come with ODBC Driver Manager. When using an ODBC application you must ensure that an ODBC Driver Manager is installed and users who will use ODBC have access to it.

► ODBC.ini

The ODBC Driver Manager uses two initialization files:

- */etc/unixODBC/odbcinst.ini*, in which you need to add the following:

```
[IBM DB2 ODBC DRIVER]
Driver=/home/<instance name>/sql1lib/lib/db2.o
```

- */home/<instance name>/odbc.ini*, in which you need to configure the data source; for setting up a data source you need to add the following:

```
In [ODBC Data Source] stanza add
itsodb2= IBM DB2 ODBC DRIVER

Add [itsodb2] stanza with
Driver=/home/<instance name>/sql1lib/lib/db2.o
Description=itsodb2 DB2 ODBC Database

in [ODBC] stanza add
InstallDir=/home/<instance name>/sql1lib/odbc1ib
```

- ▶ **MyODBC**

As shown in Figure 7-6, you do not need to use MyODBC now, instead you will use DB2 ODBC Driver.

- ▶ **MySQL Server**

The MySQL database server is replaced by the DB2 UDB server; this is discussed in detail in previous chapters.

You may optionally configure DB2 ODBC Driver to modify the behavior of the DB2 ODBC Driver. This can be done by changing the *db2cli.ini* file.

7.2.6 Condition handling in DB2

Condition handling is one of the least sparkling topics in the application development arena. Foreseeing and handling exceptions in the application is paramount for a robust, industrial strength programming style. This section attempts an introduction to various ways to handle conditions in DB2.

Error checking with SQLCODE and SQLSTATE

Each time an SQL statement is executed two values are returned by the DB2 engine and placed in the SQL communication area (SQLCA): SQLCODE and SQLSTATE.

Within your application program you can retrieve these values to determine the state of the previously executed SQL statement. They are identifiers to get more detailed information about the condition of the statement.

SQLCODE is the variable conventionally used for error handling in applications coded against the DB2 family. Therefore, SQLCODE is probably of the finest granularity when it comes to DB2 exception handling.

However, the value for SQLCODE is IBM defined. To achieve the highest portability of applications, you should only build dependencies on a subset of DB2 SQLSTATEs that are defined by ODBC Version 3 and ISO SQL/CLI specifications. Whenever you build your exception handling on IBM supplied SQLSTATEs or SQLCODEs, the dependencies should be carefully documented. The specifications can be found using the search words ISO/IEC and standards 9075-1, 9075-2, and 9075-3 for SQL Foundation.

SQLSTATE is a five character string conforming with the ANSI SQL92 standard. The first two characters are known as the SQLSTATE class code. For example:

- ▶ 00 means successful completion
- ▶ 01 is a warning
- ▶ HY is generated by the DB2 CLI (call level interface) or ODBC driver
- ▶ IM is generated by the ODBC driver manager

If, for example, your application would signal SQLSTATE 23000, the DB2 description reports an *integrity constraint violation*, which is quite similar to MySQL's rudimentary description ER NON UNIQ ERROR or ER DUP KEY. Hence, condition handling for both database management systems could almost execute the same code.

Error returns in MySQL

Although it was necessary to limit the scope of this document to MySQL Version 4.0, fairness requires us to mention that Version 4.1. of MySQL provides exception handling using SQLSTATE. It will only appear for Version 4.1 or later, and was added for compatibility with X/Open/ANSI/ODBC behavior in an attempt to attract standard compliant applications, and encourage existing MySQL applications to become more standard compliant.

The definition for SQLSTATE can be found in the MySQL source code file *include/sql_state.h*.

SQLSTATE for DB2 CLI

Follow these guidelines for using SQLSTATEs within your CLI application:

- ▶ Always check the function return code before calling *SQLGetDiagRec()* to determine if diagnostic information is available. Please refer to the IBM DB2 UDB manual *Call Level Interface Guide and Reference, Volume 2*, SC09-4850 for more information on this API.
- ▶ Use the SQLSTATEs rather than the native error code.
- ▶ To increase your application's portability, only build dependencies on the subset of DB2 CLI SQLSTATEs that are defined by the ODBC version 3 and ISO SQL/CLI specifications, and return the additional ones as information only.

- ▶ It may be useful to build dependencies on the class (the first two characters) of the SQLSTATES.
- ▶ For maximum diagnostic information, return the text message along with the SQLSTATE (if applicable, the text message will also include the IBM defined SQLSTATE). It is also useful for the application to print out the name of the function that returns the error.

The following code segment from *utilcli.c* shows how diagnostic information such as SQLSTATES can be retrieved and displayed.

Example 7-50 Handling SQLSTATE in CLI

```
void HandleDiagnosticsPrint(SQLSMALLINT htype, /* handle type identifier */
                           SQLHANDLE hndl /* handle */ )
{
    SQLCHAR message[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
    SQLINTEGER sqlcode;
    SQLSMALLINT length, i;

    i = 1;

    /* get multiple field settings of diagnostic record */
    while (SQLGetDiagRec(htype,
                        hndl,
                        i,
                        sqlstate,
                        &sqlcode,
                        message,
                        SQL_MAX_MESSAGE_LENGTH + 1,
                        &length) == SQL_SUCCESS)
    {
        printf("\n SQLSTATE          = %s\n", sqlstate);
        printf(" Native Error Code = %ld\n", sqlcode);
        printf("%s\n", message);
        i++;
    }

    printf("-----\n");
}

```

Note: The code snippets provided in this chapter are for illustration purposes only. *Utilcli.c* is sample code shipped with DB2 and can be found in the *SQLLIB/samples* directory.

Handling SQL errors in an SQLj application

SQLj clauses use the JDBC class *java.sql.SQLException* for error handling. SQLj generates an *SQLException* under the following circumstances:

- ▶ When any SQL statement returns a negative SQL error code
- ▶ When a SELECT INTO SQL statement returns a +100 SQL error code

You can use the *getErrorCode* method to retrieve SQL error codes and the *getSQLState* method to retrieve SQLSTATES.

To handle SQL errors in your SQLj application, import the *java.sql.SQLException* class, and use the Java error handling try/catch blocks to modify program flow when an SQL error occurs (see Example 7-51).

Example 7-51 SQL Exception with SQLj

```
try {
    #sql [ctxt] {SELECT LASTNAME INTO :empname
                FROM EMPLOYEE WHERE EMPNO='000010'};
}
catch(SQLException e) {
    System.out.println("Error code returned: " + e.getErrorCode());
}
```

For exception handling in Java it is important to know that DB2 provides several types of JDBC drivers with slightly different characteristics. With the DB2 Universal JDBC Driver, you can retrieve the SQLCA. For the DB2 JDBC type 2 driver for Linux, UNIX, and Windows (DB2 JDBC type 2 driver), use the standard *SQLException* to retrieve SQL error information.

SQLException under the DB2 Universal JDBC Driver

As in all Java programs, error handling is done using try/catch blocks. Methods throw exceptions when an error occurs, and the code in the catch block handles those exceptions.

JDBC provides the *SQLException* class for handling errors. All JDBC methods throw an instance of *SQLException* when an error occurs during their execution. According to the JDBC specification, an *SQLException* object contains the following information:

- ▶ A string object that contains a description of the error or null
- ▶ A string object that contains the SQLSTATE or null
- ▶ An int value that contains an error code
- ▶ A pointer to the next *SQLException* or null

The DB2 Universal JDBC Driver provides an extension to the *SQLException* class, which gives you more information about errors that occur when DB2 is

accessed. If the JDBC driver detects an error, this `SQLException` class gives you the same information as the standard `SQLException` class. However, if DB2 detects the error, this `SQLException` class gives you the standard information, along with the contents of the SQLCA that DB2 returns. If you plan to run your JDBC applications only on a system that uses the DB2 Universal JDBC Driver, you can use this extended `SQLException` class.

Under the DB2 Universal JDBC Driver, `SQLExceptions` from DB2 implement the `com.ibm.db2.jcc.DB2Diagnosable` interface. An `SQLException` from DB2 contains the following information:

- ▶ A `java.lang.Throwable` object that caused the `SQLException` or null if no such object exists. The `java.lang.Throwable` class is the superclass of all errors and exceptions in the Java language.
- ▶ The information that is provided by a standard `SQLException`
- ▶ An object of DB2-defined type `DB2Sqlca` that contains the SQLCA. This object contains the following objects:
 - An `INT` value that contains an SQL error code
 - A `String` object that contains the `SQLERRMC` values
 - A `String` object that contains the `SQLERRP` value
 - An array of `INT` values that contains the `SQLERRD` values
 - An array of `CHAR` values that contains the `SQLWARN` values
 - A `String` object that contains the `SQLSTATE`

The basic steps for handling an `SQLException` in a JDBC program that runs under the DB2 Universal JDBC Driver are:

1. Import the required classes for DB2 UDB error handling, `com.ibm.db2.jcc.DB2Diagnosable` for getting diagnostic data, and `com.ibm.db2.jcc.DB2Sqlca` for error messages.
2. In your code catch `SQLException` and use it to get SQLCA. This is allowed only if the exception thrown is an instance of the `DB2Diagnosable` class.
3. Once you have `DB2Sqlca`, it can be used to get `SQLCODE`, messages, SQL errors, and warnings as shown in Example 7-52.

Example 7-52 Processing an SQLException under the DB2 Universal JDBC Driver

```
import com.ibm.db2.jcc.DB2Diagnosable;
import com.ibm.db2.jcc.DB2Sqlca;

try {
    // Code that could throw SQLExceptions
} catch(SQLException sqle) {
    while(sqle != null) {
        if (sqle instanceof DB2Diagnosable) {
            DB2Sqlca sqlca = ((DB2Diagnosable)sqle).getSqlca();
```

```

        if (sqlca != null) {
            System.err.println ("SqlCode: " + sqlca.getSqlCode());
            System.err.println ("SQLERRMC: " + sqlca.getSqlErrmc());
            System.err.println ("SQLERRP: " + sqlca.getSqlErrp() );
            String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
            for (int i=0; i< sqlErrmcTokens.length; i++) {
                System.err.println (" token " + i + ": " + sqlErrmcTokens[i]);
            }
            int[] sqlErrd = sqlca.getSqlErrd();
            char[] sqlWarn = sqlca.getSqlWarn();
            System.err.println ("SQLSTATE: " + sqlca.getSqlState());
            System.err.println ("message: " + sqlca.getMessage());
        }
    }
    sqlc=sqlc.getNextException();
}
}

```

Error handling using the WHENEVER statement

The WHENEVER statement causes the SQL precompiler to generate source code that directs the application to go to a specified label if either an error, a warning, or no rows are found during execution. The WHENEVER statement affects all subsequent executable SQL statements until another WHENEVER statement alters the situation.

The WHENEVER statement has three basic forms:

```

EXEC SQL WHENEVER SQLERROR  action
EXEC SQL WHENEVER SQLWARNING action
EXEC SQL WHENEVER NOT FOUND action

```

In the above statements:

- ▶ SQLERROR: Identifies any condition where SQLCODE < 0.
- ▶ SQLWARNING: Identifies any condition where SQLWARN(0) = W or SQLCODE > 0 but is not equal to 100.
- ▶ NOT FOUND: Identifies any condition where SQLCODE = 100.

In each case, the *action* can be either of the following:

- ▶ CONTINUE: Indicates to continue with the next instruction in the application.
- ▶ GO TO *label*: Indicates to go to the statement immediately following the label specified after GO TO. (GO TO can be two words, or one word, GOTO.)

If the WHENEVER statement is not used, the default action is to continue processing if an error, warning, or exception condition occurs during execution.

The WHENEVER statement must appear before the SQL statements you want to affect. Otherwise, the precompiler does not know that additional error-handling code should be generated for the executable SQL statements. You can have any combination of the three basic forms active at any time. The order in which you declare the three forms is not significant.

To avoid an infinite looping situation, ensure that you undo the WHENEVER handling before any SQL statements are executed inside the handler. You can do this using the WHENEVER SQLERROR CONTINUE statement.

Declaring the SQLCA for error handling

You can declare the SQLCA in your application program so that the database manager can return information to your application. When you preprocess your program, the database manager inserts host language variable declarations in place of the INCLUDE SQLCA statement. The system communicates with your program using the variables for warning flags, error codes, and diagnostic information.

After executing each SQL statement, the system returns a return code in both SQLCODE and SQLSTATE. SQLCODE is an integer value that summarizes the execution of the statement, and SQLSTATE is a character field that provides common error codes across IBM's relational database products. SQLSTATE also conforms to the ISO/ANS SQL92 and FIPS 127-2 standard.

Note that if SQLCODE is less than 0, it means an error has occurred and the statement has not been processed. If the SQLCODE is greater than 0, it means a warning has been issued, but the statement is still processed.

For a DB2 application written in C or C++, if the application is made up of multiple source files, only one of the files should include the EXEC SQL INCLUDE SQLCA statement to avoid multiple definitions of the SQLCA. The remaining source files should use the following lines:

```
#include "sqlca.h"
extern struct sqlca sqlca;
```

Condition handling in DB2 stored procedure

Although the content of this book is based on MySQL Version 4, and stored procedure support is advertised for Version 5 of MySQL, it seems appropriate to include a few examples of DB2 condition handling. For detailed information on condition handlers you may refer to DB2 UDB manual *Application Development Guide: Programming Server Applications*, SC09-4827.

The general form of a handler declaration is:

```
---DECLARE-----+---CONTINUE+----- HANDLER-- FOR---condition----->
```

```

+-EXIT-----+
+-UNDO-----+
>-----SQL-procedure-statement-----|

```

When DB2 raises a condition that matches a condition, DB2 passes control to the condition handler. The condition handler performs the action indicated by *handler-type*, and then executes SQL-procedure-statement.

DB2 provides three general conditions:

- ▶ **NOT FOUND:**
Identifies any condition that results in an SQLCODE of +100 or an SQLSTATE beginning with the characters '02'.
- ▶ **SQLEXCEPTION:**
Identifies any condition that results in a negative SQLCODE.
- ▶ **SQLWARNING:**
Identifies any condition that results in a warning condition (SQLWARN0 is 'W'), or that results in a positive SQL return code other than +100. The corresponding SQLSTATE value will begin with the characters '01'.

You can also use the DECLARE statement to define your own condition for a specific SQLSTATE.

Example 7-53 shows the general flow of the condition handler in a stored procedure.

Example 7-53 General example for condition handling

```

Begin
  declare exit handler
    for sqlexception
    begin
      statement3;
      statement4;
    end;

  statement1;
  statement2;
End

```

Example 7-54 shows a CONTINUE handler for delete and update operations on a table named EMP. Again, please note that this code is intended for illustration only.

Example 7-54 Example of a DB2 CONTINUE handler

```
CREATE PROCEDURE PROC1()  
LANGUAGE SQL  
BEGIN  
    DECLARE SQLCODE, v_error INT;  
    DECLARE CONTINUE HANDLER FOR  
                                SQLEXCEPTION,  
                                SET v_error = SQLCODE;  
  
    DELETE FROM emp  
    WHERE empno BETWEEN 100 and 200;  
    IF (v_error = -147 ) THEN  
        INSERT . . .  
  
    UPDATE staff SET salary = salary * 1.25;  
    IF (v_error <> 0 ) THEN  
        RETURN -1;  
    END IF;  
END
```

7.2.7 Special conversions

MySQL provides access control on the level of hosts. That means specific privileges are granted depending on the host from which the user connects. DB2 UDB does not provide this control mechanism. So, if you use this MySQL feature, you have to implement a workaround in the application. Applications which use the MySQL host authentication feature to control user privileges on a database or global level will require code change.

There are many ways to implement this authentication mechanism on the application level. Here we demonstrate a workaround using a simple example application that has two functions SELECT and INSERT, which use the MySQL security feature to limit selecting and inserting data on the host level. Example 7-55 shows that the MySQL host access data for four users controlled by our example.

Example 7-55 MySQL host access data

```
mysql> select user, host, select_priv, insert_priv from user;
```

user	host	select_priv	insert_priv
	localhost	N	N
Michael	%.ibm.com	Y	N
Klaus	%	N	Y
Rakesh	%.in.ibm.com	Y	Y
itsosj	localhost	Y	N
itsosj	%.ibm.com	Y	Y

This information has to be ported into a DB2 table. When a user attempts to access the data in the DB2 database, the application will verify every user's database access rights along with the host system information where one is connecting from.

Two tables are needed for our DB2 conversion: one to store the user privileges information ported from MySQL and one is a working table. The table definitions and some sample values are shown in Example 7-56.

Example 7-56 Creation of the tables for host authentication

```
-- script for creating the tables used by our example application

-- connect to the database
connect to SAMPLE user itsosj using itsosj;

-- table ACCESSLIST
-- it stores access rights for specific users connecting from specific hosts
-- remark: there should be different access-flags for different functions

-- fields:
-- username, whom access to the function should be granted
-- hostname or ip-address, from which the user must connect
-- select access flag (Y/N), if SELECT is granted
-- insert access flag (Y/N), if INSERT is granted

drop table ACCESSLIST;
create table ACCESSLIST (
  USERNAME  varchar(8),
  HOST      varchar(30),
  ACCESS_SEL char(1),
  ACCESS_INS char(1)
);
```

```

-- insert some sample values, according to the MySQL values (see above)
insert into ACCESSLIST values('Michael', '%.ibm.com', 'Y', 'N');
insert into ACCESSLIST values('Klaus', '%', 'N', 'Y');
insert into ACCESSLIST values('Rakesh', '%.in.ibm.com', 'Y', 'Y');
insert into ACCESSLIST values('itsosj', 'localhost', 'Y', 'N');
insert into ACCESSLIST values('itsosj', '%.ibm.com', 'Y', 'Y');

-- table APPLACCESS
-- it stores the info about users and their host asking for access
-- this table is filled automatically by the sample application

-- fields:
-- username, who asks for access to the function
-- hostname, from which the user connects
-- ip-address, from which the user connects
-- timestamp, when the user asks for access

drop table APPLACCESS;
create table APPLACCESS (
    USERNAME varchar(8),
    HOSTNAME varchar(30),
    IPADDR   varchar(30),
    TS       timestamp
);

```

The tables ACCESSLIST and APPLACCESS are used by the authentication mechanism:

► ACCESSLIST table

This table stores all the combinations of users and hosts (either name or IP-address) and specific database access privileges are granted to the user and host combination. In our example, we control two access rights: SELECT and INSERT, so that we have two access fields ACCESS_SEL and ACCESS_INS. In a real world application more functions would probably be controlled, so this table should be expanded to have one ACCESS field for each function to be controlled based on the corresponding MySQL privilege.

We insert some sample values into this table for demonstration purposes.

► APPLACCESS table

This table is filled by the authentication application during runtime. When a user asks for access, the application inserts the user ID, the host name, and the IP-address from where the user connects. The timestamp is used as a key in this table as records are not deleted from it.

The application code is listed in Example 7-57; remember that the code is just for demonstration purposes.

Example 7-57 Authentication mechanism example

```
import java.lang.*;
import java.io.*;
import java.sql.*;
import java.net.*;

public class AccessControl
{
    // in our example we use fixed values, you should make this variable
    private static final String DB2DB = "sample"; // database name
    private static final String DB2USR = "itsosj"; // database user
    private static final String DB2PWD = "itsosj"; // database password

    private static Connection db2Conn; // DB2 connection object

    public static void main(String[] args) throws SQLException, Exception
    {
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();

        db2Conn=DriverManager.getConnection("jdbc:db2:"+DB2DB,DB2USR,DB2PWD);

        // This example shows how to verify accesss to specific functions
        // depending on the host where the program is executed.

        // We assume that this program runs locally and connects directly to DB2
        // In a client/server-app, the server must detect the client's host.

        // Java provides two methods for this purpose:
        //   socket.getInetAddress() ... for socket-connections
        //   request.getRemoteHost() ... for HTTP-connections

        // In this example the username is the same as the DB2 user.
        // The application user and the database user could also be different.
        // This example implements no error handling.

        // This example shows access to two specific functions:
        //   SELECT...this is indicated by mode='SEL'
        //   INSERT...this is indicated by mode='INS'
        // mode should be a variable parameter

        String mode = "SEL";

        InetAddress addr = InetAddress.getLocalHost(); // appl executes locally
        String ipname = addr.getHostName();           // get the hostname
        String ipaddr = addr.getHostAddress();        // get the ip-address
        System.out.println("Host name = " + ipname);
        System.out.println("Host addr = " + ipaddr);
    }
}
```

```

// get records for the specified username with access to the function
String query1 = "select HOST, current timestamp as TS from ACCESSLIST "+
    "where USERNAME='" + DB2USR + "' and ACCESS_" + mode + "='Y'";
System.out.println("Query = " + query1);
PreparedStatement ps1 = db2Conn.prepareStatement(query1);
ResultSet rs = ps1.executeQuery(); // run the query

if (! rs.next()) // no rows found?
    System.out.println("no authorization for this username...");
else
{
    String ts = rs.getString("TS"); // retrieve the timestamp (key)
    String hostval = rs.getString("HOST"); // retrieve allowed hostname

    // write the current connection info into a table
    // with this table it is possible to use the SQL like function
    String insertStr = "insert into APPLACCESS values " +
        "('"+DB2USR+"', '"+ipname+"', '"+ipaddr+"', '"+ts+"'";
    System.out.println("Insert = " + insertStr);
    PreparedStatement ps0 = db2Conn.prepareStatement(insertStr);
    ps0.execute(); // run the insert

    // check if the current connection info has an equivalent host entry
    // (either IP-name or IP-address)
    String query2 = "select 1 from APPLACCESS where "+
        "TS='"+ts+"' and "+
        "HOSTNAME like '"+hostval+"' or IPADDR like '"+hostval + "'";
    while (rs.next()) // there can be more than one permitted hosts
    {
        hostval = rs.getString("HOST"); // retrieve the allowed hostname
        query2 = query2 + " or HOSTNAME like '" + hostval +
            "' or IPADDR like '" + hostval + "'";
    }
    System.out.println("Query = " + query2);
    PreparedStatement ps2 = db2Conn.prepareStatement(query2);
    ResultSet rs2 = ps2.executeQuery(); // run the query
    if (! rs2.next()) // no accordance found?
        System.out.println("no authorization...");
    else
    {
        System.out.println("You are authorized to go on!");
        // here should be the call to the access controlled function
        if (mode.equals("SEL")
        {
            // call the SELECT function
        }
        if (mode.equals("INS")
        {
            // call the INSERT function
        }
    }
}

```

```
}
}
}
}
```

This example application works in the following ways:

- ▶ The first step is to get all hosts out of the ACCESSLIST table from which the specified user has access to the requested function.
- ▶ The second step is to insert the information about the access request into the table APPLACCESS. The main reason for this step is that if this information is stored in a table, the SQL LIKE function can be used in the next step. The LIKE function handles wildcards ('%' and '_') in the host information correctly.
- ▶ The third step is to verify if the host name or IP address has the access rights by comparing the entry with the host name retrieved in the first step.
- ▶ If so, access is allowed and the function should be executed. You can also implement a method that has a return code telling if access is allowed or not.

7.3 Additional application considerations

Once converted to DB2 Version 8.1, an application's run-time performance can be impacted by a number of factors. The following section describes what locking and transaction isolation does to your application when running in a multi-user environment.

7.3.1 What is the purpose of locking?

When many users access the same data source through your application, or any other interface that allows data manipulation, some unwanted effects may occur. These effects are called:

- ▶ **Lost update:**
Two concurrent users retrieve and update the same data. The last successful change is kept while the first change is overridden.
- ▶ **Uncommitted (or dirty) read:**
User A can read or view data changed by User B, but those changes have not been committed yet.
- ▶ **Non-repeatable read:**
Within the same transaction User A runs the identical SELECT statement multiple times with different results because User B *modified* records in User A's result set.

- ▶ Phantom read:
Within the same transaction User A runs a SELECT statement multiple times and gets addition records because user B *added* records in User A's result set.

One of the more advanced features of a data management system is to define *modification rules* to control the use of data and guarantee the integrity of the data to prevent above undesirable effects.

7.3.2 Concurrency control and transaction isolation

From a bird's eye view, two methods for concurrency control can be differentiated:

- ▶ The optimistic concurrency approach:
A strategy to increase concurrency in which rows are not locked. Transactions are divided into *read*, *validate*, and *write* phases. Instead, during the validation phase before they are updated or deleted, a cursor checks to see if they have been changed since they were last read. If so, the update or delete fails.
- ▶ The pessimistic concurrency or locking approach:
A strategy for implementing serializability in which rows are locked so that other transactions cannot change them. Transaction requests locks to update resources. Other transactions have to wait or time-out. The resource is released on transaction completion or commit and rollback.

Both methods have their pros and cons, but by far the most popular method is the latter approach. Both MySQL and DB2 follow this approach to various degrees of sophistication and implementation differences.

7.3.3 DB2 isolation levels

In general DB2 UDBs default setting do nt require changes in the application to work around a different locking behavior.

DB2 provides transaction isolation levels to segregate data and prevent the undesirable effects discussed in 7.3.1, "What is the purpose of locking?" on page 226. Table 7-10 gives an overview over DB2 isolation levels.

Table 7-10 DB2 isolation level

DB2 UDB isolation level	Description
Uncommitted Read	<ul style="list-style-type: none"> ▶ Access to uncommitted data from other transaction ▶ No record locks unless updates occur

DB2 UDB isolation level	Description
Cursor Stability	<ul style="list-style-type: none"> ▶ Addresses <i>dirty read</i> issue ▶ Sees only committed data from other transaction ▶ Lock is only held on cursor position unless update occurs ▶ Update lock is held until transaction completed = commit
Read Stability	<ul style="list-style-type: none"> ▶ Addresses <i>non-repeatable read</i> issue ▶ Sees only committed data from other transaction ▶ Locks are held on every row fetched (Inserts permitted) ▶ Locks are held for duration of transaction (commit/rollback)
Repeatable Read	<ul style="list-style-type: none"> ▶ Addresses <i>phantom read</i> issue ▶ All record locks held for duration of transaction ▶ A repeated query within the same transaction will get the same result set (Inserts are prevented)

As one can see, the isolation levels listed in Table 7-10 are ordered descendent according to the number and duration of locks held during the transaction, and therefore the degree of concurrency or locking is required to ensure the desired level of data integrity. However, as we can see too much locking drastically reduces concurrency. Poor application design and coding may cause locking problems such as:

- ▶ Deadlocks
- ▶ Lock waits
- ▶ Lock escalation
- ▶ Lock time-outs

By default DB2 operates with the isolation level *cursor stability*. Transaction isolation can be specified at many different levels as discussed in 7.3.5, “Specifying the isolation level in DB2” on page 230. For good performance, verify the lowest isolation level required for your migrated application.

For additional information on the DB2 UDB concurrency implementation, please refer to the DB2 UDB manual *Administration Guide: Performance*, SC09-4821.

7.3.4 Locking

Some MySQL applications when ported to DB2 appear to behave identically, and the topic of concurrency can be ignored. However, if your applications involve frequent access to the same tables you may experience a different behavior. By default, MySQL runs in a mode that is called *autocommit*. This means that MySQL considers each and every SQL statement as an atomic *unit of work* or transaction.

In contrast, DB2 by default considers a group of SQL statements with the corresponding unit of work boundaries set by a commit respectively a rollback statement as single or atomic transaction. Only certain interfaces such as the DB2 command line processor (CLP) or the JDBC interface run in autocommit mode. For all other application interfaces, autocommit is by default turned off.

Another matter causing heated discussions among experts is the level of locking that needs to be implemented on the database level. Should the locking approach be implemented with the lowest level of overhead, and therefore maintain locks on a table level? Or, is it better to lock on a lower level for example on page level? Should the granularity be even finer and locking occur on row level?

As usual, the correct answer to these questions is: *it depends!*

MySQL development decided to go the *multi storage engine* way and decided to implement lock levels based on the type of table. Table types can be mixed within a database and even a statement, and types can be altered. The default storage engine for MySQL supports only table level locking. MyISAM table, Merge and HEAP tables use a default storage engine and have table level locking. The *InnoDB* storage engine was released as a transactional table handler of MySQL with a lock manager for row level locking mechanisms. Hence, the MySQL table type InnoDB defines tables most alike DB2 tables. In addition to the two storage engines already discussed, MySQL integrated the BDB or *Berkley DB* table type. Table 7-11 gives a superficial comparison of the different flavors of MySQL tables with DB2 tables:

Table 7-11 MySQL and DB2 table comparison

Characteristics	DB2 tables	MyISAM tables	InnoDB tables	BDB tables
Lock level	Row level, Table level only on explicit request	None or table level	Row level and table level	Page level and table level

Characteristics	DB2 tables	MyISAM tables	InnoDB tables	BDB tables
Commitment Control	Yes	No	Yes	Yes
Isolation level	UR, CS, RS, RR	No	RU, RC, RR, Serializable	
Rollback on DDL	Yes	No	No	No

Consider using table level locking for:

- ▶ Applications that use mostly reads such as Data Warehouse and Business Intelligence applications
- ▶ Applications reading and updating through key positioning such as UPDATE... WHERE Custno = ?
- ▶ Applications using INSERTs with subselects and only a small number of UPDATE and DELETEs

Consider to use for row level locking for:

- ▶ Applications requiring a high level of concurrency and OLTP capabilities
- ▶ Many SELECTs with only small result sets
- ▶ Applications with high UPDATE/INSERT/DELETE frequency

However, let us attempt to summarize the concurrency issues that may arise when migrating a MySQL application to DB2 based on the two MySQL table types, which we consider significant:

- ▶ MyISAM tables provide a high level of concurrency since SQL processing occurs in autocommit mode and no row level locks are maintained. When migrating to DB2 ensure your application operates in autocommit mode, which is by default not the case. Verify the lowest isolation level required for your application and MyISAM tables.
- ▶ InnoDB tables provide concurrency control very similar to DB2. Please be aware that default transaction isolation for InnoDB is *repeatable read* while DB2 operates by default with *cursor stability*.

7.3.5 Specifying the isolation level in DB2

Because the isolation level determines how data is locked and isolated from other processes while the data is being accessed, you should select an isolation level that balances the requirements of concurrency and data integrity. The isolation level that you specify is in effect for the duration of the *unit of work*.

The isolation level can be specified in several different ways. The following heuristics are used in determining which isolation level will be used in compiling an SQL statement:

- ▶ **Static SQL:**
 - If an isolation clause is specified in the statement, then the value of that clause is used.
 - If no isolation clause is specified in the statement, then the isolation level used is the one specified for the package at the time when the package was bound to the database.
- ▶ **Dynamic SQL:**
 - If an isolation clause is specified in the statement, then the value of that clause is used.
 - If no isolation clause is specified in the statement, and a SET CURRENT ISOLATION statement has been issued within the current session, then the value of the CURRENT ISOLATION special register is used.
 - If no isolation clause is specified in the statement, and no SET CURRENT ISOLATION statement has been issued within the current session, then the isolation level used is the one specified for the package at the time when the package was bound to the database.

Note: Many commercially written applications provide a method for choosing the isolation level. Refer to the application documentation for information.

SQL procedure and isolation level

This section discusses when to specify the isolation level for a SQL procedure.

At precompile or bind time

For an application written in a supported compiled language, use the ISOLATION option of the Command Line Processor **PREP** or **BIND** commands. You can also use the PREP or BIND APIs to specify the isolation level.

- ▶ If you create a bind file at precompile time, the isolation level is stored in the bind file. If you do not specify an isolation level at bind time, the default is the isolation level used during precompilation.
- ▶ If you do not specify an isolation level, the default of cursor stability is used.

Tip: To determine the isolation level of a package, execute the following query:

```
SELECT ISOLATION FROM SYSCAT.PACKAGES
WHERE PKGNAME = 'XXXXXXXX'
AND PKGSCHEMA = 'YYYYYYYY'
```

where XXXXXXXX is the name of the package and YYYYYYYY is the schema name of the package. Both of these names must be in all capital letters.

On database servers that support REXX

When a database is created, multiple bind files that support the different isolation levels for SQL in REXX are bound to the database. Other command-line processor packages are also bound to the database when a database is created.

REXX and the command line processor connect to a database using a default isolation level of cursor stability. Changing to a different isolation level does not change the connection state. It must be executed in the CONNECTABLE AND UNCONNECTED state, or in the IMPLICITLY CONNECTABLE state.

At the statement level

Use the WITH clause. The statement-level isolation level overrides the isolation level specified for the package in which the statement appears.

You can specify an isolation level for the following SQL statements:

- ▶ SELECT
- ▶ SELECT INTO
- ▶ Searched DELETE
- ▶ INSERT
- ▶ Searched UPDATE
- ▶ DECLARE CURSOR

The following conditions apply to isolation levels specified for statements:

- ▶ The WITH clause cannot be used on subqueries.
- ▶ The WITH UR option applies only to read-only operations. In other cases, the statement is automatically changed from UR to CS.

From CLI or ODBC at runtime

Use the CHANGE ISOLATION LEVEL command. For DB2 Call Level Interface (DB2 CLI), you can change the isolation level as part of the DB2 CLI configuration. At runtime, use the *SQLSetConnectAttr* function with the SQL_ATTR_TXN_ISOLATION attribute to set the transaction isolation level for the current connection referenced by the *ConnectionHandle*. You can also use the TXNISOLATION keyword in the *db2cli.ini* file .

When working with JDBC or SQLj at runtime

Note: JDBC and SQLj are implemented with CLI on DB2, which means the *db2cli.ini* settings might affect what is written and run using JDBC and SQLj.

Use the *setTransactionIsolation* method in the *java.sql* interface connection.

In SQLj, you run the *db2prof* SQLJ optimizer to create a package. The options that you can specify for this package include its isolation level.

For dynamic SQL within the current session

Use the SET CURRENT ISOLATION statement to set the isolation level for dynamic SQL issued within a session. Issuing this statement sets the CURRENT ISOLATION special register to a value that specifies the level of isolation for any dynamic SQL statements issued within the current session. Once set, the CURRENT ISOLATION special register provides the isolation level for any subsequent dynamic SQL statement compiled within the session, regardless of the package issuing the statement. This isolation level will apply until the session is ended, or until a SET CURRENT ISOLATION statement is issued with the RESET option.



Database administration

In this chapter, we focus on the database administration features of a DB2 UDB, keeping in mind all the MySQL features. We provide a general introduction to MySQL administrations programs, utilities, and tools, and then provide a detailed description of DB2 UDB options corresponding to them. We also cover a few of the salient features available in DB2 UDB but missing in MySQL.

We present key attributes of database administration for both MySQL and DB2 UDB such as:

- ▶ Database recovery
- ▶ Database replication
- ▶ Data movement utilities
- ▶ High availability
- ▶ Automated tasks/jobs
- ▶ Database configuration

We also explore following graphical and command line tools:

- ▶ MySQL phpMyAdmin and Control Center
- ▶ DB2 UDB Control Center
- ▶ DB2 UDB Web Command Center

8.1 Database recovery

Database recovery is the action the database system or user takes to recuperate the database in case of system, hardware, software, or application failure. This includes precautionary measures and curing measures. Different and multiple recovery methods are followed by all the database systems for warding off the data lose condition.

8.1.1 MySQL recovery

MySQL provide two recovery options: the first one is using table maintenance feature and the second is using backup and restore utilities. MySQL also provides automatic recovery of crashed tables, which can be enabled by running `mysqld` with the `--myisam-recover=#` option. This works only for MyISAM tables and internally uses the `myisamchk` utility. MySQL recovery options can be done by:

- ▶ Checking and repairing tables for disorder

MySQL provides *myisamchk* (or *isamchk*) for checking and repairing MyISAM (or ISAM) tables:

```
bash>myisamchk --silent --force --fast --update-state
/home/itso/mysql/data/*.MYI
```

```
bash>isamchk --silent --force /home/itso/mysql/data/*.ISM
```

If you are using *mysqld* with the `--skip-external-locking` option, you cannot use `mysqladmin` to check the table if it is locked by `mysqld`. In this case you should do `mysqladmin flush-tables` before checking the table.

Also, you may use the **check table** or **REPAIR TABLE** command for doing this; using **REPAIR TABLE** is same as using `myisamchk -r`

CHECK TABLE works only with MyISAM and InnoDB table and the **REPAIR TABLE** option works with only MyISAM tables.

- ▶ Using backup and restore

Another technique the MySQL user can use for crash recovery is by doing precautionary database backup frequently. In case of a crash, obtaining a stable state of database by restoring the data from backup and binary logs. The steps for doing this are:

- Precautionary frequent database backup by using one of these:

```
bash> mysqldump --tab=/home/itso/mysql/backup --opt itz'sodb
bash> mysqlhotcopy itsodb /home/itso/mysql/backup
```

- Restart `mysqld` with the `--log-bin` option; this will write the information required to replicate all the changes to the database after backup.

```
bash>mysqlld --log-bin=/home/itso/mysql/logs
```

- Restore backup data after crash recovery; this can be done using: `mysqlimport` or `LOAD DATA INFILE`,

```
bash>mysqlimport itsodb tablename.txt
```

- Re-running the update binary logs that occurred after backup

This can be done by running the `mysqlbinlog` command and redirecting the output to `mysql` using Linux pipes as shown below:

```
bash>mysqlbinlog hostname-bin.[0-9]* | mysql
```

8.1.2 DB2 UDB database recovery

DB2 UDB supports database recovery using database backup in conduction with two recovery logs: active logs and archived logs:

- ▶ Active logs

Active logs contain current transaction data. These logs are required in crash recovery.

- ▶ Achieved logs

Archived logs contain committed transactions. These logs are used in rollforward recovery, which can recover the database to the state immediately before the failure. Archie logging can be enabled by setting `logretain` and the `userexit` database configuration parameter to ON.

Database backup

Backup DB2 UDB database can use DB2 **backup** command. This command can be used to back up database to disk, tape, or names pipes in UNIX. DB2 UDB supports both offline and online backup.

```
db2>backup database itsodb2 to /home/itsodb/backup
```

In addition to backing up the entire database every time, DB2 UDB also supports incremental backup where you can back up large databases on a regular basis incrementally. This requires a `trackmod` database configuration parameter to be set to `yes`. Incremental backup can be a *cumulative backup*, which stores data changes since the last successful full backup, or *delta backup* which is the last successful backup irrespective of whether that backup was full, delta, or cumulative. Figure 8-1 and Example 8-1 show the cumulative and delta backup technique.

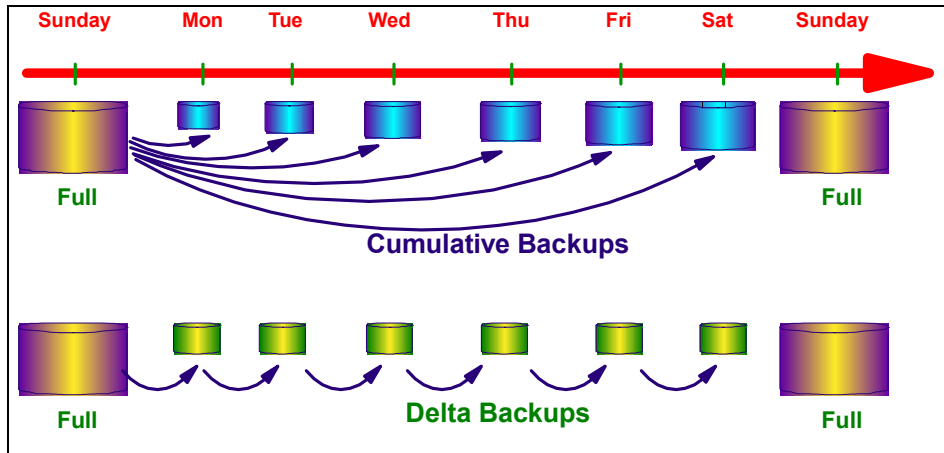


Figure 8-1 Incremental backup

Example 8-1 Incremental backup

```
db2>backup database itsodb2 to /home/itso/backup
db2>backup database itsodb2 INCREMENTAL to /home/itso/backup_inc
db2>backup database itsodb2 INCREMENTAL DELTA to /home/itso/backup_delta
```

Database restore

DB2 UDB database restore is as easy as taking database backup. This can be done by using the **RESTORE** utility. The restore database command rebuilds the database data or table space that was backed up using the backup command above. This utility supports full and incremental database restore. Incremental database restore can be automatic or manual. Example 8-2 shows automatic incremental restore, and Example 8-3 shows manual incremental restore.

Example 8-2 Automatic incremental restore

```
db2>restore database itsodb2 INCREMENTAL AUTOMATIC from /home/itso/backup taken
at 20040116
```

Example 8-3 Manual incremental restore

```
db2>create database itsodb2new1
db2>restore database itsodb2 INCREMENTAL from /home/itso/backup taken at
20040114
db2>restore database itsodb2 INCREMENTAL from /home/itso/backup taken at
20040114
db2>restore database itsodb2 INCREMENTAL from /home/itso/backup taken at
20040114
```

Using the above log files and backup files, DB2 UDB can support database recovery by various methods and to different levels:

- ▶ **Crash/restart recovery**

This is a technique for an automated database recovery to recover the database to a stage where all the transactions are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

This is done automatically by the database manager if you set the automatic restart (autorestart) database configuration parameter to ON by:

```
db2>db2 update database configuration for itsodb2 using autorestart ON
```

or by restarting the database when a database failure occurs; this is done by calling:

```
db2>restart database itsodb2
```

DB2 UDB maintains log files, the recovery history file, and table space change history file to recover data that is lost or damaged.

- ▶ **Version recovery**

This is recovery option used for non-recoverable databases or databases without achieved logs. In this recovery option, previous backup versions of the database are restored in case of an emergency. Typically, database backups are taken in tapes and kept in remote places.

It is very necessary to take a backup on regular basis because this recovery method loses the changes made in database after backup. This is done by restoring the database using incremental, delta, or full backup.

- ▶ **Rollforward recovery**

In rollforward recovery, you can specify the local time to which you want to roll forward your database. In this method backup data is used in conjugation with database transaction records to achieve a goal of taking DB2 UDB to the state immediately before the failure. Figure 8-2 shows the roll forward recovery technique of DB2 UDB.

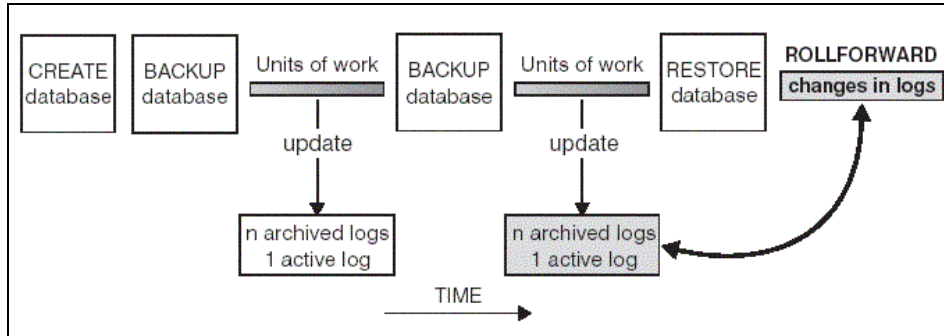


Figure 8-2 DB2 UDB roll forward restore

This can be achieved by executing the rollforward command after the backup image is restored as shown below:

```
db2>rollforward database itsodb2 complete
```

Tip: DB2 UDB backup and restore can also be done using Control Center.

8.2 Database replication

Database replication is a process of maintaining the database in multiple systems. It involves synchronizing changes from one database (a source) to another database (a target). This feature is very useful for load sharing, fast disaster recovery, and high availability.

MySQL allows one-way replication between two MySQL servers only. It is based on the master server keeping track of all the changes to the database in the binary log; and all the slave servers need to get this binary log from the master server and rerun all the statements on the slave database from the previous stable state. The slave server can be set up using the **LOAD DATA FROM MASTER** command; this is supported only for the MyISAM table type. Once the slave setup is done and running, it will connect to the master and wait for updates to process. This is a continuous process and the slave will always listen for an update.

DB2 UDB supports replication using publish subscribe replication setup, wherein you can replicate data not only in two DB2 UDB systems running on different platforms, but also on the following non-DB2 databases: Informix, Microsoft Jet, Microsoft SQL Server, Oracle, Sybase, and Sybase SQLAnywhere.

DB2 UDB provides a tool called the DB2 UDB *Replication Center*; it is a tool which can be used to set up and administer your replication environment, and to

run the Capture, Apply, and Replication Alert Monitor programs. It can be used to perform administrative tasks such as:



8.3.1 MySQL data movement

In MySQL, you can transfer tables (MyISAM and ISAM) by copying data (*.ISD, *.MYD) and index files (*.ISM, *.MYI) but only if your source and target operating system and machine architecture are same. Otherwise, you have to use following utilities for data movement:

- ▶ **mysqldump**

This is most popular MySQL utility for transferring the data from one MySQL server to another. It actually dumps a MySQL database into a file as SQL statements or as tab-separated text files. You can back up and restore databases or tables as shown in Example 8-4.

Example 8-4 mysqldump and restore

```
bash>mysqldump --opt itsodb > mysql.sql
bash>mysql itsodbtemp< mysql.sql
or
bash>mysql -e "source mysql.sql" itsodbtemp
```

This utility supports dumping from remote machine, multiple databases dumping, flush log before dumping, and locking the table before dumping options.

- ▶ **mysqlhotcopy**

If you are dumping data on server machine you may prefer to use **mysqlhotcopy**. This command internally calls MySQL commands **LOCK TABLES**, **FLUSH TABLES**, and then copies files into the output directory. It works only for UNIX machines and MyISAM and ISAM tables. It can be executed as shown below:

```
bash>mysqlhotcopy itsodb /home/itso/mysql/dbcopy
```

- ▶ **mysqlimport**

It is a simple utility to load data into the tables. It takes a comma or space separated data file as input and load it into the table. It is equivalent to calling the **LOAD DATA INFILE SQL** statement. This command can be executed as shown below:

```
bash>mysqlimport itsodb tablename.txt
```

8.3.2 DB2 UDB data movement

On the contrary, DB2 UDB provides faster and efficient tools and utilities for data movement across the different systems or reorganizing data on same system.

EXPORT utility

DB2 EXPORT is a powerful tool to export your DB2 data quickly from DB2 UDB to one of the file system outside DB2 system. DB2 EXPORT can be used to export tables, views, large objects, or typed tables to one of the three external file formats:

- ▶ .DEL: delimited ASCII format file
- ▶ .WSF: work sheet format like lotus 1-2-3®
- ▶ .IXF: integrated exchange format

The EXPORT utility can be invoked through:

- ▶ The Command Line Processor (CLP)

The EXPORT utility can be used from CLP by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables as shown below:

```
db2>export to itos.ixf of ixf select * from itsodb.katalog
```

- ▶ The Control Center

DB2 UDB data can be exported using a graphical user interface, which allows you to set export options for each table visually from Control Center. Figure 8-4 shows the usage of export through Control Center.

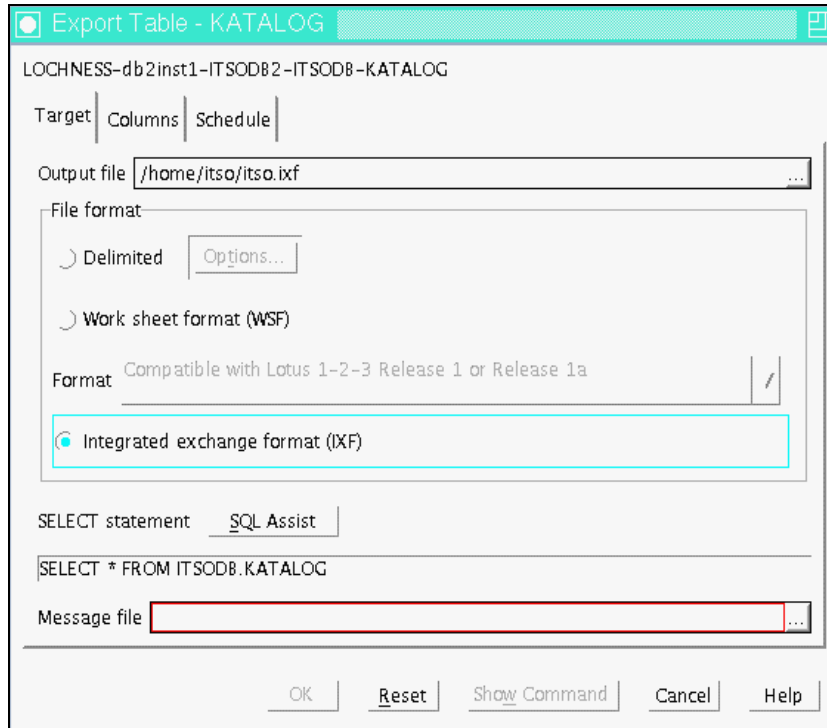


Figure 8-4 Export using control centre

- ▶ An application programming interface (API)
DB2 UDB provides an API for exporting data, which can be used to export data programmatically. This can be used by importing *sqlutil.h* and the method *sqluexpr*.

The IMPORT utility

The files created by EXPORT utility can be used to populate data into a new DB2 UDB database on the same system, or they can be transferred to another workstation platform and imported or loaded to a DB2 database that resides on that platform. IMPORT utility supports the following file formats:

- ▶ ASC: non-delimited ASCII format file
- ▶ .DEL: delimited ASCII format file
- ▶ .WSF: work sheet format like lotus 1-2-3
- ▶ .IXF: integrated exchange format

As EXPORT, IMPORT utility can also be used from:

- ▶ The Command line processor (CLP)

The IMPORT utility can be used from CLP by supplying an SQL INSERT, INSERT UPDATE, REPLACE or REPLAE_CREATE option. Also you can import the table hierarchy using the CREATE INTO option. The example below shows the simple IMPORT statement:

```
db2>import from itso.ixf of ixf messages msg.txt insert into itsodb.katalog
```

► The Control Center

Control Center can be used for importing data visually. You can get a wizard shown in Figure 8-5 by right-clicking <tablename> and selecting **Import...** from the **System -> server name -> Instance -> Databases -> database name -> Tables** window.

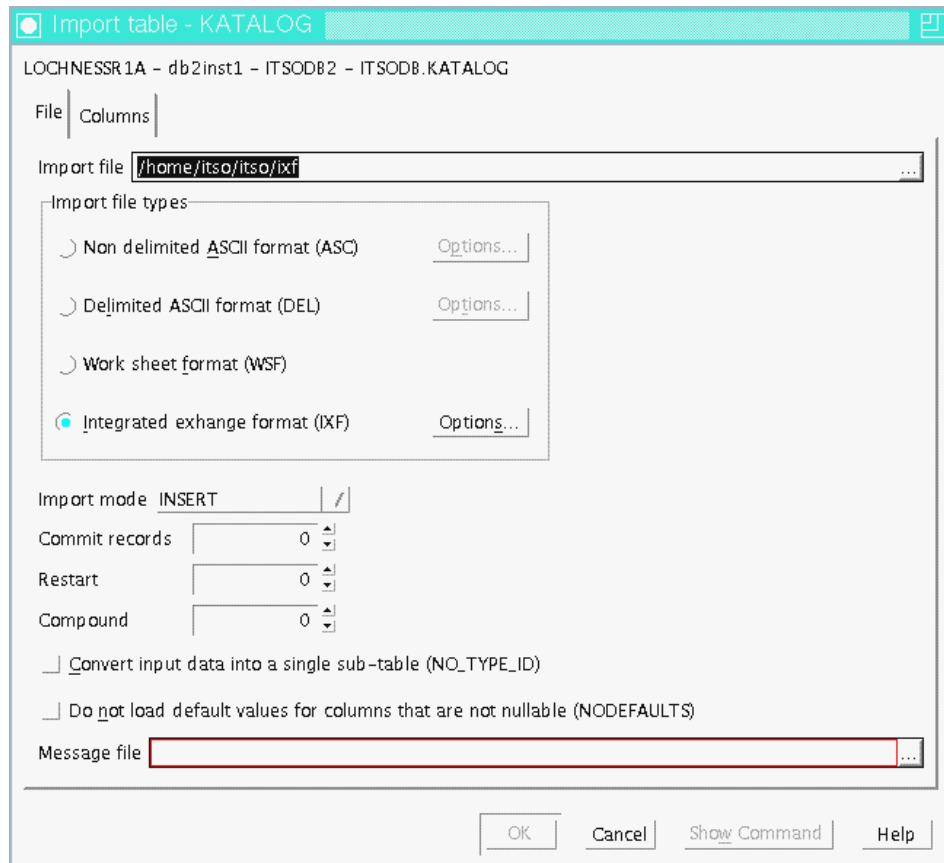


Figure 8-5 Import using control center

► An application programming interface (API)

DB2 UDB provides an API for importing data from files exported using the export tool. This provides an option to do importing programmatically by using `sqlutil.h`, and a method `sqlimpr`.

The DB2MOVE utility

Data can be copied using the DB2 **EXPORT** and **IMPORT** utilities. But a more efficient way to copy an entire DB2 UDB database is by using the DB2 *db2move* utility. This utility queries the system catalog tables for the specified database, and exports the table structure and contents of each table found to a PC/IXF formatted file. These files can be used to populate another DB2 UDB database. It can be run in three modes:

► **EXPORT mode**

In this mode, the **db2move** utility invokes DB2 **EXPORT** utility to extract data from one or more tables and write to PC/IXF formatted files. It also creates a file named `db2move.lst` that contains the names of all tables exported and the names of the files that the table's data was written to. **EXPORT** mode can be used as below:

```
bash>db2move itsodb2 export
```

► **IMPORT mode**

In this mode, the **db2move** utility invokes DB2 **IMPORT** utility to recreate tables and indexes from data stored in PC/IXF formatted files. The file `db2move.lst` generated in **EXPORT** mode can be used to get information about tables in the exported files. The above exported files can be imported using:

```
bash>db2move itsodb2 import
```

► **LOAD mode**

In this mode, the **db2move** utility invokes DB2 **LOAD** utility to populate tables that already exist with data stored in PC/IXF formatted files. The file `db2move.lst` generated in **EXPORT** mode can be used to get information about tables. The above exported files can be loaded using:

```
bash>db2move itsodb2 load -1 /home/itso/backup
```

8.4 High availability

High availability is a business requirement on the system to provide services all the time, and survive disaster, system crash, and glitches without or with minimal interrupting of the service. This is achieved by a number of techniques like online management of the database system, advanced instance management, suspended I/O, and clustered servers.

One of the advanced feature used for HA is failover, wherein a workload is transferred from one system to another quickly and automatically in case of failure.

MySQL does not support automatic high availability, instead you have to use replication features of MySQL to support high availability. You have to set up a master and slaves, and write scripts that will monitor the master to see if it is up and running, and as soon as the master gets down one of the slave, it acts like a master.

DB2 UDB has an advanced continuous checkpointing system and a parallel recovery capability that allows for extremely fast crash recovery. DB2 UDB fail-safe can be achieved in number of ways, most frequently used once are:

▶ Log shipping

In log shipping, whole log files are shipped to a standby machine though achieve file or user exit program running against primary database. The standby database is continuously rolling forward through the log files produced by the production machine. Now, as soon as main server gets down, the remaining logs are transferred to the standby machine, and the standby database rolls forward and the client resumes its services by reconnecting to the standby machine.

▶ Suspended I/O

Suspended I/O supports a quick initialization of new database by making a split mirror. A split mirror is instantaneous copy of the database that can be made by mirroring the disks containing the data, and splitting the mirror when a copy is required.

The **db2inidb** command initializes the split mirror so that it can be used:

- ▶ As a clone database
- ▶ As a standby database
- ▶ As a backup image

DB2 UDB failover can also be achieved using platform specific software:

- ▶ High Availability Cluster Multi-Processing, Enhanced Scalability, for AIX
- ▶ Microsoft Cluster Server, for Windows operating systems
- ▶ Sun Cluster, or VERITAS Cluster Server, for the Solaris Operating Environment
- ▶ Steeleye's Linux based high reliability solution for DB2 UDB

Apart from failover, DB2 UDB uses features like online buffer pools, online backup, and loading of data, online table, and index organization for providing uninterrupted service.

8.5 Automated tasks/jobs

Automated tasks lets you automate your database management job by scheduling the activities according to your requirement. They can be really useful for performing regular maintenance tasks such as backup, space monitoring, error checking, and maintenance, etc.

MySQL reckons on operating system functionality for scheduling. On Windows platforms, MySQL uses the *Task Scheduler* services to schedule jobs. On Linux platforms, it uses *cron* to schedule jobs. Few scheduling tools are available in market that can be used for scheduling:

- ▶ SQLyog Job Agent (SJA)
SJA is MySQL data synchronization and migration tool.
- ▶ PhpJobScheduler
PhpJobScheduler is a tool to automate tasks by scheduling PHP scripts to run at set intervals.

DB2 UDB offers an administrative interface called *Task Center* to automate and schedule tasks. The Task Center can be used to execute tasks, either immediately or according to a schedule, and to notify people about the status of completed tasks. It provides the ultimate flexibility for centralized scheduling and server-side execution.

With the help of Task Center you can:

- ▶ Schedule the task
- ▶ Specify success and failure conditions
- ▶ Specify success and failure actions
- ▶ Specify notification details
- ▶ Specify conditional coding using task actions.

The Task Center is tightly coupled with Control Center, so it can be invoked either directly by calling `db2tm` or from the **Control Center Tools -> Task Center** menu. Tasks can be created, saved, and reopened using a dialog or wizard within the Control Center. Figure 8-6 shows creation of tasks using Task Center.

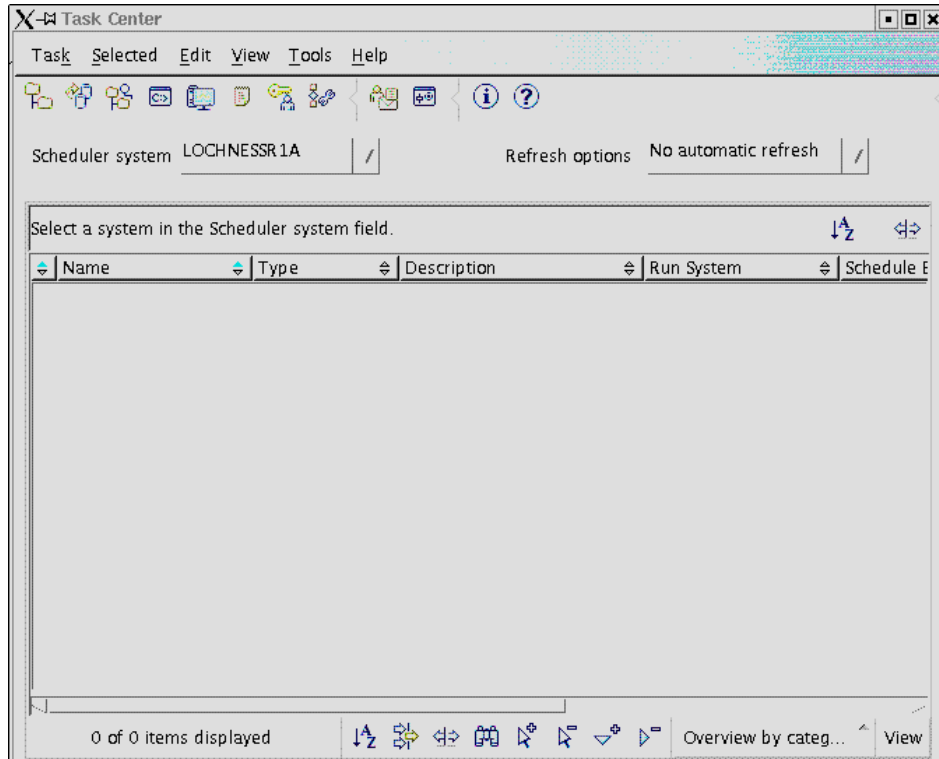


Figure 8-6 The Task Center console

8.6 Database configuration

Database configuration is another very important task for database administrators. It involves setting up parameters for database, system, database manager, etc. to optimum values for getting the best performance for your application. In this section we discuss how MySQL and DB2 UDB database parameters are tuned.

8.6.1 MySQL configuration

MySQL database configuration involves setting up system level parameters, server parameters, and database parameters. You can use database parameters to define various characteristics of a database. When you install the MySQL, a default configuration is stored; either this configuration can be used or it can be modified to suit any specific requirements you may have.

MySQL server options can either be given as a parameter to `mysqld`, or it can be written in the MySQL option files. The options are read from different files in the order of preference as shown in Table 8-1.

Table 8-1 MySQL options files

Preference	Filename	Purpose
1	/etc/my.cfg	Global options
2	DataDir/my.cfg	Server-specific options
3	defaults-extra-file	The file specified with --defaults-extra-file=path
4	~/my.cnf	User -specific options

Another way to tune MySQL is by running `mysqld` with `--name=value` to set the variable.

You can also set MySQL database variables using the **SET** command when your server is running. It can be executed with two modes:

- ▶ Global

This mode sets options at the server level; if a new connection is created the new values are used:

```
mysql>SET GLOBAL sort_buffer_size=1000000;
```

- ▶ Session

In this mode, the option you set will remain valid until the current session ends, or until you set the option to different values. It can be used as the following:

```
mysql>SET SESSION sort_buffer_size=1000000;
```

8.6.2 DB2 UDB configuration

DB2 UDB has two levels of configuration:

DB2 profile registry

DB2 profile registry variables control the DB2 environment. Information stored in these profile registries are used by the DB2 server instance, and the DB2 applications started after the changes are made. DB2 UDB provides the following different levels of profile registry:

- ▶ The DB2 Instance Level Profile Registry
- ▶ The DB2 Global Level Profile Registry
- ▶ The DB2 Instance Node Level Profile Registry

► The DB2 Instance Profile Registry

All these registry profile variables can be set using the **db2set** command. Example 8-5 shows different modes in which the **db2set** command can be used.

Example 8-5 Changing registry and environment variables using db2set

```
bash>db2set -all --this shows the current registry variables
[i] DB2COMM=tcPIP
[i] DB2AUTOSTART=YES
[g] DB2SYSTEM=1ochnessr1a
[g] DB2ADMINSERVER=dasusr1

bash>db2set DB2AUTOSTART=NO -i db2inst1 --this sets registry for all
databases in particular instance

bash>db2set DB2AUTOSTART=NO -g --this sets registry variable for all
instances

bash>db2set DB2AUTOSTART=NO -i db2inst1 65000 --this sets registry variable
for particular node
```

DB2 UDB configures the operating environment by checking for registry values and environment variables, and resolving them in the following order:

1. Environment variables set using the **export** command
2. Registry values set with the instance node level profile using **db2set -i instname nodenum**
3. Registry values set with the instance level profile using **db2set -i**
4. Registry values set with the global level profile using **db2set -g**

Configuration files

In addition to DB2 Profile Registry, DB2 UDB also has instance and database configuration files, which provide users the flexibility to configure the database and the database engine to fit business and application needs. These files are created when the DB2 instance or database is created. They contain the parameters that define values such as resource allocated to DB2 UDB, diagnostic level, log files location, etc. As shown in Figure 8-7, there are two DB2 configuration files and additional operating system configurations.

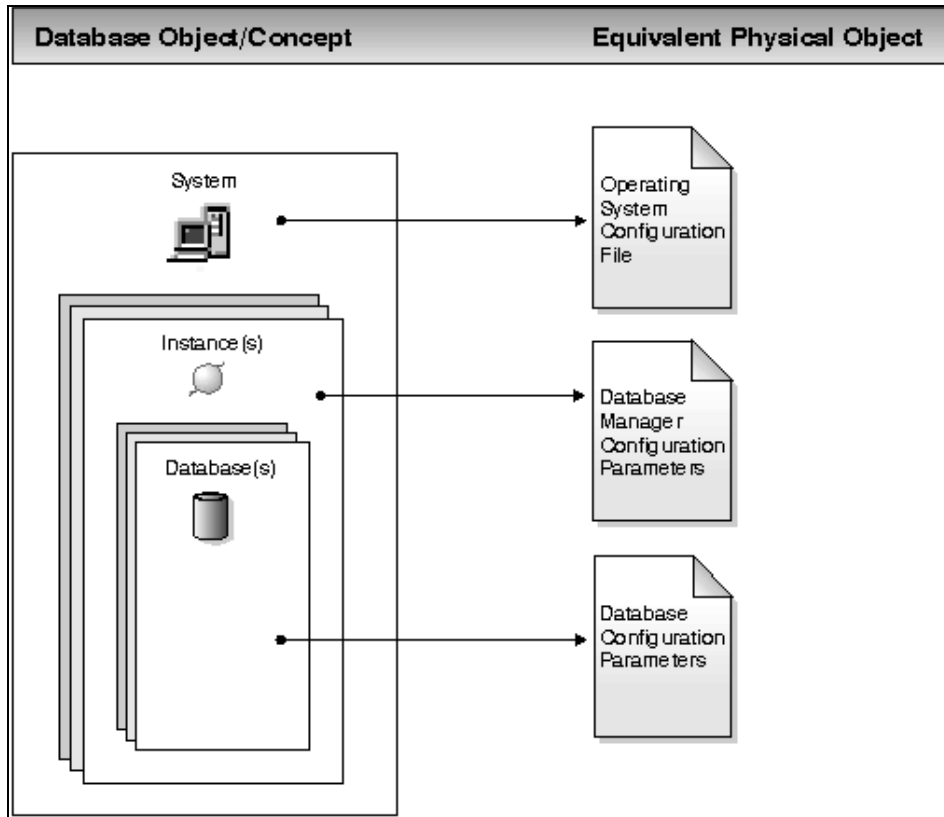


Figure 8-7 Configuration parameter files

► The database manager configuration file

This file is created with the DB2 instance and it affects the DB2 instance level. It is stored in **db2system** file under the *sqliib* subdirectory of instance. The values in this file can be seen using:

```
bash>db2 get database manager configuration
```

and it can be set using:

```
bash>db2 update dbm cfg using DIAGLEVEL 1
```

► The database configuration file

For each database there exists a database configuration file, which is created when a database is created. It resides in the directory where the database resides. It defines the resources and variables for the particular database. The values in the database configuration can be obtained using:

```
bash>get database configuration for itsodb2
```

and it can be set using:

```
bash>update database configuration for itsodb2 using ...
```

You can also use DB2 UDB *Configuration Assistant* to update DB2 UDB configuration parameters. Configuration Assistant is a graphical tool tightly integrated with Control Center. It allows you to configure both the DB2 UDB Profile Registry and the DB2 configuration files on the local machine as well as remotely. It can be launched from Control Center or by calling the **db2ca** utility. The Configuration Assistant also has an advanced view, which uses a notebook to organize connection information by object: Systems, Instance Nodes, Databases, Database Connection Services (DCS), and Data Sources. Figure 8-8 shows how to change the database manager configuration using Configuration Assistant.

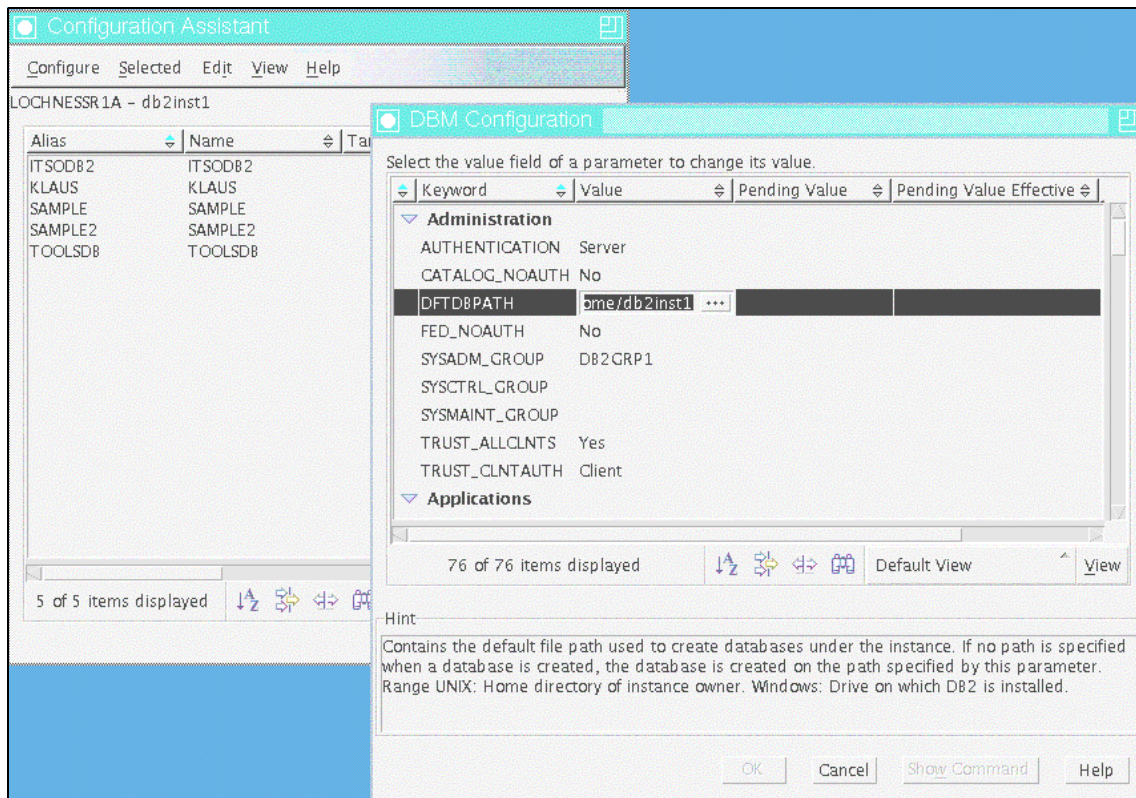


Figure 8-8 DB2 Configuration Assistant

8.7 Database management tools

In this section we introduce the graphical tools available for managing MySQL and DB2 UDB database servers. Although you can do all the tasks in both the database from shell command prompt, these tools play an important role in making the database administrators' job easier.

By default MySQL does not come with any control tool for database management, but MySQL AB provides MySQL Control Center, also known as MySQLCC for graphical administration of the MySQL server. Also, the SourceForge open source community provides a tool called phpMyAdmin, which is more popular, and can manage the MySQL server and database from the Web browser.

On the other hand, DB2 UDB supports impressive suite of tools for database management and services. All the daily database operations can be easily and effectively done using following GUI tools and wizards:

- ▶ Control Center

The Control Center is a tool provided by DB2 UDB to manage systems, DB2 instances, and database objects such as tables and views. It can be used to launch other centers and tools to help you optimize queries, jobs, and scripts, perform data warehousing tasks, create stored procedures, and work with DB2 commands. It can be opened by calling the **db2cc** command.

- ▶ Command Center

DB2 UDB Command Center interface can be used to execute SQL statements and DB2 CLP commands, to execute operating system commands, to work with command scripts, and to view a graphical representation of the access plan for explained SQL statements. It can be launched by calling the **db2cmdctr** command.

- ▶ Development Center

Development Center is easy to use interface for developing and debugging routines such as user defined functions and stored procedures.

- ▶ Project Deployment Center

This tool helps you deploy objects that you exported to a file system using the export wizard in the Development Center. This wizard can be launched using the **db2dcdp1** command.

- ▶ Journal

The Journal is the monitoring tool for viewing historical information generated within the Control Center and its components. It historical information about tasks, database actions and operations, Control Center actions, messages,

and alerts. It can be launched from Control Center or by calling the **db2journal** command.

▶ Replication Center

The Replication Center is a graphical user interface to administer the relational data replication. From the Replication Center, you can define your replication environments, copy designated changes from one location to another, and synchronize the data in both locations. Replication Center can be launched using the **db2rc** command.

▶ Task Center

The Task Center is a interface provided by DB2 UDB for scheduling, running tasks, and notifying people about the status of completed tasks. It can be executed by calling the **db2tc** command.

▶ Information Center

Information Center is central point for accessing documenting, help, instructions, conceptual information, answers, and other information about DB2 UDB. It can be activated using the **db2ic** command.

▶ Event Analyzer

An Event Monitor is a tool provided by DB2 UDB for collecting information on database activities such as transaction executed, CPU used by a statement etc. over a period of time. It provides a summary to determine how well a database activity was performed. Event Analyzer is a tool to analyze the data collected by Event Monitor. The Event Analyzer can be actuated by calling the **db2eva** command.

▶ Health Center

DB2 UDB Health Center is a tool provided to analyze and improve the health of DB2 UDB database. It provides an interface for determining and resolving problems related to memory, space, transaction, and other system resources. It can be launched by calling **db2hc**.

▶ Indoubt Transaction Manager

Indoubt Transaction Manager is a tool to administer global transactions that are left in an indoubt state. It can be used to initiate in-sync action or launch heuristic actions on those transactions. It can be triggered by calling the **db2indbt** command.

▶ Configuration Assistant

The Configuration Assistant is a tool to configure and maintain the database objects, bind application, set database configuration parameters, and import/export configuration information. It can be launched by issuing **db2ca** from your shell prompt.

- ▶ Satellite Synchronizer

Satellite Synchronizer can also be used to start, stop, and monitor the progress of a synchronization session, and to upload a satellite's configuration information to its control server.

Apart from this, DB2 UDB provides DB2 Web tools suite targeted for Web browsers, mobile laptops and notebooks, and Web-enabled PDAs and Palm devices. This suite includes:

- ▶ DB2 Web Command Center
- ▶ DB2 Web Health Center

IBM also provides DB2 for enhancing the performance of DB2 UDB on a multiplatform. The IBM DB2 offering includes:

- ▶ DB2 Performance Expert
- ▶ DB2 Recovery Expert
- ▶ DB2 High Performance Unload
- ▶ DB2 Table Editor
- ▶ DB2 Web Query Tool

8.7.1 MySQL phpMyAdmin and Control Center

phpMyAdmin is a SourceForge open community tool written in php for managing MySQL server and MySQL databases from a Web browser. It supports:

- ▶ Managing databases
- ▶ Table maintenance
- ▶ SQL-statement execution
- ▶ Exporting, loading and dumping of tables
- ▶ Managing users and privileges
- ▶ Checking referential integrity in MyISAM tables

Figure 8-9 shows phpMyAdmin console with the database structure of our sample application.

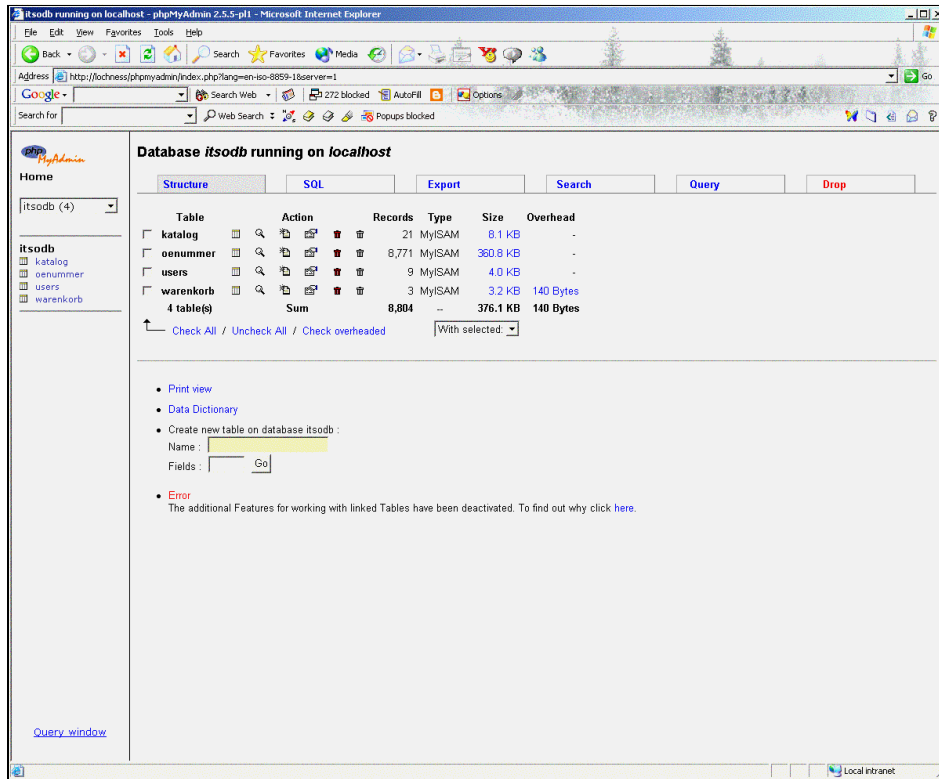


Figure 8-9 phpMyAdmin console

MySQL Control Center is a platform-independent GUI Administration Client for the MySQL database server, which can be used for table, database, or server administration. It can be used to construct queries using a syntax-highlighting text editor, and then views the results in a configurable table display. Currently, it is in beta stage. Figure 8-10 shows the usage for MySQL Control Center.

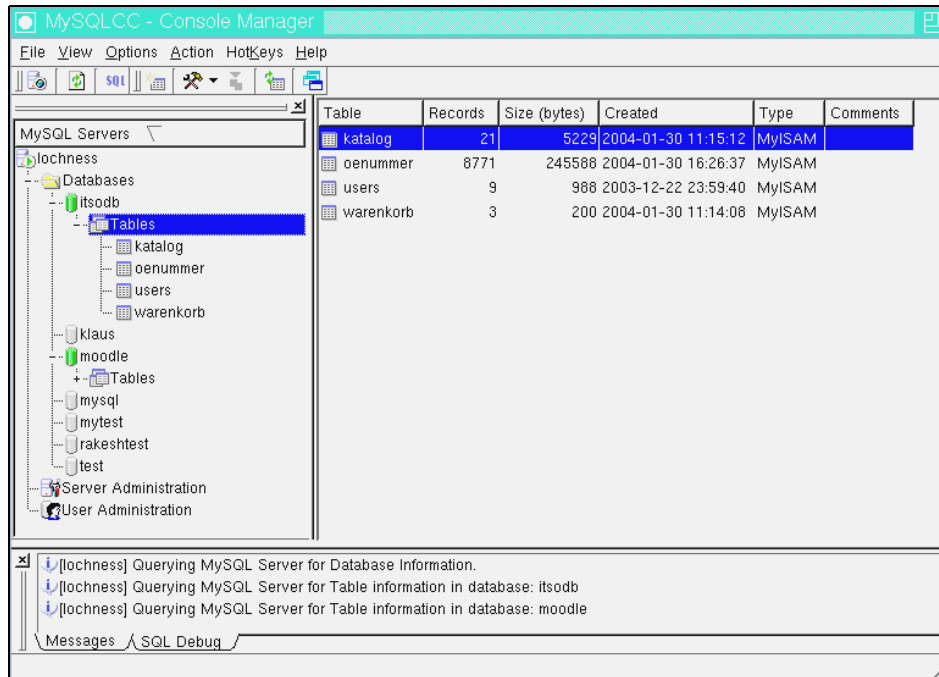


Figure 8-10 MySQL control center console

8.7.2 DB2 UDB Control Center

IBM DB2 Control Center is the central point from which you can manage your family of DB2 databases, running on an array of operating systems in your workplace. A user-friendly graphical interface makes your job easier by guiding you through each of the steps in managing DB2 Universal Database.

DB2 Control Center provides a common interface for managing DB2 databases on different platforms. You can run DB2 commands, create DDL statements, and execute DB2 utilities. DB2 Control Center's point-and-click navigation capabilities make it easy to find objects, whether you have hundreds or tens of thousands in your database environment. It can be used to administer system, instances, tables, views, indexes, triggers, user-defined types, user-defined functions, packages, aliases, users, or groups.

It is tightly coupled with other DB2 tools; Figure 8-11 shows a hierarchy of database objects on the left hand panel, and details on right hand panel.

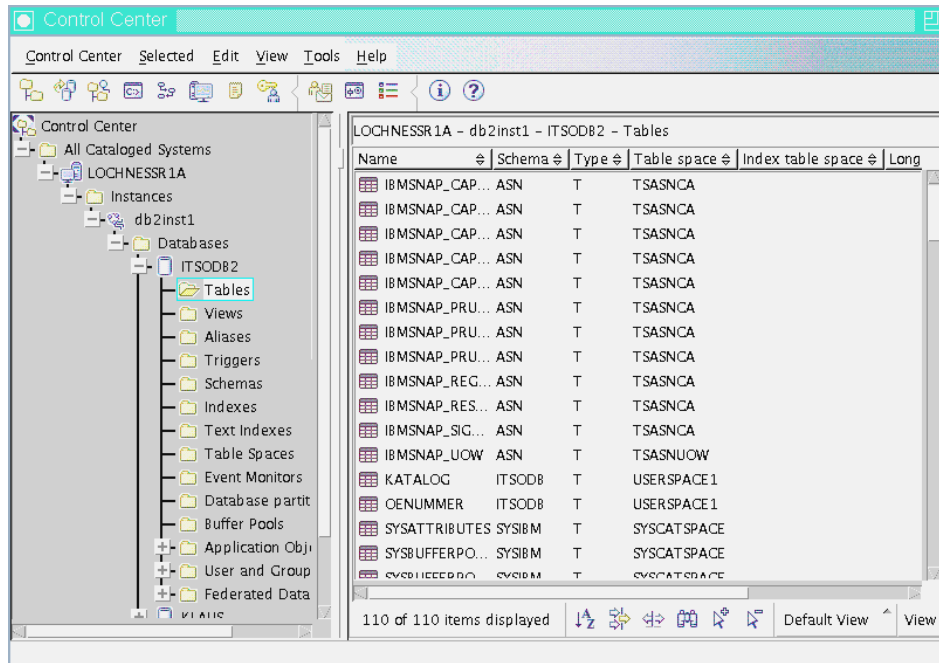


Figure 8-11 DB2 control center

The following tools can be launched from the Control Center Tools menu:

- ▶ Replication Center
- ▶ Satellite Administration Center
- ▶ Command Center
- ▶ Task Center
- ▶ Health Center
- ▶ Journal
- ▶ License Center
- ▶ Development Center

DB2 Control Center also provide set of wizards for completing specific administration tasks by taking you through each step one at a time. The following DB2 wizards are available though Control Center:

- ▶ Add partitions launchpad
- ▶ Backup wizard
- ▶ Create database wizard
- ▶ Create table space wizard
- ▶ Create table wizard
- ▶ Design Advisor
- ▶ Load wizard

- ▶ Configuration Advisor
- ▶ Restore data wizard
- ▶ Configure database logging wizard

8.7.3 DB2 UDB Web Command Center

Web Command Center is Web version of the Command Center and is targeted for Web browsers for mobile laptops, notepads, and Web-enabled PDAs and Palm devices. It supports most of the features of the Command Center, but it does not currently support Visual Explain or SQL Assist. It is based on a three-tier architecture wherein the first tier is a Web browser, the second is application server, and the third is DB2 Server. Figure 8-12 shows DB2 UDB Web Command Center in Internet Explorer.

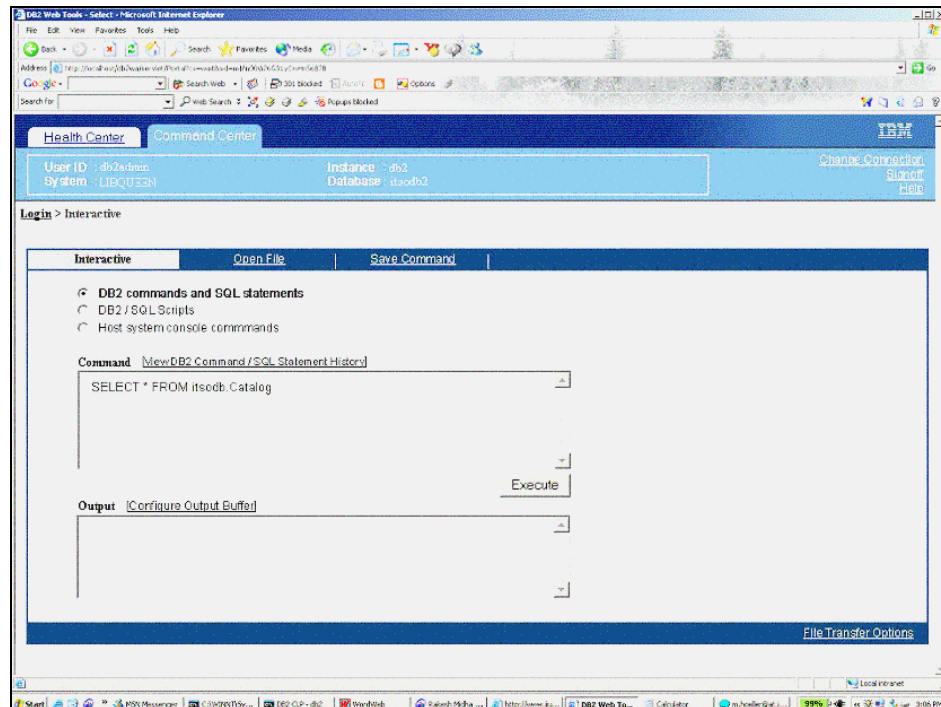


Figure 8-12 DB2 Web command center console

For using DB2 UDB Web Command Center, you need to install **db2wa.war** in your application server. This is available in the `sqllib/tools/web` directory.

It can be used for running:

- ▶ DB2 commands and SQL statements

- ▶ DB2/SQL scripts
- ▶ Operating system commands



Testing and tuning

This chapter discusses the following steps to verify that data and application functionality were ported completely and correctly:

- ▶ Test planning
- ▶ Data checking
- ▶ Code and application testing
- ▶ Troubleshooting

It provides information on how you can check that system behavior has not been changed during migration in a way that was not intended.

Methods and tools on how to tune your database in order to get as much performance as possible out of the DB2 UDB are also discussed in this chapter.

9.1 Test planning

The test planning details the activities, dependencies, and effort required to conduct the test of the converted solution.

9.1.1 Principles of software tests

Keep in mind the principles of software tests in general:

- ▶ It is not possible to test a non-trivial 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.

9.1.2 Test documentation

The test documentation is a very important part of the project. The *ANSI/IEEE Standard 829-1983 for Software Test Documentation* describes its content exactly. We give you a high level overview here.

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 and 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.

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 user's viewpoint of what the system does. This is not a technical description of the software, but a user's 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 user's viewpoint of what the system does, and a configuration management view. This is not a technical description of the software, but a user's view of the functions.

Identify *why* the feature is not to be tested; there can be any number of reasons.

Test strategy

This is your overall test strategy for this test plan. It should be appropriate to 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?
- ▶ Which combinations of hardware, software, and other vendor packages are used?

- ▶ 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 deliverable

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, etc.
- ▶ 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 tests defined. 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?

9.1.3 Test phases

Series of well designed tests should validate all stages of the migration process. A detailed test plan should describe all the 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 test 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 can 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, 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. Acceptance tests are usually the last step before going into production with the new system.

Post migration tests

Because 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.

9.1.4 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 9-1 shows the test phases during a typical migration project. The definition of the test plans happen in a very early moment. The test cases, and all its following tasks, must be done for all test phases.

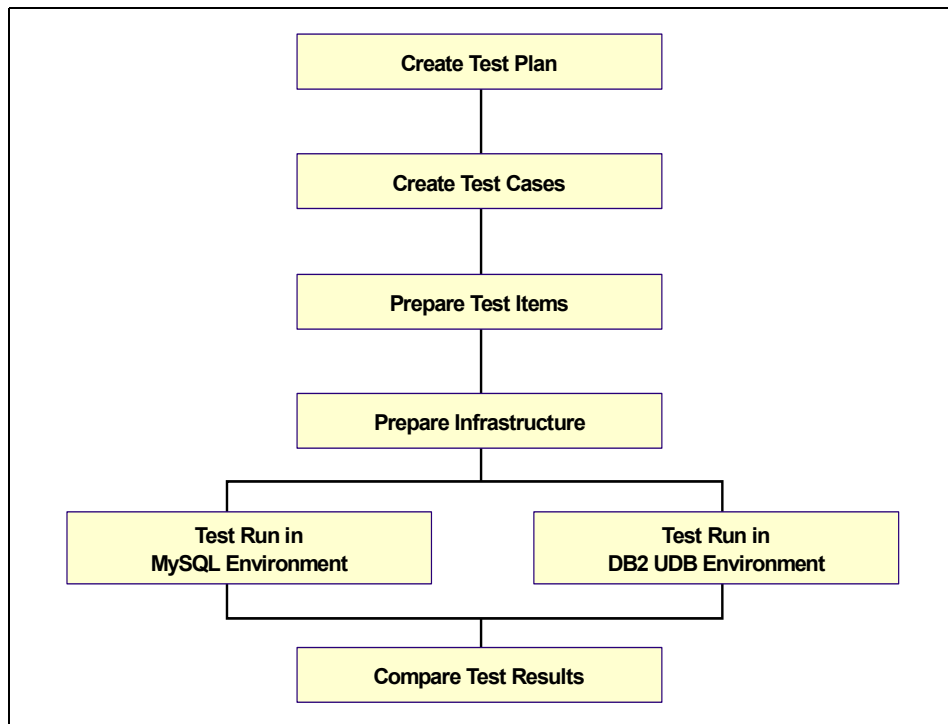


Figure 9-1 Test phases during a migration project

The time exposure of tests depends on the availability of an existing 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.

9.2 Data checking techniques

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.

9.2.1 **IMPORT/LOAD** messages

You should always check the messages generated by the **IMPORT** or **LOAD** commands. Example 9-1 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 9-1 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 "ITSO.TABLE01.SQL03081222227680". 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 9-1, the table definition is presented in Figure 9-2, and the data file in Figure 9-3.

```
CREATE TABLE TABLE01 (
  C1 SMALLINT,
  C2 CHAR(3),
  C3 SMALLINT CHECK( C3 IN (1,2,3)))
```

Figure 9-2 Table definition for Example 9-1

```
1,"abc",1
2,"abc",4
32768,"abc",2
4,"abcd",3
```

Figure 9-3 Data file for Example 9-1

The first row from the input file (Figure 9-3) was inserted without any warnings. The second row was rejected because it violated check constraints (warnings SQL3148W, SQL0545N, SQL3185W). A value of 32768 from the third row was

changed to null because it was out of 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 9-3) into *table01* (Figure 9-2) with the LOAD utility generates messages without SQL3148W, SQL0545N, SQL3185W warnings as shown in Example 9-2.

Example 9-2 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 check pending state. Accessing the table with SQL queries generates a 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 loaded table into a usable state. Example 9-3 shows a way to validate constraints. All rows that violated constraints will be moved to exception table *table01_e*.

Example 9-3 Turning integrity checking 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. To read more about the SET INTEGRITY command refer to:

<http://www-306.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8document.d2w/report?fn=r0000998.htm>

9.2.2 Data checking

Scripts performing logical data integrity checks automate the data verification process and save administrator effort.

For small tables (with less than 50,000 rows) you can write a program that compares data byte-by-byte. The program can extract sorted rows from MySQL and DB2 UDB to files in the same ASCII format. The files should be then binary compared (on Linux use the `diff` command) and checked to determine if they are the same.

For larger tables, comparing all rows byte-by-byte 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 MySQL database. Table CK_ROW_COUNT presented in Example 9-4 can be used for that purpose.

Example 9-4 Table for storing number of rows (MySQL)

```
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: MYS or DB2  
  TIME_INS DATE -- time when the count was performed
```

For each table you should count the 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(*), 'MYS', sysdate()  
from TAB_NAME
```

The table CK_ROW_COUNTS *and its data* can be manually migrated to the target DB2 database. Example 9-5 presents the DB2 version of the table.

Example 9-5 Table for storing number of rows (DB2 UDB)

```
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, DB2 table CK_ROW_COUNT should contain information about the number of rows counted on MySQL and DB2 databases. The records in the table should look like Example 9-6.

Example 9-6 Sample table CK_ROW_COUNTS contents

```
select TAB_NAME, ROW_COUNT, SYS_NAME, TIME_INS from CK_ROW_COUNT
[...]
```

TABLE_A	39001	MYS	2004-02-13-10.13.39
TABLE_A	39001	DB2	2004-02-13-10.32.13
TABLE_B	60003	MYS	2004-02-13-10.15.29
TABLE_B	60002	DB2	2004-02-13-10.33.49

```
[...]
```

Having the 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 database:

```
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)
```

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.
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 9-1 provides computations for selected database types. The argument for the DB2 SUM() function is converted to 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 9-1 Aggregations for data migration verification

Data type	MySQL operation	DB2 operation
numeric(<precision>,<scale>)	sum(<u>val</u>)	sum(cast(<u>val</u> as decimal(31,<scale>)))
date	sum(trunc(<u>val</u> - to_date('0001/01/02','yyyy/mm/dd')))	sum(cast(days(<u>val</u>) as decimal(31,1)))
variable length character	sum(length(<u>val</u>))	sum(cast(length(<u>val</u>) as decimal(31,0)))
fixed length character	sum(length(rtrim(<u>val</u>)))	sum(cast(length(rtrim(<u>val</u>)) as decimal(31,0)))

9.3 Code and application testing

The most important part of the testing process is to verify that each component of the system functions as it was before migrating. All components of the RDBMS system should be verified whether they are working properly.

9.3.1 Application code check

The scope of application testing depends on the migrated application.

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 screen form should be run and be checked for errors or improper functionality. All supported interfaces should also be checked.

The very important issue is to document all the tests conditions such as what operations were performed, which application screens were opened, what input data was used for testing, and what the result was. For larger projects, the

documenting part can become overwhelming, so usually specialized software is 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 be checked also against performance. Since there are many architectural differences between MySQL and DB2 UDB, some SQL operations might require further optimization. Observing the performance differences on 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, like locking issues, which can be observed only in a production environment.

9.3.2 Security testing

Before going into production, security must be checked in detail. MySQL handles security quite differently than DB2, so it is not trivial to compare the user rights between the two systems.

MySQL users and privileges are resolved in DB2 with operating system users and groups. A list of MySQL users should be compared to the 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 also should be verified.

9.3.3 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 Suite® TestStudio® can be used for that purpose.

IBM Rational Suite TestStudio is a set of tools for testers and developers. It automates regression, functionality, and performance testing, and provides background runtime analysis for increased product reliability. IBM Rational Suite TestStudio also includes tools for control, management, and reporting of all test activities, defect, and change tracking, software configuration management, and requirements management. IBM Rational Suite TestStudio addresses everything from test process standardization to results analysis, requirement determination to impact analysis, and test automation to defect tracking and reporting.

For more information about testing products, go to the IBM Rational Web site at:
<http://www.ibm.com/software/rational>

9.4 Troubleshooting

The first step of problem determination is to know what information is available to you. Whenever DB2 UDB 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 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 the problem
- ▶ Ask IBM for support

9.4.1 Interpreting DB2 informational messages

Start your investigation from the return code. DB2 UDB provides a return code for every operation performed in the form of CCCnnnnnS. The prefix CCC identifies the DB2 UDB component that is returning the message; the nnnnn is a four or five digit number which is also referred to as SQLCODE; and the S is a severity indicator. For example, in SQL0289N, the SQL identifier represents a message from the Database Manager, the SQLCODE is 0289, and N indicates an error message.

Here is the complete list for DB2 UDB error messages to prefix your reference:

- ▶ SQL: Database Manager messages
- ▶ DB2: Command Line Processor messages
- ▶ ASN: Replication messages
- ▶ CLI: Call Level Interface messages
- ▶ SQJ: Embedded SQLJ in Java messages
- ▶ SPM: Synch Point Manager messages
- ▶ DBI: Installation or configuration messages
- ▶ DBA: Control Center and Database Administration Utility messages
- ▶ CCA: Client Configuration Assistant messages

- ▶ DWC: Data Warehouse Center messages
- ▶ FLG: Information Catalog Manager messages
- ▶ SAT: Satellite messages

The three severity indicators are:

- ▶ W: Indicates warning or informational messages
- ▶ N: Indicates error messages
- ▶ C: Indicates critical system errors

DB2 UDB also provides detailed information for each message. The full error message describes the nature of the problem in detail and the potential user responses. To display the DB2 UDB return code full message, you can use the DB2 command `db2 ? error-code`. In Linux or AIX, since `?` (question mark) is a special character; you need to separate the DB2 command and the error code with a double quote (`"`). See Example 9-7.

Example 9-7 Explaining error codes

```
db2 "? sql0289"  
SQL0289N Unable to allocate new pages in tablespace  
        "<tablespace-name>".
```

Explanation:

One of the following conditions is true:

1. One of the containers assigned to this SMS tablespace has reached the maximum file size. This is the likely cause of the error.
2. All the containers assigned to this DMS tablespace 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 *Messages Reference*, Volumes 1 and 2, GC09-4840-00, and GC09-4841-00.

9.4.2 DB2 diagnostic logs

DB2 UDB 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 UDB. These logs are:

- ▶ `db2diag.log`

- ▶ Notify files
- ▶ Trap files
- ▶ Dump files
- ▶ Messages files

db2diag.log

The db2diag.log is the most often used file for DB2 problem investigation. You can find this file in the DB2 UDB 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:

Linux and 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 the *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 the 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, i.e. backup/restore start and finish

Example 9-8 contains an extract of a db2diag.log taken at DIAGLEVEL 3.

Example 9-8 Example of db2diag.log file

```
(1) 2004-02-10-02.26.33.004000 (2) Instance:DB2INST1 (3) Node:000
(4) PID:1012(db2syscs.exe) (5) TID:1996 (6) Appid:*LOCAL.DB2.00E049090924
(7) data management (8) sqlEscalateLocks (9) Probe:3 (10) Database:DB2_EMP2

(11) ADM5502W The escalation of "1251" locks on table "DB2INST1 .TABLE01" to
lock intent "X" was successful.
```

Explanations of the db2diag.log entries are included below. The number in parenthesis corresponds to the following numbers:

- ▶ (1) Date and timestamp of the entry made into the log
- ▶ (2) Name of the instance
- ▶ (3) Node or partition number
This number is always 0 in a single partition configuration.
- ▶ (4) Process ID of the application or agent
- ▶ (5) Thread ID of the application or agent
This is only used on the Windows platform.
- ▶ (6) The application ID
This corresponds to the LIST APPLICATIONS command output, each application has a unique application ID.
- ▶ (7) Component name
- ▶ (8) Name of the function in the component that is reporting an error or information
- ▶ (9) Probe point in the function
This corresponds to a location in the source code of the function that has returned an error or information.
- ▶ (10) Name of the accessed database that generated the message
- ▶ (11) Diagnostic information
In this example this is a administration warning telling about lock escalation (1251 row locks where successfully replaced by one table lock) on table DB2INST1.TABLE01.

Notify files

DB2 UDB also provides diagnosis information to the administration notification log in case of a failure. On Linux and UNIX platforms, the administration notification log is a text file called *<instance>.nfy*, where *<instance>* is the name of the instance. It is located in the same directory as the *db2diag.log* file. 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 DB2 UDB can write to the notify logs, but also the Health Monitor, the Capture and Apply programs, and user applications using the `db2AdminMsgWrite` API function.

Trap files

Whenever a DB2 UDB 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. The files are created using the following naming convention:

Linux and UNIX:

- ▶ `tpppppp.nnn`
 - `pppppp`: The process ID (PID)
 - `nnn`: The node where the trap occurred
 - Example: `t123456.000`

Windows:

- ▶ `DBppptt.TRP`
 - `ppp` : The process ID (PID)
 - `ttt` : The thread ID (TID)
 - Example: `DB123654.TRP`

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.

Dump files

When DB2 determines that internal information needs to be collected, it will often create binary dump files in the diagnostic path. 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 node 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 node where the problem occurred
 - Example: 123654.000

Messages files

Some DB2 UDB utilities like BIND, LOAD, EXPORT and IMPORT provide an option to dump out a messages file to a user-defined location. These files contain useful information to report the progress, success, or failure of the utility that was run.

9.4.3 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 version of DB2 UDB. As presented in Figure 9-4, the utility returns information about the installed maintenance updates (FixPaks), the length of word used by the instance (32-bit or 64-bit), the build date, and other code identifiers. It is recommended to periodically check if the newest available FixPaks are installed. DB2 maintenance updates are freely available at:

<ftp://ftp.software.ibm.com/ps/products/db2/fixes>

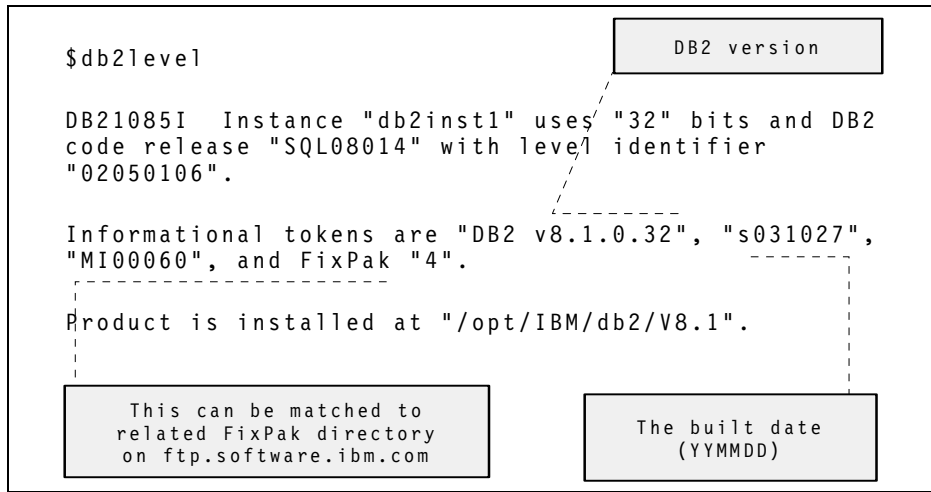


Figure 9-4 Sample db2level output

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 at the 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. Below is a sample call of the utility:

```
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 to gather information about database objects such as table spaces, tables, or packages.

DB2 Technical Support site

An invaluable place to look if experiencing a problem is the *DB2 Technical Support* site for Linux, Windows, and UNIX located on the Web at: <http://www.ibm.com/software/data/db2/udb/winos2unix/support>

The site has the most recent copies of the documentation, the knowledge base to search for technical recommendations or DB2 UDB defects, links for product updates, the latest support news, and many useful DB2 UDB related links.

To find related problems, prepare words that describe the issue like the commands that were run, the symptoms, and tokens from the diagnostics messages, and use them as each terms in the DB2 Knowledge Base. The Knowledge Base offers an option to search through DB2 UDB documentation, TechNotes, and DB2 UDB defects (APARs).

TechNotes is a set of recommendations and solutions for specific problems.

Authorized Program Analysis Reports (APARs) are defects in DB2's 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 UDB FixPaks.

On the DB2 support site there is a possibility to search for closed, open, and HIPER APARs. A status of closed APAR indicates the resolution for the problem has been verified and included in the FixPaks. Open APARs represent DB2 UDB defects that are currently being worked upon 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 Technical Support site offers e-mail notification of critical or pervasive DB2 UDB customer support issues including HIPER APARs and FixPak alerts. To subscribe to it, follow the **DB2 Alert** link on the Technical Support main page.

Calling IBM support

If the problem seems to be too complex to solve on your own, you can contact the *IBM Software Support Center*. In order to understand and resolve your support service request in the most expedient way, 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 calling IBM support) as well as the telephone numbers are available in the IBM Software Support Guide at:

<http://techsupport.services.ibm.com/guides/handbook.html>

9.4.4 Problem determination tools

Tuning and troubleshooting a database can be a complex process. DB2 UDB comes with a great number of tools, functions, and applications that make this task much simpler.

Monitoring tools

DB2 UDB monitoring utilities can collect information on many different system activities like usage of buffer pools, locks held by applications, sorts performed by 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. Because snapshots provide the point in time data, they are usually executed in scripts on regular intervals.

Snapshots can be taken from the command line, using custom API programs or through SQL using *table functions*. Example 9-9 shows the extract from a sample snapshot invoked from the command line.

Example 9-9 Example snapshot

```
db2 get snapshot for database on db2_emp
```

```
Database Snapshot
```

```
Database name           = DB2_EMP
Database path           =
/db2/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
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 counters' initialization. The counters can be reset to zero by the RESET MONITOR command, or by the appropriate DB2 event. With the mentioned Example 9-9, database deactivation and activation will reset all the database level counters. Example 9-10 shows how to reset monitors for an entire instance and for the specific database.

Example 9-10 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. Because of that reason, in Example 9-9 the value Not Collected was displayed for the buffer pool statistics. DB2 UDB 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 with the GET MONITOR SWITCHES command as shown in Example 9-11.

Example 9-11 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 02-10-2004 13:01:51.019864
Unit of Work Information	(UOW) = OFF

The monitor switches can be turned on at the instance level or at an application level. To switch the monitors at the instance level, modify the appropriate database manager parameter. After modifying the `DFT_MON_BUFFERPOOL` parameter, as shown in Example 9-12, all users with administration authorities will be able to collect buffer pool statistics on any database in the instance.

Example 9-12 Updating monitor switches at instance level

```
db2 update dbm cfg using DFT_MON_BUFFERPOOL ON
```

To switch the monitors at the application level, issue the **UPDATE MONITOR SWITCHES** command from the command line. The changes will only be applicable to that particular prompt window. Example 9-13 shows how to update the suitable monitor switch for collecting buffer pool information.

Example 9-13 Updating monitor switches at application level

```
db2 update monitor switches using BUFFERPOOL ON
```

The complete list of monitor switches and related database manager parameters is presented on Table 9-2.

Table 9-2 List of monitor switches and related DBM parameters

Database manager parameter	Monitor switch	Information provided
DFT_MON_BUFFERPOOL	BUFFERPOOL	Number of reads and 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)

Database manager parameter	Monitor switch	Information provided
DFT_MON_UOW	UOW (Unit Of Work)	Start/end times, completion status
DFT_MON_TIMESTAMP	TIMESTAMP	Timestamps

Sample snapshots

The *database manager snapshot* (Example 9-14) 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 9-14 Database manager snapshot

```
db2 get snapshot for database manager
```

The *lock snapshot* (Example 9-15) is very useful in determining what locks an application currently is 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 9-15 Lock snapshot

```
db2 get snapshot for locks on db2_emp
```

The *table snapshot* (Example 9-16) contains information on 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 9-16 Table snapshot

```
db2 get snapshot for tables on db2_emp
```

The *table space* and *buffer pool snapshots* (Example 9-17) contain similar information. The table space snapshot returns information on 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 pool, 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 9-17 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* (Example 9-18) 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 pre-bound will not show up in this list. The snapshot is an aggregate of the information concerning the SQL statements. If a SQL statement is executed 102 times, then there will be one entry with the summary of the total behavior of the 102 executions.

Example 9-18 Dynamic SQL snapshot

```
db2 get snapshot for dynamic sql on db2_emp
```

Snapshot table functions

As mentioned earlier, DB2 UDB features the capability to capture snapshots using SQL *table functions*. Accessing snapshot information through an SQL interface is very useful, because the requested information can be filtered and sorted, thereby presented in a more readable format. The snapshot *table functions* can be also very helpful in analyzing system utilization over a time period.

Most of the snapshot *table functions* accept two input parameters. The first is a string representing the database name. Entering NULL value for the database name parameter instructs the function to get snapshot information for all databases in the instance. The second parameter represents the partition number. To capture a snapshot for the currently connected partition, enter a value of -1 or a NULL.

The query in Example 9-19 uses the table function `SNAPSHOT_TABLE()` to retrieve the five table names, which have the most read and write activity on database `DB2_EMP`.

Example 9-19 Sample snapshot table function

```
db2 "select snapshot_timestamp, table_name, rows_written, rows_read,
      rows_written + rows_read as rows_accessed
from table (SNAPSHOT_TABLE('DB2_EMP', -1))as T
order by rows_accessed desc
fetch first 5 rows only"
```

TABLE_NAME	ROWS_WRITTEN	ROWS_READ	ROWS_ACCESSED
EMPLOYEE	0	256	256

STAFF	35	105	140
SYSTABLES	0	30	30
SYSROUTINES	0	10	10
INTERNAL	0	5	5

Example 9-20 illustrates a usage of the `SNAPSHOT_DYN_SQL()` function, which is very useful for finding the SQL statements that are taking the most time in the database.

Example 9-20 Sample snapshot table function

```
SELECT stmt_text, total_exec_time, num_executions
FROM TABLE( SNAPSHOT_DYN_SQL('DB2_EMP', -1)) as dynSnapTab
ORDER BY total_exec_time desc
FETCH FIRST 5 ROW ONLY
```

Example 9-21 finds the five SQL statements with the worst average execution time.

Example 9-21 Sample snapshot table function

```
SELECT CASE WHEN num_executions = 0
            THEN 0
            ELSE (total_exec_time / num_executions)
END avgExecTime,
num_executions,
stmt_text
FROM TABLE( SNAPSHOT_DYN_SQL('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, create a table based on the snapshot query such as presented in Example 9-22; include the `SNAPSHOT_TIMESTAMP` column in the snapshot query, and periodically store the results of the query in the table.

Example 9-22 Storing snapshot data in a table

```
db2 create table table_snap_hist as
(select snapshot_timestamp, table_name, rows_written, rows_read,
rows_written + rows_read as rows_accessed
from table (SNAPSHOT_TABLE('DB2_EMP', -1))as T) definition only

db2 "insert into table_snap_hist
select snapshot_timestamp, table_name, rows_written, rows_read,
rows_written + rows_read as rows_accessed
from table (SNAPSHOT_TABLE('DB2_EMP', -1))as T
```


order by rows_accessed desc fetch first 5 rows only"

Table 9-3 lists the more commonly used snapshot table functions. A complete list and detailed descriptions of snapshot table functions can be found in the *System Monitor Guide and Reference*, SC09-4847.

Table 9-3 Common snapshot table functions

Snapshot table function	Information returned
SNAPSHOT_DBM	Database manager information
SNAPSHOT_DATABASE	Database information. Information is returned only if there is at least one application connected to the database.
SNAPSHOT_APPL	General application information for each application that is connected to the database on the partition. This includes cumulative counters, status information, and the most recent SQL statement executed (if the statement monitor switch is set).
SNAPSHOT_APPL_INFO	General application identification information for each application that is connected to the database on the partition.
SNAPSHOT_LOCKWAIT	Application information regarding lock waits for the applications connected to the database on the partition.
SNAPSHOT_STATEMENT	Application information regarding statements for the applications connected to the database on the partition. This includes the most recent SQL statement executed (if the statement monitor switch is set).
SNAPSHOT_TABLE	Table activity information for each table that was accessed by an application connected to the database. Requires the table monitor switch.
SNAPSHOT_LOCK	Lock information at the database level, and application level for each application connected to the database. Requires the lock monitor switch.
SNAPSHOT_TBS	Information about table space activity at the database level, the application level for each application connected to the database, and the table space level for each table space that has been accessed by an application connected to the database. Requires the buffer pool monitor switch.
SNAPSHOT_BP	Buffer pool activity counters for the specified database. Requires the buffer pool monitor switch.

Snapshot table function	Information returned
SNAPSHOT_DYN_SQL	Point-in-time statement information from the SQL statement cache for the database.

Similar to snapshot commands, the amount of information returned from table snapshots functions is controlled by the monitor switches. Because 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 memory heap, 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. This monitor also has an additional option WITH DETAILS. This option will capture additional information, like which SQL statement was being executed when the deadlock occurred, and what locks were held by the application that encountered the deadlock. The information captured by the monitor focuses on the locks involved in the deadlock and the applications that own them.
- ▶ **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 ends. The monitor records statement's start and stop time, CPU used, text of dynamic SQL, the return code of the 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 9-23 creates a sample event monitor named `DEADLOCK_EVMON`. The query in the example accesses the `SYSCAT.EVENTMONITORS` view and displays names of event monitors that have been created in the database.

Example 9-23 Creating sample event monitor

```
db2 create event monitor deadlock_evmon for deadlocks with details
      write to table manualstart
```

```
db2 select evmonname from syscat.eventmonitors
```

The output of the `DEADLOCK_EVMON` monitor will be 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 `db2evtb1` tool as shown in the Example 9-24.

Example 9-24 Generating table syntax for specified event monitor

```
db2evtb1 -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,
```

```
[...]
```

Because 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 the value 1 and use the EVENT_MON_STATE() function to check for the current state as shown in Example 9-25 (when calling EVENT_MON_STATE() use the event monitor name in upper case). After activation of DEADLOCK_EVMON, each time a deadlock occurs in the database, it will be recorded into the event monitor tables.

Example 9-25 Enabling event monitor

```
db2 set event monitor deadlock_evmon state = 1

db2 values event_mon_state('DEADLOCK_EVMON')
```

To browse the data collected by the event monitor, you can directly access the tables, or use the GUI tool db2eva. For more information about db2eva, refer to the *Command Reference*.

Event monitors 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 an uncontrolled amount of data.

Example 9-26 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). If the total amount of collected data exceeds 5000 pages (4 KB) the event monitor will stop.

Example 9-26 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 user readable form use the db2evmon utility as shown in Example 9-27.

Example 9-27 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: SQL08014
Version of event monitor data: 7
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: SAMPLE
Database Path: C:\DB2\NODE0000\SQL00001\
First connection timestamp: 02-01-2004 13:25:43.028006
Event Monitor Start time: 02-06-2003 09:51:57.663712

3) Deadlock Event ...
Deadlock ID: 4
Number of applications deadlocked: 2
Deadlock detection time: 02-06-2003 09:53:11.952919
Rolled back Appl participant no: 2
Rolled back Appl Id: *LOCAL.DB2.010686063633
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 the 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 SQL** or **Show Explained Statements History** option. You can input an SQL statement manually or import the SQL statement through the **Get** button available in the Explain SQL window. You can also specify the optimization class for the SQL statement in the same window. The optimization class implies the effort the DB2 optimizer will spend on preparing the execution plan (higher value means more sophisticated optimization). Figure 9-5 shows an example of an access plan graph.

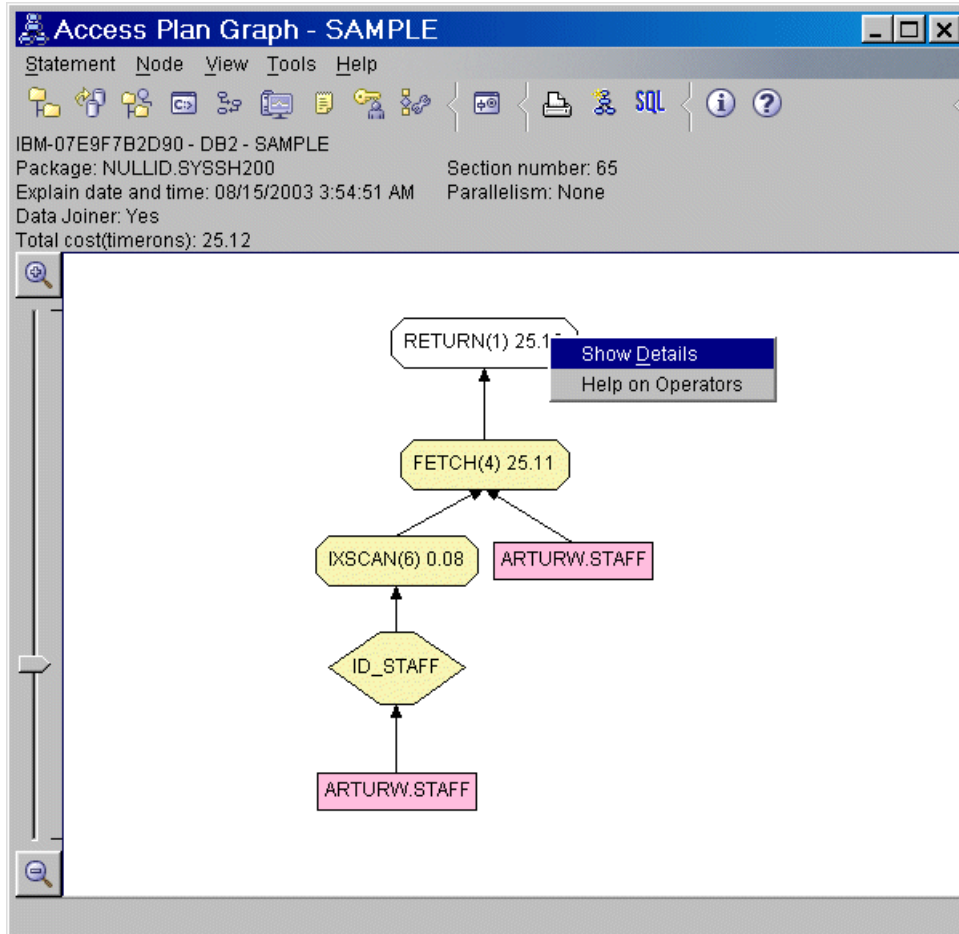


Figure 9-5 A Visual Explain access plan graph

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 right-click the desired graph element.

9.5 Initial tuning

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. Since databases created with default values are suited for computers with relatively small memory and disk storage, you may need to modify them to fit your environment. This section focuses on a number of DB2 UDB performance tuning tips that may be used for initial configuration.

9.5.1 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

By default all the three table spaces are created as *System Managed Spaces (SMS)*, which means that the regular operating system functions will be used for handling I/O operations.

Reading and writing data from tables will be buffered by the operating system, and space will be allocated according to the operating system conventions: files with *DAT* extension for tables and *INX* files for table indexes. When the table is initially created only one page is allocated on disk. When records are inserted into a table, DB2 UDB will extend the files by one page at a time.

On heavy inserts, extending files by only one page at a time can be very expensive. To minimize internal overhead for the table space extension, you can enable multi-page file allocation. With multi-page file allocation enabled for SMS table spaces, disk space is allocated one extent at a time (contiguous groups of pages defined for the table space).

To check whether the feature is enabled, look at the database configuration and search for the `Multi-page` word.

Example 9-28 Checking for current page allocation status

```
$db2 get db cfg for sample
...
Rollforward pending                = NO
Restore pending                    = NO

Multi-page file allocation enabled  = NO

Log retain for recovery status      = NO
```

```
User exit for logging status          = NO
...
```

In Example 9-28 Multi-page was not enabled. This can be changed by running the db2empfa program on the target database. Since db2empfa connects to the database in exclusive mode, all other users should be disconnected from the database. After db2empfa execution against the target database, check the multi-page file allocation parameter for the status (see Example 9-29).

Example 9-29 Enabling multi page allocation

```
$db2empfa sample
$db2 get db cfg for sample
...
Rollforward pending                 = NO
Restore pending                     = NO

Multi-page file allocation enabled   = YES

Log retain for recovery status       = NO
User exit for logging status         = NO
...
```

Better insert performance can be achieved with *Database Managed Spaces (DMS)* because containers are pre-allocated and management of the I/O operations is shifted to the database engine. In DB2 UDB Version 8 you can easily add new containers, drop, or modify the size of existing containers. Data is rebalanced to other containers automatically unless instructed. The administrative overhead is not so significant when compared to the SMS table spaces.

For optimal performance, large volume data and indexes should be placed on DMS table spaces; if possible, split to separate raw devices. Initially, system catalogs and system temporary table spaces should stay on the SMS table spaces. System catalogs contain large objects, which are not cached by the DB2 UDB engine, and can be cached by the operating system cache. In an OLTP 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.

9.5.2 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 as possible and at least five to ten disks per processor (for high I/O OLTP workload use even more). The reality is 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.

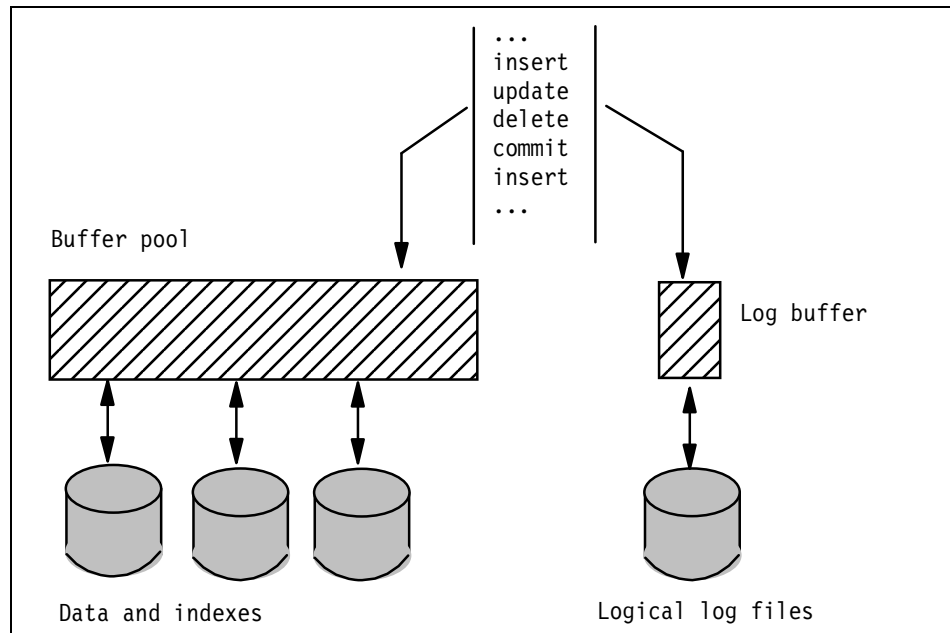


Figure 9-6 Explaining logical log

As shown in Figure 9-6, all data modifications are not only written to table space containers, but are also logged to ensure recoverability. Because 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 logical log buffer to disk by its nature is sequential and should not be interfered 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 9-30. The logs will be relocated to the new path on the next database activation (this can take some time to create the files).

Example 9-30 Relocation of logical logs

```
db2 update db cfg for sample using NEWLOGPATH /db2/logs
```

When creating a DMS table space with many containers, DB2 UDB 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_IO_CLEANERS) and I/O servers (NUM_IOSERVERS) should be adjusted. The best values for these two parameters depends on the type of workload and available resources. You can start your configuration with the following values:

- ▶ NUM_IOSERVERS = Number of physical devices, but not less than three and no more than five times the number of CPUs.
- ▶ NUM_IO_CLEANERS = Number of CPUs

Example 9-31 shows how to set the initial values of the parameters for two processor machines with six disks available to DB2.

Example 9-31 Updating IO related processes

```
db2 update db cfg for sample using NUM_IOSERVERS 6
db2 update db cfg for sample using NUM_IOCLEANERS 2
```

If there is a relatively small number of disks available, it can be difficult to keep logical logs, data, indexes, system temporary table spaces (more important for processing large queries in a warehousing environment), backup files, or the operating system paging file on separate physical devices. A compromise solution is to have one large file system striped by a 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. If you want parallel I/O operations on a single container, the DB2_PARALLEL_IO registry variable should be set before starting the DB2 UDB engine.

By performing 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 only for 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
```

9.5.3 Buffer pools

The default size for buffer pools is very small: only 250 pages (~ 1 MB) for Windows and 1000 pages (~ 4 MB) for Linux and UNIX platforms. The overall buffer size has a great effect on DB2 UDB performance since it can significantly reduce I/O, which is the most time consuming operation. We recommend to increase the default values. However, the total buffer pool size should not be set too high, because there might be not enough memory to allocate them. To calculate the maximum buffer size, all other DB2 memory related parameters like 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. DB2 version 8 allows changing buffer pool sizes without shutting down the database. The ALTER BUFFERPOOL statement with the IMMEDIATE option will take effect right away, 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 periodical changes in use, for example, switching from daytime interactive use to nighttime batch work.

Once the total available size is determined, this area can be divided into different buffer pools to improve utilization. Having more than one buffer pool can 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 that 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 additional buffer pools for caching data and leave the IBMDEFAULTBP for system catalogs. Creating an extra buffer pool for system temporary data also can be valuable for the 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 operation 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 9-32 shows how to create buffer pools assuming that an 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.

Example 9-32 Increasing buffer pools

```
connect to sample;
  -- creating two buffer pools 256 MB and 64 MB
create bufferpool DATA_BP immediate size 65536 pagesize 4k;
create bufferpool TEMP_BP immediate size 16384 pagesize 4k;

  -- changing size of the default buffer pool
alter bufferpool IBMDEFAULTBP immediate size 16384;

  -- binding the tablespaces 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	16384	4096	SYSCATSPACE
IBMDEFAULTBP	16384	4096	USERSPACE1
DATA_BP	65536	4096	DATASPACE
TEMP_BP	16384	4096	TEMPSPACE1

The `CHNGPGS_THRESH` 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 on Figure 9-7. To guarantee more consistent response time and also shorter recovery phase, lower the value to 50 or 40 using the following command:

```
db2 update db cfg for sample using CHNGPGS_THRESH 40
```

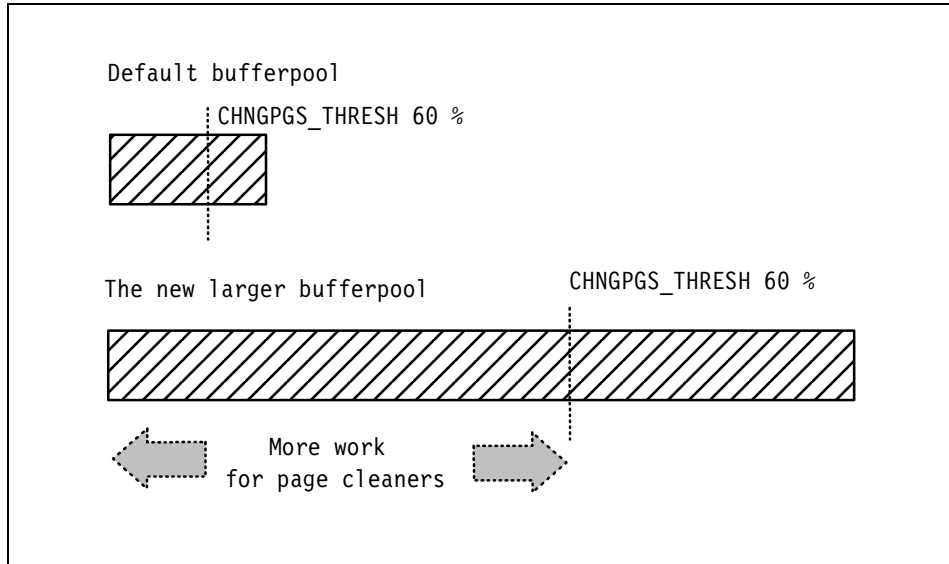


Figure 9-7 Visualizing CHNGPGS_THRESH parameter

9.5.4 Large transactions

By default, databases are created with relatively small space for transactional logs, only three log files with each 250 pages on Windows and 1000 pages on Linux and UNIX.

A single transaction should fit into the available log space to be completed; if it does not fit, the transaction is rolled back by the system (SQL0964C The transaction log for the database is full). To process transactions which are modifying large numbers of rows, adequate log space is needed.

The currently total *log space* available for transactions can be calculated by multiplying the size of one log file (database parameter LOGFILSIZ) and the number of logs (database parameter LOGPRIMARY).

From the performance perspective, it is better to have a larger log file size because of the cost for 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.

The following Example 9-33 allocates 400 MB of total log space.

Example 9-33 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, which 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 is 50 pages (200 KB) for Windows and 100 pages (400 KB) for Linux and UNIX. On 32-bit platforms, each lock requires 36 or 72 bytes of the lock list, depending on whether other locks are held on the object or not. For the default values, the maximum of 5688 (Windows) or 11377 (Linux and UNIX) locks can be allocated as shown in Figure 9-8.

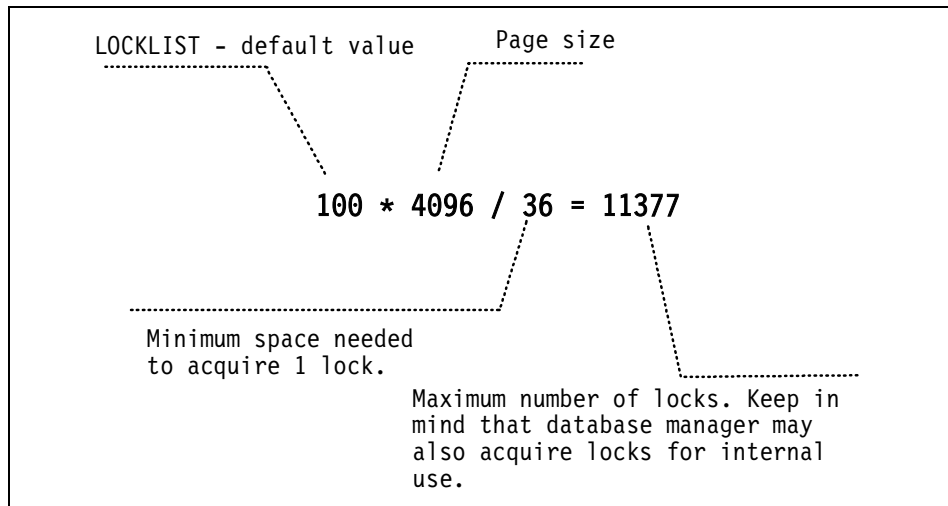


Figure 9-8 Maximum number of locks available for default settings on Linux

When the maximum number of lock requests has been reached the database manager will replace existing row level locks with table locks (*lock escalation*). This operation will reduce the requirements for locks 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.

The lock escalation can also be forced by the MAXLOCKS database parameter, which defines a limit for the maximum percentage of the lock storage held by one application. The default value for Linux and UNIX is 10 (22 for Windows), which means that if one application requests more than 10% of total locks space (LOCKLIST), an escalation will occur for the locks held by that application. As an example, inserting 1137 rows on Linux with one transaction will result in lock escalation, because the transaction requests 1138 locks (one per each inserted row plus one internal lock), which requires at least $1138 * 36 = 40968$ bytes - more than 10% of global lock memory defined by the default LOCKLIST parameter.

Initial values for LOCKLIST and MAXLOCKS should be based on the maximum number of applications and average number of locks requested by the transaction (for OLTP systems start with 512 locks for every application). When setting MAXLOCKS, you should take into account lock consuming batch processes that run during daytime hours. To check current usage of locks use snapshots such as in Example 9-34.

Example 9-34 Invoking snapshot for locks on database sample

```
db2 get snapshot for locks on sample
```

The snapshot will collect the requested information at the time the command was issued. On Figure 9-9 you can find a sample lock snapshot output. For the time the snapshot was run there were two applications connected to the database SAMPLE, 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 the locks.

```

Database Lock Snapshot

Database name           = SAMPLE
Database path          =
/home/db2inst1/db2inst1/NODE0000/SQL00001/
Input database alias   = SAMPLE
Locks held              = 1151
Applications currently connected = 2
Agents currently waiting on locks = 0
Snapshot timestamp     = 02-10-2004 16:38:58.759890

Application handle     = 21
Application ID         = *LOCAL.db2inst1.0C3B32182808
Sequence number        = 0001
Application name       = db2bp
CONNECT Authorization ID = DB2INST1
Application status     = UOW Waiting
Status change time    = Not Collected
Application code page  = 819
Locks held             = 13
Total wait time (ms)  = Not Collected

List Of Locks
Lock Name              = 0x00030005001052540000000052
Lock Attributes        = 0x00000000
Release Flags          = 0x40000000
Lock Count             = 255
Hold Count             = 0
Lock Object Name       = 1069652
Object Type            = Row
Tablespace Name        = DATASPACE
Table Schema           = DB2INST1
Table Name             = TABLE01
Mode                   = X

Total number of locks allocated for the database
Total number of locks held by the application
This lock was placed on a row in table "TABLE01" by application *LOCAL.db2inst1.0C3B32182808

```

[... the listing was cut here ...]

Figure 9-9 Explaining lock snapshot information

To check lock escalations occurrences look at the *db2diag.log* file. The lock escalation message should look like Example 9-35.

Example 9-35 Lock escalation message in db2diag.log file

```

2004-02-10-19.05.05.888741 Instance:db2inst1 Node:000
PID:56408(db2agent (SAMPLE) 0) TID:1 Appid:*LOCAL.db2inst1.0DB5F2004313
data management sqlEscalateLocks Probe:3 Database:SAMPLE
ADM5502W The escalation of "1136" locks on table "DB2INST1.TABLE01" to lock
intent "X" was successful.

```

Logical log buffer

The default size for logical log buffer is eight pages (32 KB), which is 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 issue 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. 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 presented in Example 9-36.

Example 9-36 Current usage of log space by applications

```
$db2 update monitor switches using uow on
$db2 get snapshot for applications on sample | 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 9-37 shows how to get information about log I/O activity.

Example 9-37 Checking log I/O activity

```
db2 reset monitor for database sample
   # let the transactions run for a while
```

```
db2 get snapshot for database on sample > 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 the snapshot 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 Linux **grep/egrep** 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 the average size of transactions was, 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 the log buffer to the 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 guarantees 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.

9.5.5 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 manually with the **RUNSTATS** utility, and are stored in special system catalog tables. The **RUNSTATS** command should be executed 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 executed 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 The DB2 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 9-10.

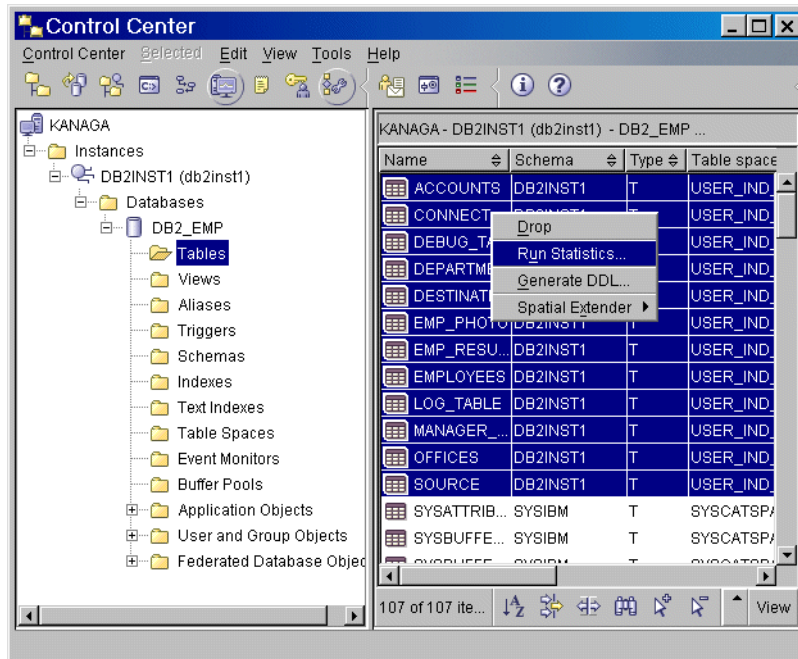


Figure 9-10 Running RUNSTATS on multiple tables

On the first **Tables** tab move all items from the **Available** list to the **Selected** list by clicking the >> button. Figure 9-11 presents the sample result of the operation.

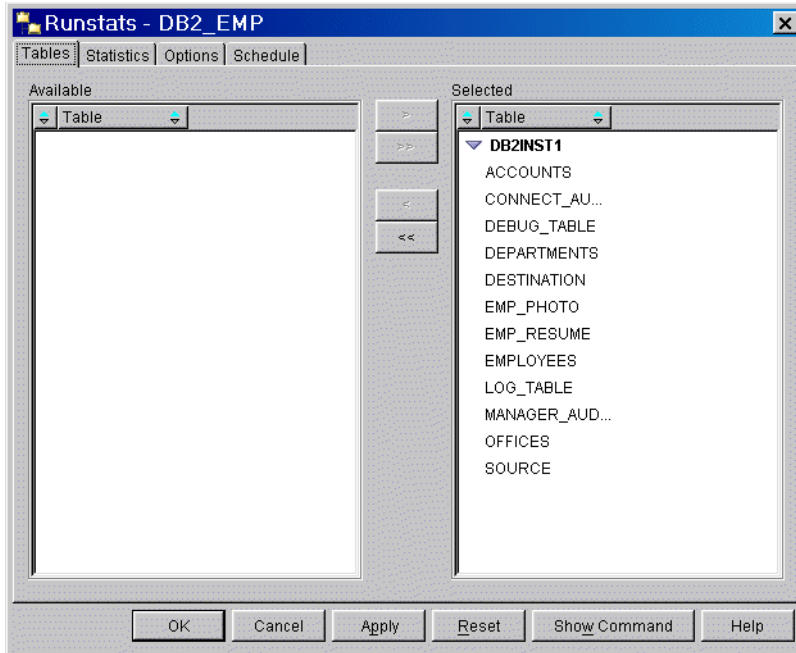


Figure 9-11 Selecting tables for RUNSTATS command

On the **Statistics** tab, 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 like presented in Figure 9-12. After setting the **RUNSTATS** options, you can execute the commands by clicking the **Apply** button.

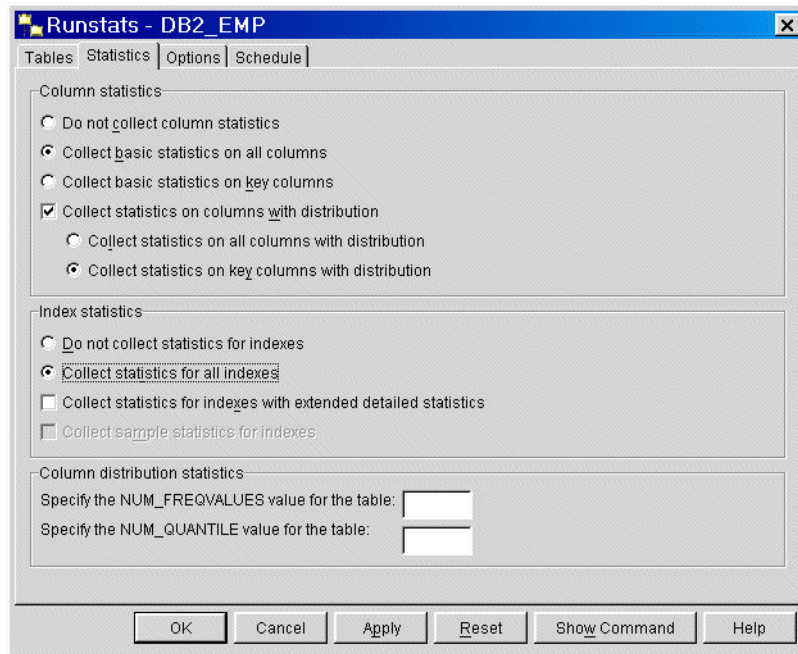


Figure 9-12 RUNSTATS command options

DB2 UDB 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 utility.

9.5.6 Configuration Advisor

The Configuration Advisor wizard is a GUI tool that can be helpful in preparing the DB2 initial configuration. The wizard requests information about the database, its data, and the purpose of the system, and then recommends new configuration parameters for the database and the instance.

To invoke this wizard from the DB2 Control Center, expand the object tree until you find the database that you want to tune. Select the icon for the database, right-click and select **Configuration Advisor**. Through several dialog windows the wizard collects information about the percentage of memory dedicated to DB2, type of workload, number of statements per transaction, transaction throughput, trade-off between recovery and database performance, number of

applications, and isolation level of applications connected to the database. Based on the supplied answers, the wizard proposes configuration changes and gives the option to apply the recommendations or save them as a task for the Task Center for later execution as shown in Figure 9-13. The result window is presented in Figure 9-14.

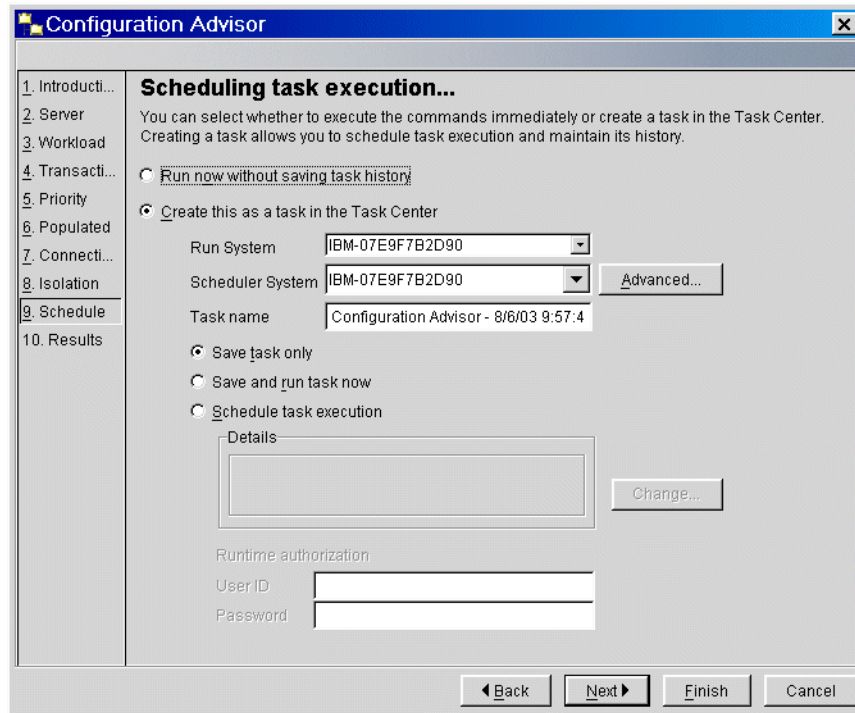


Figure 9-13 Scheduling Configuration Advisor recommendations

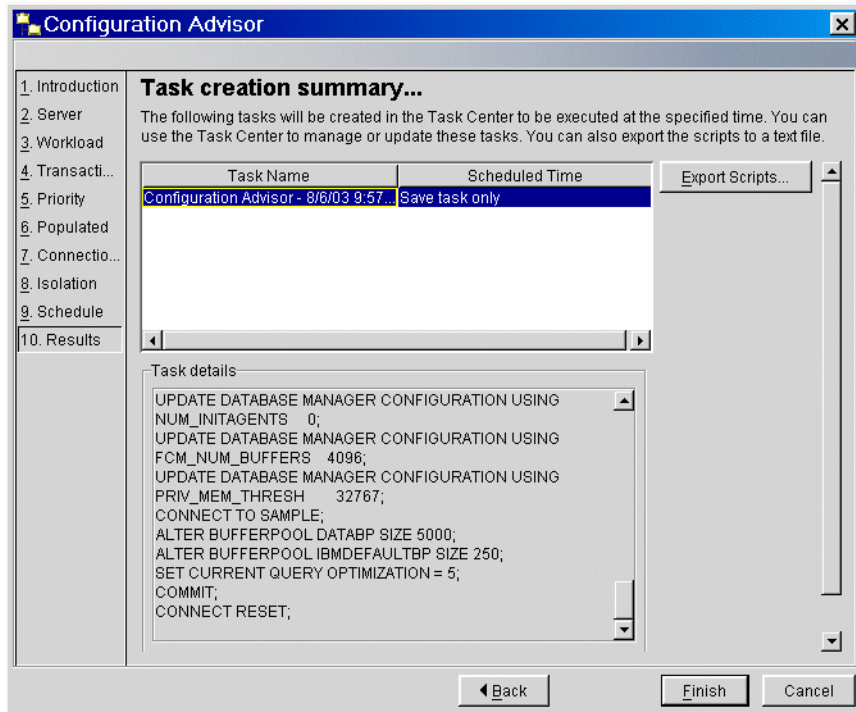


Figure 9-14 Configuration Advisor recommendations

Initial configuration recommendations can also be acquired through the text based **AUTOCONFIGURE** command. Example 9-38 shows the sample execution of the command.

Example 9-38 Sample 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 9-4 lists all **AUTOCONFIGURE** command parameters.

Table 9-4 Parameters for the autoconfigure command

Keyword	Values	Explanation
mem_percent	1-100 default: 80	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?

9.5.7 Index Advisor

Well designed indexes are essential to database performance. DB2 UDB comes with a utility called the Index Advisor, which can recommend indexes for specific SQL queries. Index Advisor can be invoked either using the **db2adv i s** command or using the Design Advisor wizard from the Command Center or Control Center. This utility accepts one or more SQL statements and their relative frequency, known as a workload.

The Index Advisor is good for:

- ▶ Finding the best indexes for a problem query

- ▶ Finding the best indexes for a specified workload. When specifying the workload, you can use the frequency parameter to prioritize the queries. You can also limit disk space for the target indexes.
- ▶ Testing an index on a workload without having to create the index

Example 9-39 shows a simple `db2adv is` call against a single SQL query; for more options run `db2adv is -h` from the command line.

Example 9-39 Finding indexes for a particular query

```
db2adv is -d db2_emp -s "select first_name, last_name, dept_name from
departments d, employees e where d.dept_code = e.dept_code and e.last_name like
'W%'"
```

```
execution started at timestamp 2004-02-10-14.15.00.408000
recommending indexes...
Initial set of proposed indexes is ready.
Found maximum set of [2] recommended indexes
Cost of workload with all indexes included [0.155868] timerons
total disk space needed for initial set [ 0.018] MB
total disk space constrained to [ -1.000] MB
 2 indexes in current solution
[ 50.3188] timerons (without indexes)
[ 0.1559] timerons (with current solution)
[%99.69] improvement

Trying variations of the solution set.
 2 indexes in current solution
[ 50.3188] timerons (without indexes)
[ 0.1559] timerons (with current solution)
[%99.69] improvement
--
--
-- LIST OF RECOMMENDED INDEXES
-- =====
-- index[1], 0.009MB
CREATE UNIQUE INDEX IDX030801141500000 ON "DB2INST1"."DEPARTMENTS"
("DEPT_CODE" ASC) INCLUDE ("DEPT_NAME") ALLOW REVERSE SCANS ;
COMMIT WORK ;
--RUNSTATS ON TABLE "DEPARTMENTS" FOR INDEX "IDX030801141500000" ;
COMMIT WORK ;
-- index[2], 0.009MB
CREATE INDEX IDX030801141500000 ON "DB2INST1"."EMPLOYEES" ("LAST_NAME" ASC,
"FIRST_NAME" ASC, "DEPT_CODE" ASC) ALLOW REVERSE SCANS ;
COMMIT WORK ;
--RUNSTATS ON TABLE "EMPLOYEES" FOR INDEX "IDX030801141500000" ;
COMMIT WORK ;
-- =====
```

--

DB2 Workload Performance Advisor tool is finished.

Launching the Index Advisor in a GUI environment

The **Index 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 all the necessary steps, and also helps to 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 9-15 presents a sample **Design Advisor** window.

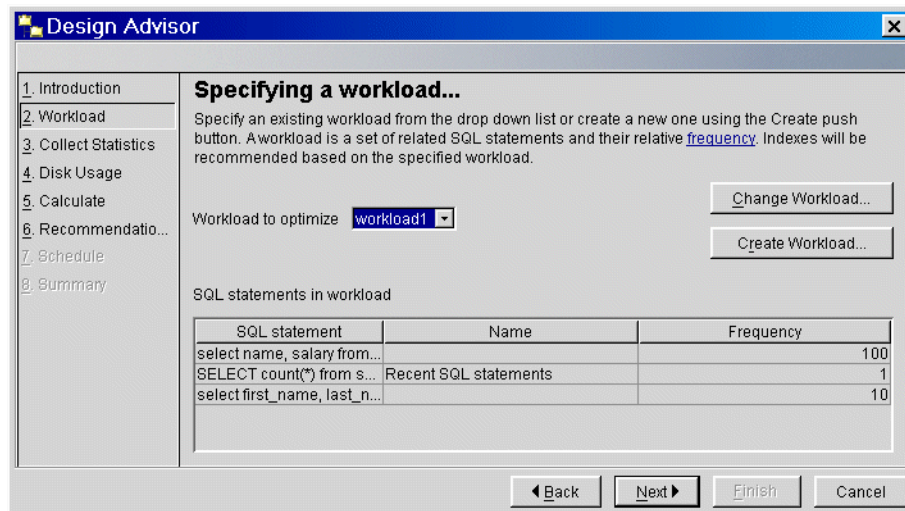


Figure 9-15 The Design Advisor

The detailed usage of Design Advisor can be found in the following redbooks:

- ▶ *DB2 UDB Evaluation Guide for Linux and Windows*, SG24-6934
- ▶ *DB2 UDB Exploitation of the Windows Environment*, SG24-6893
- ▶ *Up and Running with DB2 for Linux*, SG24-6899



Advanced DB2 UDB features

This chapter is about the advanced features DB2 UDB offers. When migrating a database from MySQL to DB2 UDB, you should consider which DB2 UDB features and functions can be used to enhance your application.

The features covered in this chapter are:

- ▶ Views
- ▶ Stored procedures
- ▶ Trigger
- ▶ User defined data types (UDT)
- ▶ User defined functions (UDF)
- ▶ Materialized query tables (MQT)
- ▶ Multidimensional clustering (MDC)

10.1 Views

A view provides a different way of looking at the data in one or more tables. It is a named specification of a result table.

Note: The specification is a SELECT statement that is run whenever the view is referenced in an SQL statement.

A view has columns and rows just like a base table. All views can be used just like base tables for data retrieval. Whether a view can be used in an INSERT, UPDATE, or DELETE operation depends on its definition.

You can use views to control access to sensitive data, because views allow multiple users to see different presentations of the same data. For example, several users may be accessing a table of employee data. A manager sees data about his employees but not employees in another department. A recruiting officer sees the hire dates of all employees, but not their salaries. A financial officer sees the salaries, but not the hire dates. Each of these users works with a view derived from the same base table. Each view appears to be a table and has its own name.

When the column of a view is directly derived from the column of a base table, that view column inherits any constraints that apply to the base table column. For example, if a view includes a foreign key of its base table, INSERT and UPDATE operations using that view are subject to the same referential constraints as in the base table. Also, if the base table of a view is a parent table, DELETE and UPDATE operations using that view are subject to the same rules as DELETE and UPDATE operations on the base table.

A view can derive the data type of each column from the base table, or base the types on the attributes of a user-defined structured type (*typed view*).

A view can become inoperative (for example, if the base table is dropped). If this occurs, the view is no longer available for SQL operations.

Example 10-1 shows some examples for views that could be used in our sample project.

Example 10-1 Example views for our sample project

```
-- A view that shows all existing groups of products in our catalog
-- and the number of items associated to them
create view GROUPS as
    (select PRODGR, count(*) as CNT from ITSODB.CATALOG group by PRODGR)
```

```

-- A view that shows all existing manufacturers in our catalog
create view MANUFACTURER as
  (select distinct MANUFACTURER from ITSODB.CATALOG);

-- A view that shows the inventory of each item in the catalog
create view INVENTORY as
  (select ID, SKU, STOCK from ITSODB.CATALOG);

-- A view that shows all products in the catalog
create view TYPES as
  (select ID, MANUFACTURER, MODEL, TYPE from ITSODB.CATALOG);

```

- ▶ **View GROUPS**
This view shows all existing groups of products in our sample catalog and the number of items associated to them. Granting privileges to users just on this view is a way to limit access for these users just on this data and not on the whole catalog.
- ▶ **View MANUFACTURER**
This view provides information about the different manufacturers whose products are in our catalog. This view provides a fast way to find out all these manufacturers.
- ▶ **View INVENTORY**
This view shows the inventory of all items in the catalog. Again, granting privileges on this view could be used for limited access to the catalog data (e.g., users having access to this table may see the inventory of the item, but not the price for which it is bought and sold).
- ▶ **View TYPES**
This view is a qualified overview of all products in the catalog that could be used for a limited view on the catalog.

Example 10-2 shows the sample output of these views. Please notice that the SELECT statements on the views are much simpler than the SELECT statements on the tables the view is based on.

Example 10-2 Sample view data

```

db2> select * from GROUPS
PR CNT
-- -----
10      16
85      5

```

```

db2> select * from MANUFACTURER
MANUFACTURER
-----
AUDI
VW

db2> select * from INVENTORY fetch first 10 rows only
ID      SKU      STOCK
-----
      2 16633      10
      3 16633      10
      5 16633       7
      8 951041     10
      9 951041     10
     12 951041      9
     14 951041     10
     15 951041     10
     17 16272      9
     19 16272     10

db2> select * from TYPES fetch first 10 rows only
ID      MANUFACTURER  MODEL              TYPE
-----
      2 VW          PASSAT (3B2)       1.6
      3 VW          PASSAT (3B2)       1.8
      5 VW          PASSAT (3B2)       1.8 T
      8 VW          POLO (6N1)         45 1.0
      9 VW          POLO (6N1)         50 1.0
     12 VW          POLO (6N1)         55 1.3
     14 VW          POLO (6N1)         55 1.4
     15 VW          POLO (6N1)         60 1.4
     17 AUDI        80 Avant (8C, B4)  2.0
     19 AUDI        80 Avant (8C, B4)  2.0 E

```

10.2 Stored procedures

An extension to SQL defined by ANSI (ISO/IEC 9075) is called SQL/PSM or *persistent, stored modules* and extends SQL to store application logic in the form of procedures, functions, or triggers as database objects. The programming style is a mix of conventional SQL statements combined with procedural logic; for example, IF, WHILE for flow control. PSMs allow us to do things we cannot do in SQL alone.

Reasons for implementing stored procedures may be:

- ▶ In today's globalized world it is very likely that multiple applications written in multiple languages working on multiple platforms may use the same database for multiple purposes. A stored procedure may ensure that this database stays consistent and applies to the business rules defined by the owner of the database.
- ▶ Enriching and securing the application. For example, if there is a requirement for additional auditing that is beyond the DBMS capabilities, stored procedures can provide additional function.

This section gives you a brief introduction to the implementation of SQL/PSM as SQL stored procedures in DB2 UDB. The introduction includes the setup of the application development environment and SQL procedure fundamentals.

Setting up the environment

To successfully work with DB2 UDB stored procedures, the DB2 Application Development Client and a DB2 supported C/C++ compiler are required. Here is a list of supported compilers for a UNIX environment:

- ▶ AIX: IBM VisualAge® C++ 5.0
- ▶ Solaris: Forte C++ Version 5.0
- ▶ Linux: GNU/Linux g++
- ▶ HP-UX: HP aC++ Version A.03.31

You may have to configure your C/C++ compiler environment if the default setting is not appropriate. Use the **db2set** command to configure the DB2 registry:

- ▶ DB2_SQLROUTINE_COMPILER_PATH and
- ▶ DB2_SQLROUTINE_COMPILE_COMMAND

Example 10-3 illustrates the use of the **db2set** command to set the compile command to a non-default compiler.

Example 10-3 Configuring the compiler environment

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=cc -fpic -D_REENTRANT \
      -I$HOME/sql1lib/include SQLROUTINE_FILENAME.c \
      -shared -lpthread -o SQLROUTINE_FILENAME -L$HOME/sql1lib/lib -ldb2
```

To ensure your application development environment is set up correctly, you may want to run the CREATE statement shown in Example 10-4 as the most simple procedure possibly created.

Example 10-4 Simple CREATE PROCEDURE

```
db2 =>
db2 => create procedure myfirst begin end
```

DB20000I The SQL command completed successfully.
db2 =>

Stored procedure fundamentals

Figure 10-1 illustrates the creation process for stored procedures and the objects a stored procedure is composed of:

- ▶ The **create procedure** command invokes the SQL precompiler, which replaces SQL statements with appropriate API calls. The result is for this example a C source file with the extension *.sql*.
- ▶ During the further compilation of the stored procedure, it is split into a file containing the executable code plus a *bind* file, which ultimately is stored in DB2 UDB as an SQL package.
- ▶ Code level consistency between the *executable* object and the *package* object stored within the db2 engine is ensured by a *consistency token*, which is exchanged on every execution.

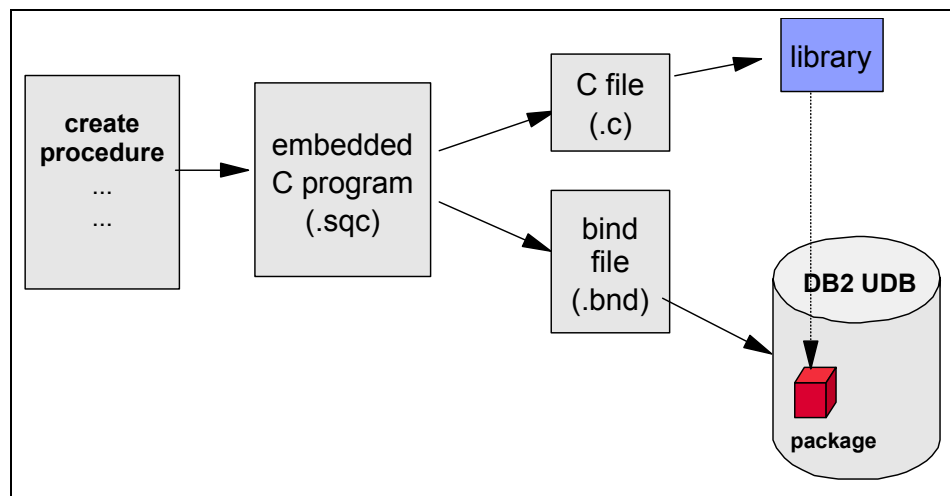


Figure 10-1 Creating stored procedures

Stored procedures can get quite complex. For details please refer to *Application Development Guide: Programming Server Applications*, SC09-4827.

The basic form a stored procedure looks like:

```
CREATE PROCEDURE proc [( {optional parameters} )]  
[optional procedure attributes]  
<statement>
```


whereby <statement> is considered to be a *single* statement even if a set of statements is grouped between *begin* and *end*. Unlike the usual parameter style, SQL stored procedures use a mode-name-type triplet for argument declaration, where *mode* can be:

- ▶ IN: Input value is not changed by procedural code
- ▶ OUT: Procedural code modifies argument but is not required as input parameter. In the CALL statement OUT parameters need to be supplied as parameter markers.
- ▶ INOUT: Value is supplied by the caller, returned to the caller and may be modified in between

In 7.2.6, “Condition handling in DB2” on page 213, you can find additional code snippets of stored procedures dealing specifically with condition handling.

Stored procedure considerations

While stored procedures are often implemented to achieve performance improvements and often yield such benefits, it is not necessarily guaranteed that your queries run faster after moving them to a stored procedure. Improvements are not as easily achieved as moving a statement from the application to a stored procedure.

Indeed, there are circumstances under which the use of stored procedures can actually cause performance to degrade. For example, if you create a stored procedure that simply issues one INSERT statement and calls this procedure from a remote application, network traffic will not be reduced. Your application still has to make a network call to invoke the procedure, just as it would to issue an INSERT statement.

Furthermore, the DBMS may need to load the procedure and incur inter-process communications overhead to execute it. Thus, your application may actually run slower by using such a stored procedure.

If you are planning on creating new stored procedures to support, or if you are trying to tune existing stored procedures, you should be aware that language issues and creation options can significantly influence your results. For example, in some DBMS products, procedures written in Java may perform more poorly than an equivalent procedure written in C or SQL. In addition, procedures that run in a separate address space from the DBMS (“fenced” procedures) perform more poorly than procedures that run in the same address space as the DBMS (“unfenced” procedures).

10.3 Trigger

Similar to SQL stored procedures, database triggers are an implementation of SQL/PSM. A database trigger is a database object containing application logic, which is activated when a particular event or operation occurs on a database table. For example, you could set up a stored procedure that checks the credit rating for a customer, and is triggered on each insert of a new customer in the customer table. DB2 UDB provides a matured implementation of database triggers. However, since V5.1 of MySQL is not just around the corner, we limit this section to the very basics of database triggers.

Unlike stored procedures triggers are associated with a database *table*, an *operation* (INSERT, UPDATE, DELETE) on the table and a *point in time* (BEFOR, AFTER) which all have to apply for the trigger to be activated. Here are some examples where triggers might be useful:

- ▶ When inserting rows in a table, triggers can be used to supply, validate or manipulate data before allowing an insert operation to occur.
- ▶ When updating rows, triggers can be used to compare old and new values and allow proper state transition. For example, a date value can only change to a future never to a past value.
- ▶ Auditing and additional logging can be implemented upon deletion of rows.

Example 10-5 shows a simple example which inserts the value of `start + 45` minutes into column `end` in table `mytab` if there is no value for column `end` supplied on the `insert` statement activating the trigger

Example 10-5 Simple TRIGGER

```
CREATE TRIGGER FirstTrg
NO CASCADE BEFORE INSERT ON mytable
REFERENCING NEW AS n
REFERENCING OLD AS o
FOR EACH ROW
MODE DB2SQL
WHEN (n.end IS NULL)
    SET n.end = n.start + 45 MINUTES
```

10.4 User-defined data types (UDT)

A lot of application programming languages are based on object-oriented analysis and design because of reasons like simplicity, scalability, and easier modeling of complex business objects and services. DB2 UDB supports a few object-oriented programming features which, you can incorporate object-oriented

(OO) concepts and methodology's into your relational database by extending it with richer sets of types and functions. With these features, you can store instances of object-oriented data types in columns of tables, and operate on them by means of functions in SQL statements.

The basic object oriented support is provided by DB2 UDB using user-defined types (UDTs) and user-defined functions (UDFs) and LOBs. They can be used to build extensions to DB2 UDB, which include a whole set of customized types and functionality.

User-defined data types are customized data types derived from the existing built-in data types in DB2 UDB. They are useful when existing data types do not serve your application requirements, and when you need data integrity, which can be achieved by strong typing and encapsulation.

There are three types of user-defined types:

► Distinct type

A distinct type is a user-defined data type that is based on an existing built-in data type. Internally, it is stored as an existing data type, but it is considered as a separate and incompatible type. The main advantages of using distinct type are extensibility, strong typing, encapsulation, and customization.

A distinct type can be created by issuing the CREATE DISTINCT TYPE statement. The following statement defines a new distinct type for our sample application where we want all the identification numbers to have common properties and functions. To achieve this goal we create a distinct type ID, which contains INTERGER values:

```
db2>create distinct type ID as integer with comparisons
```

► Structured type

A structured type is a user-defined data type that has a well defined structure consisting of existing built-in or user-defined data types. A structured type has attributes and methods defined. The attribute defines its data storage properties and methods define its behavior.

A structured type may be used as the type of a table, view, or column. When used as the type for a table or view, that table or view is known as a typed table or typed view respectively. In this case, attributes of structured type becomes columns of a typed table or view.

A structured type can be created using the CREATE TYPE statement. For example, if we want our example database ITS0DB design to use the UDT features, we can define a *product* and *sku* type, which can be used to create typed tables as shown in Example 10-6. Figure 10-2 shows its hierarchy.

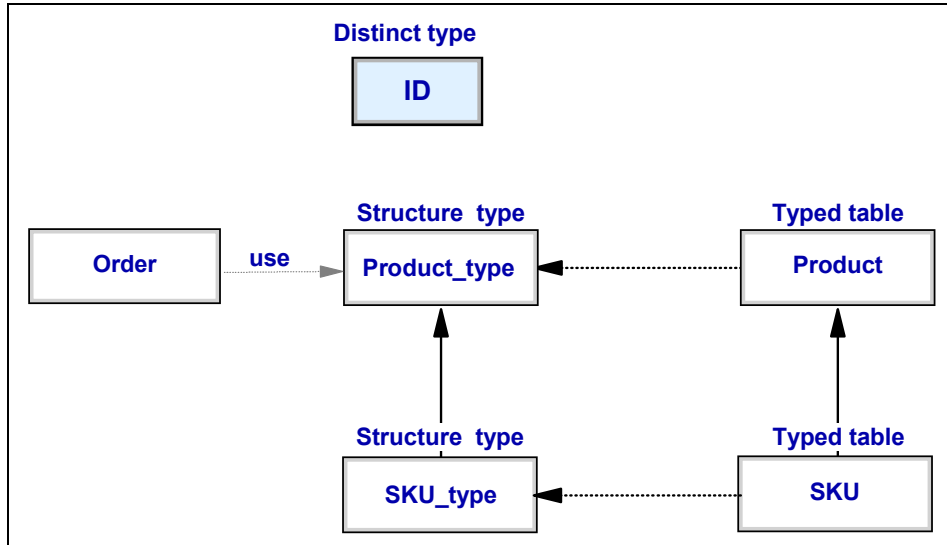


Figure 10-2 User defined data types

Example 10-6 Structured types and typed tables

```

db2>create type product_type as (
    name varchar(20),
    description varchar(200),
    brand varchar(30))
    instantiable ref using integer mode db2sql

db2>create type sku_type
    under product_type as (quantity integer)
    instantiable mode db2sql

db2>create table product
    of product_type (ref is id user generated)

db2>create table sku of sku_type
    under product inherit select privileges

db2>create table order(id ID,sku sku_type)
  
```

We can also use this type as a type for a column as shown in the last statement of Example 10-6.

► Reference type

A reference type is a companion type to a structured type; it is quite similar to a distinct type. The reference type representation is defined when the root type of a type hierarchy is created. When using a reference type, a structured

type is specified as a parameter of the type. This parameter is called the target type of the reference. The target of a reference is always a row in a typed table or a typed view. When a reference type is used, it may have a scope defined.

10.5 User-defined functions

DB2 UDB provides feature for creating additional functions, which can be used as utilities or a methods on existing data types. This provides you the capability to extend and customize your SQL according to your requirements. They can be invoked in same way as your built-in DB2 UDB functions.

Another main advantage of using UDF is it allows you to perform your logic inside the DB2 UDB server. By this way you can pass some of your application logic to server-side for faster execution. Also, DB2 UDB supports implementation of these user-defined functions in SQL or number of external programming languages. This provides you a feature to use the power of languages like C, C++, Java, OLE, etc.

The UDF can be of the types:

- ▶ Scalar function

A scalar function is a user-defined function, which returns only single values all the time. They are quite useful while writing a utility functions based on existing SQL functions. A scalar function can be referenced in the same contexts as any built-in function.

A scalar function can be sourced or external. A sourced function is defined to the database with a reference to another built-in or user-defined function, whereas an external function is defined to the database with a reference to an external library or code. Example 10-7 shows both source and external creation for the function *stringId*, which ID type to varchar(20).

Example 10-7 Scalar functions

```
db2>create function stringId(ID) returns varchar(20) source DIGITS(INTEGER)
db2>create function stringId(ID) returns varchar(10) external name 'itso!id'
language C parameter style SQL deterministic no SQL no external action
```

- ▶ Column function

A column function is a user-defined function which single-valued answers from a set of like values. They are quite useful when you want to aggregate a result from set of rows. Currently, only sourced column functions are supported in DB2 UDB. It means you can create a column function sourced upon one of the built-in column functions, and you cannot write a column

function using an external source. Columns functions can be created as following:

```
db2>create function average(ID) returns ID source avg(integer)
```

► Table function

A table function is a user-defined function, which returns a data in tabular form to the SQL statement that calls it. So, a table function can only be invoked in the FROM clause of a SQL statement. Such a function can be used to apply SQL language processing power to non DB2 UDB data or to convert such data into a DB2 table. Table functions are always external. They are useful when you want to convert your file text data to tabular form, or convert your XML data to tabular form. Example 10-8 shows the creation and usage of a tabular function, which fetches the user's data from an existing file in a tabular format.

Example 10-8 Table function creation and usage

```
db2> create function usernames()  
      returns table(  
          integer id,  
          vorname varchar(20),  
          name varchar(3),  
          email varchar(20))  
      external name 'itso.users'  
      language C parameter style SQL  
      not deterministic called on null  
      input scratchpad final call  
      no sql external action disallow parallel
```

```
db2>select * from TABLE(usernames()) as usernames
```

10.6 Materialized query tables (MQT)

A *materialized query table (MQT)* is a table whose definition is based on the result of a query, and whose data is in the form of precomputed results that are taken from one or more tables on which the materialized query table definition is based. Sometimes they are also referred to as *materialized views*.

Prior to Version 8, DB2 UDB supported summary tables also known as *automatic summary tables (ASTs)* or *replication summary tables*. Summary tables are now considered to be a special type of MQT whose fullselect contains a **GROUP BY** clause summarizing data from the tables referenced in the fullselect.

The main characteristics of MQTs are:

- ▶ MQTs can contain pre-computed and/or subsets of data.
- ▶ MQTs can have indexes; the **RUNSTATS** command can be performed against them.
- ▶ MQTs can use multi dimensionally clustered and regular tables as source tables.
- ▶ MQTs can be refreshed incremental.
- ▶ MQTs are automatically used by the optimizer if applicable.

The following MQT enhancements can result in improved query performance:

▶ **Query routing enhancements**

Queries can now be routed to MQTs whose definitions contain a join that is not aggregated. Prior to Version 8, an MQT definition could only reference a join that was aggregated. For example, in Version 8 the table described in Example 10-9, which contains a join, can be created to store the customer and account information for bad accounts.

Example 10-9 Sample MQT

```
CREATE TABLE bad_account AS (
  SELECT customer_name, customer_id, a.balance
  FROM account a, customers c
  WHERE
    status IN ('delinquent', 'problematic', 'hot')
    AND a.customer_id = c.customer_id)
DATA INITIALLY DEFERRED REFRESH DEFERRED
```

If a user asks whether an account is delinquent, the DB2 UDB optimizer recognizes that the MQT has cached the requested information, and instead of accessing the base table ACCOUNT, DB2 accesses the MQT named BAD_ACCOUNT, which provides a better response time and can be used to return customer information.

▶ **User-maintained materialized query tables**

Many custom applications maintain and load tables that are really precomputed data representing the result of a query. By identifying a table as a user-maintained materialized query table, dynamic query performance can be improved. Such MQTs are maintained by users rather than by the system. UPDATE, INSERT, and DELETE operations are permitted against user-maintained MQTs.

Setting appropriate special registers allows the query optimizer to take advantage of the precomputed query result that is already contained in the user-maintained MQT.

▶ **Materialized query tables on nicknames**

This feature allows you to cache remote data locally on your DB2 Universal Database instance. Remote data resides in databases that are supported by relational DBMS instances such as Oracle or Sybase, or even other instances of DB2 UDB.

MQTs can reference a combination of nicknames and local tables. Such materialized query tables can be created with the **REFRESH DEFERRED** option only. Queries against nicknames or tables are rewritten and optimized in relation to these MQTs.

Routing a query to the MQT when all criteria for matching and routing are satisfied, yields better performance than getting results from the remote table.

It is possible to query a nickname even if the remote table for which the nickname was created becomes unavailable. If this nickname has a materialized query table defined on it, and all routing criteria match, the query will only need to select data from the MQT.

Maintenance is performed locally by means of the **REFRESH TABLE** command. There is no way to keep track of updates to tables in a remote database. Maintenance is always deferred; refresh immediate materialized query tables (defined on nicknames) are not supported.

► **Incremental maintenance of materialized query tables using a staging table**

You can incrementally refresh an MQT defined with the **REFRESH DEFERRED** option. If a refresh deferred MQT is to be incrementally maintained, it must have a staging table associated with it. The staging table associated with an MQT is created with the `CREATE TABLE SQL` statement.

When `INSERT`, `DELETE` or `UPDATE` statements modify the underlying tables of an MQT, the changes resulting from these modifications are propagated, and are immediately appended to a staging table as part of the same statement. The propagation of these changes to the staging table is similar to the propagation of changes that occurs during the incremental refresh of immediate MQTs.

A `REFRESH TABLE` statement is used to incrementally refresh the MQT. If a staging table is associated with the MQT, the system may be able to use the staging table that supports the MQT to incrementally refresh it. The staging table is pruned when the refresh is complete. Prior to Version 8, a refresh deferred MQT was regenerated from scratch when performing a refresh table operation. MQTs can now be incrementally maintained, providing significant performance improvement. For information about the situations under which a staging table will not be used to incrementally refresh an MQT, see the *SQL Reference* manual.

You can also use this new facility to eliminate the high lock contention caused by the immediate maintenance of refresh immediate MQTs. If the data in the

MQT does not need to be current to the second, changes can be captured in a staging table and applied on any schedule.

10.7 Multidimensional clustering (MDC)

This section gives a brief introduction about the *Multidimensional Clustering (MDC)* feature provided by DB2 UDB, Version 8.1.

DB2 UDB in earlier versions provides the feature for single dimensional data clustering. This was maintained by physically clustering the data on insert, according to the order of one single clustering index. The number of clustering indexes per table is limited to one.

Figure 10-3 shows data clustering according to one single index, in this case the Region index.

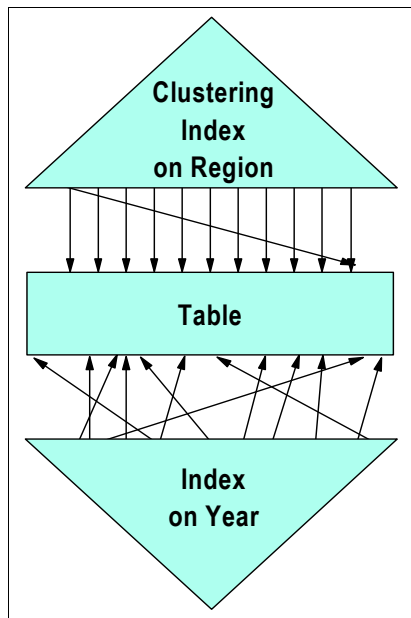


Figure 10-3 One clustering index

One of the main drawbacks of this clustering method is that clustering degrades over time and requires the use of the REORG utility from time to time to put the index back in order.

Another drawback of this method is that the indexes are record based, and is what makes them often very large.

MDC addresses these drawbacks. Data is clustered physically in multiple dimensions. The indexes for each dimension are not record based but block based, thus reducing their size (and effort needed for logging and maintaining) dramatically. Reorganization of the table in order to re-cluster is not necessary.

At the bottom line, MDC provides a powerful method for improving the performance of SELECT, INSERT, UPDATE, and DELETE statements.

Example 10-10 shows the CREATE TABLE statement of an MDC table clustered in two columns: Region and Year. The block indexes for each dimension are created automatically.

Example 10-10 Creating an MDC table

```
CREATE TABLE sales (
  Customer VARCHAR(80),
  Region CHAR(5),
  Year INT
)
ORGANIZE BY DIMENSIONS (Region, Year);
```

Figure 10-4 shows the data clustering according to two dimensions as defined in Example 10-10. Actually, the number of dimensions for a table is only limited by the system size.

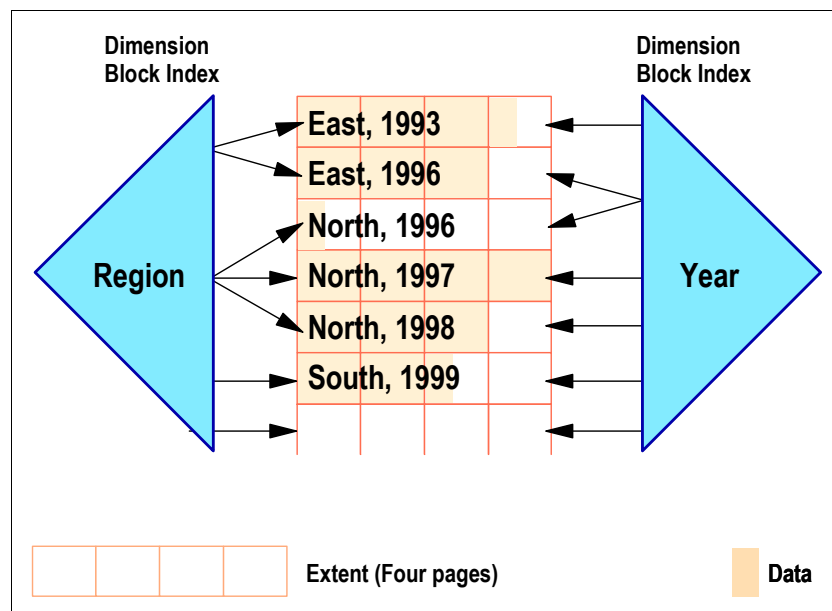


Figure 10-4 MDC table and indexes

Data in MDC tables is organized in blocks along extent boundaries according to the dimensions (clustering values). A block (extent) is a set of contiguous pages on disk, so access to these records is sequential and with minimal I/O operations. The page size is specified at table space creation.

In Figure 10-4 the extents' boundaries are defined by the two dimensions: Region and Year.

Benefits of MDC

These are the benefits of MDC:

- ▶ Range scans on any dimension use clustered data access, because each block corresponds to a set of sequential parts in the table guaranteed to contain data having that dimension value.
- ▶ Dimensions can be accessed independently from each other through their block indexes without compromising the clustering of any other.
- ▶ Block index scans can be combined by AND and OR logical operations. The resulting scan also uses clustered data access.
- ▶ Access to clustered data is much faster than access to data through clustering indexes (single dimensional data clustering), because with MDC there is one pointer per qualifying block of pages versus one pointer per qualifying row in single dimensional data clustering.
- ▶ With a specified block ID from a block index, scans on that block are very efficient and much faster than accessing each row through row ID.



Sample code for user defined functions

This appendix provides sample code to implement various MySQL built-in functions not provided by DB2 UDB. We would like to thank the respective authors of the UDFs.

A.1 Sample code for BIT_AND

Example A-1 shows conversion code for MySQLs BIT_AND function.

Example: A-1 User-defined function to map BIT_AND

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/08/29
--
-- Name of UDF: BIT_AND (N1 Integer, N2 Integer)
--
-- Used UDF: None
--
-- Description: Returns bit by bit and of both parameters.
--
-- Author: TOKUNAGA, Takashi
--
```

```
-----
CREATE FUNCTION BITAND (N1 Integer, N2 Integer)
  RETURNS Integer
  SPECIFIC BITANDMySQL
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN
WITH
Repeat (S, M1, M2, Ans) AS
(Values (0, N1, N2, 0)
Union All
Select S+1, M1/2, M2/2, Ans+MOD(M1,2)*MOD(M2,2)*power(2,S)
  From Repeat
  Where M1 > 0
     AND M2 > 0
     AND S < 32
)
SELECT ANS
  FROM Repeat
 WHERE S = (SELECT MAX(S)
            FROM Repeat)
;
```

Example A-2 shows the results of the BITAND user-defined function.

Example: A-2 Results for UDB BITAND

SQL0347W The recursive common table expression "DB2ADMIN.REPEAT"

may contain an infinite loop. SQLSTATE=01605

```
-----  
values bitand(10,8);  
-----
```

```
1
```

```
-----  
          8
```

```
-----  
values bitand(14,3);  
-----
```

```
1
```

```
-----  
          2
```

```
-----  
values bitand(1038,78);  
-----
```

```
1
```

```
-----  
         14
```

A.2 Sample code for FORMAT function

This section provides UDF for FORMAT function. Example A-3 shows the code for a user-defined function emulating FORMAT.

Example: A-3 FORMAT user-defined function

```
--  
-- DB2 UDB UDF(User-Defined Function) Samples for Migration  
--  
-- Created: 2004/02/29  
--  
-- Name of UDF: FORMAT (X Decimal(31,10), D Integer)  
--  
-- Used UDF: None  
--  
-- Description: Returns truncated to the precision specified by D and a "," for  
-- each 3 digits as a separator.  
--  
-- Author: TOKUNAGA, Takashi  
--
```

```

----- Command Entered -----
CREATE FUNCTION FORMAT (X Decimal(31,10), D Integer)
  RETURNS VARCHAR(50)
  LANGUAGE SQL
  SPECIFIC FORMAT_MySQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
BEGIN ATOMIC
DECLARE XN      DECIMAL(21,0);
DECLARE RetVal VARCHAR(50);
SET RetVal = SUBSTR(CHAR(MOD(ABS(X), 1)), 22, D+1);
SET XN = ABS(X);

Main_Loop:
WHILE XN > 0 DO
  SET RetVal = SUBSTR(CHAR(MOD(XN,1000)),19,3) || RetVal;
  SET XN = XN/1000;
  IF XN > 0 THEN
    SET RetVal = ',' || RetVal;
  ELSE
    LEAVE Main_Loop;
  END IF;
END WHILE;

RETURN CASE WHEN X < 0 THEN '-' ELSE '' END
        || TRANSLATE(LTRIM(TRANSLATE(RetVal,' ','0')),'0',' ');
END
!

```

Example A-4 shows the results of the converted FORMAT.

Example: A-4 Converted FORMAT UDF result

```

----- Command Entered -----
SELECT N
      , FORMAT(N, 2)
      , FORMAT(N, 0)
FROM (VALUES 12.34567, -12.34567, 120034.567, 123400123456789.) S(N)!
-----
--Return result

```

N	2	3
12.34567	12.34	12.
-12.34567	-12.34	-12.
120034.56700	120,034.56	120,034.


```
23400123456789.00000 123,400,123,456,789.00 123,400,123,456,789.
```

```
4 record(s) selected.
```

A.3 Sample code for RPAD and LPAD functions

This section provides UDF for LPD and RPAD functions. Example A-5 shows code for a user-defined function emulating RPAD.

Example: A-5 CREATE FUNCTION RPAD and sample usage

```
-- 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))
--                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 right(RPAD) or left(LPAD) 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)
;
```

Example A-6 shows the results of the converted RPAD function.

Example: A-6 Usage of UDF RPAD

```
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.
```

UDF RPAD with the third parameter omitted is shown in Example A-7.

Example: A-7 RPAD Omitting the third parameter

```
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,' ' )  
;
```

Running the RPAD function gives you the results shown in Example A-8.

Example: A-8 Results of RPAD omitting the third parameter

```
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.

```

Function RPAD allows a set of different input arguments. Example A-9 shows two more RPAD UDFs.

Example: A-9 RPAD with first parameter as integer, 2, and 3 parameters

```

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)
;

```

```

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,' ')
;

```

And Example A-10 shows the results of the previous UDFs.

Example: A-10 Results of RPAD with first parameter as integer, 2, and 3 parameters

```
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.

SELECT char(rpad(9021,3),20) FROM SYSIBM.SYSDUMMY1;

1

902

1 record(s) selected.

The counterpart for RPAD are the LPAD functions, which are shown in Example A-11.

Example: A-11 LPAD: CREATE FUNCTION and sample usage

```
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
;

```

Results of LPAD look like Example A-12.

Example: A-12 Results of LPAD: CREATE FUNCTION and sample usage

```

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.

```

As RPAD allows LPAD a different number and data type for input arguments, Example A-13 shows LPAD without the third parameter.

Example: A-13 LPAD: Omitting the third parameter

```
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,' ')
;
```

And the results of Example A-13 should look like those in Example A-14.

Example: A-14 Result of LPAD: Omitting the third parameter

```
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.
```

Two more LPAD UDFs with different characteristics are shown in Example A-15.

Example: A-15 LPAD: The first parameter is integer, 2, and 3 parameters

```
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)
;
```

```
-----
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,' ')
;
```

And the results are shown in Example A-16.

Example: A-16 Results of LPAD: The first parameter is integer, 2, and 3 parameter

```
SELECT char(1pad(9021,15,'*'),50) FROM SYSIBM.SYSDUMMY1;
```

```
-----
1
```

```
*.*.*.*.*9021
```

1 record(s) selected.

```
-----
SELECT char(1pad(9021,15,''),50) FROM SYSIBM.SYSDUMMY1;
```

```
-----
1
```

```
9021
```

1 record(s) selected.

```
-----
SELECT char(1pad(9021,3),20) FROM SYSIBM.SYSDUMMY1;
```

```
-----
1
```

1 record(s) selected.

A.4 Sample code for GREATEST function

Example A-17 is a set of UDF examples emulating the behavior of MySQL's GREATEST function. The various UDFs accept input parameters in *varchar* and from 2 to 10 input parameters.

Example: A-17 User-defined functions to map GREATEST

```
--
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/08/28, 08/29
--
-- Name of UDF: GREATEST (P1 VarChar(254), P2 VarChar(254), ...)
--
--
-- Used UDF: None
--
-- Description: Returns greatest value of list of data.
--
-- Author: TOKUNAGA, Takashi
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254))
  RETURNS VarChar(254)
  LANGUAGE SQL
  SPECIFIC GREATESTOracle2
  DETERMINISTIC
  CONTAINS SQL
  NO EXTERNAL ACTION
  RETURN
  CASE
  WHEN P1 >= P2 THEN P1
  ELSE          P2
  END
;
-----
--
-- GREATEST function with three parameters
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254))
  RETURNS VarChar(254)
```



```

LANGUAGE SQL
SPECIFIC GREATESTOracle3
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
CASE
WHEN P1 >= P2
THEN CASE
    WHEN P1 >= P3 THEN P1
    ELSE P3
    END
ELSE CASE
    WHEN P2 >= P3 THEN P2
    ELSE P3
    END
END
;
-----
--
-- GREATEST function with four parameters
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC GREATESTOracle4
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
CASE
WHEN P1 >= P2
THEN CASE
    WHEN P1 >= P3
    THEN CASE
        WHEN P1 >= P4 THEN P1
        ELSE P4
        END
    ELSE CASE
        WHEN P3 >= P4 THEN P3
        ELSE P4
        END
    END
ELSE CASE
    WHEN P2 >= P3
    THEN CASE
        WHEN P2 >= P4 THEN P2

```

```

        ELSE          P4
        END
    ELSE CASE
        WHEN P3 >= P4 THEN P3
        ELSE          P4
        END
    END
END
;
-----
--
-- GREATEST function with five parameters
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254), P5 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC GREATESTOracle5
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
CASE
WHEN P1 >= P2
THEN CASE
    WHEN P1 >= P3
    THEN CASE
        WHEN P1 >= P4
        THEN CASE
            WHEN P1 >= P5 THEN P1
            ELSE          P5
            END
        ELSE CASE
            WHEN P4 >= P5 THEN P4
            ELSE          P5
            END
        END
    ELSE CASE
        WHEN P3 >= P4
        THEN CASE
            WHEN P3 >= P5 THEN P3
            ELSE          P5
            END
        ELSE CASE
            WHEN P4 >= P5 THEN P4
            ELSE          P5
            END
        END
    END
END

```

```

        END
ELSE CASE
    WHEN P2 >= P3
    THEN CASE
        WHEN P2 >= P4
        THEN CASE
            WHEN P2 >= P5 THEN P2
            ELSE          P5
            END
        ELSE CASE
            WHEN P4 >= P5 THEN P4
            ELSE          P5
            END
        END
    ELSE CASE
        WHEN P3 >= P4
        THEN CASE
            WHEN P3 >= P5 THEN P3
            ELSE          P5
            END
        ELSE CASE
            WHEN P4 >= P5 THEN P4
            ELSE          P5
            END
        END
    END
END
;
-----
--
-- GREATEST function with six parameters
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
                        , P5 VarChar(254), P6 VarChar(254))
    RETURNS VarChar(254)
    LANGUAGE SQL
    SPECIFIC GREATESTOracle6
    DETERMINISTIC
    CONTAINS SQL
    NO EXTERNAL ACTION
    RETURN
GREATEST(GREATEST(P1,P2,P3),GREATEST(P4,P5,P6))
;
-----
--
-- GREATEST function with seven parameters
--

```

```

-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC GREATESTOracle7
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
GREATEST(GREATEST(P1,P2,P3,P4),GREATEST(P5,P6,P7))
;
-----
--
-- GREATEST function with eight parameters
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254), P8
VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC GREATESTOracle8
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
GREATEST(GREATEST(P1,P2,P3,P4),GREATEST(P5,P6,P7,P8))
;
-----
--
-- GREATEST function with nine parameters
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254), P8
VarChar(254)
, P9 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC GREATESTOracle9
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
GREATEST(GREATEST(P1,P2,P3,P4,P5),GREATEST(P6,P7,P8,P9))

```

```

;
-----
--
-- GREATEST function with ten parameters
--
-----
CREATE FUNCTION GREATEST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254), P8
VarChar(254)
, P9 VarChar(254),P10 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC GREATESTOracle10
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
GREATEST(GREATEST(P1,P2,P3,P4,P5),GREATEST(P6,P7,P8,P9,P10))
;
-----

```

Example A-18 shows the results of UDFs GREATEST.

Example: A-18 Result of UDFs mapping GREATEST

```

SELECT char(greatest('abcdefg','abcfgh'),20) FROM sysibm.sysdummy1;
-----

```

```

1
-----
abcfgh

1 record(s) selected.

```

```

-----
SELECT char(greatest('abcdefg','defgh','abcfgh'),20) FROM sysibm.sysdummy1;
-----

```

```

1
-----
defgh

1 record(s) selected.

```

```
-----  
SELECT char(greatest('abcdefg','defgh','abcfgh','endof...'),20) FROM  
sysibm.sysdummy1;  
-----
```

```
1  
-----
```

```
endof...
```

```
1 record(s) selected.
```

```
-----  
SELECT char(greatest('abcdefg','defgh','abcfgh','endof...','add on'),20) FROM  
sysibm.sysdummy1;  
-----
```

```
1  
-----
```

```
endof...
```

```
1 record(s) selected.
```

```
-----  
SELECT char(greatest('abcdefg','defgh','abcfgh','endof...','add  
on','extra'),20) FROM sysibm.sysdummy1;  
-----
```

```
1  
-----
```

```
extra
```

```
1 record(s) selected.
```

```
-----  
SELECT char(greatest('abcdefg','defgh','abcfgh','endof...','add on','extra','a  
bit of'),20) FROM sysibm.sysdummy1;  
-----
```

```
1  
-----
```

```
extra
```

```
1 record(s) selected.
```

```
-----  
SELECT char(greatest('abcdefg','defgh','abcfgh','endof...','add on','extra','a  
bit of','more'),20) FROM sysibm.sysdummy1;  
-----
```

```
1  
-----
```

```
more
```

```
1 record(s) selected.
```

```
-----  
SELECT char(greatest('abcdefg','defgh','abcfgh','endof...','add on','extra','a  
bit of','more','more and '),20) FROM sysibm.sysdummy1;  
-----
```

```
1  
-----
```

```
more and
```

```
1 record(s) selected.
```

```
-----  
SELECT char(greatest('abcdefg','defgh','abcfgh','endof...','add on','extra','a  
bit of','more','more and ',' something'),20) FROM sysibm.sysdummy1;  
-----
```

```
1  
-----
```

```
more and
```

```
1 record(s) selected.
```

A.5 Sample code for LEAST

Example A-19 is a set of UDF examples emulating the behavior of MySQL's LEAST function. The various UDFs accept input parameters in *varchar* and from 2 to 10 input parameters.

Example: A-19 User-defined functions to map LEAST

```
-- DB2 UDB UDF(User-Defined Function) Samples for Migration
--
-- 2001/08/28, 08/29
--
-- Name of UDF: LEAST (P1 VarChar(254), P2 VarChar(254))
--
--
-- Used UDF: None
--
-- Description: Returns least value of list of data.
--
-- Author: TOKUNAGA, Takashi
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254))
  RETURNS VarChar(254)
  LANGUAGE SQL
  SPECIFIC LEASTOracle2
  DETERMINISTIC
  CONTAINS SQL
  NO EXTERNAL ACTION
  RETURN
  CASE
  WHEN P1 <= P2 THEN P1
  ELSE          P2
  END
  ;
-----
--
-- LEAST function with three parameters
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254))
  RETURNS VarChar(254)
  LANGUAGE SQL
  SPECIFIC LEASTOracle3
  DETERMINISTIC
  CONTAINS SQL
  NO EXTERNAL ACTION
  RETURN
  CASE
  WHEN P1 <= P2
  THEN CASE
        WHEN P1 <= P3 THEN P1
        ELSE          P3
        END
  ELSE CASE
```



```

        WHEN P2 <= P3 THEN P2
        ELSE          P3
        END
END
;
-----
--
-- LEAST function with four parameters
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254))
    RETURNS VarChar(254)
    LANGUAGE SQL
    SPECIFIC LEASTOracle4
    DETERMINISTIC
    CONTAINS SQL
    NO EXTERNAL ACTION
    RETURN
CASE
WHEN P1 <= P2
THEN CASE
    WHEN P1 <= P3
    THEN CASE
        WHEN P1 <= P4 THEN P1
        ELSE          P4
        END
    ELSE CASE
        WHEN P3 <= P4 THEN P3
        ELSE          P4
        END
    END
ELSE CASE
    WHEN P2 <= P3
    THEN CASE
        WHEN P2 <= P4 THEN P2
        ELSE          P4
        END
    ELSE CASE
        WHEN P3 <= P4 THEN P3
        ELSE          P4
        END
    END
END
END
;
-----
--
-- LEAST function with five parameters
--

```

```

-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC LEASTOracle5
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
CASE
WHEN P1 <= P2
THEN CASE
    WHEN P1 <= P3
    THEN CASE
        WHEN P1 <= P4
        THEN CASE
            WHEN P1 <= P5 THEN P1
            ELSE P5
            END
        ELSE CASE
            WHEN P4 <= P5 THEN P4
            ELSE P5
            END
        END
    ELSE CASE
        WHEN P3 <= P4
        THEN CASE
            WHEN P3 <= P5 THEN P3
            ELSE P5
            END
        ELSE CASE
            WHEN P4 <= P5 THEN P4
            ELSE P5
            END
        END
    END
ELSE CASE
    WHEN P2 <= P3
    THEN CASE
        WHEN P2 <= P4
        THEN CASE
            WHEN P2 <= P5 THEN P2
            ELSE P5
            END
        ELSE CASE
            WHEN P4 <= P5 THEN P4
            ELSE P5
            END
        END
    END

```

```

        END
    END
ELSE CASE
    WHEN P3 <= P4
    THEN CASE
        WHEN P3 <= P5 THEN P3
        ELSE P5
        END
    ELSE CASE
        WHEN P4 <= P5 THEN P4
        ELSE P5
        END
    END
END
END
END
;
-----
--
-- LEAST function with six parameters
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC LEASTOracle6
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
LEAST(LEAST(P1,P2,P3),LEAST(P4,P5,P6))
;
-----
--
-- LEAST function with seven parameters
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC LEASTOracle7
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
LEAST(LEAST(P1,P2,P3,P4),LEAST(P5,P6,P7))

```

```

;
-----
--
-- LEAST function with eight parameters
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254), P8
VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC LEASTOracle8
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
LEAST(LEAST(P1,P2,P3,P4),LEAST(P5,P6,P7,P8))
;
-----
--
-- LEAST function with nine parameters
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254), P8
VarChar(254)
, P9 VarChar(254))
RETURNS VarChar(254)
LANGUAGE SQL
SPECIFIC LEASTOracle9
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
LEAST(LEAST(P1,P2,P3,P4,P5),LEAST(P6,P7,P8,P9))
;
-----
--
-- LEAST function with ten parameters
--
-----
CREATE FUNCTION LEAST (P1 VarChar(254), P2 VarChar(254), P3 VarChar(254), P4
VarChar(254)
, P5 VarChar(254), P6 VarChar(254), P7 VarChar(254), P8
VarChar(254)
, P9 VarChar(254),P10 VarChar(254))
RETURNS VarChar(254)

```

```

LANGUAGE SQL
SPECIFIC LEASTOracle10
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN
LEAST(LEAST(P1,P2,P3,P4,P5),LEAST(P6,P7,P8,P9,P10))
;

```

Example A-20 shows the results of UDF's LEAST.

Example: A-20 Results of UDFs mapping LEAST

```

SELECT least('HARRY','HARRIOT','HAROLD') FROM sysibm.sysdummy1;
-----

1
-----
-----
HAROLD

1 record(s) selected

```

A.6 Sample code for BIT_COUNT

Example A-21 is a UDF example emulating the behavior of MySQL's BIT_COUNT function. It returns the number of set bits in the parameter (the number of 1 in the binary value of the parameter) assuming that the parameter is a 32-bit **INTEGER**.

Example: A-21 User-defined function to map BIT_COUNT

```

CREATE FUNCTION BIT_CNT (N1 Integer)
RETURNS Integer
SPECIFIC BITCNTMySQL
LANGUAGE SQL
CONTAINS SQL
NO EXTERNAL ACTION
DETERMINISTIC
RETURN
WITH
Repeat (S, M1, Ans) AS
(Values (0, N1, 0)
Union All
Select S+1, M1/2, Ans+MOD(M1,2)
From Repeat

```

```

Where M1 <> 0
  AND S < 32
)
SELECT case when ANS > 0 then ANS else 32 + ANS end
  FROM Repeat
  WHERE S = (SELECT MAX(S)
            FROM Repeat)
;

```

Example A-22 shows the sample output.

Example: A-22 Sample output of BIT_CNT

```
db2> values bit_cnt(64)
```

```

1
-----
      1

```

```
1 record(s) selected.
```

```
db2> values bit_cnt(63)
```

```

1
-----
      6

```

```
1 record(s) selected.
```

```
db2> values bit_cnt(-7)
```

```

1
-----
     29

```

```
1 record(s) selected.
```

A.7 Sample code for SUBSTRING_INDEX

Example A-23 is a UDF example emulating the behavior of MySQL's SUBSTRING_INDEX function. It returns the substring from input string before counting occurrences of the delimiter.

Example: A-23 User-defined function to map SUBSTRING_INDEX

```
create function SUBSTRING_INDEX(In varchar(2000),delimit varchar(200), n Int)
returns varchar(2000)
deterministic no external action contains sql
begin atomic
  declare out varchar(2000);
  declare dem varchar(2000);
  declare num int;
  declare pos int;
  declare temp varchar(2000);
  set dem=delimit;
  set temp=In;
  set num=n;
  set pos=1;
  if(num<0) then
    while(locate(delimit,temp)!=0) do
      set temp=substr(temp,locate(delimit,temp)+1);
      set num=num+1;
    end while;
    set num=num+1;
    set temp=In;
  end if;
  while (num>0) do
    set pos=pos+locate(delimit,temp)-1;
    set temp=substr(temp,locate(delimit,temp)+1);
    set num=num-1;
  end while;
  if(n>0) then
    return substr(In,1,pos);
  else
    return substr(In,pos+1);
  end if;
end
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 366. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Up and Running with DB2 UDB for Linux*, SG24-6899
- ▶ *Oracle to DB2 UDB Conversion Guide*, SG24-7048

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM DB2 UDB Command Reference V8*, SC09-4828
- ▶ *IBM DB2 UDB What's New V8*, SC09-4848
- ▶ *IBM DB2 UDB Administration Guide: Planning V8*, SC09-4822
- ▶ *IBM DB2 UDB Administration Guide: Implementation V8*, SC09-4820
- ▶ *IBM DB2 UDB Administration Guide: Performance V8*, SC09-4821
- ▶ *IBM DB2 UDB Data Movement Utilities Guide and Reference V8*, SC09-4830
- ▶ *IBM DB2 UDB Data Recovery and High Availability Guide and Reference V8*, SC09-4831
- ▶ *Federated Systems PIC Guide Version 8 Release 1*, GC27-1224
- ▶ *IBM DB2 UDB Guide to GUI Tools for Administration and Development*, SC09-4851
- ▶ *IBM DB2 UDB SQL Reference, Volume 1, V8*, SC09-4844
- ▶ *IBM DB2 UDB SQL Reference, Volume 2, V8*, SC09-4845
- ▶ *IBM DB2 UDB System Monitor Guide and Reference V8*, SC09-4847
- ▶ *IBM DB2 UDB Application Development Guide: Building and Running Applications V8*, SC09-4825

- ▶ *IBM DB2 UDB Application Development Guide: Programming Client Applications V8*, SC09-4826
- ▶ *IBM DB2 UDB Application Development Guide: Programming Server Applications V8*, SC09-4827
- ▶ *IBM DB2 UDB Call Level Interface Guide and Reference, Volume 1, V8*, SC09-4849
- ▶ *IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2, V8*, SC09-4850
- ▶ *Data Warehouse Center Application Integration Guide Version 8 Release 1*, SC27-1124
- ▶ *DB2 XML Extender Administration and Programming Guide Version 8 Release 1*, SC27-1234
- ▶ *IBM DB2 UDB Quick Beginnings for DB2 Clients V8*, GC09-4832
- ▶ *IBM DB2 UDB Quick Beginnings for DB2 Servers V8*, GC09-4836
- ▶ *IBM DB2 UDB Installation and Configuration Supplement V8*, GC09-4837

Online resources

These Web sites and URLs are also relevant as further information sources:

DB2

- ▶ Database and Data Management
<http://www.ibm.com/software/data/>
<http://www.ibm.com/software/data/highlights/db2tco.html>
- ▶ DB2 Developer Domain
<http://www7b.software.ibm.com/dmdd/>
- ▶ DB2 Universal Database
<http://www.ibm.com/software/data/db2/udb/>
<http://ibm.com/db2/v8>
- ▶ DB2 Universal Database for Linux
<http://www.ibm.com/software/data/db2/linux/>
<http://www.ibm.com/db2/linux/validate>
<http://ibm.com/db2/linux>
<http://www-3.ibm.com/software/data/db2/linux/validate>
- ▶ DB2 Universal Database V8 Application Development
<http://www.ibm.com/software/data/db2/udb/ad>

- ▶ DB2 Technical Support
<http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/index.d2w/report>
- ▶ DB2 Extenders
<http://www.ibm.com/software/data/db2/extenders/>
- ▶ IBM Manuals for Data Management Products
<http://www.ibm.com/software/data/db2/library/>
- ▶ DB2 NOW!
<http://www.ibm.com/db2/migration>

MySQL

- ▶ MySQL home page
<http://www.mysql.com/>

Linux

- ▶ IBM Software for Linux
<http://www.ibm.com/software/is/mp/linux/software/>
- ▶ SuSE home page
http://www.suse.com/index_us.html
- ▶ Red Hat home page
<http://www.redhat.com/>

Other

- ▶ Apache Web Development with IBM DB2 for Linux
<http://www7b.boulder.ibm.com/dmdd/library/tutorials/db2linux/db2linux.html>
- ▶ DBI.perl.org
<http://dbi.perl.org>
- ▶ DB2 Perl Database Interface
<http://www.ibm.com/software/data/db2/perl>
- ▶ Comprehensive Perl Archive Network
<http://www.cpan.org>
http://www.cpan.org/modules/by-category/07_Database_Interfaces/DBI
- ▶ Rapid e-business development for Linux
<http://www.borland.com/kylinx/index.html>
- ▶ VisualAge for Java

<http://www-3.ibm.com/software/ad/vajava>

▶ Net.data

<http://www-3.ibm.com/software/data/net.data>

▶ WebSphere Developer Domain Products

<http://www7b.boulder.ibm.com/wsdd/products>

▶ Apache HTTP Server Project

<http://httpd.apache.org>

▶ Perl.apache.org

<http://perl.apache.org/docs/1.0/guide>

PHP scripting language

<http://php.apache.org>

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

.DEL 243–244
.IXF 243–244
.WSF 243–244

A

aC++ 321
acceptance testing 268
access rights 139, 141
ACID properties 34
Active Data Object Database 183
active log 237
Active subagent 10
ActiveX Data Objects and Remote Data Objects 27
address space 7
administration 60
Administration Client 21, 82
Administration Server 80
administrative authority 142
ADO 27
ADO.NET 28
ADODB 178, 183
adodb-db2.inc.php 183
adodb-mysql.inc.php 183
ADONewConnection() 184
aliases 258
ALTER BUFFERPOOL 301
alter table 102
ANSI/ISO 95
APAR 284
applet driver 190
application 8
Application access 22
application assessment 49
Application Development Client 21
Application level processes 10
application porting 59
application profile 46
application users 138
apxs 85
archived log 237
ASC 244
ASN 277

assessment 46
AST 328
authentication mechanism 224
authorities 141
Authorized Program Analysis Reports 284
autocommit 207, 229
AUTOCONFIGURE 313–314
automatic summary tables (ASTs) 328

B

BDB 36, 100
BerkeleyDB 100
BerkeleyDB (BDB) tables 36
binary data 134–135
Binary Large Objects 134
bind 322
BLOB 134–135, 149
block 333
block based 332
bottleneck 96
buffer pool snapshot 288
buffer pools 301
built-in 90

C

cache 17
Call Level Interface 200
catalog table 13
CCA 277
CHNGPGS_THRESH 302
CLASSPATH 137
CLI 200
CLI Call 277
Client
 Administration 82
 Application Development 82
 Run-Time 82
client proces 7
Client/Server or 2-tier 22
client-server 5
column function 327
columns_priv 138–139
commands

- IMPORT 133
- LOAD 132
- RUNSTATS 150
- commit 99
- common.php 186
- Communication requirement 78
- Compiler 321
- concurrency 99, 230
- Concurrency-Control Manager 31
- configuration 249
- Configuration Advisor 311
- configuration file 11, 251
- Connect() 184
- consistency 124
- consistency token 322
- constraint 111
- CONTROL 142
- Control Center 258
- CPU 101
- crash 99
- crash recovery 236, 239
- CREATE EVENT MONITOR 293
- CREATE PROCEDUR 321
- create procedure 322
- CREATE TRIGGER 324
- cron 248
- cumulative backup 237
- cursor stability 228

D

- d2setup 79
- DAS 79–80
- dasadm1 82
- dasupdt 80
- data checking 270
- data clustering 332
- Data Definition Language 95
- data formats 134
- data integrity 230
- Data Managed Space 98
- Data Manipulation Language (DML) 59
- data migration testing 267
- data movement 241
- data porting 57, 127
- Data type
 - Data and Time Type 92
- data type 90
 - BLOB 134–135, 149

- DATETIME 134
- TIMESTAMP 134, 149
- data types
 - DATETIME 134
 - TIMESTAMP 134
- database 8, 11
- Database Definition Language (DDL) 55
- database level 143
- Database level processes 9
- Database Managed Spaces (DMS) 298
- database manager snapshot 288
- database partition group 12
- database structure porting 55
- database triggers 324
- database users 138
- DATETIME 134–135
- DB.php 186
- DB2 277
 - DB2 administrative API 24
 - DB2 Call Level Interface (DB2 CLI) 25
 - DB2 CLI 25
 - DB2 communication manager 10
 - DB2 coordinating agent 10
 - DB2 daemon spawner 9
 - DB2 Data Warehouse Enterprise Edition 4
 - DB2 Data Warehouse Standard Edition 4
 - DB2 database objects 10
 - DB2 deadlock detector 10
 - DB2 diagnostic logs 278
 - DB2 Everyplace 6
 - DB2 format log 9
 - DB2 Knowledge Base 284
 - DB2 log reader 9
 - DB2 log writer 9
 - DB2 Migrate Now! 49
 - DB2 optimizer 295, 308
 - DB2 page cleaner 9
 - DB2 prefetcher 9
 - DB2 security system 138, 140
 - DB2 subagent 10
 - DB2 system controller 9
 - DB2 system logger 9
 - DB2 TCP manager 10
 - DB2 Technical Support site 283
 - DB2 UDB architecture 6
 - DB2 UDB Connect Application Server Edition 5
 - DB2 UDB Connect Enterprise Edition 5
 - DB2 UDB Connect Personal Edition 5
 - DB2 UDB Connect Unlimited Edition 5

DB2 UDB Enterprise Server Edition 4
 DB2 UDB Express 4
 DB2 UDB Personal Developer's Edition 5
 DB2 UDB Personal Edition 3
 DB2 UDB processes 8
 DB2 UDB Workgroup Server Edition 3
 DB2 UDB Workgroup Server Unlimited Edition 4
 DB2 Universal Database 2
 DB2 Universal Developer's Edition 6
 DB2 version 282
 DB2 watchdog 9
 DB2_PARALLEL_IO 300
 DB2_SQLROUTINE_COMPILE_COMMAND 321
 DB2_SQLROUTINE_COMPILER_PATH 321
 db2AdminMsgWrite 281
 db2advis 314–315
 db2agent 10
 db2agnta 10
 db2agntp 10
 db2ca 255
 db2cc 254
 db2cmdctr 254
 DB2COMM 21
 db2dcdpl 254
 db2diag.log 278–279, 281, 306
 db2dlock 10
 db2empfa 298
 db2eva 255, 294
 db2evmon 294
 db2evtbl 293
 db2fenc1 81
 db2fntlg 9
 db2gds 9
 db2hc 255
 db2ic 255
 db2icrt 81
 db2indbt 255
 db2inst1 81
 db2ipccm 10
 db2isetup 81
 db2iupd 80
 db2jd 190
 db2journal 255
 db2level 282
 db2loggr 9
 db2logw 9
 db2move 246
 db2pclnr 9
 db2pfchr 9
 db2rc 255
 db2set 321
 db2setup 80
 db2setup.log 80
 db2support 283
 db2support.zip 283
 db2sysc 9
 db2syslog 9
 db2tc 255
 db2tccm 10
 db2tm 248
 db2wdog 9
 DBA 277
 DBADM 142
 DBHEAP 307
 DBI 174, 277
 DBI->connect 175
 DDL 95
 delta backup 237
 describe 110
 Design Advisor 314, 316
 DFT_MON_BUFPOOL 287
 DIAGLEVEL 279
 diagnostic logs 278
 Differences 134
 dimension 333
 disconnect 177
 Disk requirement 77
 distinct type 325
 distributions
 Linux 76
 DML 59
 DMS 14, 98, 298
 do(\$sql_statement) 177
 DRDA 191
 Driver Manager 212
 DriverManager 195
 dump files 282
 DWC 278
 Dynamic SQL 25
 dynamic SQL snapshot 289

E

EDU 8
 Embedded DML Precompiler 30
 Embedded SQL for Java (SQLj) 27
 Embedded SQL statements in applications 24
 Engine Dispatchable Units 8

- enterprise 6
- environment variable 17
- event monitoring 285, 292
- EVENT_MON_STATE() 294
- execute() 176
- Execution Engine 31
- executive summary 264
- EXPORT 243
- extent 333

F

- failover 247
- fenced procedures 323
- FixPak 80, 282, 284
- FLG 278
- Forte 321
- functional testing 268

G

- GET MONITOR SWITCHES 286
- GET SNAPSHOT 305
- GNU/Linux g++ 321
- grant 95
- group 140

H

- Hardware requirement 76
- hashing 104
- heap table 36, 104
- high availability 246
- HIPER APAR 284
- host 138–139
- host authentication 221

I

- IBM DB2 Migration Toolkit (MTK) 53–54
- IBM migration offering 48
- IBM Software Support Center 284
- IBM support 284
- IBMDEFAULTBP 301
- IMMEDIATE 301
- implicit privileges 143
- IMPORT 128, 131, 133, 244
- index 98, 106
- Index Advisor 314, 316
- Index Sequential Access Method 35
- individual privileges 142

- initial tuning 296
- InnoDB 100
- InnoDB tables 36
- installation 55
- Installation procedure 78
- installFixPak 80
- instance 8, 11, 96
- instance creation 81
- Instance level processes 9
- in-sync 255
- integration testing 268
- iODBC with ODBC-compliant drivers 177
- ISAM 35, 101
- ISAM tables 35
- isamchk 30
- isolation level 230
- ISV 28

J

- Java Database Connectivity application (JDBC) 25
- Java embedded SQ 27
- JDBC 25
- JDBC driver 137
- JDBC Universal Drive 191

L

- libdb2 211
- Linux
 - IBM validation program 76
- LIST APPLICATIONS 280
- LOAD 128, 131–132, 135, 142, 150
- load authority 142
- load stress testing 268
- loading tools 131
- lock escalation 304, 306
- lock snapshot 288, 305
- Locking 304
- LOCKLIST 304–305
- Log Manager 31
- log shipping 247
- log space 303
- LOGBUFSZ 307–308
- LOGFILSIZ 303
- logical log 299
- logical log buffer 306
- logical model 116
- LOGPRIMARY 303

M

- manual commit 207
- Materialized 328
- materialized query table 104, 328
- materialized views 328
- MAXLOCKS 305
- MDC 105, 331
- Memory Manager 32
- Memory requirement 78
- merge table 103
- messages files 282
- metadata 18
- metadata transport 111
- methodology 186
- MigBlob utility 136
- migration planning 45
- migration process 54
- migration scenarios 52
- migration tools 53
- mobile 6
- mode
 - EXPORT 246
 - IMPORT 246
 - LOAD 246
- MON_HEAP_SZ 292
- monitoring tools 285
- mounting option 96
- MQT 104–105, 328
- MTK 53–54, 73, 112
- Multidimensional clustering 105
- Multidimensional clustering (MDC) 331
- multi-page 297
- multiple server 107
- multi-tier 5, 23
- MyISAM 101
- MyISAM tables 35
- myisamchk 236
- MyODBC 211
- mysql 30
- MySQL AB 28
- MySQL architecture 29
- MySQL clients 30
- MySQL data type
 - Numeric type 91
 - BIGINT 92
 - BIT/BOOL/BOOLEAN 91
 - DOUBLE 92
 - FLOAT 92
 - INTEGER/INT 92

- MEDIUMINT 92

- SMALLINT 91

- TINYINT 91

- MySQL database 28

- MySQL standard SQL compliance 36

- MySQL table type 34

- mysql.columns_priv 138

- mysql.columns_priv tables 139

- mysql.db 138

- mysql.host 138–139

- Mysql.pm 174

- mysql.tables_priv 138–139

- mysql.user 138–139

- Mysql->Connect 174

- mysql_close(\$db) 182

- mysql_connect 179

- mysql_fetch_row() 181

- mysql_pconnect() 179

- mysql_select_db 179

- mysqlaccess 147

- mysqladmin 30

- mysqldump 30, 128, 242

- mysqlhotcopy 242

- mysqlimport 242

N

- native-API driver 189

- NConnect() 184

- NewADOConnection() 184

- NEWLOGPATH 299

- nodegroup 12, 97

- non-persistent 104

- notify files 281

- NOTIFYLEVEL 281

- NUM_IO_CLEANERS 300

- NUM_IOSERVERS 300

O

- Object

- Database 11

- object 99

- object oriented 186, 324

- ODBC 25, 211

- odbc.php 186

- odbc_close(\$db) 182

- odbc_connect 179

- odbc_exec() 181

- odbc_fetch_into() 181

odbc_pconnect() 180
off-line 96
OLE DB 27
open source 28
overflow 104
ownership privilege 142

P

package 322
partition 97
passwords 138, 141
PConnect() 184
PEAR 178
PEAR DB 185
performance 263
performance testing 268
Perl DBI 28
personal digital assistant 6
phpMyAdmin 256
physical model 116
planning 45
point-and-click 258
porting preparation 55
prepare(\$sql_statement) 176
primary key 111
principles of software tests 264
privileges 139–141
 column level 140
 database level 140
 global level 140
 table level 140
problem determination 277
problem determination tools 285
profile registry 250
protocol 21
PUBLIC 144

Q

QLSetConnectAttr 232
Query processing 30
Query Processor 31
query(\$sql_statement) 176

R

RDO 27
recovery 236
recovery log 11

Recovery Management 31
Recovery Manager 31
Redbooks Web site 366
 Contact us xviii
reference type 326
referential integrity 37, 95, 111, 124
regular table 101
Remote Data Objects 27
REORG 308
replication 240
replication summary tables 328
RESET MONITOR 286
Resource Management 31
Resource Manager 32
restart recovery 239
restore 238
result set 187
revoke 95
rollback 99
rollforward recovery 239
root user 79
row level 100
RUNSTATS 150, 308, 310
runtime 232
Run-Time Client 21, 82

S

SAT 278
scalar function 327
schema 14, 33, 99
schema level 143
security 33
services file 21
SET INTEGRITY 272
severity indicator 277
show 110
Single-tier 22
SMPO 48
SMS 14, 98, 297
snapshot monitoring 285
snapshot table functions 289, 291
SNAPSHOT_DYN_SQL() 290
SNAPSHOT_TABLE() 289
SNAPSHOT_TIMESTAMP 290
snapshots 288
Software Migration Project Office (SMPO) 48
special conversions 221
specification 14, 103

- SPM 277
- SQJ 277
- SQL 14, 95, 277
- SQL execution plan 308
- SQL table functions 289
- SQL/PSM 320
- SQL-92 36
- SQL-99 36
- SQLCODE 277
- SQLException 216
- SQLj 27, 192
- sqllogdir 17
- SQLPorter 54
- sqluexpr 244
- SQLWays 54
- standby 247
- Static SQL statements 24
- Storage Manager 32
- stored procedure 38, 320
- structured query language 14, 95
- structured type 325
- Subqueries 37
- suspended I/O 247
- symbolic link 96
- SYSADM 142
- SYSCAT.EVENTMONITORS 293
- SYSCTRL 142
- SYSMOINT 142
- system authority 142
- System catalog table 13
- System Managed Space 98, 297
- system planning 51

T

- table 14, 98
- table function 289, 291
- table functions 289
- table funtion 328
- table level 144
- table snapshot 288
- table space 13, 98, 110, 297
- table space level 143
- table space snapshot 288
- tables_priv 138–139
- TechNotes 284
- temporary table 104
- test deliverables 266
- test documentation 264

- test phases 267, 269
- test planning 264
- test strategy 265
- testing 60, 263
- TIMESTAMP 134–135, 149
- TIMESTAMPFORMAT 134
- tools catalog 80
- Transaction Management 31
- Transactions 36
- transaction-safe 99
- trap files 281
- trigger 38
- tuning 60, 263
- two-tier architecture 51
- Type 2 driver 26
- Type 2 driver, 189
- Type 3 driver 26, 190
- Type 4 driver 26
- typed view 318

U

- UDT 90
- unfenced procedures 323
- Unified ODBC 178
- UNION ALL 104
- unique key 111
- unit of work 229
- unixODBC 178
- UPDATE MONITOR SWITCHES 287
- user 138–139
- user account management 138
 - differences 138
- user data 138, 140
- user defined data type 90
- username 139

V

- version recovery 239
- view 14, 37, 318
- view level 144
- Views 318
- views 318
- Visual Explain 295
- VisualAge 321
- volume stress testing 268

W

wildcard 139
WITH DETAILS 292
work-around 221

Y

ysql.db 139



MySQL to DB2 UDB Conversion Guide

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



MySQL to DB2 UDB Conversion Guide



Redbooks

Complete guide to migrate MySQL database and application to DB2 UDB

DB2 Universal Database (DB2 UDB) has long been known for its technology leadership. This IBM Redbook is an informative guide that describes how to migrate the database system from MySQL to DB2 UDB Version 8.1 on Linux, and how to convert applications to use DB2 UDB instead of MySQL.

Application enrichment through advanced DB2 UDB features

This guide presents the best practices in migration strategy and planning, migration tools, and practical migration examples. It is intended for technical staff involved in a MySQL to DB2 UDB conversion project.

Application migration with detailed examples

This redbook also provides step-by-step instructions for installing and using the IBM DB2 Migration Toolkit (MTK) to port the database objects and data from MySQL to DB2 UDB.

Application programming and conversion considerations are discussed along with the differences in features and functionality of MySQL and DB2 UDB.

Examples are used throughout the book to illustrate conversions of database access, database administration, SQL statements (DDL, DML) and others, as well as testing and tuning your DB2 UDB system.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks