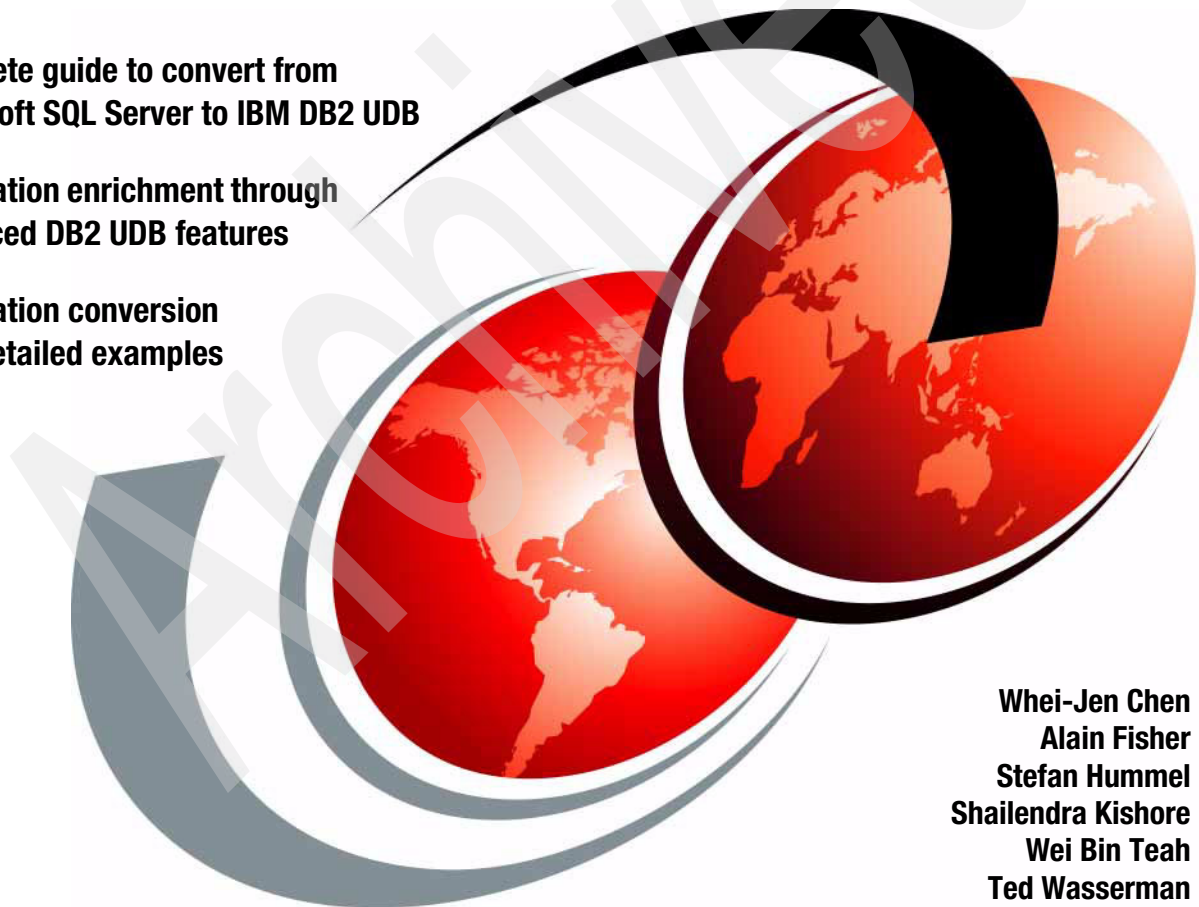


Microsoft SQL Server to IBM DB2 UDB Conversion Guide

Complete guide to convert from
Microsoft SQL Server to IBM DB2 UDB

Application enrichment through
advanced DB2 UDB features

Application conversion
with detailed examples



Whei-Jen Chen
Alain Fisher
Stefan Hummel
Shailendra Kishore
Wei Bin Teah
Ted Wasserman



International Technical Support Organization

**Microsoft SQL Server to IBM DB2 UDB
Conversion Guide**

June 2005

Archived

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

First Edition (June 2005)

This edition applies to DB2 UDB Version 8.2, Microsoft SQL Server 2000, and Windows 2000 Server.

© Copyright International Business Machines Corporation 2005. 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	xiii
Tables	xix
Notices	xxi
Trademarks	xxii
Preface	xxiii
The team that wrote this redbook	xxiv
Acknowledgements	xxv
Become a published author	xxvi
Comments welcome	xxvii
Chapter 1. Introduction	1
1.1 DB2 Universal Database - a high-level overview	2
1.2 Project environment	7
1.3 IBM migration offering	8
1.3.1 DB2 UDB promotion	8
1.3.2 DB2 migration services and support	9
Chapter 2. Architecture	11
2.1 Instances	12
2.1.1 SQL Server instance definition	12
2.1.2 DB2 UDB instance definition	12
2.2 Memory allocation	14
2.2.1 SQL Server memory allocation	14
2.2.2 DB UDB memory allocation	14
2.3 Processes	16
2.3.1 SQL Server process model	16
2.3.2 DB2 UDB process model	17
2.4 Allocating disk space	18
2.4.1 SQL Server disk allocation	18
2.4.2 DB2 UDB disk allocation	18
2.5 Transaction logs	19
2.5.1 SQL Server transaction log overview	20
2.5.2 DB2 UDB transaction log overview	20
2.6 Query optimization	21
2.6.1 SQL Server query optimization	22
2.6.2 DB2 UDB query optimization	22

2.7 Parallelism	23
2.7.1 SQL Server parallelism	23
2.7.2 DB2 UDB parallelism	24
2.8 Message logs	24
2.8.1 SQL Server error logs	24
2.8.2 DB2 error logs	24
2.9 Security	26
2.9.1 SQL Server security	26
2.9.2 DB2 UDB Security	26
2.9.3 Security enhancements in V8.2	29
2.10 High availability	30
2.10.1 SQL Server high availability strategies	30
2.10.2 DB2 high availability strategies	30
Chapter 3. DB2 UDB administration tools	31
3.1 GUI tools	32
3.1.1 Control Center	32
3.1.2 Command Editor	33
3.1.3 SQL Assist	34
3.1.4 Visual Explain	35
3.1.5 Task Center	36
3.1.6 Journal	37
3.1.7 Health Center	38
3.1.8 Development Center	39
3.1.9 Configuration Assistant	40
3.1.10 Information Center	41
3.1.11 License Center	42
3.1.12 Replication Center	43
3.1.13 Data Warehouse Center	44
3.1.14 Information Catalog Center	45
3.1.15 Satellite Administration Center	46
3.1.16 Web administration	47
3.2 Wizards	49
3.3 Advisors	50
3.4 Command Line	52
3.5 Utilities	53
3.5.1 Maintaining database integrity	55
3.5.2 Throttling utilities	57
3.5.3 Validating a backup	58
3.5.4 DDL extraction	58
3.6 Performance monitor integration	61
3.7 Optional tools	62
3.7.1 DB2 Performance Expert	62

3.7.2 DB2 Recovery Expert	62
3.7.3 DB2 High Performance Unload	63
3.7.4 DB2 Test Database Generator	63
3.7.5 DB2 Table Editor	64
3.7.6 DB2 Web Query Tool	64
Chapter 4. SQL considerations	65
4.1 SQL standard compliance	66
4.2 Data types	66
4.2.1 Data type mapping	66
4.2.2 Strong type casting	66
4.2.3 ROWVERSION data type	67
4.2.4 NULL value considerations	69
4.2.5 Large object (LOB) considerations	69
4.3 Date and time considerations	70
4.3.1 Retrieving date and time values	71
4.3.2 Date and time conversion	71
4.3.3 Date and time arithmetic	73
4.3.4 Examining date and time components	74
4.3.5 Additional date and time examples	75
4.4 String considerations	76
4.5 Case sensitivity	77
4.6 SQL language syntax and semantics	79
4.6.1 SELECT statements	79
4.6.2 SELECT INTO	81
4.6.3 INSERT statements	82
4.6.4 UPDATE and DELETE statements	84
4.6.5 TRUNCATE TABLE	84
4.6.6 ANSI joins	85
4.6.7 ORDER BY and GROUP BY clauses	87
4.6.8 Top n clause	87
4.6.9 Cursors	88
4.7 Built-in SQL functions	90
4.8 System catalog queries	91
4.8.1 Catalog interrogation and instance/database metadata	92
4.8.2 System table mapping	94
4.8.3 Authorizations on system catalog tables	95
4.9 Transact-SQL to SQL PL translation	96
4.9.1 EXECUTE <string>	97
4.9.2 PRINT	98
4.9.3 @@ERROR	99
4.9.4 RAISERROR	101
4.9.5 @@SQLSTATUS	101

4.9.6 @@ROWCOUNT	102
4.9.7 @@TRANCOUNT	102
4.10 XML	103
4.10.1 FOR XML statement	104
4.10.2 OPENXML statement	113
4.11 SQL limits	117
Chapter 5. Planning for a conversion	119
5.1 Preparation	120
5.1.1 Performing a porting assessment	120
5.1.2 Understanding and selecting conversion tools	121
5.1.3 Estimating the effort required	122
5.1.4 Planning the conversion	123
5.1.5 Becoming educated on DB2 UDB	123
5.1.6 Environment preparation	124
5.2 Conversion	124
5.2.1 Converting the database structure	124
5.2.2 Converting database objects	126
5.2.3 Porting additional database components and products	126
5.2.4 Modifying the application	127
5.2.5 Modifying the database interface	128
5.2.6 Migrating the data	131
5.3 Post-conversion	132
5.3.1 Performance tuning	132
5.3.2 Utilizing additional DB2 utilities	134
5.3.3 Defining a maintenance strategy	134
5.4 Additional references	135
Chapter 6. The IBM DB2 Migration Toolkit for SQL Server	137
6.1 MTK technical overview	139
6.1.1 Supported operating system and versions	139
6.1.2 Hardware requirements	139
6.1.3 MTK software requirements	139
6.1.4 Where to install MTK	139
6.2 MTK setup	140
6.3 Using MTK	142
6.3.1 MTK GUI interface	142
6.3.2 Migration tasks	143
6.3.3 The MTK SQL Translator	148
Chapter 7. Database structure conversion	151
7.1 Databases	152
7.2 Table spaces	152
7.3 Tables	153

7.3.1 Data types	154
7.3.2 IDENTITY	154
7.3.3 Computed columns	155
7.3.4 Constraints	156
7.4 Indexes	158
7.5 Schemas	159
7.6 Views	160
7.7 Privileges	160
7.8 Examples	162
7.8.1 Database structure conversion using MTK	162
7.8.2 Manually converting database structure	165
Chapter 8. Data and script migration	169
8.1 Introduction	170
8.2 A typical data migration process	170
8.2.1 Migrating a subset of data using MTK	171
8.2.2 Verifying referential integrity	174
8.3 Using Export, Import, and Load	175
8.4 Using WebSphere Information Integrator	177
8.4.1 Accessing SQL Server data from DB2 UDB	178
8.4.2 Replicating data from SQL Server to DB2 UDB	180
8.5 Using other tools	182
8.6 Converting scripts	182
Chapter 9. Converting database objects	185
9.1 Temporary tables	186
9.1.1 Private temporary tables	187
9.1.2 Global temporary tables	188
9.1.3 Additional information	188
9.2 Stored procedures	189
9.2.1 Temporary stored procedures	190
9.2.2 Extended stored procedures	190
9.2.3 Remote procedure calls	191
9.2.4 @@PROCID	191
9.2.5 Simple conversion example	192
9.2.6 Default parameter values	193
9.2.7 Passing parameter values	196
9.2.8 Returning a cursor to the caller	199
9.2.9 Exception handling and transaction control	202
9.2.10 Additional information	208
9.3 User defined functions	208
9.3.1 Scalar functions	209
9.3.2 Table functions	210

9.3.3	Overcoming inline SQL PL limitations in user defined functions . . .	211
9.3.4	Additional information	214
9.4	Triggers	215
9.4.1	Accessing transition values in triggers	216
9.4.2	Simple conversion example	216
9.4.3	IF UPDATE(column)	218
9.4.4	Overcoming inline SQL PL limitations in triggers	219
9.4.5	INSTEAD OF triggers	222
9.4.6	Additional information	223
Chapter 10.	Application conversion considerations	225
10.1	Introduction	226
10.2	Converting to DB2 UDB compatible SQL syntax	226
10.2.1	Using MTK's SQL Translator	226
10.2.2	Manual conversion	228
10.3	Identifying transaction differences	228
10.4	Identifying locking and lock escalation differences	229
10.4.1	Types of locks	230
10.4.2	Lock escalation	233
10.4.3	Deadlock	233
10.5	Identifying isolation level differences	234
10.5.1	Repeatable Read (RR)	236
10.5.2	Read Stability (RS)	237
10.5.3	Cursor Stability (CS)	237
10.5.4	Uncommitted Read (UR)	238
10.6	Identifying and modifying database interfaces	238
10.6.1	ActiveX Data Objects .NET (ADO.NET) interfaces	239
10.6.2	ActiveX Data Object (ADO)	242
10.6.3	Java Database Connectivity (JDBC)	249
10.6.4	Embedded SQL for C (ESQL/C)	249
10.7	Integrating DB2 UDB in Integration Development Environment (IDE)	253
10.7.1	Microsoft .NET add-in (Visual Studio)	253
10.7.2	IBM WebSphere Studio Application Developer	255
10.8	Approaching packaged application migration	256
10.8.1	SAP	257
10.8.2	Siebel	260
Chapter 11.	Performing administrative tasks in DB2 UDB	263
11.1	Working with instances	264
11.1.1	Create, drop, and list instances	264
11.1.2	Attaching / switching instances	265
11.1.3	Start, stop, and quiesce instances	266
11.1.4	Configure instances	266

11.2 Working with databases	267
11.2.1 Create, drop, and list databases	267
11.2.2 Activate and terminate databases	270
11.2.3 Connect, disconnect, and quiesce databases	270
11.2.4 Configure databases	271
11.2.5 List and force off applications	272
11.3 Managing database storage	272
11.3.1 Table spaces and containers	273
11.3.2 Monitoring table space and container storage	284
11.3.3 Transaction logging	289
11.4 Working with buffer pools	294
11.5 Task Center and the DB2 Tools Catalog	296
11.5.1 DB2 Administration Server and Tools Catalog Database	297
11.5.2 Task Center and Scheduler	298
11.6 Backup, recovery, and log administration	305
11.6.1 Automatic backup maintenance	306
11.6.2 Backup using throttling mode	308
11.6.3 Log file inclusion in backup images	308
11.6.4 Backup compression	310
11.6.5 Automated log file management	311
11.6.6 Undo management	312
11.7 High availability	313
11.7.1 Failover clustering	313
11.7.2 HADR	313
11.7.3 Log mirroring	327
11.7.4 Replication	328
11.7.5 Online split mirror and suspended I/O support	329
11.8 REORG and RUNSTATS	330
11.8.1 Database reorganization	330
11.8.2 Database statistics	335
Chapter 12. Post-conversion tuning considerations	339
12.1 Performance tuning	340
12.2 Quick-start tips for performance tuning	340
12.2.1 Design Advisor	341
12.2.2 Configuration Advisor	342
12.2.3 ACTIVATE DATABASE command	345
12.2.4 RUNSTATS and REORG	346
12.3 Configuring automatic maintenance	346
12.4 Other performance tuning advice	350
12.4.1 Table spaces	350
12.4.2 Physical placement of database objects	352
12.4.3 Buffer pools	355

12.4.4	Large transactions	358
12.4.5	Process tuning	363
12.5	Data access strategies	364
12.5.1	Indexing	364
12.5.2	DB2 UDB index expansions	368
12.5.3	Index reorganization	372
12.6	Advanced access methods	372
12.6.1	Materialized query tables	373
12.6.2	Multidimensional clustering (MDC) tables	379
12.7	Optimizer	381
12.7.1	Optimizer analysis	383
12.7.2	Optimizer directives	391
Chapter 13.	Testing and troubleshooting	397
13.1	Planning your testing	398
13.1.1	Principals of software tests	398
13.1.2	Test documentation	398
13.1.3	Test phases	401
13.1.4	Time planning and time exposure	402
13.2	Data checking techniques	403
13.2.1	IMPORT/LOAD messages	404
13.2.2	Data checking	406
13.3	Code and application testing	408
13.3.1	T-SQL to SQL PL object check	408
13.3.2	Application code check	409
13.3.3	Security testing	410
13.3.4	Tools for testing and problem tracking	410
13.4	Troubleshooting	410
13.4.1	Interpreting DB2 informational messages	411
13.4.2	DB2 diagnostic logs	412
13.4.3	DB2 support information	417
Chapter 14.	Conversion scenario	421
14.1	Set up the conversion environment	422
14.2	Create an MTK project	423
14.3	Core database extraction and deployment	424
14.3.1	Specify Source	425
14.3.2	Convert	429
14.3.3	Refine	433
14.3.4	Generate Data Transfer Scripts	438
14.3.5	Deploy to DB2	441
14.4	Other database object conversion	444
14.4.1	View conversion	445

14.4.2	Function conversion	448
14.4.3	Stored procedure conversion	452
14.4.4	Trigger conversion	456
14.5	Application conversion	461
14.5.1	Overview of the C# sample application	461
14.5.2	Set up Visual Studio .NET for the DB2 UDB environment	462
14.5.3	Add a DB2 library reference to .NET classes	463
14.5.4	Replace interface-specific objects	463
14.5.5	Database connection statement	464
14.5.6	Summary	464
Appendix A. Terminology mapping		465
A.1	SQL Server to DB2 UDB terminology comparison	466
Appendix B. Data type mapping		469
B.1	Mapping SQL Server data types to DB2 data types	470
B.2	Supported SQL data types in C/C++	471
B.3	SQL data types for the DB2 .NET Data Provider	476
B.4	Supported SQL data types in Java	478
Appendix C. Function mapping		481
C.1	Installation of additional built-in functions	482
C.2	Mathematical functions	483
C.3	Character and string functions	484
C.4	Boolean functions	486
C.5	Date and time functions	487
C.6	Metadata functions	488
C.7	Aggregate functions	490
C.8	System functions	491
C.9	Security functions	493
C.10	Miscellaneous functions	494
Appendix D. Operator mapping		495
D.1	Arithmetic operators	496
D.2	Assignment operators	496
D.3	String concatenation operators	496
D.4	Comparison operators	497
D.5	Logical operators	497
D.6	Bitwise operators	498
Appendix E. Administrative tasks mapping		501
Appendix F. SQL limits		505
F.1	Identifier length limits	506

F.2 Database limits	506
Appendix G. Additional material	509
Locating the Web material	509
Using the Web material	509
System requirements for downloading the Web material	511
How to use the Web material	511
Related publications	513
IBM Redbooks	513
Other publications	513
Online resources	514
How to get IBM Redbooks	515
Help from IBM	515
Index	517

Figures

1-1	Project environment	7
2-1	DB2 architecture showing a single instance	13
2-2	DB2 shared memory architecture	15
2-3	db2diag.log example	25
3-1	Control Center	33
3-2	Command Editor	34
3-3	Sample SQL Assist output	35
3-4	Visual Explain output	36
3-5	Task Center	37
3-6	Journal	38
3-7	Health Center	39
3-8	Development Center	40
3-9	Configuration Assistant	41
3-10	Information Center	42
3-11	License Center	43
3-12	Replication Center Q replication launchpad	44
3-13	Data Warehouse Center	45
3-14	Information Catalog Center	46
3-15	Satellite Administration Center	47
3-16	Web Command Center showing script execution	48
3-17	Web Command Center with script execution result	49
3-18	Wizards Launchpad	50
3-19	Configuration Advisor output	51
3-20	CLP in interactive mode	52
3-21	CLP in command mode	53
3-22	DB2LOOK GUI utility	59
6-1	MTK Installation - Welcome screen	140
6-2	MTK Installation - Agreement screen	141
6-3	MTK Installation- Installation folder	141
6-4	MTK Installation - Progress screen	142
6-5	MTK Installation - Install Complete Screen	142
6-6	MTK Main window	143
6-7	Specify source	144
6-8	Convert	145
6-9	Refine	146
6-10	Generate data transfer script	146
6-11	Deploy to DB2	147
6-12	Overview of MTK conversion tasks	148

6-13	MTK SQL Translator	149
8-1	Suggested data migration process	171
8-2	Extract table data in the REDBOOK database within SQL Server . . .	172
8-3	Global Type Mapping summary	172
8-4	Generate Data Transfer Scripts in MTK.	173
8-5	Migrating a subset of the database data using MTK	173
8-6	Creating nickname using Control Center	178
8-7	SQL Server tables accessed in Control Center through nicknames . .	180
8-8	Show command pop-up window	183
8-9	Importing SQL scripts into Task Center	184
10-1	Opening the SQL Translator in MTK	227
10-2	SQL Translator after converting a SQL Server SQL statement	228
10-3	DB2 UDB LOCK TABLE syntax	232
10-4	CONNECT database statement in DB2 UDB with lock mode	233
10-5	Providers in .NET to connect to DB2 UDB.	240
10-6	ODBC Driver Manager environment versus DB2 CLI environment . .	246
10-7	Procedure to build ESQL/C application with DB2 UDB	252
10-8	Development Center integrated into WebSphere	256
10-9	SAP migration project plan	259
10-10	Siebel migration process	260
11-1	Attaching to an instance using Control Center.	265
11-2	Creating database in Control Center	268
11-3	Show Command from the standard create database wizard	269
11-4	Configure database logging after creation of database	269
11-5	Connect, disconnect, quiesce, and unquiesce menu options	271
11-6	Configuring database parameters in Control Center	271
11-7	Default Table Spaces.	273
11-8	Create Table Space Wizard.	278
11-9	Create Table Space Wizard: Type of table space	279
11-10	Create Table Space Wizard: select buffer pool	280
11-11	Create table space wizard: container options	281
11-12	Create table space: summary	282
11-13	Add table space container wizard	284
11-14	Storage management interface	286
11-15	Configure Database Logging Wizard	292
11-16	Configure Database Logging Wizard - Summary screen.	293
11-17	Create buffer pool window	296
11-18	Creating the Tools Catalog from the Control Center	298
11-19	Entering task information in Task Center.	300
11-20	Entering script information in Task Center.	301
11-21	Entering task run properties.	301
11-22	Scheduling the task in Task Center	302
11-23	Entering notification information in Task Center	303

11-24 Specifying task action in Task Center	304
11-25 Specifying task security in Task Center	304
11-26 Saving and scheduling a task from the Backup Wizard	305
11-27 Configure Automatic Maintenance Wizard	307
11-28 Enable throttling and set priority for backup utility	308
11-29 Include log files in backup images during online backup operation	309
11-30 Restore log files into a specified directory from a backup image	310
11-31 HADR Wizard: Confirm primary database selection	315
11-32 HADR Wizard: Select a standby database and initialization method	316
11-33 HADR Wizard: Identify a backup image to populate standby database	317
11-34 HADR Wizard: Restore the database on the standby system	318
11-35 HADR Wizard: Copy objects to the standby system	319
11-36 HADR Wizard: Configure Communications	320
11-37 HADR Wizard: Configure automatic client re-route	321
11-38 HADR Wizard: Selecting synchronization mode	322
11-39 HADR Wizard: Summary of information	323
11-40 HADR Wizard: HADR setup confirmation dialog	325
11-41 Manage HADR	326
11-42 HADR versus queue replication	328
11-43 Reorganizing a table in Control Center	334
11-44 Executing RUNSTATS using Control Center	337
11-45 Query plans before and after a RUNSTATS run	338
12-1 Design Advisor	342
12-2 Scheduling Configuration Advisor recommendations	343
12-3 Configuration Advisor recommendations	344
12-4 Configure automatic maintenance summary	349
12-5 Explaining logical log	353
12-6 Visualizing CHNGPGS_THRESH parameter	357
12-7 Maximum number of locks available for default settings on Linux	359
12-8 Explaining lock snapshot information	361
12-9 Estimating the index size using the Create Index wizard	367
12-10 Schematic view of a composite index	369
12-11 Schematic view of an INCLUDE index	370
12-12 MQT routing	376
12-13 Visualization of a two-dimensional MDC table	380
12-14 Block access using an MDC table	381
12-15 Steps performed by the SQL compiler	382
12-16 Launch Visual Explain from Control Center	384
12-17 Explain SQL statement in Visual Explain	385
12-18 Query plan generated by Visual Explain	386
12-19 Details of an optimizer step Visual Explain	387
12-20 Comparison of query plans with optimization classes 0 and 5	395
13-1 Test phases during a migration project	403

13-2	Table definition for Example 13-1	405
13-3	Data file for Example 13-1	405
13-4	Development Center debug window	409
14-1	MTK - Welcome window	423
14-2	MTK - Project Management dialog window	424
14-3	Database conversion process using MTK	425
14-4	MTK - Specify Source tab	426
14-5	MTK - Connect to Database dialog window	427
14-6	MTK - Extract dialog window for tables	428
14-7	MTK - Specify Source tab with extracted table script selected	429
14-8	MTK - Convert tab after converting tables	430
14-9	MTK - Global Type Mapping window	431
14-10	MTK - Data Type Editor dialog window	431
14-11	MTK - Global Type Mapping changes	432
14-12	MTK - Advance Options window (for conversions)	433
14-13	MTK - Refine tab	434
14-14	MTK - Translator Information message tree (for tables)	434
14-15	MTK - Translator Warning message tree (for tables)	435
14-16	MTK - Translator Warning message tree (for tables)	435
14-17	MTK - Refine tab including source and target code subtabs, for tables	436
14-18	MTK - Refine tab (change recommended object name for indexes) ..	437
14-19	MTK - Refine tab (Translation success ratio)	438
14-20	MTK - Generate Data Transfer Scripts tab	439
14-21	MTK - Generate Data Transfer Scripts tab (data loading options) ..	439
14-22	MTK - Generate Data Transfer Scripts tab (LOAD/IMPORT mode) ..	440
14-23	MTK - Generate Data Transfer Scripts tab (file format options)	440
14-24	MTK - Deploy to DB2 tab (for tables)	442
14-25	MTK - Deploy to DB2 tab (first deployment step)	443
14-26	MTK - Deploy to DB2 tab (second deployment step)	443
14-27	MTK - Deploy to DB2 tab (third deployment option)	444
14-28	MTK - Deploy to DB2 verification report (for tables)	444
14-29	MTK - Extract dialog window for Views	445
14-30	MTK - Convert tab after setting the context (for Views)	446
14-31	MTK - Refine tab after conversion (translation success ratio for Views) ..	447
14-32	MTK - After deploying functions to DB2 UDB	448
14-33	MTK - Convert tab (choosing a previous conversion activity)	448
14-34	MTK - Extract dialog window for user defined functions	449
14-35	MTK - Set Context window (for functions)	450
14-36	MTK - Convert context indicator (for functions)	450
14-37	MTK - Refining converted functions	451
14-38	MTK - Refine tab translation message error (for stored procedures) ..	453
14-39	MTK - Refine tab after modifying a SQL Server stored procedure ..	455
14-40	MTK - Refine tab (for triggers)	456

14-41 MTK - Refine tab (for triggers)	457
14-42 MTK - Refine tab (renaming of multi-statement triggers)	458
14-43 Sample C# application - Login window	461
14-44 Sample C# application - Author Search window	462

Archived

Tables

2-1	DB2 UDB process list (Windows)	17
3-1	SQL Server and DB2 utilities comparison	54
4-1	Mapping of T-SQL DATEPART() function parameters	74
4-2	Lengths of various character values	76
4-3	Mapping proprietary outer join syntax to DB2 UDB	86
4-4	Mapping of cross join syntax to DB2 UDB	87
4-5	Mapping of SQL Server system tables to DB2 UDB catalog tables.	95
4-6	Mapping of common T-SQL programming constructs to SQL PL	96
7-1	CREATE TABLE syntax - compare between SQL Server, DB2 UDB	154
7-2	IDENTITY column syntax - compare between SQL Server, DB2 UDB	155
7-3	Computed column syntax - compare between SQL Server, DB2 UDB	156
7-4	Primary key syntax - compare between SQL Server and DB2 UDB	156
7-5	Check constraint syntax - compare between SQL Server, DB2 UDB	157
7-6	Clustered index syntax-compare between SQL Server and DB2 UDB	158
7-7	Non-clustered index syntax-compare between SQL Server, DB2 UDB	158
7-8	View syntax - compare between SQL Server and DB2 UDB.	160
7-9	SQL Server user list for the REDBOOK database.	161
7-10	DB2 UDB object privilege list for the REDBOOK database.	161
8-1	Compare between query using nickname in SQL Server, DB2 UDB	180
9-1	Different invocation of SQL Server procedure shown in Example 9-14	197
9-2	Different invocation of the equivalent DB2 UDB procedure	198
10-1	Comparison of SQL Server and DB2 UDB locking levels	229
10-2	A complete list of DB2 UDB locks	230
10-3	A comparison of SQL Server and DB2 UDB isolation levels	234
10-4	.NET provider code-level comparison	240
10-5	Mapping of OLE DB data types between SQL Server and DB2 UDB	244
10-6	IBM OLE DB Driver non-supported components and interfaces	245
10-7	DB2 CLI and ODBC function mapping.	246
11-1	Recommendations for default table spaces.	275
11-2	Basic statistics command - compare between SQL Server, DB2 UDB	336
12-1	Parameters for the AUTOCONFIGURE command	345
14-1	Input to MTK project management dialog	423
14-2	A comparison of DB2 UDB's IMPORT and LOAD utilities	440
A-1	Physical objects	466
A-2	Logical objects	466
A-3	Database objects	466
A-4	Administration and usage	467
B-1	SQL Server data types mapped to DB2 data types	470

B-2	DB2 UDB SQL data types mapped to C/C++ data types	472
B-3	SQL data types mapped to .NET data type	476
B-4	SQL data types mapped to Java declarations	478
C-1	Mathematical function mapping for SQL Server to DB2	483
C-2	Character and string function mapping for SQL Server to DB2 UDB	484
C-3	Boolean function mapping from SQL Server to DB2 UDB	486
C-4	Date and time function mapping from SQL Server to DB2 UDB	487
C-5	Metadata function mapping from SQL Server to DB2 UDB	488
C-6	Aggregate function mapping from SQL Server to DB2 UDB	490
C-7	System function mapping from SQL Server to DB2 UDB	491
C-8	Security function mapping from SQL Server to DB2 UDB	493
C-9	Miscellaneous function mapping from SQL Server to DB2 UDB	494
D-1	Arithmetic operator mapping from SQL Server to DB2 UDB	496
D-2	Assignment operator mapping from SQL Server to DB2 UDB	496
D-3	String concatenation operator mapping from SQL Server to DB2 UDB	496
D-4	Comparison operator mapping from SQL Server to DB2 UDB	497
D-5	Logical operator mapping from SQL Server to DB2 UDB	497
D-6	Bitwise operators mapping from SQL Server to DB2 UDB	499
14-3	SQL Server and DB2 UDB comparable tasks	501
F-1	SQL Server 2000 and DB2 UDB V8.2 identifier limits	506
F-2	SQL Server 2000 and DB2 UDB V8.2 database limits	506

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:

@server™
developerWorks®
e-business on demand™
ibm.com®
iSeries™
i5/OS™
xSeries®
z/OS®
zSeries®
AIX®
AS/400®

DB2 Extenders™
DB2 Universal Database™
DB2®
Everyplace®
HACMP™
Informix®
IBM®
MVS™
MVS/ESA™
Notes®
OS/390®

POWER™
Rational Suite®
Rational®
Redbooks™
Redbooks (logo) ™
RETAIN®
S/390®
TestStudio®
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

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.

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

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

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

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

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

Preface

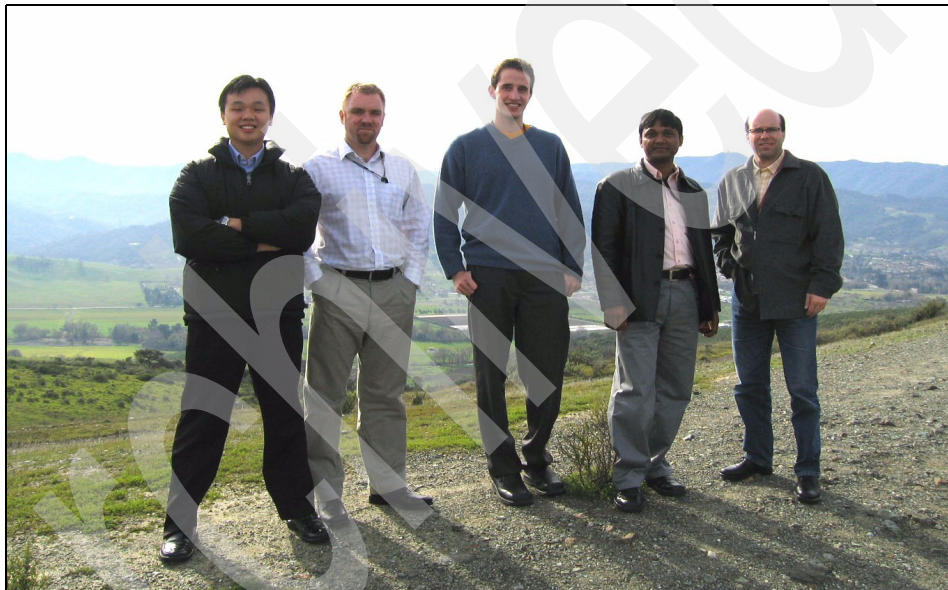
This IBM® Redbook provides procedures and examples for converting a database and application from Microsoft® SQL Server to DB2® Universal Database™. We focus on the technical considerations and methodology involved in performing the database and application conversion. The redbook is organized as follows.

- ▶ **Chapter 1** provides an brief overview of the DB2 UDB family, the project environment used in this book and the IBM migration offering for customers, independent software vendors (ISVs) and IBM Business Partners.
- ▶ **Chapter 2** provides an architecture comparison between Microsoft SQL Server and DB2 UDB.
- ▶ **Chapter 3** provides an overview of the administrative tools, Wizards and Advisors that help in the transition to DB2 UDB, as well as many other tools and utilities that are available.
- ▶ **Chapter 4** highlights the SQL language syntactical and semantic differences between SQL Server and DB2 UDB as well as other SQL language and limitation differences.
- ▶ **Chapter 5** describes the process of migration planning, including the preparation, conversion, and post conversion tasks.
- ▶ **Chapter 6** introduces the IBM DB2 Migration Toolkit (MTK) for Microsoft SQL Server including a MTK technical overview, as well as setup and usage guideline.
- ▶ **Chapter 7** discusses the process of converting the database structure from SQL Server to DB2 UDB. It also provides a database structure comparison and conversion examples.
- ▶ **Chapter 8** describes how to migrate data from SQL Server to DB2 UDB, while ensuring that data is migrated completely, consistently, and within a reasonable time frame.
- ▶ **Chapter 9** highlights the key differences between database objects in SQL Server and DB2 UDB and provides examples that can serve as a reference during a conversion to DB2 UDB.
- ▶ **Chapter 10** provides guidance in application and interface modifications that are necessary for applications to run on DB2 UDB.
- ▶ **Chapter 11** discusses basic and intermediate administration tasks including working with instances and databases, storage management, and automating administrative tasks.

- ▶ **Chapter 12** highlights the important areas where initial performance tuning efforts should be directed.
- ▶ **Chapter 13** discusses the steps to verify that data and application functionality are ported completely and correctly.
- ▶ **Chapter 14** provides a complete migration scenario using the MTK.

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.



Left to right: Wei Bin Teah, Alain Fisher, Ted J. Wasserman, Shailendra Kishore, Stefan Hummel

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.

Alain Fisher is a Software Support Specialist for IBM Technical Support and Remote Operations in the United Kingdom. He has eight years of experience in the IT field and as a member of the North Region Information Management team

supports a wide range of DB2 customers on the Windows®, UNIX® and Linux® platforms. He also specializes in VMWare support.

Stefan Hummel is an IT Specialist for competitive database migrations for IBM Germany. He has more than eleven years of IT experience using a wide range of client and server platforms. His experience includes application development as well as database design, implementation, and administration of SQL Server, Oracle, MySQL and DB2 databases on UNIX, Linux, and Windows. He is a IBM Certified Solutions Expert for DB2 UDB Application Development and Administration.

Shailendra Kishore is a Technical Consultant at the IBM Innovation Center in India. He has seven years of IT experience in the IT field. He provides technical support in Database migrations to IBM Business Partners. He is an IBM Certified Database Administrator and Application Developer.

Wei Bin Teah is an IT Specialist for ASEAN and A/NZ Techline for IBM Malaysia. He holds a bachelor's degree in Computer Engineering from Purdue University, West Lafayette, Indiana. His area of expertise is in pre-sales technical sales support for DB2 information management software. He is an IBM Certified Advanced Database Administrator, IBM Certified Solution Designer for DB2 Business Intelligence, and IBM Certified Specialist for Grid Computing Technical Sales.

Ted Wasserman is an IBM employee based out of the IBM Silicon Valley Laboratory where parts of the DB2 product are developed. Ted works as a Database Consultant on the DB2 Business Partner Technical Enablement team where his main responsibility is to provide technical assistance to IBM Business Partners during migrations to DB2 UDB. Ted has a master's degree in Computer Science, as well as a Hon. B.Sc. in Computer Science from Queen's University (Kingston, Ontario, Canada).

Acknowledgements

The authors express their deep gratitude for the help received from **Jie Zhang** who contributed advice, support, and written content.

Jie Zhang is a Technical Consultant specializing in WebSphere® Information Integrator (II). He works in the IBM Information Management Business Partner Technical Enablement (BPTE) organization. Jie joined the BPTE organization in 2001, and has worked on many WebSphere II and DB2 related projects, including conducting workshops on WebSphere II worldwide.

We also thank the following people for their support and contributions to this book:

Grant Hutchison
IBM DB2 Marketing, Software Group, Toronto Laboratory, Canada

Michael Gao
IBM Software Group, Toronto Laboratory, Canada

Marina Greenstein
Arthur V. Sammartino
IBM Software Migration Project Office, USA

Abdul Al-Azzawe
IBM Software Group, San Francisco, USA

Martin Mezger
Sven-Uwe Kusche
IBM SAP DB2 Center of Expertise, IBM Germany

Eva Billich
IBM Software Group, IBM Germany

Michael Spoden
IBM Global Service, IBM Germany

Sandy Gregus
Michael Logan
Jay Pederson
Sam Qita
IBM DB2 Product Manager, Software Group

Shawn Moe
IBM Software Group, IBM DB2 Migration Toolkit

Emma Jacobs
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 E-mail 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

Database management software is now the core of enterprise computing. Companies need access to a wide range of information such as XML documents, streaming video, and other rich media types. New ways of working bring new requirements, including digital rights management. The e-business evolution makes continuous availability a necessity and is driving the convergence of transaction, business intelligence, and content management applications as companies integrate their business operations. DB2 Universal Database (DB2 UDB) for Linux, UNIX, and Windows can help your organization meet these challenges.

The goal of this book is to provide information that can help SQL Server customers understand, plan for, and execute a migration to DB2 UDB in the easiest possible manner.

This book introduces DB2 UDB products and their capabilities, discusses converting databases and applications, and describes the most important aspects of converting applications from Microsoft SQL Server to DB2 UDB. It describes the differences between their architecture, the tools for administration and development, the data definition language (DDL), the data modeling, SQL considerations, data conversion, and application conversion.

1.1 DB2 Universal Database - a high-level overview

DB2 UDB is a true cross-platform relational database management system (RDBMS), running on a wide variety of systems including Windows, Solaris, HP-UX, AIX®, and Linux.

DB2 UDB responds quickly to peaks in transaction demand, expands to hold growing amounts of information that may be distributed over a number of different databases, and grows with your information infrastructure from one processor, to multiple processors, to massively parallel clusters. The integration of partitioning and clustering technology into DB2® Universal Database Enterprise Server Edition (ESE) means that it is flexible enough to meet future growth.

As the real database leader in several technologies, DB2 UDB provides the following capabilities:

- ▶ Integrated support for complex data such as text documents, images, video and audio clips
- ▶ Integrated Web access through native support for Java™, Java Database Connectivity (JDBC), embedded SQL for Java (SQLJ) and Microsoft .NET
- ▶ Integrated system management tools
- ▶ Data replication services
- ▶ High availability disaster recovery (HADR)

The DB2 information management software portfolio provides the functionality you need to enable your on demand business. DB2 encompasses a broad range of products under the DB2 brand. In this discussion, we highlight the DB2 UDB products that pertain to users making the transition from SQL Server.

At the core of the DB2 portfolio is the DB2 UDB database server. IBM DB2 UDB Version 8 for Linux, UNIX, and Windows is leading the evolution of the relational database. DB2 UDB is the database of choice for the development and deployment of critical solutions such as:

- ▶ e-business
- ▶ Business intelligence
- ▶ Content management
- ▶ Enterprise resource planning
- ▶ Customer relationship management

Innovative manageability

DB2 UDB greatly reduces the complexity of data management by simplifying, automating, or eliminating many tasks traditionally associated with maintaining

an enterprise-class database. Some of these advances represent steps toward making full autonomic computing a reality for database implementations. Some examples of innovative manageability include:

- ▶ Configuration Advisor puts the knowledge of a seasoned DBA at your fingertips.
- ▶ Health Center/monitor keeps your database functioning and provides real-time alerts to a variety of interfaces, such as e-mail and pagers.
- ▶ Memory Visualizer lets you dynamically view and control DB2 UDB memory usage.
- ▶ Advisors deliver expert advice about optimizing database access.
- ▶ Simplified management of large-scale partitioned databases.
- ▶ Automated database maintenance, including database statistics and reorganization.

Integrated information

DB2 UDB provides a strong foundation of information integration technologies, including federation, replication, Web services, and XML. With DB2 UDB built-in capabilities, you can query, update, and replicate data across DB2 UDB and other data sources. This is a key capability for those transition to DB2 UDB. Information integration technologies include:

- ▶ Federated Web services give you the ability to publish and consume data from various sources, such as another DBMS and XML.
- ▶ XML productivity tools simplify integrating XML, giving you the ability to store, retrieve, and decompose XML structures.
- ▶ Enriched data type support. DB2 provides rich type support for spatial data, text data, and flat files. Tools that enable these capabilities include the DB2 Spatial Extender, DB2 Net Search Extender, and DB2 Data Links Manager.

Robust foundation for 24x7 operations

Around-the-clock operations are now requirements in the e-business on demand™ environment. With DB2 UDB, IBM has provided operational features to meet this demand. Examples include:

- ▶ High Availability Disaster Recovery (HADR) is new with V8.2. It enables you to fail over your DB2 environment to a backup environment in the event of an outage, without relying on external software.
- ▶ Connection Concentrator for more user scalability. The Connection Concentrator is included with DB2 UDB and provides for multiple user sessions over fewer physical connections, thus conserving system resources.

- ▶ Dynamic parameter configuration enables you to reconfigure key parameters without having to bring down the database or instance.
- ▶ In-place online reorganization.
- ▶ Online load.
- ▶ Online storage management.
- ▶ Mobility on demand feature. DB2 UDB customers can easily extend their solutions to include mobile data by leveraging the new mobility on demand capability available with DB2 UDB. The mobility on demand capability, based on DB2 Everyplace® technology, includes the high-performance, robust DB2 Everyplace database and a powerful synchronization solution for use with an existing DB2 UDB deployment. This enables wireless and pervasive devices to occasionally run connected to DB2 UDB applications.

Integrated business intelligence

Business intelligence capabilities and performance are key strengths of DB2 UDB. The philosophy is to identify features for query performance and drive them directly into the database engine, greatly speeding response time. As examples:

- ▶ Multidimensional clustering (MDC) improves performance of complex queries by storing table data physically clustered in blocks, according to common values of multiple columns.
- ▶ Advisors that analyze your data and schema, and then recommend appropriate indexes and Materialized Query Tables (MQT, precomputed physical views of data based on a query) for improved performance with less administrative effort.

Enhanced application development productivity

DB2 UDB offers an extensive toolkit for building applications. These application development tools focus on maximizing programmer productivity by providing support for major application frameworks popular with both Java and Microsoft application programmers. Capabilities include:

- ▶ DB2 Development Center for creating server-side objects, such as stored procedures in SQL or Common Language Runtime (CLR) procedures in C#, and user-defined functions in SQL, MQ, and XML.
- ▶ Integration in Microsoft Visual Studio as well as in IBM WebSphere.
- ▶ Enhanced drivers for applications written to .NET, ADO, ODBC, OLE DB, DB2 CLI, JDBC, and SQLJ programming interfaces.

- Visual Studio .NET wizards to create DB2 UDB server side application objects

Note: One of the most exciting new features of DB2 UDB Version 8.2 is support for Common Language Runtime (CLR) procedures. You can code C# and VB.NET methods in class assemblies and then deploy them as DB2 routines. DB2 CLR routines enable you to code your procedures using your choice of .NET language.

This feature is not available in SQL Server yet.

DB2 family of products

The IBM Information Management software portfolio offers a wide range of products to accommodate different business needs and technical requirements that provide customers with a robust and scalable enterprise wide solution.

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

- **DB2 UDB Enterprise Server Edition (ESE)**
DB2 UDB ESE is designed to meet the database server needs of midsize to large businesses. ESE's high scalability, availability, and reliability features provide customers an ideal database management system for all types of transactions.
 - **The Database Partitioning Feature (DPF)**
The Database Partitioning Feature is a licensing option that allows ESE customers to partition a database within a single system or across a cluster of systems. The DPF capability provides the customer with multiple benefits including scalability to support very large databases, or complex workloads and increased parallelism for administration tasks.
- **DB2 Workgroup Server Edition (WSE)**
This Edition is designed for deployment at a departmental level or in a small business environment with a small number of users. It can be deployed on a server with up to four CPUs.
- **DB2 Workgroup Server Unlimited Edition (WSUE)**
This product offers a simplified per processor licensing for deployment at a departmental level or in a small business environment.

- ▶ **DB2 Personal Edition (PE)**
This flavor of DB2 provides a database management system for a single user database.
- ▶ **DB2 UDB Developers Edition**
This product offers a package for single application developer to design and build an application.
- ▶ **DB2 UDB Personal Developer's Edition (PDE)**
Similar to the Developer's edition, this product enables the developer to build a single user desktop application.
- ▶ **DB2 Express**
This is the newest member of DB2 UDB product family. The key features include simplified deployment, autonomic management capabilities, and application development support and design for 7/24 operation. This flavor of DB2 UDB is aimed at Independent Software Vendors (ISV) who want to integrate DB2 UDB as part of their application with low cost, and have the capability to expand easily in the future.

DB2 Information Management software extends the functionality of DB2 UDB by offering different technologies called *extenders*. These extenders include:

- ▶ Text Extender: for performing text searches
- ▶ Image Extender: for image manipulation operations
- ▶ XML Extender: enables storing and retrieving data in the form of XML documents
- ▶ Spatial Extend: manipulates spatial data
- ▶ Audio and Video Extenders: handles both audio and video files respectively.

IBM also provides federated technologies to extend the functionality of DB2 UDB. With WebSphere Information Integrator (WebSphere II) you can access objects in many different databases such as Oracle and SQL Server with a single query.

In addition, DB2 Connect Editions offer you the capability to extend your enterprise system to access legacy systems.

Tip: For further information, refer to the IBM Web site

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

Another helpful article for this topic is "Which distributed edition of DB2 Version 8 is right for you?" available at:

<http://www.ibm.com/developerworks/db2/library/techarticle/0211zikopoulos/0211zikopoulos.html>

1.2 Project environment

For this redbook, we built an environment with two database servers shown in Figure 1-1 that simulates a real customer environment. On the first server, we installed Microsoft SQL Server 2000 with a sample database named REDB00K. We use this database for our sample code contained in the following chapters. A complete description of our database is provided in Chapter 14, “Conversion scenario” on page 421. The second machine is the target DB2 UDB server. For the client, we used ordinary workstation PCs. The source files of the REDB00K database for SQL Server and DB2 UDB are available as download in Appendix G, “Additional material” on page 509.

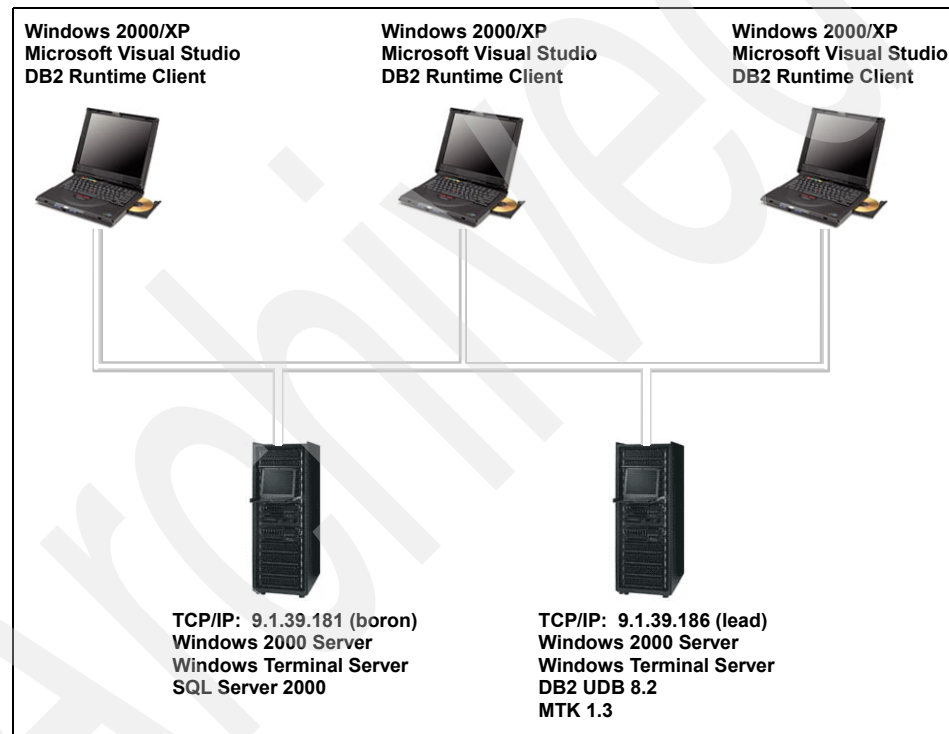


Figure 1-1 Project environment

In summary, our environment is as follows:

- 2 x IBM @server xSeries® 330, 1 GHz, RAM 4 GB
Windows 2000 Server with SP4
Windows Terminal Server

- ▶ Workstation PCs
e.g., IBM Thinkpad T40, Pentium® M 1.5 GHz, RAM 1GB
Windows 2000 or XP
- ▶ DB2 Version 8.2 Enterprise Server Edition with FixPak 8
- ▶ Microsoft SQL Server 2000
- ▶ DB2 Migration Toolkit (MTK) Version 1.3
- ▶ Microsoft SQL Server ODBC Driver Version 03.81.9031

We do not cover Recovery and High Availability (HA) in our environment; however it is given some mention in Chapter 11, “Performing administrative tasks in DB2 UDB” on page 263 and we also discuss HA considerations in Chapter 5, “Planning for a conversion” on page 119.

1.3 IBM migration offering

IBM gathered significant experience in performing database migrations over the past several years. This know-how is offered to everybody in the form of free of charge promotions as well as low priced software licenses and services.

1.3.1 DB2 UDB promotion

Migrate Now! for DB2 Universal Database facilitates the migration from Microsoft SQL Server, Oracle, Sybase, and additional database platforms to DB2 UDB at a special price with a significant discount. *Migrate Now!* is an end-to-end offering that includes:

- ▶ Migration tool kits
- ▶ Free online education
- ▶ Sales teams and resources to assist you in planning and implementing your migration based on the IBM proven methodology

Take advantage of this special offer and Migrate Now!

For more information about this promotion, refer to the following Web site:

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

Or, contact your IBM representative.

1.3.2 DB2 migration services and support

IBM employs a proven migration methodology which helps reduce costs and risks associated with a migration while addressing its major components:

- ▶ Applications
- ▶ Database design
- ▶ Data

Migration consultants can advise your organization regarding a phased migration approach. This approach includes:

- ▶ Assessment of the database conversion effort
- ▶ DB2 UDB migration assessment
- ▶ Installation of DB2 UDB and other products you may require
- ▶ Pilot migration
- ▶ Full migration of remaining data and applications

IBM migration specialists have provided advice and counsel to over 70 customers and performed over 3500 migrations to DB2 UDB from non-IBM database systems like Microsoft SQL Server, Oracle, Sybase, MySQL, ProgreSQL, ADABAS, IDMS, SUPRA, TOTAL, CA-IDMS, CA-Datcom and VSAM. If you would like IBM to perform your migration, please send an e-mail to the appropriate contact on the following Web site:

<http://www.ibm.com/software/solutions/softwaremigration/contacts.html>

Technical training

IBM offers a workshop designed for ISVs and IBM Business Partners intending to sell applications on DB2 UDB, and who are new to DB2 UDB. It begins with building fundamental DB2 UDB skills, where participants learn how to migrate their existing database to DB2 UDB. You actually bring your own database to the workshop. In some cases (depending on database complexity), attendees have been able to completely migrate their database within the five day period. Detail information for DB2 Enablement workshops can be found at:

http://www.developer.ibm.com/spc/events/db2_en.html

Contacts

IBM services are available for assistance during any part the migration. If you have a migration project in mind or you want to obtain the latest conversion information (e.g., porting guides), please contact:

- ▶ 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:dungj@hkl.ibm.com>

Architecture

In this chapter we compare the architectures of the Microsoft SQL Server (SQL Server) and DB2 Universal Database (DB2 UDB) relational database management system (RDBMS) products. Understanding these differences will aid in the transition from SQL Server to DB2 UDB. The architectural differences and terminologies are compared at a high level. It is not within the scope of this book to cover in-depth configuration or performance tuning.

Topics covered in this chapter include:

- ▶ Instances
- ▶ Memory overview
- ▶ Processes
- ▶ Disk Space Allocation
- ▶ Transaction logs
- ▶ Query Optimization
- ▶ Message logs
- ▶ Security
- ▶ High Availability

2.1 Instances

The term instance is used in both SQL server and DB2 UDB while they are conceptually the same, their implementation is very different.

2.1.1 SQL Server instance definition

SQL Server supports multiple “instances” or installations on the same host computer. Each instance runs independently from all others and has its own set of installation code, configuration parameters, system and user databases, memory allocation, and security configuration. When multiple instances are running simultaneously, the amount of memory can be dynamically allocated for each specific instance by an algorithm or manually specified by an administrator.

2.1.2 DB2 UDB instance definition

DB2 UDB differs from SQL Server in that only one installation of the code is needed per server or machine. DB2 UDB uses instances to provide separate environments within the same machine and like SQL Server, a number of instances can exist on a single machine while each of these environments or instances has its own configuration file, security configuration and resources (memory and disk space), although they do share the same binary code. Each instance is responsible for managing its own databases.

The resource settings for each instance, including memory, can be automatically managed by DB2 UDB or can be user-defined and stored in individual configuration files, making the way an instance uses memory highly configurable. The instance runs as a Windows service and single application process (process) called *db2syscs.exe*.

Note: In DB2 UDB, an instance is also referred to as the *Database Manager*

An example the DB2 UDB architecture is shown in Figure 2-1 and has the following components:

- ▶ One DB2 Administration Server (DAS)
- ▶ One instance with a database manager
- ▶ Two databases, database1 and database2
- ▶ The following allocations for each database:
 - Processes
 - Log buffers and log files

- Buffer pools: two for database2, IBMDEFAULTBP is default buffer pool created while database is created, and fact_table is created by user; one default buffer pool for database1.
- Table spaces:
 - Default table spaces (built when instance is created), *syscatspace*, *userspace1*, and *tempspace1* for both databases
 - Table and index table spaces for database2, *fact_table*, and *fact_index*

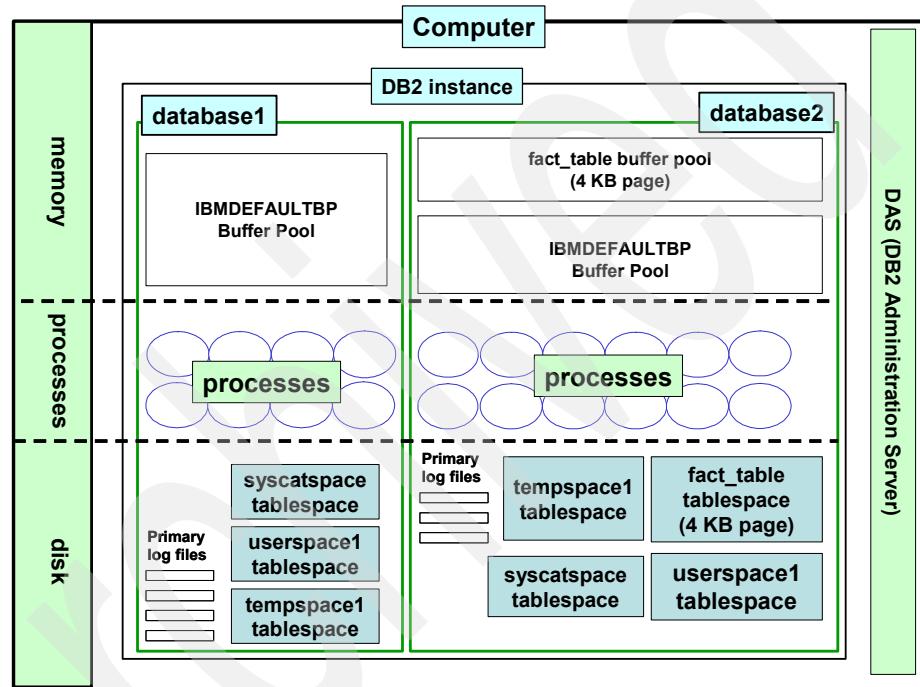


Figure 2-1 DB2 architecture showing a single instance

The DB2 UDB administration server (DAS) performs a similar function to SQL Server's SQL Agent process. The purpose of the DAS is to provide a facility for job scheduling and management through use of the Tools database allowing DB2 UDB to utilize both DB2 UDB command scripts and operating system scripts. The DAS also handles remote administration requests from clients.

2.2 Memory allocation

Memory allocation is key in achieving optimal performance of a database. It is important to understand how memory is allocated to an instance in DB2 UDB. DB2 UDB's memory model allows for a significant amount of tuning, making it highly adaptable to many environments. First, we briefly describe about how SQL Server's memory is allocated, then describe in more detail how DB2 UDB allocates memory.

2.2.1 SQL Server memory allocation

Each instance of SQL Server has its own memory address space which is divided into two areas, executable code and memory pool. DLLs and executable code used by the SQL Server Engine and Net-Libraries are loaded into the executable memory area. The memory pool is the main area of memory for SQL Server and almost all objects are loaded into the memory pool. Some of the objects contained in the memory pool are:

- ▶ System data structure
- ▶ Buffer cache
- ▶ Procedure cache
- ▶ Log cache
- ▶ Connection context

The memory pool size and division continuously change because the regions within the memory pool are constantly being adjusted to optimize performance. In addition to SQL Server adjusting memory allocation it is possible for an administrator to manually allocate memory to the instance. SQL Server is also able to utilize memory above the 2GB-4GB limit through the use of the Address Windowing Extensions (AWE) API.

2.2.2 DB UDB memory allocation

DB2 manages memory in four areas which are discussed in further detail:

- ▶ Instance shared memory
- ▶ Database shared memory
- ▶ Application shared memory
- ▶ Agent private memory

DB2 UDB differs from SQL Server in that many of the memory structures are allocated at much finer granularities, such as per instance, per database, per connection, and per agent making DB2 UDB highly configurable for optimum performance.

Figure 2-2 on page 15 shows the DB2 memory architecture.

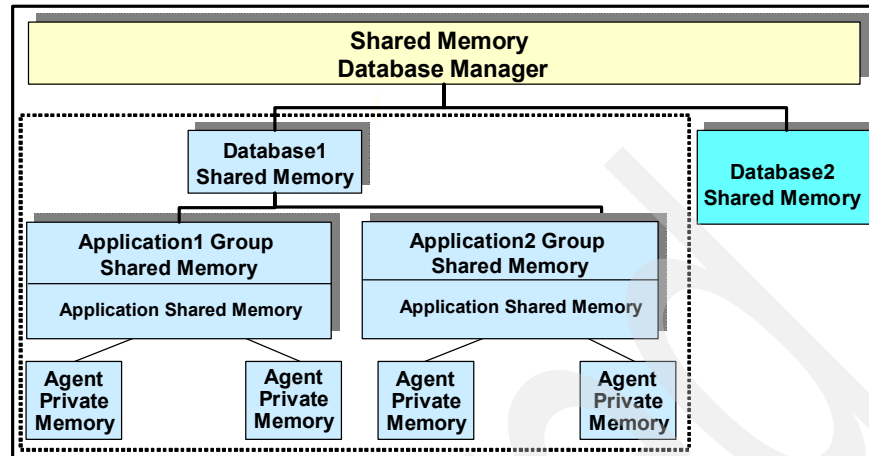


Figure 2-2 DB2 shared memory architecture

Instance shared memory

Instance shared memory, also referred to database manager shared memory, is allocated when the instance is started and freed when the instance is stopped. This area of memory is used for instance level tasks such as monitoring, auditing and inter-node communication. It is common to allow DB2 UDB to manage instance memory automatically; however, a DBA may also configure memory usage manually to suit a particular environment.

Database shared memory

Database shared memory is allocated when a database is activated or the first connection is made. IT is freed when the database is deactivated (if it was first activated) or the last connection to the database is disconnected. There is one shared memory set per database. This memory is typically used for tasks such as SQL execution, locking, backup/restore and loads.

The largest component of database global memory is the *buffer pools* and is where all index and regular data is handled. A database must have at least one buffer pool allocated, but can have a number of additional buffer pools depending on the database workload and page sizes used in the database.

It is possible for DB2 UDB to utilize memory above the 2GB-4GB limit imposed by 32-bit operating systems through the use of the AWE API.

Like the instance memory, database memory can be configured through the use of various database configuration parameters. Both the database manager and database configuration parameters can be set through Control Center or from the

command line Processor (CLP). More information about the Control Center and CLP can be found in Chapter 3, “DB2 UDB administration tools” on page 31.

Application shared memory

Application shared memory is used for data exchange between DB2 UDB and a connected application. It is allocated from the database shared memory set and is only allocated in certain environments:

- ▶ Non-partitioned databases where intra-parallel processing is used.
- ▶ Multi-partitioned databases.
- ▶ Databases where the connection concentrator is enabled.

For all other environments, application shared memory does not exist.

Agent private memory

Agent private memory is allocated to DB2 UDB agents to perform work. Each database that an application connects to has various processes or threads called *agents* which carry out tasks such as performing sorts, gathering statistics, logging, prefetching and handling communication. On Windows systems the agents run as threads within the database manager process.

More information about DB2 UDB memory management can be found in the DB2 UDB documentation *DB2 Administration Guide: Performance V8*, SC09-4821 or the online Information Center.

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

2.3 Processes

In this section we describe the differences between process implementation in SQL Server and DB2 UDB. Both SQL Server and DB2 UDB (Windows version) use a single process/multi-threaded model.

2.3.1 SQL Server process model

SQL Server runs as a single process called `sqlservr.exe`, which represents a single instance. Within the SQL Server process there are various threads running such as the read-ahead manager, used for retrieving data from disk to memory, and the backup process. There are also threads running for the transaction log writer and for user commands. The threads are managed by Open Data Services and have their CPU time scheduled by User Mode Scheduler.

SQL Server also has a process for the SQL Agent (sqlagent.exe), Distributed Transaction Coordinator (msdtc.exe), and Active Directory Helper (sqladhlp.exe).

2.3.2 DB2 UDB process model

DB2 has processes that are allocated to the database manager (instance), the database, or the requesting application connection. Typically, they are called *agents*, or engine dispatchable units (EDUs). You may remember from the section “Agent private memory” on page 16 that an EDU is one of the many agents that performs tasks on behalf of the database manager or an application.

On the Windows platform the instance runs as a single process (db2syscs.exe) and the EDUs are implemented as threads within this process. On the UNIX platform EDUs run as processes.

Important: Manipulating any DB2 UDB process is not recommended. Killing db2syscs.exe from Task Manager for example will bring the entire instance down affecting all databases within that instance.

Table 2-1 shows a list of the DB2 UDB processes running on typical Windows installation.

Table 2-1 DB2 UDB process list (Windows)

Process	Note
db2syscs	Instance process
db2dasrm.exe	Administration server (DAS) process
db2fmp.exe	Fenced mode process
db2jds.exe	JDBC server process
db2licd.exe	DB2 license service process
db2rcmd.exe	Remote command service process
db2sec.exe	Security service process
db2systray.exe	System tray tool process (similar to SQL Server service manager)

2.4 Allocating disk space

In this section, we describe the different methods SQL Server and DB2 UDB use for data storage. You will see that SQL Server and DB2 UDB have very different approaches to disk space allocation.

2.4.1 SQL Server disk allocation

SQL Server organizes its data storage into database files and filegroups. A database consists of one primary data file but can have secondary data files with the log files having their own database files. Database files can automatically grow at a specified increment until they reach a defined limit or you run out of available space on the disk is exhausted. The data files can then be grouped in *filegroups*. SQL Server uses filegroups to simplify data organization, enabling an administrator to group tables that hold certain types of data.

2.4.2 DB2 UDB disk allocation

DB2 UDB uses the concept of *tablespaces* to organize data, making it highly versatile compared to SQL Server. The physical space within a database is organized into a collection of table spaces. Each table space consists of a collection of *containers*, each of which is either a directory in the file system, a physical file, or a raw device. When a table is created, it is assigned to a table space. Data is written to table spaces in pages or group of pages called *extent*. An extent contains data for only one table and all extents are the same size in each table space. After the initial extent fills, additional extents are created on each additional container defined for in the table space in a round-robin fashion. DB2 UDB automates the striping of data across containers. Distributing data in this way enables parallel input and output and optimizes performance.

Table space types

Two types of table spaces are available: *System Managed Space* (SMS) and *Database Managed Space* (DMS). SMS table spaces are managed by the operating system and DMS table spaces are managed by DB2 UDB.

SMS table spaces are better suited for database contains many small tables. These table spaces are also easier to maintain. The space in a SMS table space is defined as a directory and is not pre allocated. Data can be added as long as there is enough physical space available. More than one directory can be assigned to the table space. With SMS table space, data, indexes, and large objects are all stored together in the same table space.

DMS table spaces are better suited for large database where performance is a key factor. Containers are added to DMS table spaces in the form of a file or raw

device. Space is pre-allocated and fixed in size. Additional containers can be added to support growth. In addition, DMS containers can be resized and extended when necessary. DMS table spaces support the separation of data, indexes, and large objects. DB2 UDB provides two types of GUI wizards to assist a DBA in table space creation: the Create Table Space wizard and the Create Table Space notebook.

SMS and DMS table spaces

The following points contrast SMS and DMS table spaces:

- ▶ Physical Implementation:
 - A SMS table space is created using directory containers.
 - A DMS table space is created using either file containers or device containers.
- ▶ Managed by:
 - SMS is system managed space (managed by the operating system).
 - DMS is database managed space (managed by DB2 UDB).

Advantages of a SMS table space

A SMS table space has the following advantages:

- ▶ Space is not allocated by the system until it is required.
- ▶ Creating a table space requires less initial work, because you do not have to redefine the containers.
- ▶ Large objects (LOBs) may benefit from operating system caching.

Advantages of a DMS table space

A DMS table space has the following advantages:

- ▶ The size of a table space can be increased by adding or extending containers, using the ALTER TABLESPACE statement. Existing data can be automatically rebalanced across the new set of containers to retain optimal I/O efficiency.
- ▶ Table data, indexes, and large objects can be split across multiple table spaces, based on the type of data being stored.

2.5 Transaction logs

Transaction logging plays an important part in the recoverability and performance of a database. Transaction logs are a record of all modifications that have occurred in the database.

2.5.1 SQL Server transaction log overview

SQL Server uses the transaction log of each database to enable database recovery. The transaction log records the start of each transaction, the changes to the data, and should the need arise, enough information to undo any modifications made within each transaction.

SQL Server uses three recovery models for its transaction logging.

- ▶ Simple
 - The simplest to manage.
 - Only full database backups are allowed.
 - Cannot back up just the changes made since the last full backup.
 - Transaction logs will not become full as the log space is reclaimed when transactions are committed.
 - Non-logged operations are allowed.
- ▶ Full
 - Can create complete backups of the database
 - Incremental backups of only the changes that have occurred since the last full backup are possible.
 - Allows point in time recovery.
 - Space in the transaction log is only reclaimed when a backup of the transaction log is made.
 - Non-logged operations are not allowed.
- ▶ Bulk-Logged
 - Lies between the other two models.
 - Incremental backups of the database are possible.
 - Bulk copy operations are only minimally logged.
 - If a bulk copy operation occurs, point-in-time recovery is not possible.

SQL Server log files are logically divided into “virtual” log files which are the unit of truncation for the transaction log.

2.5.2 DB2 UDB transaction log overview

DB2 UDB has two types of logging, *circular logging* and *archive logging*.

- ▶ Circular
 - Similar to SQL Server’s Simple recovery model.

- Default behavior when a new database is created.
 - Only full, offline backups of the database are allowed.
 - Does not allow a database to be rolled forward through the transaction log.
 - Uses a “ring” of three or more log files to provide recovery from system crashes or transaction failures.
 - If a failure occurs and transaction cannot be recovered, all changes since the last full backup are lost.
- Archive
- Similar to SQL Server’s Full recovery model.
 - Enables the ability to take online backups. Users can continue to work.
 - Permits roll forward recovery to a point in time or to the end of the log.
 - Archived logs can be used to recover changes made after the last backup was taken.

Both circular and archive logging allow non-logged operations to be carried out. Unlike SQL Server, DB2 UDB is able to take incremental and delta backups for both logging types. An incremental backup copies all database data that has changed since the most recent full backup and a delta backup copies all database data that has changed since the last backup (full, incremental, or delta).

There are a number of database configuration parameters that can be used to configure log file activity. For example the number, size and location of the logs can be configured. A new feature in version 8.2 allows for log files to be contained in a backup image, eliminating the possibility of losing log files required for recovery.

More information about the logging and recovery options can be found in the DB2 UDB documentation *Data Recovery and High Availability Guide*, SC09-4831.

2.6 Query optimization

The optimizer is the part of a database engine that is responsible for creating query access plans. An access plan describes how to obtain the data from disk, what indexes can be used, and what join methods can be employed. If the optimizer produces a suboptimal plan, a client might experience relatively poor query performance.

The optimization process is one of the last elements in a chain of events preceding SQL statement execution. Both DB2 UDB and SQL Server use a cost based optimizer that determines the cost of a plan based on stored information including database metadata and statistical parameters.

2.6.1 SQL Server query optimization

SQL Server's cost-based optimizer automatically determines the best access plan based on several steps. The steps the optimizer goes through are:

- ▶ Trivial plan optimization
- ▶ Syntax check
- ▶ Examine table statistics
- ▶ Cost-based optimization

The first step in optimization is called "Trivial Plan Optimization". The optimizer tries to find the most inexpensive access plan based on the SQL statement being executed. If a simple plan is not found, it then looks at the syntax of the statement being executed to see if any corrections are necessary. If the optimizer thinks that the best access plan still has not been found, it examines statistical information it has on the indexes and starts its cost-based optimization process. SQL Server's table and index statistics are created and maintained with the `sp_updatestats` system stored procedure and the `CREATE STATISTICS` T-SQL command.

2.6.2 DB2 UDB query optimization

DB2 UDB's cost-based optimizer uses statistical information stored in the database system catalog to determine the best access plan. The optimizer first checks the statement for operational effectiveness (joins, predicate application, aggregation) then evaluates whether an index scan or table scan should be used.

DB2 UDB also has a query rewrite feature that will rewrite the original query to obtain the optimal access plan. Catalog statistics are updated using the `runstats` utility. If an application uses static SQL, access plans are generated at bind time using the statistics in the catalogs and are stored as executable code in the database. If an application uses dynamic SQL, the access plan is generated at runtime using current statistics and stored in a dynamic cache. Dynamic caching allows for quick reuse of optimized code.

DB2 UDB's cost-based optimizer bases its access plan on such information as:

- ▶ Query optimization class

- DB2 has several optimization classes or levels (0 to 9) that can be specified during compilation of an SQL statement. These determine how aggressive the optimizer is in choosing the best access plan.
- ▶ Configuration parameters
 - Database configuration parameters.
- ▶ Bind options
 - The optimization class can be set during a bind.
- ▶ Statistics in system catalog tables
 - Catalog statistics are collected and updated by running the **runstats** utility.

Two new features in DB2 UDB V8.2 are *automatic statistics profiling* and *automatic table statistics collection*. Automatic statistics profiling allows DB2 UDB to determine which statistics need to be updated. DB2 UDB will then automatically run the **runstats** utility in the background to make sure statistics are kept up-to-date.

Query optimization is discussed further in 12.7, “Optimizer” on page 381.

2.7 Parallelism

Both SQL Server and DB2 UDB support parallelism. Parallelism is the ability to execute an SQL statement, perform I/O, or run certain utilities such as backup, restore or load across multiple processors.

2.7.1 SQL Server parallelism

SQL Server supports query parallelism. It automatically evaluates the best degree of parallelism to use for each parallel query execution by considering the following information:

- ▶ Number of processors that are running on the system.
- ▶ Amount of memory available to execute parallel queries.
- ▶ Number of concurrent connected users.
- ▶ Type of query being executed.

SQL Server then decides if parallel execution of the query is warranted based on the server workload and configuration. It can also utilize parallel operations for backup/restore and parallel load (bulk copy program).

2.7.2 DB2 UDB parallelism

DB2 UDB supports two types of parallelism, *intra-partition parallelism* and *Inter-partition parallelism* and within in this DB2 can carry out parallel operations for the following:

- ▶ Query Parallelism. There are two types, inter-query parallelism and intra-query parallelism.
 - Inter-Query parallelism, the ability for a database to accept queries from multiple applications at the same time.
 - Intra-Query parallelism, the ability to process multiple parts of a query at the same time.
- ▶ Input/Output (I/O)
 - Parallel I/O to one or more I/O devices.
- ▶ Utilities
 - Backup, restore, load, and index creation can take advantage of intra-parallelism.

2.8 Message logs

This section provides an overview of the error logging differences between SQL Server and DB2 UDB, highlighting the versatility of DB2 UDB's logging capabilities.

2.8.1 SQL Server error logs

SQL server has a set of six error logs per instance. They are typically found in the Microsoft SQL Server\MSSQL\LOG directory or \MSSQL\$instance name directory if e more than one instance is installed. Note that only one log is active at any one time. The error logs can be gathered with the **sqldiag** utility. Information is also recorded in the Windows application event log.

2.8.2 DB2 error logs

DB2 uses a single error log per instance called the *db2diag.log* to record informational and error messages. The amount of information recorded in the *db2diag.log* can be configured with the database manager configuration parameter **DIAGLEVEL**, giving the administrator the ability to increase the diagnostic information gathered to aid troubleshooting. The log format for DB2 version 8.2 has been improved making it easier to read. Figure 2-3 is an example of the *db2diag.log*.

```

2005-01-20-18.40.22.921000-480 E24073165H399      LEVEL: Error (OS)
PID      : 3224                      TID   : 3308      PROC  : db2dasstm.exe
INSTANCE: DB2                      NODE   : 000
FUNCTION: DB2 UDB, oper system services, sqlOSSemClose, probe:20
CALLED   : OS, -, unspecified_system_function      OSERR: 6
RETCODE  : ECF=0x9000000C=-1879048180=ECF_INVALID_PARAMETER
          Invalid parameter

2005-01-20-18.40.27.140000-480 E24073566H790      LEVEL: Event
PID      : 2592                      TID   : 2588      PROC  : db2syscs.exe
INSTANCE: DB2                      NODE   : 000
FUNCTION: DB2 UDB, base sys utilities, DB2StartMain, probe:911
MESSAGE  : ADM7513W Database manager has started.
START    : DB2 DBM
DATA #1 : Build Level, 124 bytes
Instance "DB2" uses "32" bits and DB2 code release "SQL08020"
with level identifier "03010106".
Informational tokens are "DB2 v8.1.7.664", "s040914", "WR21342", FixPak "7".
DATA #2 : System Info, 1304 bytes
System: WIN32_NT WISLA Service Pack 4 5.0 x86 Family 15, model 2, stepping 9
CPU: total:2 online:2
Physical Memory: total:2039 free:1591 available:1591
Virtual Memory: total:5387 free:4945
Swap      Memory: total:3348 free:33542005-01-20-18.40.22.921000-480

```

Figure 2-3 db2diag.log example

In addition to the db2diag.log there is an error log for the administration server (DAS) called the *db2dasdiag.log* for recording information in relation to operation of the DAS.

Both of these log files, as well other logs, configuration information, and environment information can be gathered with the **db2support** command. This is similar to SQL Server's **sqldiag** command but gathers much more information and negates the need to gather information for troubleshooting manually. The db2support command and other problem determination tools are discussed in Chapter 13, "Testing and troubleshooting" on page 397.

DB2 UDB also has an additional log used for problem determination. The *notify log*, also referred to as the *Administration Notification log* is used to record significant events that occur. On UNIX and LINUX platforms, the administration notification log is a text file called *instance_name.nfy*, typically located in the \$HOME/sql11ib/db2dump directory (where \$HOME is the home directory of the instance owner). On Windows, all administration notification messages are written to the Windows Event Logs.

2.9 Security

In this section we describe how security is implemented in SQL Server and DB2 UDB and highlight some of the enhancements made with the introduction of DB2 UDB V8.2.

2.9.1 SQL Server security

SQL Server uses two methods of user authentication. The first and most commonly used is *Windows authentication*. This allows a user to connect to SQL Server using a domain or local windows account. The second is *mixed-mode* authentication. This is a combination of Windows authentication and SQL Server authentication and is used to provide backward compatibility for applications that require SQL Server logins or if SQL Server is running on an operating system does not support Windows authentication (e.g., Windows95/98).

SQL Server uses “*Roles* to aid in controlling permissions on the database and database objects. Roles are equivalent to groups that contain users. They enable permissions to be granted more easily.

2.9.2 DB2 UDB Security

DB2 UDB controls database access using three levels of security, *authentication*, *authorities* and *privileges*. This section aims to give an understanding of how security is handled in DB2 UDB.

Just as SQL Server uses roles to assign permissions to a group of users, authorities and privileges can be granted to individual users or a group of users in DB2 UDB.

Authentication

The first step in accessing a DB2 UDB database is authentication. Before a user can access an instance or database they must be authenticated. This is normally handled by the operating system or another third party security product. DB2 UDB uses parameters in the database manager configuration file to determine the type of authentication to be used and where users are authenticated.

Authentication types that can be configured for each *instance* include:

- ▶ CLIENT
 - Specifies that authentication occurs on the database partition where the application is invoked using operating system security.
- ▶ SERVER
 - Specifies that authentication occurs on the server using local operating system security.

- ▶ **SERVER_ENCRYPT**
 - Specifies that the server accepts encrypted SERVER authentication schemes. If client authentication is not specified, the client is authenticated using the method selected at the server.
- ▶ **KERBEROS**

Used when both the DB2 UDB client and server are on operating systems that support the Kerberos security protocol.
- ▶ **KRB_SERVER_ENCRYPT**

Specifies that the server accepts either KERBEROS authentication or encrypted SERVER authentication schemes.
- ▶ **DATA_ENCRYPT**

The server will accept encrypted SERVER authentication schemes and the encryption of user data.
- ▶ **DATA_ENCRYPT_CMP**

The server will accept encrypted SERVER authentication schemes and the encryption of user data.
- ▶ **GSSPLUGIN**

Specifies that the server uses a GSS-API plug-in to perform authentication.
- ▶ **GSS_SERVER_ENCRYPT**

Specifies that the server accepts plug-in authentication or encrypted server authentication schemes.

Authentication in DB2 UDB can be likened to Windows authentication or mixed mode in SQL Server.

Authorities

Once a user has been authenticated the next step is for DB2 UDB to evaluate the authorities and privileges assigned to a user. Authorities are the right assigned to a user to perform specific administrative, maintenance or utility operations against databases within an instance. They can be compared to server and database roles in SQL Server although their implementation is different. Authorities are divided into the following groups:

- ▶ **SYSADM**

This is the highest authority and gives the user full control of the instance. It also has the authority to grant or revoke DBADM authorities and has control over database manager resources.

- ▶ **SYSCTRL**

Applies to operations affecting system resources and includes privileges to stop an instance or a database, to create, update or drop a database, and to create or drop a table space.

- ▶ **SYSMAINT**

Enables a user to perform maintenance operations on all databases in an instance. This authority gives the privilege to back up a database or a table space, update database configuration files and restore an existing database.

The above three authority levels are configurable. For each DB2 UDB instance. These authorities can only be assigned to groups. The names of the groups that are assigned these authorities are stored in the database manager configuration file associated with each instance.

DB2 UDB has further database level authorities and privileges that can be granted at a database level such as:

- ▶ **DBADM**
- ▶ **CREATEDB**
- ▶ **BINDADD**
- ▶ **CONNECT**
- ▶ **LOAD**
- ▶ **QUIESCECONNECT**

The above database-level privileges and authorities are assigned using SQL GRANT and REVOKE commands.

Privileges

Privileges convey the right of a user to perform specific operations on database objects. For example a user may be able to issue an INSERT, UPDATE, or DELETE against table but not alter the table definition.

The types of privileges that can be granted/revoked are:

- ▶ **select**
- ▶ **insert**
- ▶ **update**
- ▶ **delete**
- ▶ **control**
- ▶ **alter**
- ▶ **indexes**
- ▶ **references**

Privileges in DB2 UDB are equivalent to database object permissions in SQL Server.

2.9.3 Security enhancements in V8.2

DB2 UDB Version 8.2 provides many new security feature and function enhancements including:

Improved Windows Domain and Active Directory support

Active Directory and Domain support has been enhanced:

- ▶ Support using cached credentials for DB2 UDB authentication and group lookup.
- ▶ Support for nested group semantics
- ▶ Support for Domain Local Groups.
- ▶ Support for implicit trusts between Windows Domains.

User ID and group name enhancements

There are several enhancements to user and group naming:

- ▶ Support for special characters (!%&(){}-.^~ and space).
- ▶ Support for user and group names containing up to 30 characters
- ▶ Two part names can be used in a database connect statement or an instance to attachment. For example, specifying DOMAIN\username.

Running DB2 UDB with the Local System Account

In DB2 UDB Version 8.2, the various DB2 UDB services can now run under the Windows Local System account (LSA).

Protection for DB2 UDB system files

An installation of DB2 UDB on Windows creates a various objects such as files, directories, registry keys. During the installation of DB2 UDB, there is an option to create two groups, DB2ADMNS and DB2USERS. These groups can be used to restrict access to the critical files that DB2 UDB needs for operation. This file lock down operation can also be done post installation using the **db2secv82** command.

More information about authentication, authorities and privileges can be found in the DB2 UDB documentation:

- ▶ *SQL Reference Volume 1*, SC09-4844.
- ▶ *Administration Guide: Implementation*, SC09-4820.

2.10 High availability

High availability of systems can be achieved with correct planning and implementation. Both SQL Server and DB2 UDB support high availability strategies through the use of failover clustering, log shipping, replication and mirroring.

2.10.1 SQL Server high availability strategies

SQL Server supports various high availability strategies. Failover clustering with Microsoft Cluster Services, Log shipping (SQL Server Enterprise Edition only), Transaction Replication and Remote Mirroring through the use of third party products.

2.10.2 DB2 high availability strategies

DB2 UDB supports a number of high availability strategies, including Microsoft Cluster Services, a new feature in DB2 UDB V8.2 called High Availability Disaster Recovery (HADR), Log Shipping, Transaction Log Mirroring, and Replication and Mirroring using third party tools.

DB2UDB High Availability Disaster Recovery (HADR) is a data replication feature that provides a real-time high-availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the *standby*. A partial site failure can be caused by a hardware, network, or software (DB2 UDB or operating system) failure. Without HADR, the database management system (DBMS) server or the machine where the database resides has to be rebooted.

DB2 UDB mirroring cannot be done at the table space level; it must be performed outside of the database at the hardware level. DB2 UDB does support mirroring of the transaction logs. It also supports the increased availability of data by allowing mirrored copy of data to be split and made available for other types of processing.

DB2 UDB administration tools

As you make the transition from SQL Server to DB2 UDB, it is important to understand how to administer the database using DB2 UDB's administration tools. In this chapter, we provide an overview of the administrative tools, Wizards, and Advisors, as well as many other tools and utilities available.

The topics covered include:


- ▶ GUI tools
- ▶ Wizards
- ▶ Advisors
- ▶ Command line tool
- ▶ Utilities
- ▶ Windows Performance Monitor integration
- ▶ Optional tools

Note: The examples shown in this chapter are based on the SAMPLE database provided with DB2 UDB.

3.1 GUI tools

DB2 UDB ships with a number of Java based GUI tools and assistants available to ease the DBA's job. We highlight some of these tools and, where applicable, compare them to the equivalent tools in SQL Server. All the tools covered in this section are also accessible from the Control Center.

3.1.1 Control Center

Control Center is the main administration GUI tool for DB2 UDB and contains capabilities that extend beyond that of SQL Server's Enterprise Manager. You can launch the Control Center from **Start Menu → Programs → IBM DB2 → General Administration Tools → Control Center** or from a command line by typing **db2cc**. The Control Center can also be launched from any of the DB2 UDB GUI tools **Tools** menu or by clicking the Control Center icon . The Control Center is a Java application, which means that it can run from most platforms that has a JDK. You can use the Control Center to manage and administer systems, instances, databases, and database objects such as tables and views. In addition, there are a number of wizards and advisors designed to guide you through common administration tasks. We take a closer look at these Wizards in 3.2, "Wizards" on page 49 and 3.3, "Advisors" on page 50.

The following are some of the key tasks that you can be performed using the Control Center:

- ▶ Add DB2 UDB systems, federated systems such as DB2 UDB for z/OS and OS/390 systems, IMSysplexes, instances, databases, and database objects to the object tree.
- ▶ Manage database objects. Create, change, and drop databases, table spaces, tables, views, indexes, triggers, and schemas.
- ▶ Manage data. Load, import, export, and reorganize data. You can also gather statistics and run queries.
- ▶ Perform maintenance operations such as backups and restores.
- ▶ Configure and tune instances and databases.
- ▶ Manage database connections, such as DB2 Connect servers and subsystems.
- ▶ Manage DB2 UDB for z/OS and OS/390 subsystems.
- ▶ Manage applications.

Figure 3-1 shows the Control Center. At the top of the window, under the main menu, you will see the **Tools** option. This option provides you with access to a

variety of tools, many of which are described here. You can also access the tools from the icons below the drop down menus.

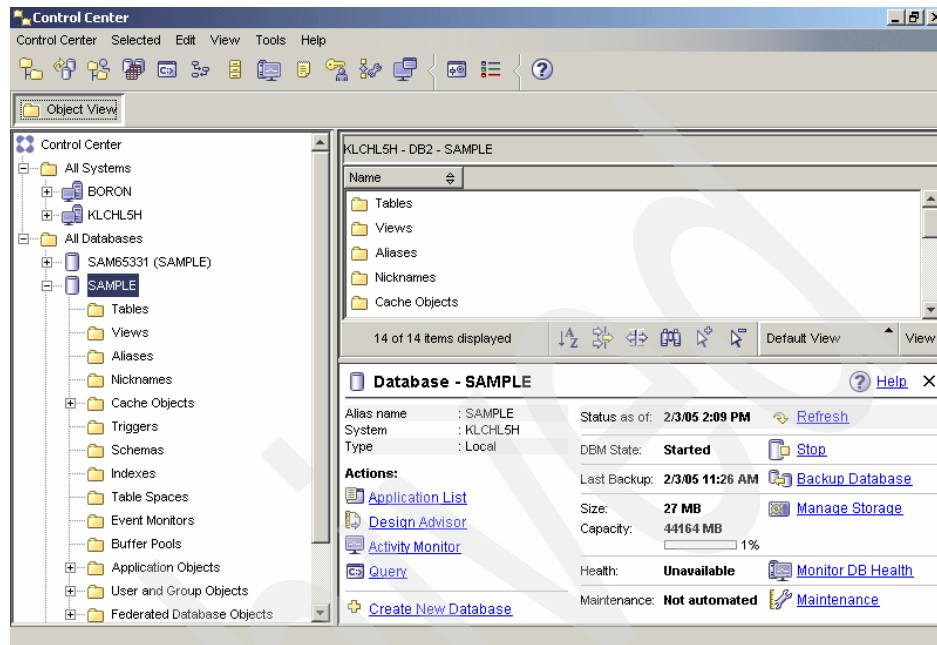


Figure 3-1 Control Center

Within Control Center you have the ability to set the configuration options at both the instance and database level. New in DB2 UDB V8.2 is the ability to set configuration parameters online. Changes to these online configuration parameters take effect immediately without the need to stop and re-start the instance, or deactivate and activate the database. Some SQL Server configuration options can be managed using Enterprise Manager or the `sp_configure` system stored procedure; however, some configuration options require you to stop and restart the server before the new values take effect.

3.1.2 Command Editor

Command Editor is similar to Query Analyzer found in SQL Server. Command Editor is used to generate, edit, execute, and manipulate SQL statements and DB2 commands. It can also be used to work with the resulting output at SQL statements, and to view a graphical representation of the access plan for explained SQL statements.

You can launch the Command Editor from **Start Menu → Programs → IBM DB2 → Command Line Tools → Command Editor** or from a command line by


typing **db2cmdctr**. You can also launch the Command Editor from any of the DB2 UDB GUI tools. Tools menu or by clicking the Command Editor icon . Command Editor can be opened from within Control Center (embedded) or as a standalone application. Both modes offer the same set of functionality and enable you to open multiple Command Editors session.

Figure 3-2 shows a Command Editor window. There are three “tabs”, *Command*, *Query Results*, and *Access Plan*. The Command tab is used for entering DB2 UDB commands and SQL statements. The Query Results tab is where the results of any queries are displayed. The Access Plan tab show a graphical representation of the access plan used for an SQL statement.

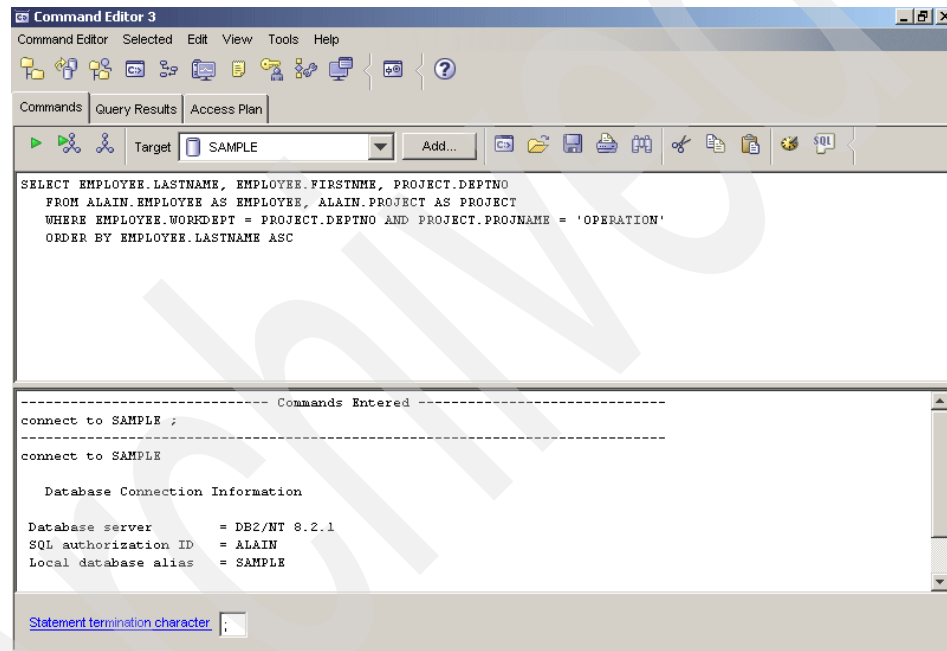



Figure 3-2 Command Editor

3.1.3 SQL Assist

The *SQL Assist* wizard can be used to help format SQL queries (i.e., SELECT statements). SQL Assist can be launched by clicking the SQL Assist icon  in the Command Editor and can also be launched from the Development Center. With the SQL Assist wizard, you can create SQL SELECT statements, including complex statements with JOINS. Figure 3-3 shows how the SQL Assist can be used to create a query that joins the employee and project tables.

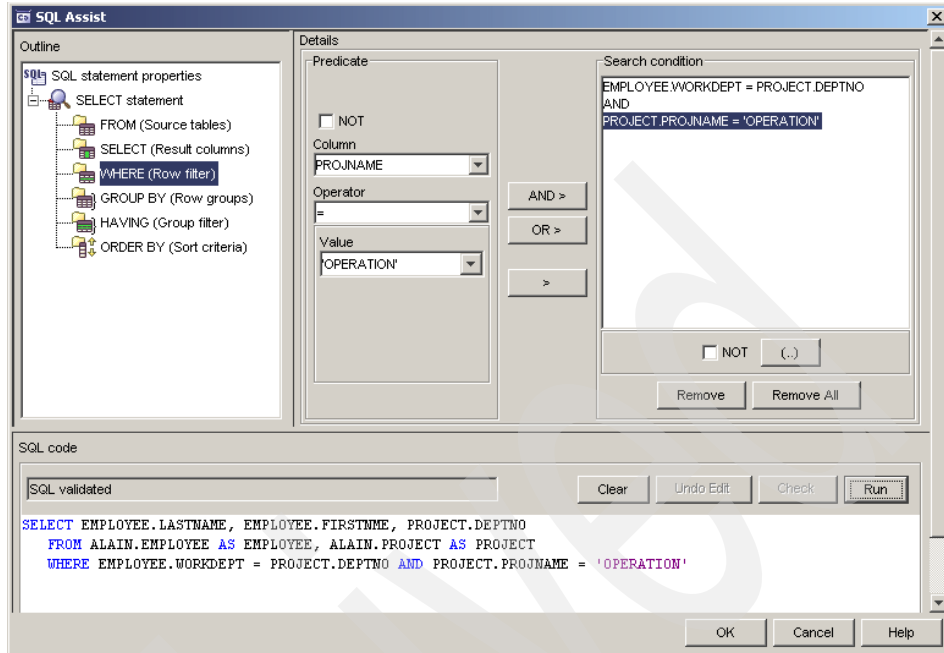


Figure 3-3 Sample SQL Assist output

3.1.4 Visual Explain

Visual Explain displays the access plan for explained SQL statements. It is similar to the Execution Plan view in SQL Server's Query Analyzer. You can use the information available from the graph to tune your SQL queries for better performance. You can access Visual Explain from within Command Center and in Control Center by right-clicking on a database and selecting **Explain SQL**.

An access plan graph shows:

- ▶ Tables (and their associated columns) and indexes
- ▶ Operators (such as table scans, sorts, and joins)
- ▶ Table spaces and functions
- ▶ Total estimated cost and number of rows retrieved (cardinality)

In addition, Visual Explain displays the statistics that were used at the time of optimization. You can then compare these statistics to the current catalog statistics to help you determine whether rebinding the package might improve performance. You can determine whether or not an index was used to access a table. If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed. Visual Explain is particularly useful for displaying the effects of performance tuning.

Figure 3-4 shows the explain output for the SELECT statement shown in Command Center (Figure 3-2.)

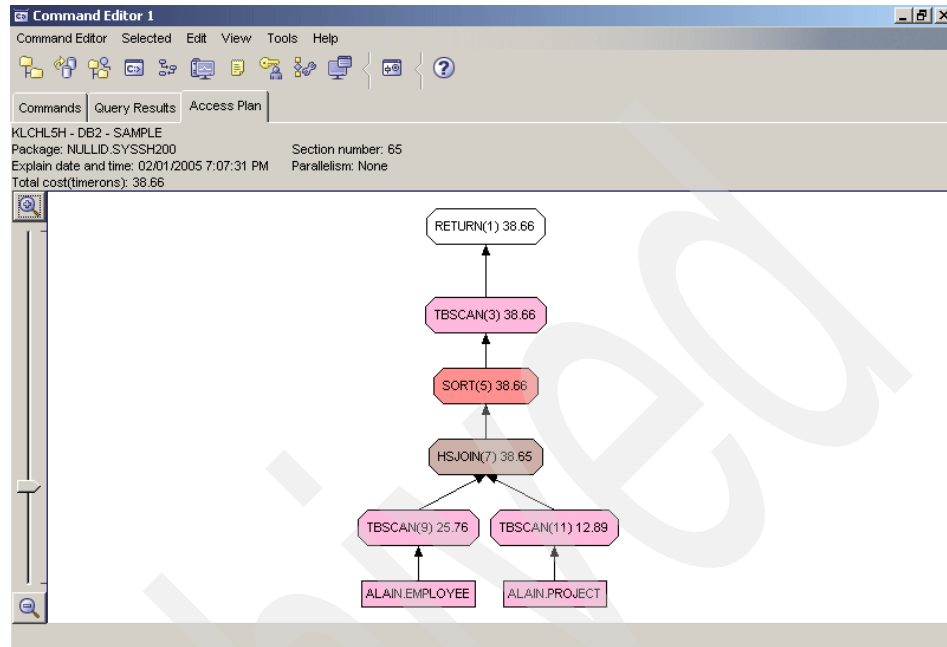



Figure 3-4 Visual Explain output

3.1.5 Task Center

Task Center is similar to the SQL Server's Agent. It is used for running and scheduling tasks such as DB2 UDB and operating system scripts. The Task Center can be launched from **Start Menu** → **Programs** → **IBM DB2** → **General Administration Tools** → **Task Center** or from a command line by typing **db2tc**. You can also launch the Task Center from any of the DB2 UDB GUI tools from the Tools menu or by selecting the Task Center icon .

Task schedules are managed by a scheduler, included with DB2, while the tasks are run on one or more systems, called run systems. You define the conditions for a task to fail or succeed with a success code set. Based on the success or failure of a task, or group of tasks, you can run additional tasks, disable scheduled tasks, and other actions. You can also define notifications to send after a task completes. You can send an e-mail notification to people in the contacts list, or you can send a notification to the Journal. The Task Center can be launched either from the Control Center **Tools** menu or by clicking the Task Center icon. You can save tasks to Task Center from many of the other GUID administration Wizards. For example, a backup job created using the backup

wizard can be saved to Task Center for execution at a later date. Figure 3-5 shows Backup and Reorg jobs created and scheduled in Task Center.

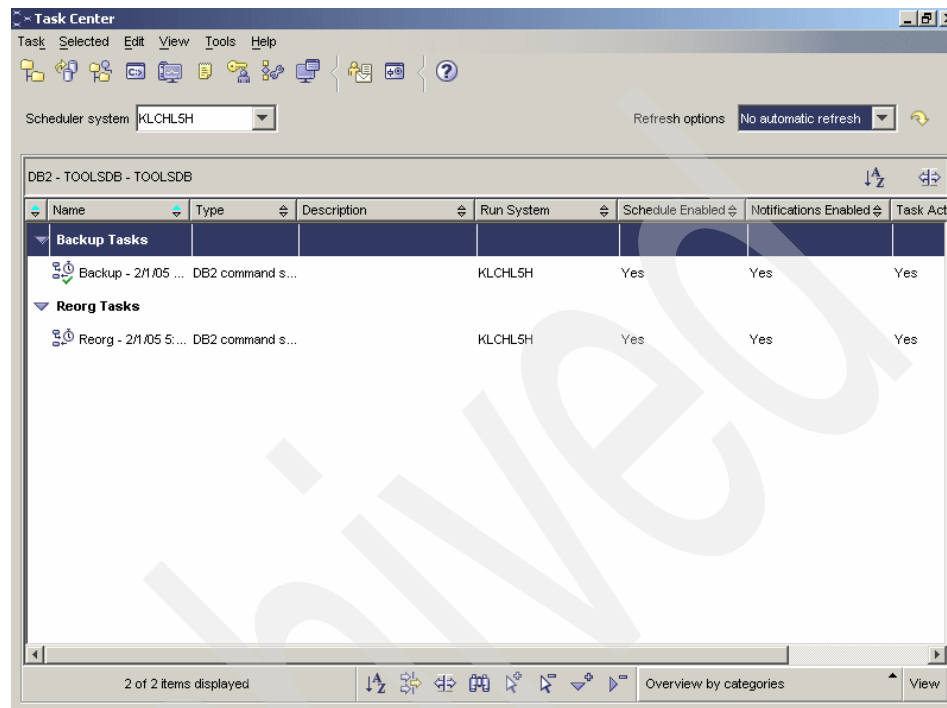



Figure 3-5 Task Center

3.1.6 Journal

The *Journal* is similar to the “alerts” functionality built into the SQL Server Agent and is used to view historical information about tasks, actions and operations carried out against a database as well as alerts and messages. You can launch the Journal from **Start Menu → Programs → IBM DB2 → General Administration Tools → Journal** or from a command line by typing `db2journal`. You can also launch the Journal from any of the DB2 UDB GUI tools’ Tools menu or by clicking the Journal icon .

The Journal allows you to view the details and results of any task that has been executed. Figure 3-6 shows an example of Backup and Reorg tasks that completed. The Journal can also display information relating to database history, messages and information from the notification log.

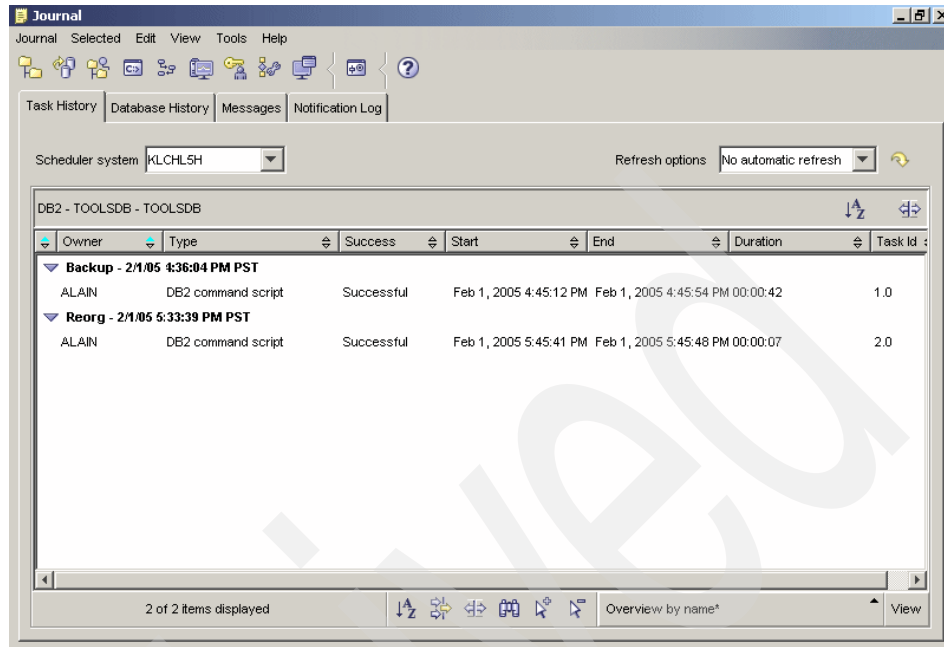



Figure 3-6 Journal

3.1.7 Health Center

The *Health Center* is a powerful tool and adds the capability to monitor the health of an instance, active databases, table spaces, and table space containers, without user interaction. The Health Monitor continuously monitors a set of indicators for problems, proactively detecting issues that might lead to unacceptable system performance or failure. When one of the configurable threshold settings is approached or has been exceeded, a warning or alarm is issued. Like Task Center, it can send notification of warnings or alarms to administrators. You can launch the Health Center from **Start Menu → Programs → IBM DB2 → Monitoring Tools → Health Center** or from a command line by typing **db2hc**. You can also launch the Health Center from any of the DB2 UDB GUI tools from the Tools menu or by clicking the Health Center icon .

The Health Center can be used to:

- ▶ View alerts associated with an object (instance, database or database object).
- ▶ Find out the environmental status.
- ▶ Find out detailed information about a particular alert.
- ▶ View recommendations to help correct alerts.

- ▶ Configure health monitor settings.
- ▶ Select the contacts who will receive alerts.

Figure 3-7 shows Health Center with an attention notification against the USERSPACE1 table space.

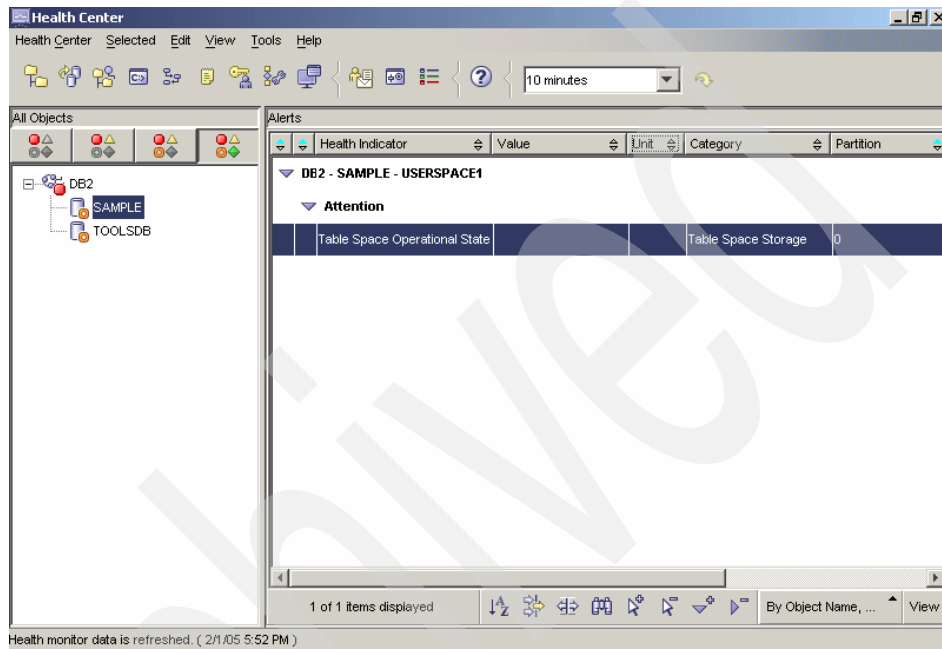



Figure 3-7 Health Center

3.1.8 Development Center

Development Center is an easy to use tool that can be used to develop and deploy new Java or SQL stored procedures and user defined functions (UDFs). The easy to use wizards allow you to create, view, and modify stored procedures and UDFs on local and remote DB2 UDB servers. You can also test and debug stored procedures or UDFs that are already installed.

You can launch the Development Center from **Start Menu → Programs → IBM DB2 → Development Tools → Development Center** or from a command line by typing **db2dc**. You can also launch the Development Center from any of the DB2 UDB GUI tools from the Tools menu or by selecting the Development Center icon . Development Center (Figure 3-8) can also be launched from Microsoft Visual Studio.NET and IBM WebSphere Studio Application Developer. Support for the .NET framework has been extended to include support of CLR procedures developed with C# and VisualBasic.NET

More information about application development for DB2 UDB can be found in the DB2 UDB documentation *Application Development Guides: Building and Running Applications V8*, SC09-4825.

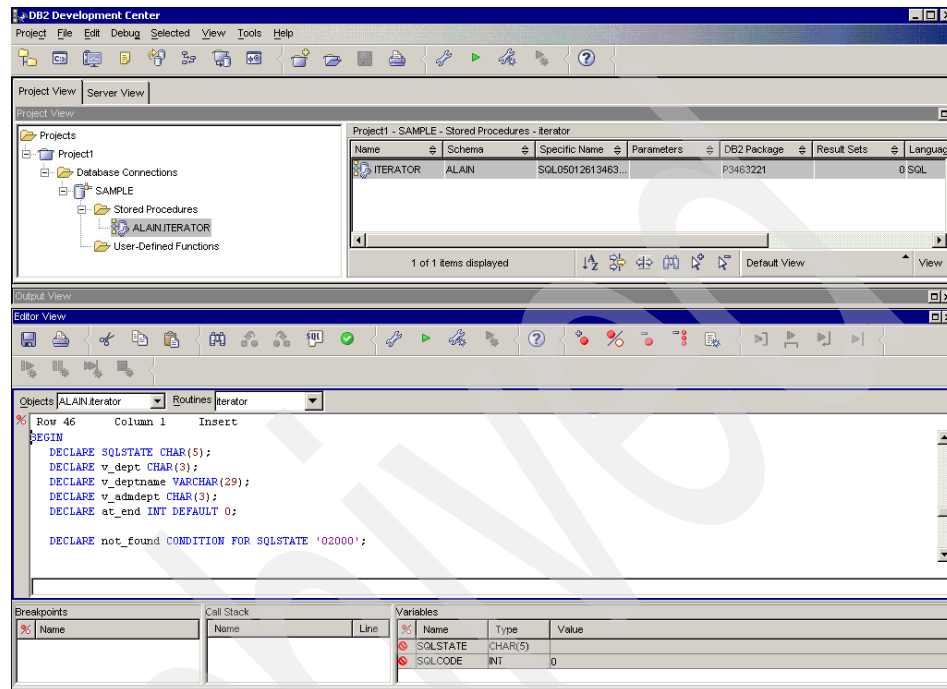



Figure 3-8 Development Center

3.1.9 Configuration Assistant

Configuration Assistant (Figure 3-9) is used to configure database connections. SQL Server uses the ODBC manager found in the Windows Control Panel to configure client to server database connectivity. DB2 UDB requires that you configure access to each database from your DB2 client before you can work with it or any database objects. From the Configuration Assistant, you can work with existing database objects, add new ones, bind application packages, set database manager configuration parameters and import and export configuration information. Notice that all of the GUI tools can be launched from Configuration Assistant. You can launch the Configuration Assistant from **Start Menu** → **Programs** → **IBM DB2** → **Setup Tools** → **Configuration Assistant** or from a command line by typing **db2ca**. You can also launch the Configuration Assistant from any of the DB2 UDB GUI tools from the **Tools** menu or by clicking the Configuration Assistant icon .

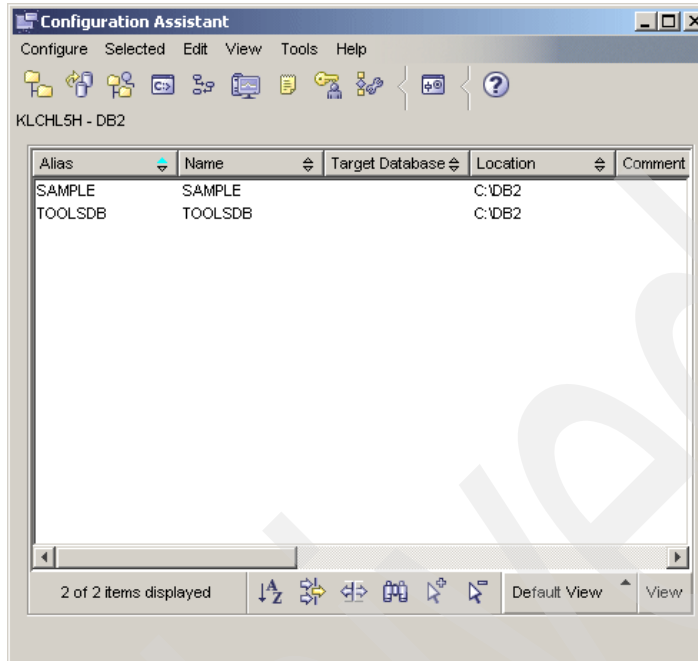


Figure 3-9 Configuration Assistant

3.1.10 Information Center

Information Center, shown in Figure 3-10, provides access to the DB2 UDB product documentation. Information Center is similar to SQL Server's Books Online, but information it contains is more in depth information. Topics covered in Information Center range from installation and configuration, administration, to tuning and troubleshooting.

The Information Center can either be installed either locally or can be accessed on the Internet. Information Center can be launched from **Start Menu** → **Programs** → **IBM DB2** → **Information** → **Information Center** or by selecting the Information Center icon (?) in any of the GUI **tools**. You can launch the online Information Center from a Web browser using the following URL:

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

If the Information Center is installed locally, you can type **db2ic** at a command line to start the Information Center. You also can access Information Center from the **Help** button in any of the GUI tools, this will display help on the particular task you are carrying out.

DB2 UDB Information Center will be a valuable tool in the transition from SQL Server to DB2 UDB.

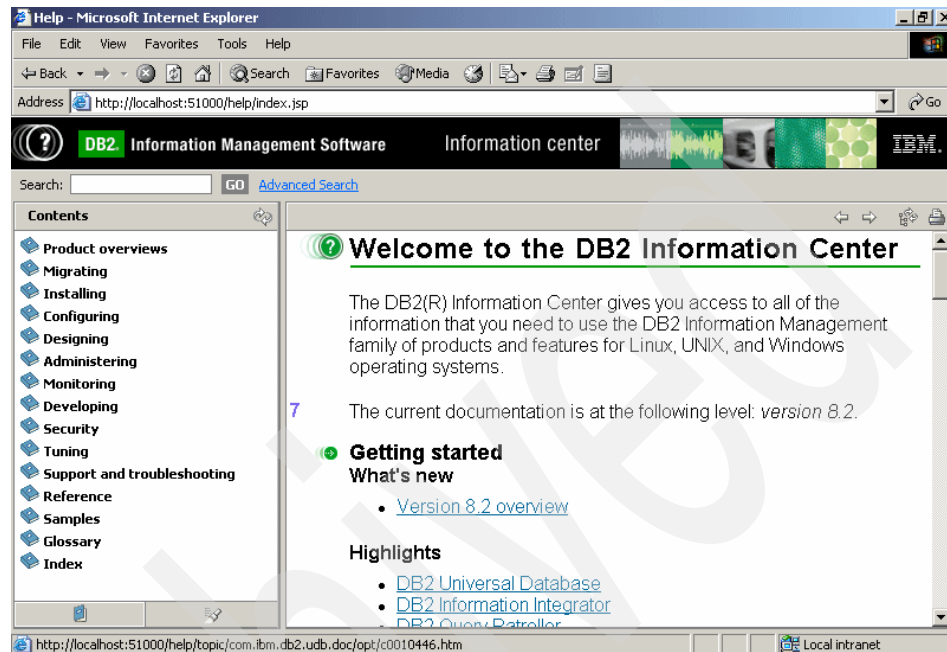



Figure 3-10 Information Center

3.1.11 License Center

License Center is used for the management of DB2 UDB product licenses. From the License Center, you can check the license information, statistics, registered users, and current users for each of your installed products. The equivalent SQL Server tool is the “SQL Server Licensing Setup” applet found in the Windows Control Panel. The License Center can be accessed from the Tools menu in any of the GUI tools or by clicking the License Center icon . The **db2licm** command line tool also performs basic license operations. Using this command, you are able to add, remove, list, and modify licenses and policies installed on your local system. Figure 3-11 shows the License Center.

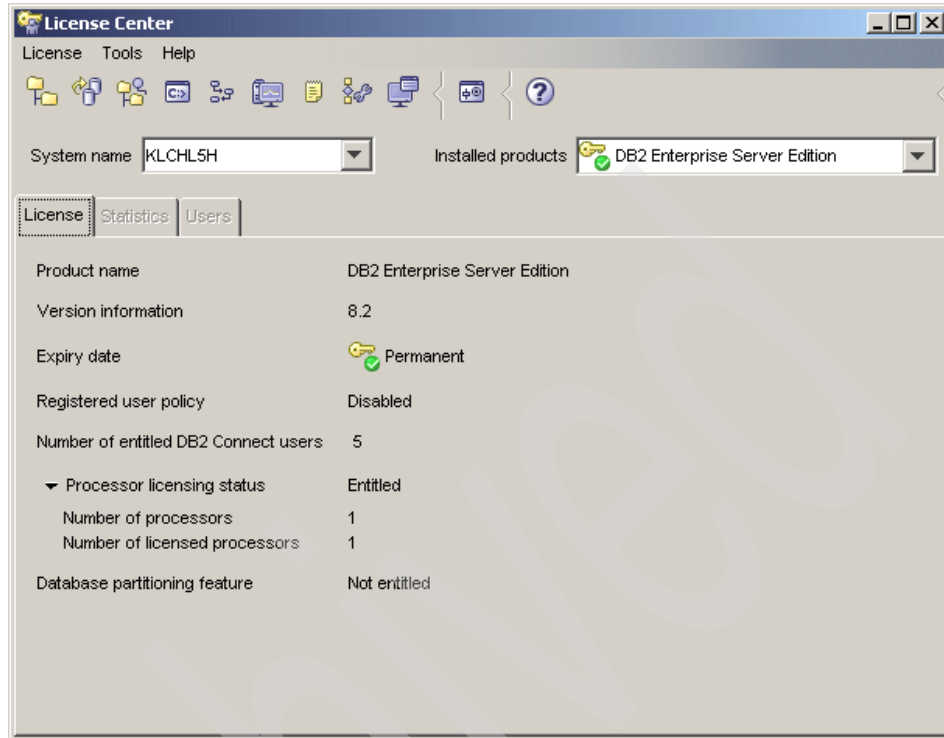


Figure 3-11 License Center

3.1.12 Replication Center


Replication Center is a tool that you can use to set up and administer a replication environment. Replication Center supports administration for DB2-to-DB2 replication environments as well as between DB2 and non-DB2 relational databases, such as SQL Server.

You can use the Replication Center to set up the three types of replication: SQL replication, queue replication, and event publishing. You can specify unidirectional and bidirectional replication with one or more servers. Use the Replication Center to:

- ▶ Create replication control tables.
- ▶ Register replication sources.
- ▶ Create subscription sets and add subscription set members to subscription sets.
- ▶ Operate the Capture program.

- ▶ Operate the Apply program.
- ▶ Monitor the replication process.

Figure 3-12 shows the Q replication launchpad.

You can launch the Replication Center from **Start Menu → Programs → IBM DB2 → General Administration Tools → Replication Center** or from a command line by typing **db2rc**. You can also launch the Replication Center from any of the DB2 UDB GUI tools from the Tools menu or by clicking the Replication Center icon .

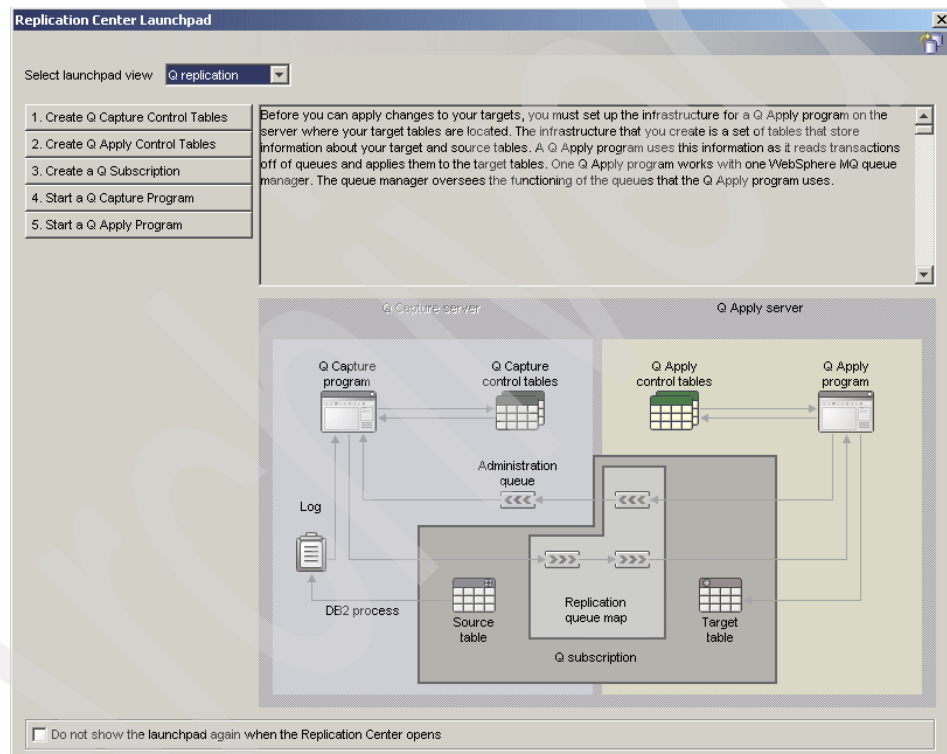



Figure 3-12 Replication Center Q replication launchpad

3.1.13 Data Warehouse Center

Data Warehouse Center (DWC) can be used to move data from operational databases to a data warehouse, which users can query for decision support. You can use the DWC to define the structure of the operational databases, called *sources*. You can then specify how the operational data is to be moved and transformed for the warehouse. You can model the structure of the tables in the

data warehouse, called *targets*, or build the tables automatically as part of the process of defining the data movement operations and loading the data.

Replication can be used to copy large quantities of data from warehouse sources into a warehouse target, and then capture any subsequent changes to the source data. These operations are supported on all of the DB2 UDB workstation operating environments, DB2 UDB for zSeries®, DB2 for iSeries™, and non-DB2 databases systems such as SQL Server, using WebSphere Information Integrator. You can also use the Data Warehouse Center to move data into an online analytical processing (OLAP) database. An expanded functionality version of DB2 Warehouse Center, called DB2 Warehouse Manager, is available as an additional cost option. Figure 3-13 shows the Data Warehouse Center with an SQL Server data source setup. You can launch Data Warehouse Center from **Start Menu → Programs → IBM DB2 → Business Intelligence Tools → Data Warehouse Manager** or from a command line by typing `db2dwc`. It can also be launched from any of the DB2 UDB GUI tools' Tools menu or by clicking the Data Warehouse Center icon  from any of the GUI tools.

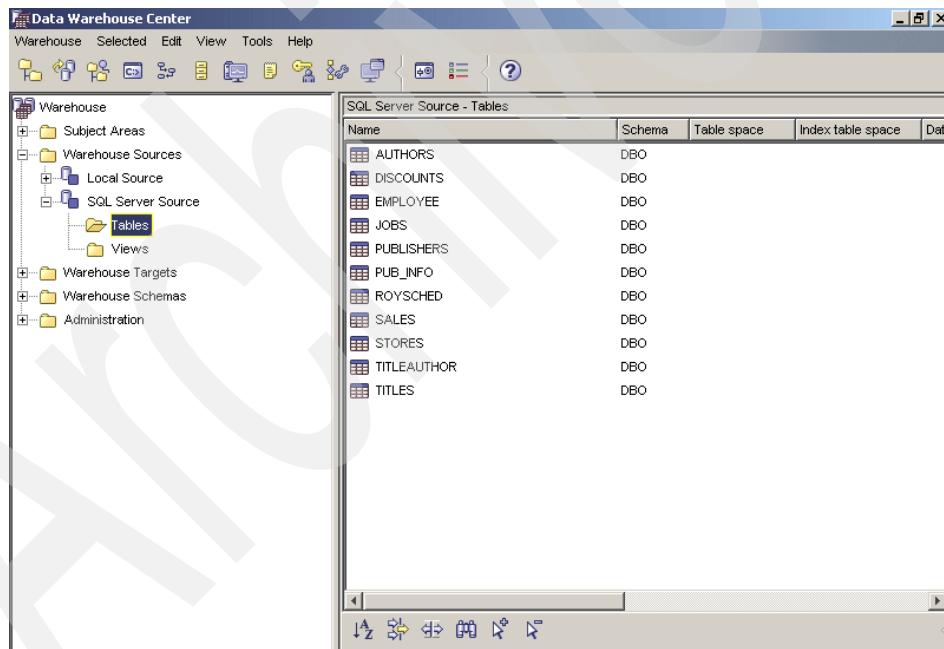



Figure 3-13 Data Warehouse Center

3.1.14 Information Catalog Center

Information Catalog Center provides the ability to manage descriptive data, known as business metadata, through information catalogs. The descriptive

data, which is organized into metadata objects, helps identify and locate information. You can search for specific objects in the information catalog and view any relationships an object participates in or an object's lineage. You can also create comments for objects. Information Catalog Center can be launched from **Start Menu → Programs → IBM DB2 → Business Intelligence Tools → Information Catalog Center** or from the command line by typing **db2icc**. It can also be launched from any of the DB2 UDB GUI tools' **Tools** menu or by clicking the Information Catalog Center icon  from any GUI tool.

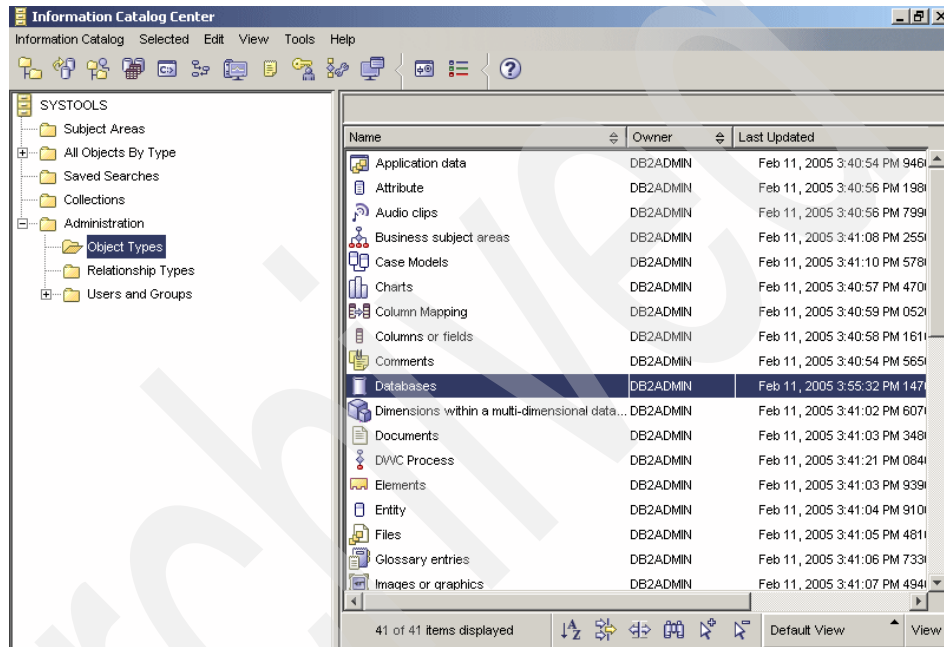


Figure 3-14 Information Catalog Center

3.1.15 Satellite Administration Center

Satellite Administration Center is a GUI tool used to administer a group of DB2 UDB servers that perform the same business function from a central point. Each DB2 UDB server in the group is known as a satellite. The servers could have a common application running on the server and a common configuration supporting the application. By having the servers grouped together administration is simplified because each satellite does not have to administer individually.

Within the Satellite Administration Center, you can create groups, application versions, satellites, batches, and authentication. Information about the satellite environment is stored in a central database known as the satellite control


database. You can launch the Satellite Administration Center from any of the DB2 UDB GUI tools' Tools menu or by clicking the Satellite Administration Center icon  from any GUI tool.

Figure 3-15 shows the Satellite Administration Center.

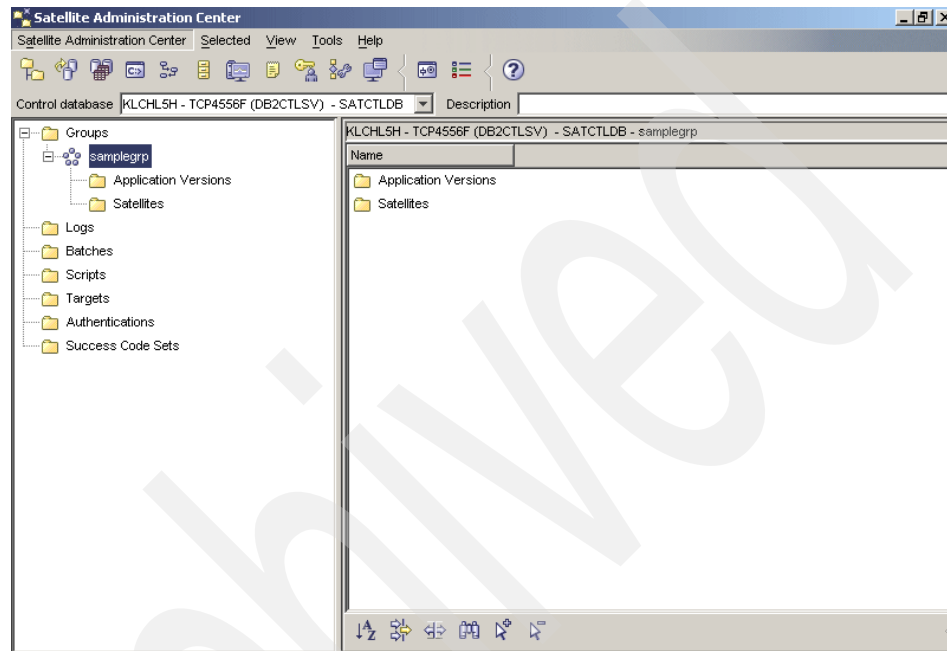


Figure 3-15 Satellite Administration Center

3.1.16 Web administration

Two Web-based tools are available to remotely administer a DB2 database: the *DB2 Web Command Center* and *DB2 Web Health Center*. These tools run as Web applications on a Web application server and provide access to DB2 servers through Web browsers.

The DB2 Web tool is based on a three-tier architecture. The first tier is the Web client HTTP browser. The middle tier is an application server that hosts the business logic such as IBM WebSphere Application Server and set of applications. This middle tier provides the underlying mechanisms for the communication (HTTP/HTTPS) with the first tier (Web client browser) and the third tier (database or transaction server).

The DB2 Web Command Center implements many of the already existing features of the Command Center but it does not contain the SQL Assist wizard or

Visual Explain. The DB2 Web Health Center displays data relating to the health of a DB2 instance. It displays internal data which is provided by the server-side health monitoring process.

Figure 3-16 shows the same script presented in 3.1.2, “Command Editor” on page 33, being executed in Web Command Center.

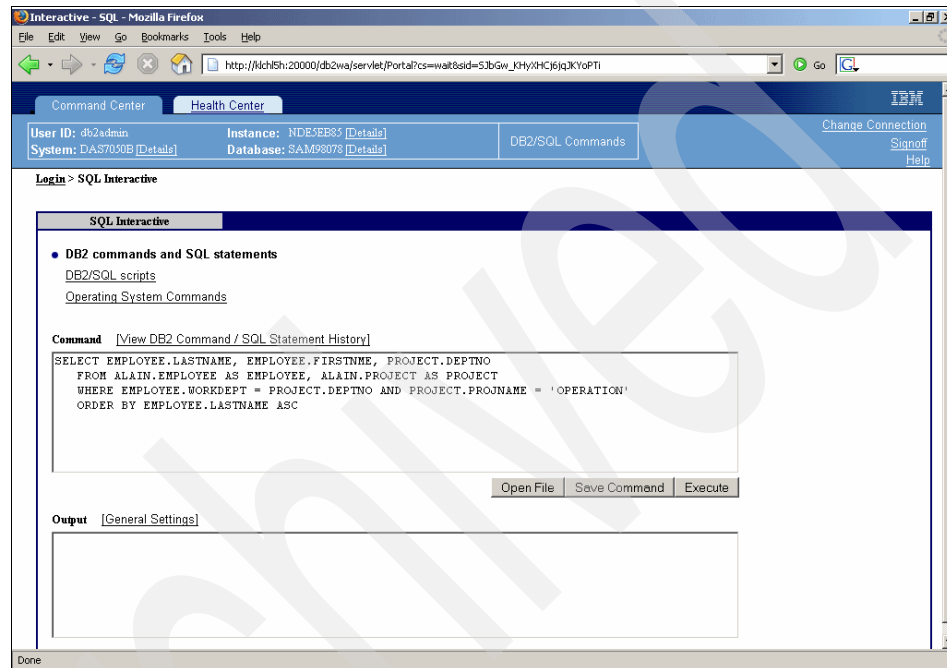


Figure 3-16 Web Command Center showing script execution

Figure 3-17 show the result of script execution.

The DB2 Web Command Center and Web Health Center are targeted for use with the HTTP clients (browsers) available on devices such as notebooks as well as Web-enabled PDAs and Palm devices. For more information about installing and setting up the Web tools refer to the DB2 UDB documentation *Installation and Configuration Supplement*, GC09-4837.

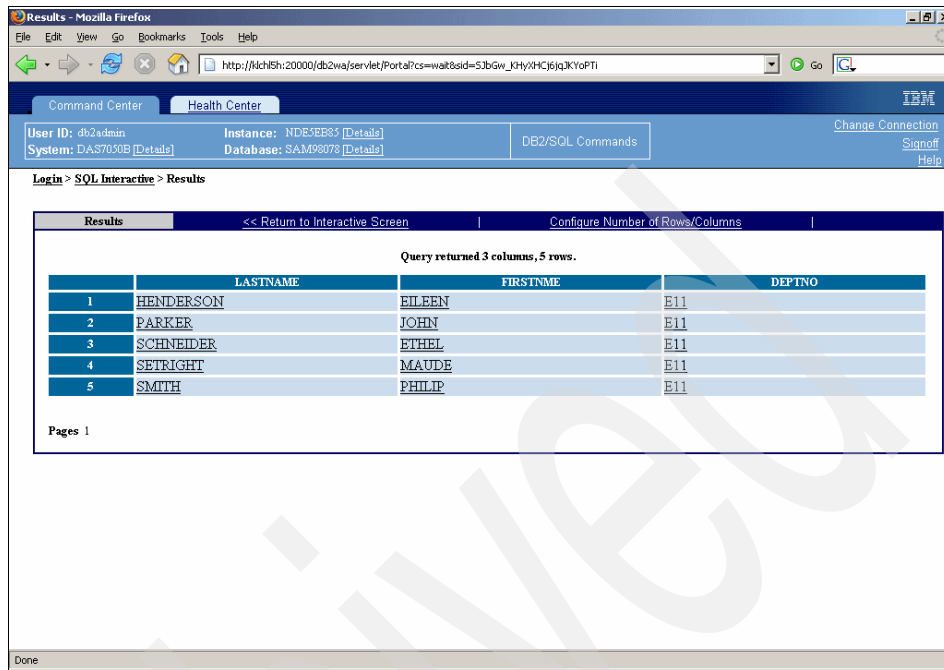


Figure 3-17 Web Command Center with script execution result

3.2 Wizards

DB2 UDB has a number of graphical wizards designed to guide the user through administration processes such as creating objects and manipulating data. The wizards greatly improve productivity, especially when making the transition from SQL Server to DB2 UDB. Some of the wizards accessible from the Control Center are:

- ▶ Create Database with Automatic Maintenance
- ▶ Create Database
- ▶ Create Table Space
- ▶ Create Table
- ▶ Design Advisor
- ▶ Configuration Advisor
- ▶ Backup
- ▶ Restore Data
- ▶ Load
- ▶ Configure Database Logging
- ▶ Add Partitions Launchpad
- ▶ Configure Automatic Maintenance

- ▶ Setup Activity Monitor
- ▶ Setup High Availability Disaster Recovery (HADR)

The wizards can be launched from context menus within Control Center or from the Wizards launchpad as shown in Figure 3-18.

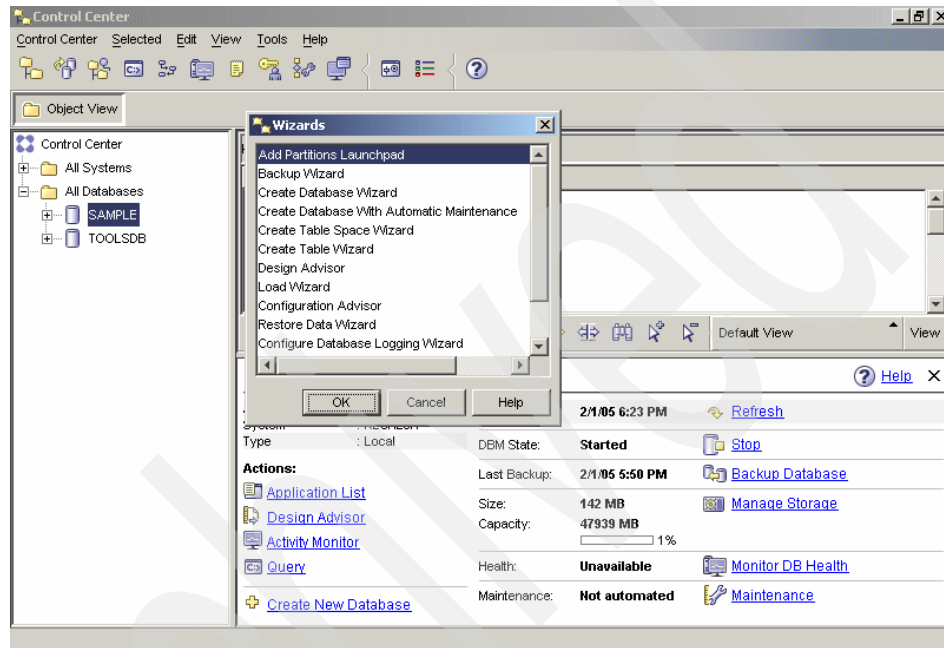


Figure 3-18 Wizards Launchpad

3.3 Advisors

Among the wizards, there are two “advisors” that help with database configuration and performance, the Configuration Advisor and the Design Advisor.

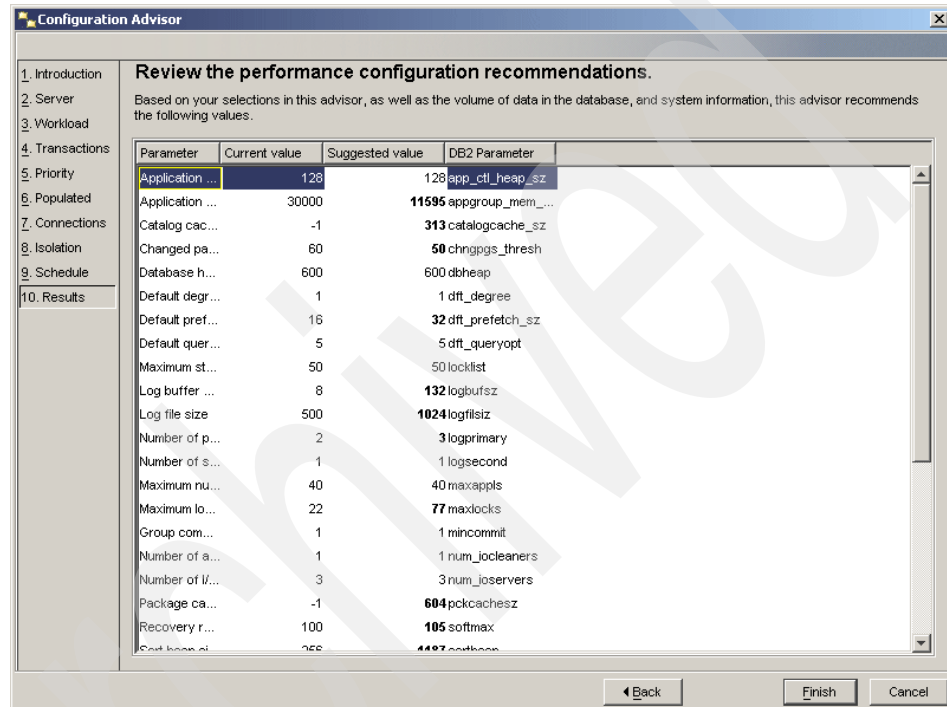
Configuration Advisor

Configuration Advisor can be used to help configure instance and database parameters that improve performance. The advisor asks for information such as:

- ▶ Amount of memory available to the Database Manager
- ▶ Type of workload expected
- ▶ Expected number of transactions
- ▶ Transaction performance optimization
- ▶ Whether or not the database is already populated with data

- The average number of local and remote application connections
- The isolation level

When the Configuration Advisor finishes evaluating the input received, a table providing the current and new recommended parameter values is displayed. Figure 3-19 shows the results displayed after running the Configuration Advisor. Changes can either be applied immediately or scheduled in the Task Center.



Configuration Advisor

Review the performance configuration recommendations.

Based on your selections in this advisor, as well as the volume of data in the database, and system information, this advisor recommends the following values.

Parameter	Current value	Suggested value	DB2 Parameter
Application ...	128	128	app_ctl_heap_sz
Application ...	30000	11595	appgroup_mem...
Catalog cac...	-1	313	catalogcache_sz
Changed pa...	60	50	chngpgs_thresh
Database h...	600	600	dbheap
Default degr...	1	1	dft_degree
Default pref...	16	32	dft_prefetch_sz
Default quer...	5	5	dft_queryopt
Maximum st...	50	50	locklist
Log buffer ...	8	132	logbufsz
Log file size	500	1024	logflsiz
Number of p...	2	3	logprimary
Number of s...	1	1	logsecond
Maximum nu...	40	40	maxappls
Maximum lo...	22	77	maxlocks
Group com...	1	1	mincommit
Number of a...	1	1	num_io cleaners
Number of l...	3	3	num_ioservers
Package ca...	-1	604	pckcachesz
Recovery r...	100	105	softmax
Sort heap si...	256	4497	sortheap

Navigation buttons: Back, Finish, Cancel

Figure 3-19 Configuration Advisor output

Design Advisor

The *Design Advisor* helps to determine the optimal set of database objects needed to achieve optimal workload performance. You provide the Design Advisor with a set of SQL statements that represent the typical workload and Design Advisor recommends additional indexes, materialized query tables (MQTs), multidimensional clustering tables, the repartitioning of tables, and the removal of objects no longer used by the specific workload.

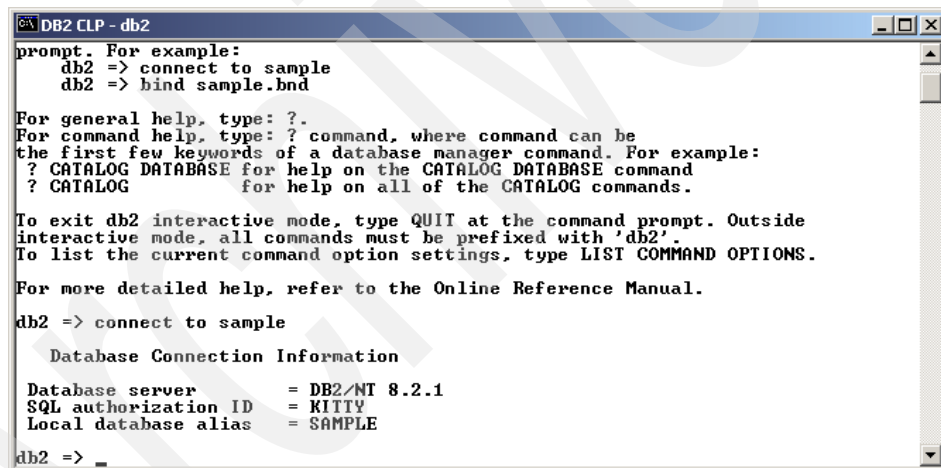
3.4 Command Line

Another useful DB2 UDB tool is the *Command Line Processor* (CLP). It is comparable to the ISQL found in SQL Server, although ISQL is more limited in what it can do. Most of the tasks that can be carried out with the GUI tools can be run at the CLP including running DB2 UDB commands, operating system commands, and SQL queries.

The CLP has three modes of operation:

- ▶ *Interactive mode*, characterized by the db2 => input prompt
- ▶ *Command mode*, where each command must be prefixed by db2
- ▶ *Batch mode*, which uses the -f file input option

When using the CLP in Interactive mode, you should only type the DB2 command. Figure 3-20 shows the “CONNECT TO <database>” command being used. To invoke an operating system command in Interactive mode, prefix the OS command with !, for example db2=>! dir.



```
DB2 CLP - db2
prompt. For example:
db2 => connect to sample
db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be
the first few keywords of a database manager command. For example:
? CATALOG DATABASE for help on the CATALOG DATABASE command
? CATALOG          for help on all of the CATALOG commands.

To exit db2 interactive mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the Online Reference Manual.

db2 => connect to sample

Database Connection Information

Database server      = DB2/NT 8.2.1
SQL authorization ID = KITY
Local database alias = SAMPLE

db2 => _
```

Figure 3-20 CLP in interactive mode

In Command Mode, each DB2 command must be prefixed with **db2**. Figure 3-21 shows the “CONNECT TO <database>” command. Operating system commands can be issued normally in this mode.

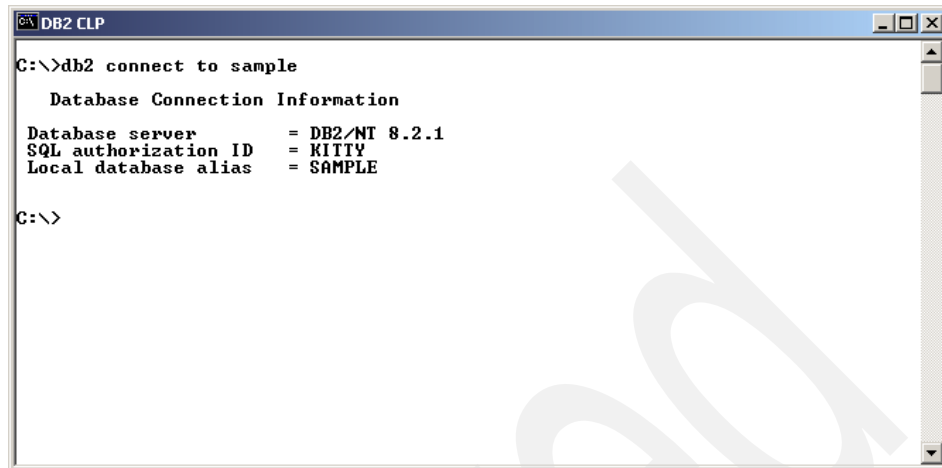


Figure 3-21 CLP in command mode

Batch mode allows a number of DB2 UDB commands contained in a file to be executed together. For example the DB2 UDB commands shown in Example 3-1 are contained in a text file called `clptest.txt` which can be invoked with the command:

```
C:\db2 -f clptest.txt
```

Example 3-1 DB2 UDB commands

```
connect to sample
list tables for schema kitty
list tablespaces show detail
connect reset
```

For more information about the CLP, refer to the following article:

<http://www.ibm.com/developerWorks/db2/library/techarticle/dm-0503melnyk/index.html>

3.5 Utilities

DB2 UDB has a complete set of maintenance utilities, available through the command line and graphical interface. In this section, we show the wizards and the command syntax for each utility.

Table 3-1 on page 54 is a comparison chart showing the major DB2 utilities with their SQL Server equivalents. For more information, see the DB2 UDB documentation *Command Reference*, SC09-4828.

Table 3-1 SQL Server and DB2 utilities comparison

Function	SQL Server	DB2 UDB
Backup/Restore	backup, dump, restore, load	backup, restore, recover
Load/Unload of data	BCP utility, import/export wizard	import (row at a time), load (bulk data), export , High Performance Unload (fast unload, a separately priced option), db2move (move database)
Reorganize	DBCC shrinkdatabase, DBCC shrinkfile , shrink database wizard; DBCC dbreindex, DBCC indexdefrag , defragment, rebuilt index wizard	reorg index, reorg table
Check if a database needs reorganization	DBCC showcohtig	reorgchk
Maintain database statistics	sp_createstats, sp_updatestats, update statistics, create statistics (can also be automated)	runstats (can also be automated)
DDL (schema) extraction	Generate SQL script wizard	db2look
Check database integrity	DBCC checkdb (various other options available)	db2dart, inspect/db2insfp
Check backup	n/a	db2ckbkp
Analyze queries	Display Estimated Execution Plan in Query Analyzer	explain, db2exfmt, db2expln , Visual Explain

3.5.1 Maintaining database integrity

There are two methods of checking for database integrity in DB2 UDB, **inspect** and **db2dart**. Both these commands are executed from the command line. There are many parameters available for each command.

Inspect

The **inspect** command inspects table spaces and tables for their architectural integrity, while the database remains online. Inspection validates table objects and table space structures. To run **inspect**, a database connection is required as are the following authorizations:

- ▶ SYSADM
- ▶ DBADM
- ▶ SYSCTRL
- ▶ SYSMANT
- ▶ CONTROL privilege if single table.

Some of the parameters that can be specified include:

- ▶ DATABASE
 - Specifies the whole database.
- ▶ BEGIN TBSPACEID (tablespaceID number)
 - Specifies that processing should begin from the table space with the given table space ID number.
- ▶ BEGIN TBSPACEID (tablespace ID number) OBJECTID (object ID number)
 - Specifies that processing should begin from the table space with the given table space ID number and object ID number
- ▶ TABLESPACE
 - NAME tablespace-name
 - Specifies a single table space by name.
 - TBSPACEID (tablespace ID number)
 - Specifies a single table space with the given table space ID number.
 - BEGIN OBJECTID (object ID number)
 - Specifies that processing should begin from a table with the given object ID number.
- ▶ TABLE
 - NAME table-name
 - Specifies a table by name.

- SCHEMA schema-name
 - Specifies a schema name for a specified table name.
- TBSPACEID n OBJECTID n
 - Specifies table with the given table space ID number and object ID number.

For example, if you wanted to check the integrity of a table space with an ID number of 4, you would issue the command:

```
db2 inspect check tablespace tbspaceid 4 results keep inspect.out
```

After you run **inspect**, the **db2inspf** command should be run to format the results, as the following example shows:

```
db2inspf inspect.out inspect.fmt
```

You can then view the results of the inspect by viewing the inspect.fmt file with a text viewer. The default location for the output file is the instance profile directory. The default instance profile directory is C:\Program Files\IBM\SQLLIB\DB2 on Windows systems. The output file is only available when there is at least one error.

For a full summary of all of the parameters available for the **inspect**, refer to DB2 UDB documentation *Command Reference*, SC09-4828, for further information.

DB2DART

The **db2dart** utility performs a similar task to **inspect**; however, no database access is allowed while the tool is running. Some of the parameters that can be specified with **db2dart** include:

- ▶ /DB
 - Inspects the entire database. This is the default option if no parameters are specified.
- ▶ /T
 - Inspects a single table.
- ▶ /TSF
 - Inspects table space files and containers.
- ▶ /TSC
 - Inspects a table space's constructs, but not tables within the table space.
- ▶ /TS

- Inspects a single table space and its tables.
- ▶ /ATSC
 - Inspects constructs of all table spaces, but not tables.

In addition, **db2dart** has the ability to perform certain repairs with the following parameters:

- ▶ /ETS
 - Extends the table limit in a 4 KB table space (DMS only).
- ▶ /MI
 - Marks an index as invalid. The database must be offline when this parameter is used.
- ▶ /MT
 - Marks a table with a *drop-pending* state. As with the /MI parameter, the database must be offline.
- ▶ /IP
 - Initializes the data page of a table as empty. Again the database must be offline to use this parameter.

For example, the command you would run to inspect the EMPLOYEE tables in the REDBOOK database would be:

```
db2dart redbook /t /tsi 2 /tn EMPLOYEE
```

This will generate a plain text file called REDBOOK.RPT in the SQLLIB\DB2\DART directory. You do not have to format the output for **db2dart**.

As with **inspect** you can find a full summary of all of the parameters available with **db2dart** refer to the DB2 UDB product documentation *Command Reference* SC09-4828 for further information.

3.5.2 Throttling utilities

DB2 UDB maintenance utilities, such as BACKUP and REBALANCE, can be very resource intensive. Running them can impact the performance of your production system. Such utilities are typically scheduled to run in off-peak hours. However, today's business demand of constant uncompromised availability makes finding any off-peak time very difficult. Deferring these utility tasks is not an option because they are required to assure data integrity and maintain query performance. With the introduction of utility throttling, you can regulate the performance impact of maintenance utilities so that they can be run concurrently with production periods. You can develop a throttling policy that will run the

utilities aggressively when the production workload is light, but will run them more conservatively as production demands increase. This functionality is unavailable in SQL Server.

The ability to throttle utilities enables you to:

- ▶ Execute maintenance tasks with total control over the performance impact to the production workload. This eliminates the need to identify off-peak hours or schedule downtime for utility tasks.
- ▶ Ensure that valuable system resources are fully used by utilities in periods of reduced demand.
- ▶ Eliminate performance impact as a consideration when monitoring a utility and configuring its parameters (for example, setting the `PARALLELISM` parameter for a database backup). After a throttling policy is established, it is the system's responsibility to ensure that the policy is obeyed.

3.5.3 Validating a backup

The `db2ckbkp` (check backup) utility can be used to test the integrity of a backup image and to determine whether or not the image can be restored. It can also be used to display the metadata stored in the backup header.

Example 3-2 shows the results of checking the integrity of a backup image of the `SAMPLE` database.

Example 3-2 db2ckbkp utility

```
C:\>db2ckbkp C:\db2backups\SAMPLE.0\DB2\NODE0000\CATN0000\20050202\180440.001
```

```
[1] Buffers processed: #####
```

```
Image Verification Complete - successful.
```

3.5.4 DDL extraction

The `db2look` utility is the equivalent to the SQL Server Generate **SQL** GUI utility and is used for extracting a database schema to an external file. These DDL statements can be used to reproduce the database objects in another database.

With this tool, it is possible to create a test database where the access plans created are similar to those that would be used on the production system.

The `db2look` utility can also be accessed from the Control Center at the database or the table level by right-clicking the database or table, and selecting **Generate DDL**.

Figure 3-22 shows the **db2look** GUI utility launched from Control Center.

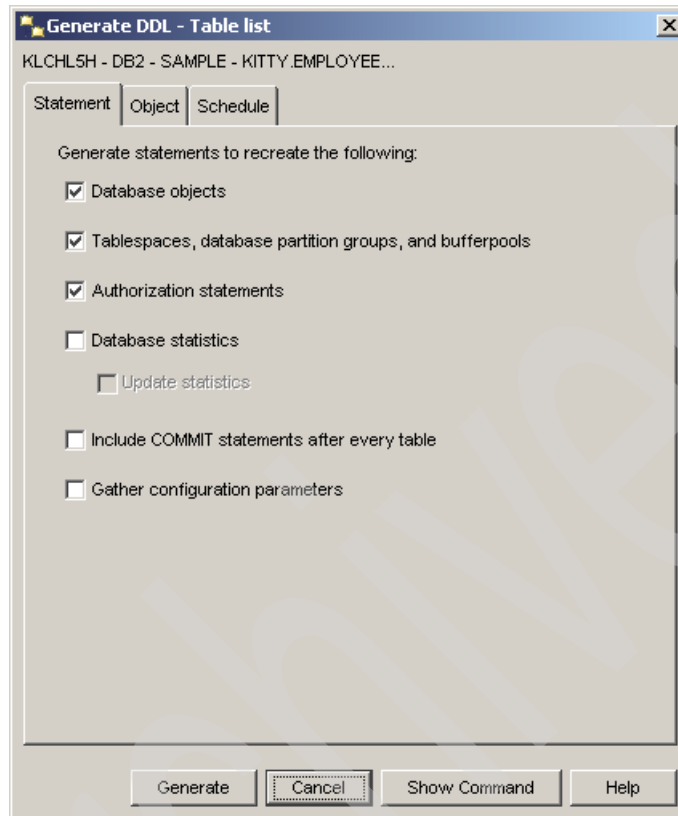


Figure 3-22 DB2LOOK GUI utility

Clicking the **Show Command** button shows the syntax of the command that can be used in the CLP. In this case, the command would be:

```
db2look -d sample -t EMPLOYEE -a -e -l -x -c
```

Example 3-3 shows the DDL generated with the **db2look** execution in Figure 3-22.

Example 3-3 DDL for the Employee table in the SAMPLE database

```
-- This CLP file was created using DB2LOOK Version 8.2
-- Timestamp: 02/03/2005 10:56:17 AM
-- Database Name: SAMPLE
-- Database Manager Version: DB2/NT Version 8.2.1
-- Database Codepage: 1252
-- Database Collating Sequence is: UNIQUE
```

```

CONNECT TO SAMPLE;

-- Mimic tablespace

ALTER TABLESPACE SYSCATSPACE
    PREFETCHSIZE AUTOMATIC
    OVERHEAD 12.670000
    TRANSFERRATE 0.180000;

ALTER TABLESPACE TEMPSPACE1
    PREFETCHSIZE AUTOMATIC
    OVERHEAD 12.670000
    TRANSFERRATE 0.180000;

ALTER TABLESPACE USERSPACE1
    PREFETCHSIZE AUTOMATIC
    OVERHEAD 12.670000
    TRANSFERRATE 0.180000;

-----
-- DDL Statements for table "KITTY"."EMPLOYEE"
-----

CREATE TABLE "KITTY"."EMPLOYEE" (
    "EMPNO" CHAR(6) NOT NULL ,
    "FIRSTNME" VARCHAR(12) NOT NULL ,
    "MIDINIT" CHAR(1) NOT NULL ,
    "LASTNAME" VARCHAR(15) NOT NULL ,
    "WORKDEPT" CHAR(3) ,
    "PHONENO" CHAR(4) ,
    "HIREDATE" DATE ,
    "JOB" CHAR(8) ,
    "EDLEVEL" SMALLINT NOT NULL ,
    "SEX" CHAR(1) ,
    "BIRTHDATE" DATE ,
    "SALARY" DECIMAL(9,2) ,
    "BONUS" DECIMAL(9,2) ,
    "COMM" DECIMAL(9,2) )
    IN "USERSPACE1" ;

COMMIT WORK;

CONNECT RESET;

```

```
TERMINATE;  
-- Generate statistics for all creators  
-- The db2look utility will consider only the specified tables  
-- Creating DDL for table(s)  
-- Binding package automatically ...  
-- Bind is successful  
-- Binding package automatically ...  
-- Bind is successful  
;
```

3.6 Performance monitor integration

DB2 UDB integrates with Windows Performance Monitor (perfmon) through the use of a plug-in. This allows monitoring of instances, databases, and DB2 UDB applications. You can also monitor DCS (host) applications and databases. The advantage with this kind of integration is that SQL Server DBAs that are familiar with perfmon can continue to use this method of performance monitoring.

You can register the DB2 UDB Performance Monitor counters with the **db2perf** command, as shown in Example 3-4:

Example 3-4 db2perf command usage

```
C:\Program Files\IBM\SQLLIB\BIN>db2perf -i  
SUCCESS : Installed DB2 Performance counters
```

```
C:\Program Files\IBM\SQLLIB\BIN>
```

There are other, third party tools that can be used to aid in performance monitoring, in addition to Performance Monitor. These tools include:

- ▶ Kernrate
<http://www.microsoft.com/downloads/details.aspx?FamilyID=d6e95259-8d9d-4c22-89c4-fad382eddc1&DisplayLang=en>
- ▶ Vtune
<http://www.intel.com/software/products/vtune/vpa/>
- ▶ CodeAnalyst
<http://www.developwithamd.com/appPartnerProg/codeanalyst/home/index.cfm?action=home>
- ▶ GA Tools Package
<http://ausgsa.ibm.com/projects/w/wbiperftools/documentation/tools/ga/packages/win32.html>

3.7 Optional tools

There are a number of optional tools designed to further enhance the management of DB2 UDB. More information about these tools can be found at the following URL:

<http://www.ibm.com/software/data/db2imstools/>

3.7.1 DB2 Performance Expert

DB2 Performance Expert offers a comprehensive view that consolidates, reports, analyzes, and recommends changes to DB2 performance-related information. The tool includes a Performance Warehouse that stores performance data and analysis tools and a Buffer Pool Analyzer that collects data and provides reports on related event activity. DB2 Performance Expert builds on IBM autonomic computing and on demand expertise, providing recommendations for system tuning to gain optimum throughput. The tool is available for DB2 UDB on the Linux, UNIX, and Windows platforms, as well as z/OS.

For more information, refer to the redbook *DB2 Performance Expert for Multiplatforms V2*, SG24- 6470.

3.7.2 DB2 Recovery Expert

DB2 Recovery Expert provides targeted, flexible, and automated recovery of database assets. DB2 Recovery Expert helps expert and novice DBAs to recover database objects safely, precisely, and quickly without having to resort to full disaster recovery processes.

Building on IBM autonomic computing expertise, this tool provides intelligent analysis and diagnostics of altered, incorrect, or missing database assets including tables, indexes, or data. It also automates the process of rebuilding those assets to a correct “point-in-time,” often without taking the database offline. In addition, you can mine the database logs to “undo” or “redo” SQL statements and to remove errant transactions without needing to do a full table space or database recovery.

For more information, refer to redbook *DB2 REcovery Expert for Multiplatforms*, SG24- 6421.

3.7.3 DB2 High Performance Unload

High Performance Unload is a high-speed DB2 utility for unloading data from DB2 tables in a database or backup images. The tool is available for DB2 UDB on z/OS, as well as on the Linux, UNIX, and Windows platforms.

Using HPU, you can quickly unload your DB2 data into a variety of formats, such as flat files, tape, or named pipes. Additional features include:

- ▶ Fast SQL-like SELECT capabilities, such as filtering columns and rows using a WHERE clause
- ▶ Unloads a sampling of data (for example, every nth row)
- ▶ Unloads data from online databases or database backups
- ▶ Unloads all data or data from selected table spaces

Consider using High Performance Unload if you need to significantly reduce your unload times or batch processing windows, or if you are performing multiple unloads against the same table.

3.7.4 DB2 Test Database Generator

DB2 Test Database Generator rapidly populates application and testing environments and simplifies problem resolution. It can easily create test data from scratch or from existing data sources and maintains referential integrity while extracting data sets from source databases. It can create complete or scaled down copies of production databases while masking sensitive production data for use in a test environment.

Consider DB2 Test Database Generator when:

- ▶ Application developers and testers need to routinely generate test databases or regenerate copies of their original test databases.
- ▶ Sales people need realistic test data to use for product demonstrations.
- ▶ DBAs need relief from the time-consuming task of generating appropriate test data for applications.

DBAs need test data to use when evaluating new software products.

3.7.5 DB2 Table Editor

DB2 Table Editor quickly and easily accesses, updates, and deletes data across multiple DB2 database platforms. Key features include:

- ▶ Navigates IBM DB2 databases, tables and views; finds related data; and quickly updates, deletes, or creates data with full support for your existing security and logon IDs.
- ▶ Edits DB2 tables everywhere with your choice of end-user entry points: Java-enabled Web browsers, Java-based interfaces launched from the IBM DB2 Control Center and Microsoft Windows.
- ▶ Provides drag-and-drop and wizards to rapidly create customized, task-specific Java-bases or Windows-based table editing forms containing built-in data validation and business rules.

Consider DB2 Table Editor when you need to edit DB2 tables, need easy-to-build forms capability for end users, or need access to DB2 data.

3.7.6 DB2 Web Query Tool

DB2 Web Query Tool connects all your users directly to multiple enterprise databases, securely and simultaneously, regardless of database size, hardware, operating system, or location.

The key features of DB2 Web Query Tool are:

- ▶ Enables complex querying, data comparisons, and customized presentations.
- ▶ Provides rapid global access to business information over e-mail clients, including WAP-enabled devices such as PDAs, wireless phones, and text pagers.
- ▶ Supports standard browsers, giving administrators, developers, and end users the ability to build queries that support multiple DB2 platforms, share and run the queries, and convert the results to XML and other highly transportable file formats.
- ▶ Is a J2EE-compliant Web application, so it can be deployed on WebSphere and other application servers.

Consider DB2 Web Query Tool when you need comprehensive query and comparison capabilities without compromising DB2 security or data integrity, require thin client access from many different devices on your network, or need access to DB2 databases across an enterprise.

SQL considerations

This chapter highlights SQL language syntactical and semantic differences between SQL Server and DB2 UDB as well as other SQL language and limit differences. In particular, we cover:

- ▶ SQL standard compliance
- ▶ Data types
- ▶ Date and time considerations
- ▶ String considerations
- ▶ Case sensitivity
- ▶ SQL language syntax and semantics
- ▶ Built-in SQL functions
- ▶ System catalog queries
- ▶ Transact-SQL to SQL PL translation
- ▶ XML
- ▶ SQL limits

4.1 SQL standard compliance

Most relational database management systems (DBMSs) are either fully compliant or conform to one of the SQL standards. Thus, many SQL statements and queries written for SQL Server can be converted to DB2 UDB without modification. However, certain SQL syntax and semantic differences exist across DBMSs, depending on what standard is implemented and level of conformance.

SQL Server complies with the entry level of ANSI SQL92 standard and with the Federal Information Processing Standards (FIPS 127-2) as established by the US National Institute of Standards and Technology (NIST). The commands `SET FIPS_FLAGGER` and `SET ANSI_DEFAULTS` change the level of compliance and SQL92 behavior.

DB2 UDB complies with the SQL92 standard at the entry-level and includes additional features from the intermediate and full levels. DB2 UDB also contains many core features of the SQL99 standard, and many features that are beyond the SQL99 Core are partially or completely supported. There are no commands to change the SQL compliance level, but the `SQLFLAG` option on the `PREP` command can be used in embedded SQL applications to check that SQL syntax conforms to the SQL92 Entry Level syntax.

4.2 Data types

There is a well-defined mapping between SQL Server and DB2 UDB data types. This section highlights the important differences between various data types in SQL Server and DB2 UDB.

4.2.1 Data type mapping

There are some differences between the data types supported in SQL Server and DB2 UDB. However, most of the data types supported in SQL Server can be mapped to equivalent DB2 UDB data types. Refer to Appendix B, “Data type mapping” on page 469 for a complete data type mapping.

4.2.2 Strong type casting

DB2 UDB provides strong type casting to avoid end-user mistakes during the assignment or comparison of different types involving real world data. Casting between data types can be done explicitly using the `CAST` operation, but may also occur implicitly during assignments involving user-defined types. For example,

casting an integer data type into a character string data type can be done in an expression or SQL statement such as the following:

```
VALUES CAST (1000 AS DOUBLE)
```

Differences in implicit type conversions between SQL Server and DB2 UDB can sometimes result in complications in the translation of DML statements. For example, the following SQL statements are valid in SQL Server, but not in DB2 UDB:

```
CREATE TABLE t1 (c1 DATETIME)
CREATE TABLE t2 (c1 VARCHAR(26))
```

```
INSERT INTO t2 SELECT * FROM t1
```

The above INSERT statement is valid in SQL Server because the DATETIME data type in column c1 is implicitly casted as a VARCHAR. DB2 UDB does not implicitly convert a TIMESTAMP into a VARCHAR. In order to correctly translate this statement, the wildcard character (*) must be replaced by the names of all the required columns, explicitly casting each column to its appropriate data type where necessary. The INSERT statement above can be converted to DB2 UDB as:

```
INSERT INTO t2 SELECT TO_CHAR(c1) FROM t1
```

where the function TO_CHAR() explicitly converts a TIMESTAMP data type into a character string data type in a specified format. Examples of working with dates and times are included in 4.3, “Date and time considerations” on page 70.

Refer to Chapter 2 in the DB2 UDB documentation *SQL Reference - Volume 1*, SC09-4844 for more information about data type promotion and casting considerations.

4.2.3 ROWVERSION data type

SQL Server provides a special data type for facilitating the automatic generation of unique binary numbers. This data type is the ROWVERSION (formally known as TIMESTAMP) data type. While DB2 UDB also has a TIMESTAMP data type; however, it is used for storing date and time values.

The direct equivalent of ROWVERSION in DB2 UDB is the TIMESTAMP data type. However, a better solution is to use an IDENTITY column. Example 4-1 shows how to define a table with a unique, automatically generated column in DB2 UDB.

Example 4-1 An automatically generated unique column in DB2 UDB

```
CREATE TABLE book_orders (  
    book_id      INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY  
                (START WITH 1, INCREMENT BY 1),  
    order_date   DATE NOT NULL,  
    customer_id  INTEGER NOT NULL,  
    quantity     INTEGER NOT NULL)  
!
```

There are three additional methods of converting ROWVERSION data types to DB2 UDB (i.e., generating unique values):

- ▶ GENERATE_UNIQUE()
- ▶ sequence objects
- ▶ stored procedures

The DB2 UDB function GENERATE_UNIQUE() in Example 4-2 generates a bit data character string that is unique between subsequent calls. This allows for an infinite number of values to be generated while ensuring that all values are unique throughout the database.

Example 4-2 Unique data with the GENERATE_UNIQUE() function

```
CREATE TABLE book_orders (  
    book_id      CHAR(13) FOR BIT DATA,  
    order_date   DATE NOT NULL,  
    customer_id  INT NOT NULL,  
    quantity     INT NOT NULL)  
!  
  
INSERT INTO book_orders  
VALUES ( GENERATE_UNIQUE(),  
        CURRENT DATE,  
        0001,  
        100 )  
!
```

Another conversion method involves the use of sequence objects. Since sequence objects are separate database objects, they can be referenced in any SQL statement. Thus, a single sequence object can be used to generate values that are unique throughout the database. The use of sequence objects defined using the BIGINT data type is recommended as this allows for the same number of possible values as ROWVERSION.

If neither of the previous methods are satisfactory, it is possible to generate unique values manually using a stored procedure written in an external language such as C or Java.

The following articles on the DB2 developerWorks® Web site describe how to generate unique values in DB2 UDB:

- *Generating Unique Values in DB2 Universal Database*

<http://www.ibm.com/developerworks/db2/library/techarticle/0205pilaka/0205pilaka2.html>

- *Automatically Generating Sequences of Numeric Values in DB2 Universal Database*

<http://www.ibm.com/developerworks/db2/library/techarticle/0302fielding/0302fielding.html>

4.2.4 NULL value considerations

SQL Server and DB2 UDB both support the use of NULL values; however, there are some differences between the way they are evaluated.

In both SQL Server and DB2 UDB, NULL is interpreted as a synonym for “*unknown*”. As such, you cannot write expressions that test for nullability using the equals (=) operator. For instance, the expression:

```
SELECT ... WHERE empno = NULL
```

is not provided for in the ANSI SQL standard and will return an error. Expressions that test for nullability must use the IS [NOT] NULL clause. The expression above can correctly be re-written as:

```
SELECT ... WHERE empno IS NULL
```

In SQL Server, the ANSI NULL behavior is set by default. However, if the ANSI NULL option is not set, SQL Server interprets equality and inequality comparisons involving a variable, such as the expression `someColumn = @someVariable`, in some nonstandard way. In particular, when both `someColumn` and `@someVariable` are NULL, the expression will evaluate to true in SQL Server, but will evaluate to NULL (as per the SQL standard) in DB2 UDB. Additionally, the result of concatenation and arithmetic operations (e.g., `NULL + 3`, `'MAR' concat NULL`) are different between SQL Server and DB2 UDB.

4.2.5 Large object (LOB) considerations

DB2 UDB supports three types of large objects (LOBs). Of particular interest are DBCLOBs and BLOBs. The DBCLOB data type is used for storing double-byte character strings over 32 KB. BLOB columns can be used to store up to 2 GB of binary data.

The NOT LOGGED option can be specified when declaring LOB columns. It is mandatory for columns greater than 1 GB in size. Generally, it is recommended

for LOBs larger than 10 MB as changes to large columns can quickly fill the log file. Even if NOT LOGGED is used, changes to LOB files during a transaction can still be successfully rolled back.

A second option that can be used with LOB columns is COMPACT. Using the COMPACT option incurs a performance cost; however, it can greatly decrease disk usage, especially on operating systems that do not support sparse file allocation.

DB2 UDB does not store LOB data in its internal buffer pools. Instead, data is read directly from the disk each time it is needed. Because of this, it is highly recommended that LOB data be placed in an SMS table space or a DMS table space that uses file containers. Both of these types of table spaces will take advantage of the operating system's file caching, reducing the I/O requirements and increasing performance.

Users should be aware of the restrictions of the use of LOBs in DB2 UDB. These restrictions include LOBs not being permitted in:

- ▶ Indexes
- ▶ A GROUP BY clause
- ▶ An ORDER BY clause
- ▶ The pattern operand in a LIKE predicate, or the search string operand in a POSSTR function

Full detail about LOBs, including the complete list of limitations, can be found in the DB2 UDB documentation *Application Development Guide: Programming Server Applications*, SC09-4827.

4.3 Date and time considerations

SQL Server and DB2 UDB have specific data types for working with dates and times, as well as functions for retrieving, converting, and performing arithmetic on their values.

SQL Server has two date and time data types, DATETIME and SMALLDATETIME. DB2 UDB has three distinct data types for date and time values, DATE, TIME and TIMESTAMP. Since DB2 UDB uses strong data-typing, implicit conversion between these data types is not performed.

Both of SQL Server's types can be mapped to DB2 UDB's TIMESTAMP data type; however, this may not be the best solution. If only the date portion of a DATETIME or SMALLDATETIME column is used by the application, it is more efficient to convert it to DB2 UDB's DATE type. Similarly, if only the time portion of a DATETIME or SMALLDATETIME column is used by the application, it is more efficient to convert

these to DB2 UDB's TIME type. In the next sections, we provide examples for manipulating dates and time values.

4.3.1 Retrieving date and time values

To retrieve the current date in SQL Server, a SELECT statement with the getdate() function is used:

```
SELECT getdate()
```

DB2 UDB stores the current date and time in special system registers. The values retrieved from these registers are based on a reading of the time-of-date clock when the SQL statement is executed at the application server. These special registers are accessible using a VALUES or SELECT statement:

```
VALUES CURRENT TIMESTAMP  
VALUES CURRENT DATE  
VALUES CURRENT TIME
```

Or,

```
SELECT CURRENT TIMESTAMP FROM SYSIBM.SYSDUMMY1  
SELECT CURRENT DATE FROM SYSIBM.SYSDUMMY1  
SELECT CURRENT TIME FROM SYSIBM.SYSDUMMY1
```

Note: If these special registers are used more than once within a single SQL statement, or mixed together within a single statement, all values retrieved are based on a single clock reading.

4.3.2 Date and time conversion

To convert a date between a string and date/time data type in SQL Server, the CONVERT() function is used:

```
CONVERT(VARCHAR(20), GETDATE())  
CONVERT(DATETIME, '7/4/2000')  
CONVERT(DATETIME, '10.12.99', 1)
```

DB2 UDB supports the functions TO_CHAR() and TO_DATE(), but only one format is natively supported, as illustrated here:

```
TO_CHAR (<timestamp_expression>, 'YYYY-MM-DD HH24:MI:SS')  
TO_DATE (<string_expression>, 'YYYY-MM-DD HH24:MI:SS')
```

A DB2 UDB user defined function (UDF) is an ideal solution for creating a customized to_char() or to_date() casting function. The following examples (Example 4-3 and Example 4-4) show what these functions might look like.

Example 4-3 Sample to_char() user defined function in DB2 UDB

```
CREATE FUNCTION MS7.to_char (   timestamp_in TIMESTAMP,
                               format      VARCHAR(10))

RETURNS VARCHAR(10)
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
RETURN
    WITH PARTS(yyyy, mm, dd, hh, mi, ss, tttttt) AS
    (
        VALUES( SUBSTR(CHAR(timestamp_in), 1, 4),
                  SUBSTR(CHAR(timestamp_in), 6, 2),
                  SUBSTR(CHAR(timestamp_in), 9, 2),
                  SUBSTR(CHAR(timestamp_in), 12, 2),
                  SUBSTR(CHAR(timestamp_in), 15, 2),
                  SUBSTR(CHAR(timestamp_in), 18, 2),
                  SUBSTR(CHAR(timestamp_in), 21, 6)
        )
    )
SELECT
CASE
    WHEN UCASE(format) = 'MM/DD/YYYY' THEN mm || '/' || dd || '/' || yyyy
    WHEN UCASE(format) = 'MM/DD/YY' THEN mm || '/' || dd || '/' ||
        SUBSTR(yyyy, 3, 2)
    WHEN UCASE(format) = 'HH24:MI:SS' THEN hh || ':' || mi || ':' || ss
    ELSE mm || '/' || dd || '/' || yyyy
END
FROM PARTS
!
```

Example 4-4 Sample to_date() user defined function in DB2 UDB

```
CREATE FUNCTION MS7.to_date (timestamp_in VARCHAR(21),
                              format VARCHAR(21))

RETURNS TIMESTAMP
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
RETURN
    WITH PARTS(mm, dd, yyyy, hh, mi, ss) AS
    (
        VALUES( SUBSTR(timestamp_in, 1, 2),
                  SUBSTR(timestamp_in, 4, 2),
                  SUBSTR(timestamp_in, 7, 4),
                  SUBSTR(timestamp_in, 12, 2),
                  SUBSTR(timestamp_in, 15, 2),
                  SUBSTR(timestamp_in, 18, 2)
        )
    )
```

```

        )
SELECT
CASE
    WHEN UCASE(format) = 'MM/DD/YYYY@HH24:MI:SS' THEN
        TIMESTAMP(yyyy||mm||dd||hh||mi||ss)
    WHEN UCASE(format) = 'MM-DD-YYYY' or UCASE(format) = 'MM/DD/YYYY' THEN
        TIMESTAMP(yyyy||mm||dd||'000000')
END
FROM PARTS
!

```

Note that the above functions are different than the built-in `to_char()` and `to_date()` functions, since they offer the ability for customization.

The `CHAR()` function in DB2 UDB can be used to convert a date or time value to a character string in a local or specified standard format, such as:

```

CHAR(CURRENT DATE, ISO);
CHAR(CURRENT DATE, USA);

```

The following examples show how DB2 UDB dates can be converted into different formats:

```

CHAR(CURRENT DATE) => '10/01/2003'
CHAR(CURRENT DATE + 3 DAYS) = '10/04/2003'
CHAR(CURRENT DATE, ISO) = '2003-10-01'
CHAR(CURRENT DATE, EUR) = '01.10.2003'
CHAR(CURRENT DATE, JIS) = '2003-10-01'
CHAR(CURRENT TIME, USA) = '02:21 PM'
CHAR(CURRENT TIME + 2 HOURS, EUR) = '16.21.23'

```

Note: For more information about the `CHAR()` function, refer to the DB2 UDB documentation *SQL Reference Volume 1*, SC09-4844.

4.3.3 Date and time arithmetic

Both SQL Server and DB2 UDB provide a suite of duration and date/time arithmetic capability. SQL Server built-in functions, such as `DATEADD()`, `DATEDIFF()`, and `DATEPART()`, can in some cases be translated to equivalent or similar DB2 UDB built-in functions. When a SQL Server function does not have a DB2 UDB equivalent, or the calling code is unable to be changed, a UDF can in most cases be created with the same name and behavior as the original SQL Server function.

The only arithmetic operations that DB2 UDB supports for date and time values are addition and subtraction. These operations are supported through the normal addition and subtraction operators (i.e., `+` and `-`). Supplementing these

operations is a rich set of type-casting functions (e.g., YEARS(), DAYS(), HOURS(), SECONDS(), etc.).

The following examples show how date arithmetic may be performed in SQL Server:

```
DATEADD(day, 1, GETDATE())
DATEADD(month, 1, GETDATE())
DATEADD(year, 1, GETDATE())
```

In DB2 UDB, the same functionality can be implemented as follows:

```
CURRENT DATE + 1 day
CURRENT DATE + 1 month
CURRENT DATE + 1 year
```

Additional examples of date/time arithmetic and duration operations in DB2 UDB include:

<u>Arithmetic operation</u>	<u>Result</u>
CURRENT DATE	10/02/2003
CURRENT DATE + 3 DAYS	10/05/2003
CURRENT TIMESTAMP + 2 YEARS	2005-10-02-12.33.27.667000
CURRENT TIMESTAMP - 2 MONTHS	2003-08-02-12.33.27.667002
CURRENT TIME + 5 MINUES	12:38:27

DB2 UDB also provides additional convenience functions when working with date and time values, such as DAYNAME(), MONTHNAME(), and many more.

4.3.4 Examining date and time components

In SQL Server, the function DATEPART() returns a specified component of a date, for example, would return the day (number) of the current date:

```
DATEPART(day, GETDATE())
```

DB2 UDB has similar functions for extracting components from date and time values. The equivalent DB2 UDB function to the above SQL Server DATEPART function is:

```
DAY(CURRENT DATE)
```

Table 4-1 shows the mapping of various DATEPART() parameters in SQL Server to the corresponding functions in DB2 UDB.

Table 4-1 Mapping of T-SQL DATEPART() function parameters

SQL Server	DB2 UDB
DATEPART(year, GETDATE())	YEAR(CURRENT DATE)

SQL Server	DB2 UDB
DATEPART(quarter, GETDATE())	QUARTER(CURRENT DATE)
DATEPART(month, GETDATE())	MONTH(CURRENT DATE)
DATEPART(dayofyear, GETDATE())	DAY OFYEAR(CURRENT DATE)
DATEPART(day, GETDATE())	DAY(CURRENT DATE)
DATEPART(week, GETDATE())	WEEK(CURRENT TIME)
DATEPART(weekday, GETDATE())	DAYOFWEEK(CURRENT DATE)
DATEPART(hour, GETDATE())	HOURL(CURRENT TIME)
DATEPART(minute, GETDATE())	MINUTE(CURRENT TIME)
DATEPART(second, GETDATE())	SECOND(CURRENT TIME)
DATEPART(millisecond, GETDATE())	MICROSECOND(CURRENT TIMESTAMP)

4.3.5 Additional date and time examples

Additional information and examples about DB2 UDB user defined functions that manipulate date and time values, such as DUMP(), NEW_TIME(), NEXT_DAY(), TRUNC(), and others can be found on the DB2 developerWorks Web site at:

<http://www.ibm.com/developerworks/db2/library/samples/db2/0205udfs/>

Another ideal place for using a UDF is converting the SQL Server months_between() function:

```
months_between(sysdate,v_date)
```

If you use the IBM DB2 Migration Toolkit (MTK) to automate the conversion, MTK implements the months_between() as a UDF and automatically deploys it into the target DB2 UDB database. Example 4-5 shows the source code for the months_between() function:

Example 4-5 DB2 UDB months_between() function

```
CREATE FUNCTION months_between( d1 TIMESTAMP,
                                d2 TIMESTAMP)

RETURNS FLOAT
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
CONTAINS SQL
RETURN 12 * (year(d1) - year(d2)) + month(d1) - month(d2)
      + (TIMESTAMPDIFF(2,CHAR(d1 - (d2 + (12*(year(d1) - year(d2))
```

+ month(d1) - month(d2)) MONTHS))) / 2678400.0)
!

This function uses the DB2 UDB built-in function `TIMESTAMPDIFF()`. Details about `TIMESTAMPDIFF()` can be found in the DB2 UDB documentation *SQL Reference - Volume 1*, SC09-4844.

For more information about manipulating date and time values, refer to the article *DB2 Basics: Fun with Dates and Times* on the DB2 developerWorks Web site at:

<http://www.ibm.com/developerworks/db2/library/techarticle/0211yip/0211yip3.html>

4.4 String considerations

SQL Server and DB2 UDB have similar string comparison and concatenation behaviors; however, there are some important differences to be aware of between the use of character strings in SQL Server and DB2 UDB.

In SQL Server, `CHAR` columns are of variable length whereas in DB2 UDB, they are of fixed length. The following example illustrates the implications of this. Suppose the following SQL Server table definition:

```
CREATE TABLE t (cOne      CHAR(1),
                 cNull     CHAR(5),
                 cNotNull   CHAR(5) NOT NULL);
```

The equivalent table definition in DB2 UDB would be:

```
CREATE TABLE t (cOne      CHAR(1),
                 cNull     VARCHAR(5),
                 cNotNull   CHAR(5) NOT NULL);
```

Table 4-2 shows the resulting character string lengths when the values shown are inserted into the tables defined above.

Table 4-2 Lengths of various character values

Inserted data values	Value in SQL Server for: len(cOne) len(cNull) len(cNotNull)	Value in DB2 UDB for: length(cOne) length(cNull) length(cNotNull)
INSERT INTO t VALUES ('x', 'x', 'x')	1 1 1	1 5 5

Inserted data values	Value in SQL Server for: len(cOne) len(cNull) len(cNotNull)	Value in DB2 UDB for: length(cOne) length(cNull) length(cNotNull)
INSERT INTO t VALUES (' ', ',', ',', VALUES (' ')	0 0 0	1 5 5
INSERT INTO t VALUES (' ', ',', ',', ',')	0 0 0	1 0 5

The following information should also be kept in mind while working with strings in DB2 UDB:

- ▶ Concatenating a NULL value to a string will always result in a NULL value. For instance, if a variable has not been initialized or contains a NULL value, concatenation with a character string will evaluate to a NULL value.
- ▶ The empty string literal (denoted as '' in a SQL expression) is NOT equivalent to a NULL value.
- ▶ The empty string literal is NOT equivalent to the single space string (denoted as ' ' in a SQL expression).

4.5 Case sensitivity

Both SQL Server and DB2 UDB support case-sensitive environments, where object identifiers and character strings can contain mixed upper and lower case characters; however, both achieve this functionality differently.

In SQL Server, case sensitivity is determined by the environment settings. The environment can be either case-sensitive or case-insensitive. In most cases, operations that work in a case-sensitive environment will also work in a case-insensitive environment, but not the converse. For example, the object identifier (table name) in the SQL statement `SELECT * FROM mytable` would be equivalent to the SQL statement `SELECT * FROM MYTABLE` in a case-insensitive environment, but not in a case sensitive environment. The result of comparing character strings in table columns or variables is also determined by the environment settings.

The following query can help determine whether you are operating in a case-sensitive or case-insensitive environment in SQL Server:

```
SELECT CASE WHEN ('A' = 'a') THEN -1 ELSE 1 END
```

In a case-insensitive environment, this query returns the value -1, otherwise it will return the value 1. In SQL Server, case sensitivity is controlled at different levels. Each SQL Server instance has a default collation setting, which determines the character set and sort order of characters. Databases can override the instance-level collation settings.

In DB2 UDB, all database object identifiers (tables, views, columns, etc.) are stored in the catalog tables in uppercase characters, unless they are explicitly delimited upon object creation. If a delimited identifier is used to create the object, the exact case of the identifier is stored in the catalog tables. An identifier, such as a column name or table name, is treated as case insensitive when used in an SQL statement unless it is explicitly delimited. For example, assume that the following statements are issued in DB2 UDB:

```
CREATE TABLE MyTable (id INTEGER)
CREATE TABLE "YourTable" (id INTEGER)
```

Two tables, MYTABLE and YourTable exist, and the following two statements are therefore equivalent:

```
SELECT * FROM MyTable
SELECT * FROM MYTABLE
```

However, the second statement below will fail with a SQL0204N error saying it cannot find an object (table) named YOURTABLE (uppercase):

```
SELECT * FROM "YourTable"      -- executes without error
SELECT * FROM YourTable        -- error, table not found
```

DB2 UDB strings are also case sensitive. For example, the string 'daTAbase' is not the same as the string 'DATABASE'. Case sensitivity must be taken into account when comparing strings to ensure consistent results. One strategy for doing this involves always performing string comparisons using the uppercase or lowercase representation of the string, via the UPPER() and LOWER() DB2 UDB functions. Additionally, views constructed over base tables can present the data in upper or lower case. Application users would then perform all operations against the views so that case does not become an issue.

For more information about the above case-sensitivity solutions in DB2 UDB, refer to the article, "Making DB2 Case-Insensitive" on the DB2 developerWorks Web site:

<http://www.ibm.com/developerworks/db2/library/techarticle/0203adamache/0203adamache.html>

4.6 SQL language syntax and semantics

This section highlights SQL language syntactical and semantic differences between SQL Server and DB2 UDB. Since both SQL Server and DB2 UDB conform to the same set of SQL standards, most functionality and syntax is portable between the two databases. Proprietary or nonstandard SQL Server syntax and features usually involves extra effort to convert to DB2 UDB.

4.6.1 SELECT statements

Typically, SQL Server SELECT statements do not require many modifications when converting to DB2 UDB. However, there are a few differences to be aware of during a conversion effort.

Unqualified columns

SQL Server permits the use of an unqualified column wildcard (*) alongside other elements in the SELECT clause list. DB2 UDB adheres to the SQL standard which states that a SELECT element that contains an unqualified column wildcard cannot contain anything else. DB2 UDB requires these elements to be qualified (a sequence of t1.*, t2.*, ... where t1, t2, ... are the tables in the FROM clause).

For example, the following query is valid in SQL Server, but not in DB2 UDB:

```
SELECT e.*, * FROM employee e, jobs j WHERE e.job_id = j.job_id
```

Because an unqualified column wildcard appears in the SELECT list alongside other elements, the query would be invalid in DB2 UDB. To convert this query, the wildcard column would need to be qualified, like the following modified query demonstrates:

```
SELECT e.*, j.* FROM employee e, jobs j WHERE e.job_id = j.job_id
```

Unspecified FROM clause

In SQL Server, the FROM clause of a SELECT statement is required except when the SELECT list contains only constants, variables, and arithmetic expressions (no column names). For example:

```
SELECT (4 * 5)
SELECT getdate()
```

In DB2 UDB, the FROM clause should always be specified as part of a SELECT statement. The SYSIBM.SYSDUMMY1 view can be used when a table or view name is not applicable. This view contains a single row. The above statements can be translated as:

```
SELECT (4 * 5) FROM SYSIBM.SYSDUMMY1
```

```
SELECT CURRENT TIMESTAMP FROM SYSIBM.SYSDUMMY1
```

Alternatively, the VALUES statement can be used instead of the SELECT statement, as the following examples show:

```
VALUES (4 * 5)
VALUES CURRENT TIMESTAMP
```

Variable assignments

SQL Server syntax for queries involving assignment to variables also differs from the DB2 UDB syntax. For example, the SQL Server statement:

```
SELECT @v_max=MAX(c1) FROM table1
```

corresponds to DB2 UDB's SELECT INTO statement:

```
SELECT MAX(c1) INTO v_max FROM table1
```

Special attention should be given to SELECT INTO statements in DB2 UDB since if the number of rows returned by the query is greater than one, DB2 UDB will raise an exception. The default behavior of SQL Server is to use the last row of the result set. When converting such statements to DB2 UDB, the FETCH FIRST 1 ROW ONLY clause should be used, for example:

```
SELECT c1 INTO v_c1 FROM table1 FETCH FIRST 1 ROW ONLY
```

COMPUTE

By including the COMPUTE clause in a SELECT statement, SQL Server generates totals that appear as additional summary columns at the end of the result set. When used in conjunction with the BY clause, control-breaks and subtotals are generated in the result set. For instance, the following SQL Server query generates an additional summary column for each department consisting of the sum of employee salaries for that department:

```
SELECT dept_id, salary
FROM staff order by dept_id COMPUTE sum(salary) BY dept_id
```

To convert this functionality to DB2 UDB, the UNION ALL of multiple SELECT statements can be used. The following query shows how the above SQL Server query can be converted:

```
SELECT CHAR(dept_id) dept_id, salary FROM staff
UNION ALL
SELECT CHAR(dept_id)||'Sum' dept_id, SUM(salary) salary
FROM staff GROUP BY dept_id
UNION ALL
SELECT 'Sum' dept_id, SUM(salary) salary FROM staff ORDER BY dept_id
```

Alternatively, it may also be possible to convert this functionality using DB2 UDB's ROLLUP() OLAP function, such as the following example demonstrates:

```
SELECT CHAR(dept_id) dept, salary, SUM(salary) AS sumdept
FROM STAFF
GROUP BY ROLLUP (dept, salary)
ORDER BY dept, sumdept
```

Column and table aliases

In SQL Server and DB2 UDB, column and table names in a SELECT statement can be re-named locally for that statement, albeit with different syntax.

The following SQL Server SELECT statement uses column alias names and table correlation names:

```
SELECT DepartmentNo = d.deptno,
       EmployeeNo = e.empno,
       EmployeeName = e.firstnme || ' ' || lastname
FROM   department d INNER JOIN employee e ON d.deptno = e.workdept
```

DB2 UDB syntax uses the **AS** keyword to define column aliases and table correlation names, as shown below:

```
SELECT d.deptno AS DepartmentNo,
       e.empno AS EmployeeNo,
       e.firstnme || ' ' || lastname AS EmployeeName
FROM   department AS d INNER JOIN employee AS e ON d.deptno = e.workdept
```

Alternatively, the AS keyword can also be omitted, as shown below:

```
SELECT d.deptno DepartmentNo,
       e.empno EmployeeNo,
       e.firstnme || ' ' || lastname EmployeeName
FROM   department d INNER JOIN employee e ON d.deptno = e.workdept
```

4.6.2 SELECT INTO

SQL Server's SELECT INTO statement is completely different than DB2 UDB's SELECT INTO statement. SQL Server's SELECT INTO statement is equivalent to a CREATE TABLE statement followed by an INSERT statement in DB2 UDB. Thus, the following query in SQL Server:

```
SELECT * INTO t2 FROM t1
```

is equivalent to the following statements in DB2 UDB:

```
CREATE TABLE t2 AS (SELECT t1.* FROM t1 ) DEFINITION ONLY
INSERT INTO t2 SELECT t1.* FROM t1
```

4.6.3 INSERT statements

Some INSERT statements cannot be directly translated from SQL Server to DB2 UDB because of the use of IDENTITY or ROWVERSION columns. For example, the following statements are valid in SQL Server, but not in DB2 UDB:

```
CREATE TABLE t1(c1 INT, c2 INT IDENTITY)
INSERT INTO t1 VALUES(1)
```

The second statement is not valid in DB2 UDB because the number of columns in t1 does not match the number of columns specified in the VALUES clause. DB2 UDB requires that the columns for which the values are intended are explicitly stated, as shown in the valid translation below:

```
INSERT INTO t1(c1) VALUES(1)
```

DEFAULT VALUES clause

In SQL Server, the DEFAULT VALUES clause can be specified in an INSERT statement to force the new row to contain the default values defined for each column. For example, assume the following table definition in SQL Server:

```
CREATE TABLE defaulttab (
  c1 INT IDENTITY,
  c2 VARCHAR(30)
  CONSTRAINT default_name DEFAULT ('column default'),
  c3 INT NULL)
```

The following INSERT statement inserts the values (1, column default, NULL) into a new row, for columns c1, c2, and c3, respectively.

```
INSERT INTO defaulttab DEFAULT VALUES
```

To achieve the same functionality in DB2 UDB, the default keyword must be specified as part of the INSERT statement for each column where the column's default value is to be used. Assuming the equivalent table definition in DB2 UDB:

```
CREATE TABLE defaulttab (
  c1 INT IDENTITY GENERATED ALWAYS BY DEFAULT,
  c2 VARCHAR(30) NOT NULL
  CONSTRAINT default_name DEFAULT ('column default'),
  c3 INT)
```

The above INSERT statement should be translated as:

```
INSERT INTO defaulttab(c1, c2, c3) VALUES (default, default, default)
```

in order to preserve the same functionality.

Attention: DB2 UDB attempts to use NULL as a default, if no default value is defined for a column. If a column is defined as NOT NULL and does not have a default value specified, an INSERT statement using the default keyword to supply that column's value will fail.

Alternatively, DB2 UDB tries to use a column's default value, if one exists, to supply a value for column not specified in an INSERT statement. For example, executing the following statement:

```
INSERT INTO defaulttab(c1, c3) VALUES (default, default)
```

inserts the values (1, column default, -) into table defaulttab. Notice how the default value for column c2 is used, even though it was not specified in the INSERT statement.

INSERT INTO <table_name> EXECUTE <procedure_name>

In SQL Server, an INSERT statement can be combined with an EXECUTE statement to invoke stored procedures on the same server or a remote server. The procedure must return data with SELECT or READTEXT statements. The procedure on the remote server is executed and the result sets are returned to the local server and loaded into the table on the local server. For example, the following SQL Server code creates a stored procedure, then invokes it as part of an INSERT statement:

```
CREATE PROCEDURE proc1
AS
SELECT * FROM table1
GO

INSERT INTO table1 EXECUTE proc1
GO
```

This code could be converted to DB2 UDB using two stored procedures in which one procedure invokes the other, receives the returned result set, then fetches each row from the returned result set and inserts it into the intended table. However, this may not be the best solution. In the example above, it is much simpler to duplicate this behavior using an INSERT statement with a sub-SELECT statement, like the following:

```
INSERT INTO table1 SELECT * FROM t1
```

Since there is no single technique to convert this feature to DB2 UDB, the actual technique used will depend on how/why this functionality is being used in your application.

4.6.4 UPDATE and DELETE statements

SQL Server allows more than one table to be specified in the FROM clause of UPDATE and DELETE statements. For example, the following DELETE statement is valid in SQL Server, but not in DB2 UDB:

```
DELETE t1 FROM t1, t2 WHERE t1.c1=t2.c1
```

This statement can be translated to DB2 UDB using a combination of a DELETE statement and SELECT statement:

```
DELETE FROM t1 WHERE EXISTS(SELECT * FROM t1,t2 WHERE t1.c1 = t2.c1)
```

In the case of UPDATE statements, non-determinism may make translation complex. Non-determinism arises when more than one row matches the join condition of the tables specified in the UPDATE statement. For example, suppose the following SQL Server UPDATE statement:

```
UPDATE t1 SET c1=t2.c1 FROM t1,t2 WHERE t1.c2=t2.c2
```

In this example, non-determinism arises when there is more than one matching row in t2, meaning any of the c1 values of these rows can be selected to do the update. If there is at most a single matching row in t2, the equivalent DB2 UDB statement is:

```
UPDATE t1 SET c1 = (SELECT DISTINCT t2.c1 FROM t2 WHERE t1.c2 = t2.c2)
WHERE EXISTS(SELECT * FROM t2 WHERE t1.c2 = t2.c2)
```

Attention: The only way to correctly simulate this behavior is by using a cursor-based implementation. Usually, such non-determinism is unintentional and should be corrected during a conversion to DB2 UDB.

DB2 UDB also supports retrieving result sets from SQL data-change operations (INSERT, UPDATE, and DELETE). SELECT and SELECT INTO statements can be used to retrieve result sets from SQL data-change operations embedded in the FROM clause of such statements. The following example illustrate this functionality:

```
SELECT name INTO p_name FROM NEW TABLE
(UPDATE staff SET salary = 10000.0 WHERE id = p_id)
```

This syntax is more efficient than issuing separate modification and retrieval statements, thus adopting this syntax is recommended where appropriate.

4.6.5 TRUNCATE TABLE

SQL Server's TRUNCATE TABLE statement provides a quick way to delete rows in a table. For example, the statement:

```
TRUNCATE TABLE employee
```

deletes all the rows from the employee table.

There are two ways to convert this statement to DB2 UDB:

- Using the DELETE statement.

In this case, the TRUNCATE TABLE statement above would be converted as:

```
DELETE FROM employee
```

It is important to note that the DELETE statement is logged and could therefore fail if the employee table is large, and the DELETE operation causes the log files to overflow.

An alternative method using the DELETE statement above is to create a stored procedure that deletes from the table in batches. Between batches, a COMMIT statement can be issued so that log files do not overflow.

- Using the NOT LOGGED INITIALLY clause of the ALTER TABLE statement.

If the table being deleted from was originally defined with the NOT LOGGED INITIALLY clause, there is a quick way to delete all rows from the table, by using the ALTER TABLE statement. The following ALTER TABLE statement purges all table data from the employee table referenced above without performing any transaction logging:

```
ALTER TABLE employee ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE
```

This is the preferred method of performing bulk deletes in DB2 UDB because of its high performance. However, only tables created with the NOT LOGGED INITIALLY clause can take advantage of this technique.

Note: At the completion of the current unit of work, the NOT LOGGED INITIALLY attribute is deactivated and all operations that are performed on the table in subsequent units of work are logged again.

4.6.6 ANSI joins

SQL Server supports two join syntaxes. The first is a proprietary syntax, which is not supported by DB2 UDB. This proprietary syntax uses the *= and =* operators in the WHERE clause to specify right and left outer joins, and is now only supported for backward compatibility. The second is the ANSI-style syntax.

The DB2 UDB syntax for joins is ANSI-style, with the operators:

- INNER
- LEFT [OUTER]
- RIGHT [OUTER]
- FULL [OUTER]

The ANSI join operators also follow the ANSI definitions for join behavior. The ANSI-style join syntax helps to avoid ambiguous interpretation when other conditions are specified in the WHERE clause.

Inner joins

The SQL Server syntax for inner joins is similar to the DB2 UDB syntax. The following example shows what the syntax for an inner join looks like in both SQL Server and DB2 UDB.

```
SELECT r.title, a.name
FROM authors a
INNER JOIN redbooks r
ON a.author_id = r.author_id
```

Outer joins

SQL Server supports the use of proprietary outer join operators (*= and =*) for backward compatibility. Table 4-3 shows how this proprietary syntax can be mapped to DB2 UDB supported syntax:

Table 4-3 Mapping proprietary outer join syntax to DB2 UDB

SQL Server	DB2 UDB
SELECT R.book_no, R.title ,A.name FROM redbooks R, authors A WHERE R.author_id *= A.author_id	SELECT R.book_no, R.title, A.name FROM redbooks R LEFT OUTER JOIN authors A ON A.author_id = R.author_id;
SELECT COUNT(R.book_id) amount, A.name FROM redbooks R, authors A WHERE R.author_id =* A.author_id GROUP BY A.name ORDER BY amount desc	SELECT COUNT(R.book_id) AS amount, A.name FROM redbooks R RIGHT OUTER JOIN authors A ON R.author_id = A.author_id GROUP BY A.name ORDER BY amount DESC;

If the ANSI style outer join syntax is already used, it is unlikely that changes will need to be made during a conversion to DB2 UDB.

Cross join

A cross join (also known as a cartesian product) produces every possible join permutation of the selected columns. The example in Table 4-4 translates to “every job is available in all departments”. In DB2 UDB, cross joins are specified by not including any join conditions in the WHERE clause of the SELECT statement.

Table 4-4 Mapping of cross join syntax to DB2 UDB

SQL Server	DB2 UDB
SELECT job_desc, dept_code FROM jobs CROSS JOIN departments	SELECT job_desc, dept_code FROM jobs INNER JOIN departments;

4.6.7 ORDER BY and GROUP BY clauses

SQL Server displays all NULL values at the beginning of the result set when the ORDER BY or GROUP BY clause is included in a SELECT statement. In DB2 UDB, NULL values of a column appear last when that column is ordered in ascending sequence, and first when ordered in descending sequence. Although DB2 UDB does not provide syntax for changing the sort order of NULL values, there is a way to obtain similar results by using the COALESCE() function to convert NULL to an empty string, which is at the opposite end of the collating sequence. Example 4-6 and Example 4-7 illustrate the use of this approach.

Example 4-6 SELECT statements with NULL values at the end of the result set

SELECT city FROM authors ORDER BY city;	-- [1]
SELECT city FROM authors ORDER BY city ASC;	-- [2]
SELECT COALESCE(city,'') FROM authors ORDER BY 1 DESC;	-- [3]
SELECT COALESCE(city,'') AS city FROM authors ORDER BY city DESC;	-- [4]

Example 4-7 SELECT statements with NULL values at the beginning of the result set

SELECT COALESCE(city,'') FROM authors ORDER BY 1 ASC;	-- [5]
SELECT COALESCE(city,'') AS city FROM authors ORDER BY city ASC;	-- [6]
SELECT city from authors ORDER BY city DESC;	-- [7]

Using the above approach, care must be taken because the application is returned an empty string, not a NULL value. If that must be avoided, the original column in the SELECT list can be kept and ordered on the COALESCE() form of it, as the following altered version of query [4] above illustrates:

```
SELECT city,  
       COALESCE(city,'') AS city_no_nulls  
FROM authors  
ORDER BY city_no_nulls DESC;
```

4.6.8 Top n clause

SQL Server's TOP n clause in a SELECT statement can be translated to DB2 UDB using the FETCH FIRST n ROWS clause:

```
SELECT *  
FROM redbooks
```

```
ORDER BY price ASC  
FETCH FIRST 5 ROWS ONLY
```

There is no equivalent in DB2 UDB for the TOP n PERCENT clause. To convert this functionality, customized cursor logic is required.

4.6.9 Cursors

SQL Server supports all ANSI-style cursors: static, dynamic, forward only, and keyset-driven. This includes support for INSENSITIVE and SCROLL cursor behavior and for all fetch options (FIRST, LAST, NEXT, PRIOR, RELATIVE, and ABSOLUTE). Cursor support is available through the following interfaces: ADO, OLE DB, ODBC, DB-Library, and T-SQL.

DB2 UDB supports static, forward-only, and scrollable cursors. There are two types of scrollable cursor: static and keyset-driven. The latter provides the ability to detect or make changes to the underlying data. Application support for static scrollable cursors is provided through DB2 CLI, ODBC, JDBC, SQLJ, and SQL PL. Keyset-driven scrollable cursors are supported through DB2 CLI and ODBC.

DB2 UDB has a few additional restrictions on cursor usage than SQL Server. For example, DB2 UDB's SQL PL procedural language does not support a cursor definition that includes more than one table specified in the FROM clause of the cursor's SELECT statement (columns to be updated all belong to the same table).

Transaction behavior

The default behavior in SQL Server is to have cursors remain open after a COMMIT or ROLLBACK statement is issued. This behavior strays from the ANSI SQL standard, although it is possible to configure SQL Server to use the ANSI SQL standard behavior of closing open cursors after a COMMIT or ROLLBACK statement is issued.

DB2 UDB's default behavior follows the ANSI SQL standard of closing open cursors whenever a COMMIT or ROLLBACK statement is issued. However, cursors that are declared with the WITH HOLD option remain open after a COMMIT statement is issued. In DB2 UDB, all open cursors are closed when a ROLLBACK statement is issued.

Note: In DB2 UDB, after a COMMIT statement has been issued, a cursor declared with the WITH HOLD statement remains open and is positioned before the next logical row of the results table. Additionally, all locks are released except for locks protecting the current cursor position.

DEALLOCATE

SQL Server provides a cursor clean-up command called DEALLOCATE which is mainly used as a performance benefit to eliminate the cost of declaring a new cursor once the last reference to the cursor is deallocated. The DEALLOCATE command allows a cursor to be closed and re-opened without re-declaring it.

DB2 UDB does not have a similar DEALLOCATE command. Cursors should be explicitly closed when they are no longer needed, and re-declared if and when they are needed again.

Note: As a recommended best practice in DB2 UDB, the SELECT statement of the cursor declaration should be unambiguous, that is, the clauses FOR UPDATE ONLY, FOR FETCH ONLY, and FOR READ ONLY should be included wherever possible. There are numerous performance benefits when these clauses are included. For example, they help the optimizer choose the best access path to retrieve the data as well as ensure that record blocking is used wherever possible.

Converting cursor attributes

SQL Server supports cursor attributes to obtain information about the current status of a cursor. SQLCODE/SQLSTATE values can be used to obtain the analogous information in DB2 UDB. This section shows how to match SQL Server cursor attribute functions with DB2 UDB SQLCODE/SQLSTATE values.

@@CURSOR_ROWS

In SQL Server, the @@CURSOR_ROWS function returns the number of qualifying rows from the previous cursor declaration. In DB2 UDB, a local (counter) variable can be used to store this information after each FETCH operation from the cursor. Example 4-8 shows how this can be implemented in DB2 UDB. In the example, a local variable is declared to hold the number of fetched rows [1] and is incremented each time a new row is fetched [2].

Example 4-8 DB2 UDB cursor with counting of fetched rows

```
DECLARE c1 CURSOR FOR
    SELECT author_id, name FROM authors;

DECLARE v_cursor_rows INTEGER DEFAULT 0;          -- [1]

LOOP
    FETCH c1 INTO my_ename, my_deptno;
    SET v_cursor_rows = v_cursor_rows + 1;        -- [2]
    ...
END LOOP;
```

@@FETCH_STATUS

In SQL Server, @@FETCH_STATUS returns the status of the last FETCH statement. Example 4-9 shows how to gather the status of a FETCH statement in DB2 UDB.

Example 4-9 DB2 UDB cursor processing with exception handling

```
DECLARE v_sqlstatus INTEGER DEFAULT 0;

DECLARE c1 CURSOR FOR                                -- [1]
    SELECT author_id, name FROM authors;

DECLARE CONTINUE HANDLER FOR NOT FOUND              -- [2]
    SET v_sqlstatus = -1;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION           -- [3]
    SET v_sqlstatus = -1;

OPEN c1;
WHILE 0 = v_sqlstatus DO                             -- [4]
    FETCH c1 INTO ...
    ...
END WHILE

CLOSE c1;
```

It should be noted that DB2 UDB throws an exception when a warning or an error is encountered during processing. In the variable declaration part of the code [1], a CONTINUE HANDLER is defined for the case that no data is found [2] and another CONTINUE HANDLER is defined [3] for any other exception that arises. In both CONTINUE HANDLERS, a flag variable is set, which is checked in the WHILE loop condition [4]. If the flag is set by one of the CONTINUE HANDLERS, cursor processing (the WHILE loop processing) ceases.

4.7 Built-in SQL functions

There are differences between the SQL functions supported in SQL Server and DB2 UDB. Some of the functions exist with the same name and functionality in both platforms; however, it is important to verify that the semantics of the functions' implementations are similar under both platforms.

A second class of SQL functions exist under both platforms, but are named differently (e.g., CHARINDEX() for SQL Server, which is equivalent to LOCATE() for DB2 UDB). Converting these functions can be done either by modifying the actual function calls in the application code, or by creating a sourced user defined function (UDF). A sourced UDF is defined using a reference to another

previously defined function. This can be extremely useful for extending functionality of built-in functions to user-defined data types or for creating a new function with the same functionality as an existing function. In Example 4-10, a new UDF, `CHARINDEX()`, is created and based upon the `LOCATE()` function.

Example 4-10 Sourced user defined function called CHARINDEX in DB2 UDB

```
CREATE FUNCTION CHARINDEX(  VARCHAR(4000),
                           VARCHAR(4000),
                           INTEGER)

RETURNS INTEGER
SOURCE LOCATE(VARCHAR(4000), VARCHAR(4000), INTEGER)
!
```

Finally, there are some functions which have no direct equivalent in DB2 UDB (e.g., `DATEADD()` for SQL Server). This set of functions can usually be converted to DB2 UDB using the following techniques:

- ▶ SQL UDFs

SQL UDFs are the first choice for converting unsupported SQL functions. Using DB2 UDB's SQL/PL language to create and duplicate SQL Server function behavior is usually quite successful.

- ▶ External UDFs

In situations where the desired functionality cannot be accomplished using SQL PL alone, external UDFs can be used. This involves duplicating the required behavior in a supported compiled language such as Java or C.

- ▶ Stored procedures

In some cases where advanced functionality is required, using a stored procedure instead of a function is the most appropriate method of duplicating the desired functionality if a table function cannot be used.

Refer to Appendix D, "Operator mapping" on page 495 for a complete mapping of SQL functions in SQL Server to DB2 UDB.

4.8 System catalog queries

In SQL Server, metadata is stored in the master database and is accessible using a special schema called `INFORMATION_SCHEMA`.

In DB2 UDB, each database contains its own metadata in a set of base tables and views called the *catalog*. The catalog contains information about the logical and physical structure of the database objects, object privileges, integrity information, etc.

The DB2 UDB database catalog is automatically created when a database is created. The tables and views in the catalog belong to the SYSCATSPACE table space. The catalog is divided into three schemas:

- ▶ **SYSIBM**
The SYSIBM schema contains all base tables
- ▶ **SYSCAT**
The SYSCAT schema contains read-only views pertaining to object information
- ▶ **SYSSTAT**
The SYSSTAT schema contains updateable views that include statistical information used during query optimization

The DB2 UDB system catalog views cannot be modified using traditional SQL statements. They are automatically updated each time a SQL data definition statement is executed.

Due to the differing product architectures, querying the system for information is different in SQL Server and DB2 UDB. However, the following section contains several examples that demonstrate how specific information can be retrieved from the DB2 UDB system catalog. More detailed information about the system catalog contents can be found in the DB2 UDB documentation *SQL Reference - Volume 1*, SC09-4844.

4.8.1 Catalog interrogation and instance/database metadata

To retrieve metadata from the catalog, a SELECT statement can be issued or can be retrieved using Control Center. In DB2 UDB, a connection to a database must be established prior to querying the database's catalog. For example, to obtain information about the table departments in the REDBOOK database, the following query can be issued:

```
SELECT SUBSTR(colname,1,18) AS Column_Name
      ,colno
      ,SUBSTR(typename,1,18) AS Type_Name
      ,scale
      ,length
      ,codepage
FROM SYSCAT.COLUMNS
WHERE tabname = TRANSLATE('departments');
```

The result of the query is:

COLUMN_NAME	COLNO	TYPE_NAME	SCALE	LENGTH	CODEPAGE
DEPT_ID	0	INTEGER	0	4	0
DEPT_CODE	1	CHARACTER	0	4	1252
DIVISION	2	CHARACTER	0	3	1252

NAME	3 VARCHAR	0	40	1252
LOCATION	4 VARCHAR	0	15	1252

With the DB2 Command Line Processor (CLP), it is possible to obtain database and instance metadata as well. The most common commands used for this are:

- DB2 GET ...
- DB2 LIST ...

To obtain a list of all databases that are cataloged in your environment, the following command can be entered in a CLP window:

```
DB2 LIST DATABASE DIRECTORY
```

In SQL Server, if an application needs to know if a specific database exists, this information can be obtained from the master database. Example 4-11 shows how to obtain this information using a stored procedure.

Example 4-11 SQL Server procedure to check for database existence

```
CREATE PROCEDURE exists_database
    @database_name sysname
AS
    DECLARE @result int
    SET NOCOUNT ON

    SELECT @result = COUNT(*)
    FROM MASTER.DBO.SYSDATABASES
    WHERE NAME = @database_name

    SELECT @result AS DB_EXIST

    SET NOCOUNT OFF

    RETURN @result
GO
```

In DB2 UDB, the above functionality can be implemented using the DB2 UDB SNAPSHOT functionality. In Example 4-12, a user defined function and a stored procedure is provided for this purpose. The procedure requests a database SNAPSHOT and attempts to access the information returned by the snapshot.

Example 4-12 DB2 UDB function and procedure to check for database existence

```
CREATE PROCEDURE exists_db ( IN db_name VARCHAR(8),
                           OUT db_exists INTEGER)
LANGUAGE SQL
READS SQL DATA
BEGIN
    DECLARE p_token_string VARCHAR(255);
```

```

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
BEGIN
    GET DIAGNOSTICS
        EXCEPTION 1
        p_token_string = DB2_TOKEN_STRING;

    IF (p_token_string =
        'SYSPROC.SNAPSHOT_DATABASE_SNAPSHOT_DATABASE_SQL1611') THEN
        SET db_exists = 1;
    ELSE
        SET db_exists = 0;
    END IF;

END; -- cont. handler

SET db_exists = (
    SELECT COUNT(*)
    FROM TABLE(SNAPSHOT_DATABASE(db_name,0)) AS temp_tab);
END
!

CREATE FUNCTION exists_database(v_db_name VARCHAR(128))
RETURNS INTEGER
LANGUAGE SQL
NO EXTERNAL ACTION
READS SQL DATA
BEGIN ATOMIC
    DECLARE v_rc integer DEFAULT 0;

    CALL exists_db(v_db_name, v_rc);

    RETURN v_rc;
END
!

```

With the function `exists_database()`, it is now possible to check for database existence, as illustrated in the following statement:

```
VALUES exists_database('redbook')
```

For more information about the `GET DIAGNOSTICS` and `SNAPSHOT` commands, refer to the DB2 UDB documentation *SQL Reference - Volume 2*, SC09-4845.

4.8.2 System table mapping

Many of the SQL Server *system tables* can be mapped to DB2 UDB *catalog tables*. There are some minor differences in table names and contents. Table 4-5

lists the most common SQL Server system tables as well as the corresponding catalog tables in DB2 UDB.

Table 4-5 Mapping of SQL Server system tables to DB2 UDB catalog tables

SQL Server system table	DB2 catalog table
n/a	SYSCAT.BUFFERPOOLS
n/a	SYSCAT.TABLESPACES
INFORMATION_SCHEMA.tables	SYSCAT.TABLES
INFORMATION_SCHEMA.columns	SYSCAT.COLUMNS
INFORMATION_SCHEMA.views	SYSCAT.VIEWS
n/a	SYSCAT.TRIGGERS
sysindexes	SYSCAT.INDEXES
INFORMATION_SCHEMA.routines	SYSCAT.SYSPROCEDURES
INFORMATION_SCHEMA.routines	SYSCAT.FUNCTIONS
systypes	SYSCAT.DATATYPES
sysreferences	SYSCAT.REFERENCES

4.8.3 Authorizations on system catalog tables

During database creation, SELECT privilege on the system catalog views is granted to PUBLIC. In most cases, this does not present any security problems. For very sensitive data, however, it may be inappropriate, as these tables describe every object in the database. If this is the case, consider revoking the SELECT privilege from PUBLIC; then grant the SELECT privilege as required to specific users. Granting and revoking SELECT privilege on the system catalog views is done in the same way as standard views, but SYSADM or DBADM authority is required. For example, to revoke the SELECT privilege on the SYSCAT.TABLES view from the PUBLIC group, the following statement can be issued:

```
REVOKE SELECT ON SYSCAT.TABLES FROM PUBLIC
```

Control Center can also be used to grant/revoke privileges on database objects. At a minimum, consider restricting access to the following catalog views:

- ▶ SYSCAT.DBAUTH
- ▶ SYSCAT.TABAUTH
- ▶ SYSCAT.PACKAGEAUTH
- ▶ SYSCAT.INDEXAUTH
- ▶ SYSCAT.COLAUTH

- ▶ SYSCAT.PASSTHRUAUTH
- ▶ SYSCAT.SCHEMAAUTH

4.9 Transact-SQL to SQL PL translation

Transact SQL (T-SQL) is SQL Server's extension to the ANSI SQL language. T-SQL is a dynamic database programming language. All SQL Server stored procedures, user defined functions, triggers, and SQL statements are coded using T-SQL, although not every T-SQL statement is supported in each of these database objects.

SQL PL is DB2 UDB's procedural programming language and is a subset of the ANSI SQL Persistent Stored Modules (SQL/PSM) language standard. This standard forms the basis for creating stored procedures and functions combining procedural language statements with the power of the SQL language. In DB2 UDB, only stored procedures support the advanced elements of the SQL PL. Triggers, user defined functions, and stand-alone code (SQL PL scripting) use inline SQL PL. Inline SQL PL is a procedural language that supports a subset of the SQL PL elements. As such, certain SQL PL elements that are supported in SQL stored procedures are not supported in triggers, UDFs, and stand-alone SQL PL scripts.

Since both T-SQL and SQL PL are extensions of the same standard, many of the same features are available in both languages, although, sometimes with differing syntax. Proprietary extensions are usually what cause difficulty when converting from T-SQL to SQL PL, since there is not always a direct translation from one language to the other. Table 4-6 shows an equivalency between T-SQL and SQL PL language elements and statements.

Table 4-6 Mapping of common T-SQL programming constructs to SQL PL

T-SQL	SQL PL
DECLARE @varname datatype = defaultvalue	DECLARE varname datatype DEFAULT defaultvalue;
SELECT @var1=value	SET var1 = value;
SELECT @var1=colname FROM table WHERE...	SET var1 = (SELECT colname FROM table WHERE...);
SELECT @v1=col1,@v2=col2,@v3=col3 FROM table...	SELECT col1,col2,col3 INTO v1,v2,v3 FROM table...
SELECT column_alias = column_name, ...	SELECT column_name AS column_alias, ...

T-SQL	SQL PL
WHILE expression BEGIN ... END	WHILE expression DO ... END WHILE;
CONTINUE	ITERATE
BREAK	LEAVE loop_label
IF (...) BEGIN ... END ELSE ...	IF (...) THEN ... ELSE ... END IF;
EXECUTE ('INSERT INTO t1 VALUES(2)')	INSERT INTO t1 VALUES (2); OR EXECUTE IMMEDIATE ('INSERT INTO t1 VALUES(2)');
EXECUTE procname(parm1,parm2,...)	CALL procname(parm1,parm2,...);
EXECUTE @retval=procname(parm1,parm2,...)	CALL procname(parm1,parm2,...); GET DIAGNOSTICS retval = RETURN_STATUS;
RETURN <int_value>	RETURN < int_expr>;
GOTO <label>	GOTO <lable>
PRINT	See below for more details.
RAISERROR	See below for more details.
@@ERROR	See below for more details.
@@SQLSTATUS	See below for more details.
@@ROWCOUNT	See below for more details.
@@TRANCOUNT	See below for more details.

4.9.1 EXECUTE <string>

The SQL Server EXECUTE statement followed by a string argument is converted differently depending on its context.

- If the statement occurs at the top level and the argument is a string literal, then it should be translated directly as though the contents of the string literal had occurred at the top level.
- If the statement occurs inside of a stored procedure, it can be translated to DB2 UDB using a static SQL statement or an EXECUTE IMMEDIATE statement (dynamic SQL). A static SQL statement is the preferred option when the full SQL statement is known or no parameter markers are used. Dynamic SQL is the preferred choice when the full SQL statement is not known at compile time.

Example 4-13 shows a simple SQL Server stored procedure using the EXECUTE statement to issue a SQL statement.

Example 4-13 SQL Server stored procedure using an EXECUTE statement

```
CREATE PROCEDURE p1
AS
    EXECUTE ('INSERT INTO t1 VALUES (3)')
GO
```

Example 4-14 shows the equivalent DB2 UDB procedure declaration.

Example 4-14 DB2 UDB stored procedure using an EXECUTE IMMEDIATE

```
CREATE PROCEDURE p1()
LANGUAGE SQL
MODIFIES SQL DATA
BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';
    DECLARE execStr VARCHAR(4000);
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET l_error = '00000';

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
    BEGIN
        SET l_error = SQLSTATE;
        IF SUBSTR(l_error, 1, 1) >= '5' AND
            SUBSTR(l_error, 1, 1) <= '9' THEN
            RESIGNAL;
        END IF;
    END;

    INSERT INTO t1 VALUES (3);
END
!
```

- If the statement is invoking a stored procedure, it should be translated to DB2 UDB using the CALL statement. Care must be taken to ensure that all parameters are properly specified. If the statement is invoking a function, it should be translated to DB2 UDB using the VALUES clause.

4.9.2 PRINT

In SQL Server, a PRINT statement is used to display a character string to the user. It is typically used during development to print debugging statements, but can also be used for logging and as a quick way to notify the user. PRINT is different

than the SELECT statement in that it returns a message of severity 0, the SELECT statement returns a set of data.

There is no direct equivalent to the PRINT statement in DB2 UDB. Depending on the reason it is being used, a work-around in DB2 UDB is possible. For example, if the PRINT statement is being used to log the progress of an operation to the screen, it is possible to simulate this in DB2 UDB using an external user defined function by logging results to a file instead.

A DB2 UDB user defined function called PUT_LINE() that enables file output from pure SQL is provided as additional material with this redbook. This function has similar functionality as the PRINT statement. It can be used as a tool for debugging stored procedures, but also allows any messages to be written to a specified file for other purposes.

The function code sources and an installation guide can be found on the IBM Redbooks Internet site. For more details, see Appendix G, "Additional material" on page 509.

4.9.3 @@ERROR

In T-SQL, the execution of every statement as well as the evaluation of any IF condition has the effect of setting the @@ERROR system function to a success or failure value. Execution continues to the next statement. It is the programmer's responsibility to check if the execution of any statement failed and to take appropriate action. Only fatal errors cause the transaction to be rolled back.

In SQL PL, an error encountered during the processing of any SQL statement will result in an exception being raised. If the exception is not caught by a specific exception handler, the application is terminated. The error status codes for the specific error can be retrieved from either of the SQLCODE and SQLSTATE system variables.

The most direct way of converting @@ERROR to SQL PL is to create a default exception handler in every stored procedure as shown in Example 4-15.

Example 4-15 DB2 UDB stored procedure with a default exception handler

```
CREATE PROCEDURE proc_name (OUT returncode INTEGER)
LANGUAGE SQL
BEGIN
    DECLARE SQLCODE INTEGER DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING, NOT FOUND
        SET returncode = SQLCODE;

    -- set default return value
    SET returncode = 0;
```

```
-- executable-statements  
END  
!
```

This CONTINUE handler catches all exceptions and copies the SQLCODE error value into a local variable, which is returned to the calling application. The exception handler should be declared as a CONTINUE handler so that execution continues with the statement following the statement that raised the exception.

Evaluation of @@ERROR in a condition

@@ERROR is reset by the evaluation of all statements and predicates (e.g., an IF condition). Sometimes, existing T-SQL code contains faulty error processing logic in which the programmer forgot that the value of @@ERROR was reset by a previous statement or condition. For example, in the following pseudo-code sequence the evaluation of the first IF condition causes @@ERROR to be reset to 0, hence the second IF condition will always be false:

```
insert_statement;  
if (@@ERROR = error_value)  
    error_handling_logic  
else if (@@ERROR != 0)  
    more_error_handling_logic
```

This is not likely what the original programmer intended. When converting @@ERROR to DB2 UDB, ensure that the original logic is correct and understand how errors are processed in SQL PL using exception handlers. Refer to Example 4-15 to see how @@ERROR function behavior can be converted to SQL PL using exception handlers.

Exceptions during a condition

In T-SQL, if an error occurs during the evaluation of the IF branch of an IF-THEN-ELSE condition, the condition evaluates to false and the ELSE branch of the condition is executed. In SQL PL, an exception is raised and if an exception handler has been defined, will catch the exception. Once the error processing logic is complete, statement execution continues with the statement following the IF-THEN-ELSE statement.

There is no direct way to convert this behavior to SQL PL. Rather than simulate this type of logic in DB2 UDB, we recommend that you re-design the code where this type of behavior occurs (errors in IF condition evaluation) and adopt an SQL PL style of error-handling behavior and logic.

System errors that correspond to a value of @@ERROR

When @@ERROR is tested after a procedure call and the error value corresponds to a system generated error value, ensure that the SQL exception raised by the most recent statement in DB2 UDB is propagated to the calling statement. Sometimes, the exception is accidentally caught by a more local exception handler. Ensure the most recently executed statement is outside the block (BEGIN...END) containing the local handler, or continue propagating the statement until it reaches the top level.

4.9.4 RAISERROR

A RAISERROR statement is equivalent to a PRINT statement, followed by an assignment to @@ERROR. It should be carefully converted to the appropriate SQL PL statements depending on its context:

- ▶ A RAISERROR statement that is immediately followed by a RETURN statement can be converted using the SQL PL command SIGNAL SQLSTATE.
- ▶ The commonly used sequence RAISERROR; ROLLBACK; RETURN should be converted as though it were the sequence ROLLBACK; RAISERROR; RETURN, since the @@ERROR value is reset to zero by the ROLLBACK statement in the former.
- ▶ Care must be taken when converting the sequence RAISERROR; RETURN<some_value>. This sequence causes @@ERROR to be reset to zero before returning from the procedure. When converting to SQL PL, RAISERROR might simply be ignored in this context. Translating the sequence into SQL PL's SIGNAL SQLSTATE will also not work because the procedure will not be able to return the <some_value>.

4.9.5 @@SQLSTATUS

The T-SQL system function @@SQLSTATUS is similar to @@ERROR except that it is only relevant for cursor FETCH statements. @@SQLSTATUS indicates if a FETCH operation was successful, if there was no data, or if there was some other error.

To translate @@SQLSTATUS to SQL PL, the value of SQLCODE can be checked after each FETCH operation, as shown in Example 4-16:

Example 4-16 DB2 UDB cursor with SQLCODE check

```
DECLARE cur1 CURSOR FOR
    SELECT author_id, name FROM authors;

OPEN cur1;

FETCH cur1
    INTO v_id, v_name;
```

```

WHILE SQLCODE <> 100 DO
  -- executable-statements
  FETCH cur1
    INTO v_id, v_name;
END WHILE;

```

4.9.6 @@ROWCOUNT

The T-SQL system function @@ROWCOUNT indicates the number of rows selected or affected by the previous statement.

The equivalent statement in SQL PL is the GET DIAGNOSTICS statement. This statement can be used to retrieve the number of rows affected by an INSERT, UPDATE, or DELETE statement and to estimate the number of rows expected as a result of a PREPARE statement. Example 4-17 illustrates the use of the GET DIAGNOSTICS statement.

Example 4-17 DB2 UDB logic using the GET DIAGNOSTICS statement

```

...
DECLARE rc INT DEFAULT 0;
INSERT INTO brands (brand_id, brand_name, brand_desc)
  VALUES (1, 'DB2', 'DB2 Information Management');

GET DIANOSTICS rc = ROW_COUNT;

IF ( rc > 0 )
  THEN RETURN 0;
ELSE
  RETURN -200;
END IF;
...

```

4.9.7 @@TRANCOUNT

Transactions are handled differently in SQL Server and DB2 UDB. In SQL Server, modifications to the database are committed immediately, unless a transaction is explicitly initiated with a BEGIN TRANSACTION statement.

In DB2 UDB, every statement is implicitly part of a transaction. A transaction is initiated with the first statement after a successful database connection, or after a COMMIT or ROLLBACK statement is issued.

In T-SQL, the execution of a COMMIT TRANSACTION statement does not necessarily cause the current transaction to be committed. The time at which the changes

are actually committed depends on the value of the system function @@TRANCOUNT. This function returns the depth of executed BEGIN TRANSACTION blocks. @@TRANCOUNT is initially set to zero and is incremented by one each time a BEGIN TRANSACTION statement is issued. A COMMIT TRANSACTION statement issued when the value of @@TRANCOUNT is zero has no effect. If the value of @@TRANCOUNT is greater than zero when the COMMIT TRANSACTION statement is issued, the value of @@TRANCOUNT is decremented by one. If the resulting value after decrementing is zero, only then is the transaction committed. This implies that only the outermost COMMIT statement of a nested transaction block will actually commit the data. The effect of a ROLLBACK statement is to undo the current transaction and to reset the value of @@TRANCOUNT to zero.

There is no direct equivalent to the @@TRANCOUNT function in SQL PL, since transactions are handled differently in DB2 UDB. Converting the @@TRANCOUNT function to DB2 UDB essentially amounts to ensuring that transactional statements including COMMIT and ROLLBACK are included in the appropriate places in the SQL PL code. For instance, in T-SQL, database updates that are performed when @@TRANCOUNT is zero are not part of any transaction, hence a COMMIT statement should be issued in SQL PL immediately after such updates. COMMIT statements should also be added to the SQL PL code where it is known that the value of @@TRANCOUNT is guaranteed to be one. Nested COMMIT statements in T-SQL code can often be removed when converting to SQL PL since the value of @@TRANCOUNT is not zero after the decrement operation, and hence the data is not actually committed at that point.

4.10 XML

Many applications exchange data as XML but store and query data using a relational database. This presents an added challenge to application developers and database administrators: how to manage traditional relational data and the hierarchical data structure presented by XML. Microsoft uses the FOR XML option as part of the SELECT statement to retrieve results in XML. With the FOR XML option, application developers can execute SQL queries against existing relational databases and return results as XML data, rather than as a standard sets of rows (tables).

SQL/XML, supported by DB2 UDB, is an emerging part of the ANSI and ISO SQL standard. As such, a database system like DB2 UDB aims to provide application developers with a set of publishing functions (integrated in SQL) to receive query results in XML. Using the SQL/XML standard instead of custom-coded solutions or proprietary vendor extensions greatly simplifies development, provides for more maintainable code, and allows portability across database systems.

In addition to generating XML data from relational data sources, some applications need to consume XML and store relevant information in relational data content stores. OPENXML is Microsoft's solution for this requirement. OPENXML provides a row set view of an XML document, meaning a result can be used as a source for INSERT, DELETE, or UPDATE operations against the database.

One approach to convert OPENXML to DB2 UDB is to use the DB2 XML Extender. The XML Extender provides the capability to shred an XML document into a set of relational tables. Once the XML document is shredded into relational tables, standard relational data operations can be performed.

4.10.1 FOR XML statement

SQL Server has a set of XML-related features built-in to the database engine. It allows application developers to publish relational data as XML or populate and update relational information with data retrieved from an XML document. SQL queries executed against existing relational databases can return results as XML data rather than standard row sets. The FOR XML option of the SELECT statement is used to retrieve results in XML. The basic syntax of the FOR XML clause is:

```
FOR XML mode [, XMLDATA] [, ELEMENTS][, BINARY BASE64]
```

Conversion examples

The FOR XML clause allows three different modes, RAW, AUTO, and EXPLICIT, to be used. This section contains examples to show how the FOR XML clause can be translated to DB2 UDB in each of the different modes. The examples are based on the following three table definitions, taken from the DB2 UDB SAMPLE database. They assume the following DB2 UDB tables also exist in SQL Server, with the appropriate corresponding data types.

Department Table

```
CREATE TABLE DEPARTMENT (  
  DEPTNO      CHAR(3) NOT NULL,  
  DEPTNAME    VARCHAR(29) NOT NULL,  
  MGRNO       CHAR(6),  
  ADMRDEPT    CHAR(3) NOT NULL,  
  LOCATION    CHAR(16));
```

Employee Table

```
CREATE TABLE EMPLOYEE (  
  EMPNO       CHAR(6) NOT NULL,  
  FIRSTNAME   VARCHAR(12) NOT NULL,  
  MIDINIT     CHAR(1) NOT NULL,  
  LASTNAME    VARCHAR(15) NOT NULL,
```

```

WORKDEPT  CHAR(3),
PHONENO   CHAR(4),
HIREDATE  DATE,
JOB       CHAR(8),
EDLEVEL   SMALLINT NOT NULL,
SEX       CHAR(1),
BIRTHDATE DATE,
SALARY    DECIMAL(9,2),
BONUS     DECIMAL(9,2),
COMM      DECIMAL(9,2));

```

Project Table

```

CREATE TABLE PROJECT (
  PROJNO  CHAR(6) NOT NULL,
  PROJNAME VARCHAR(24) NOT NULL,
  DEPTNO   CHAR(3) NOT NULL,
  RESPEMP  CHAR(6) NOT NULL,
  PRSTAFF  DECIMAL(5,2),
  PRSTDATE DATE,
  PRENDATE DATE,
  MAJPROJ  CHAR(6));

```

FOR XML AUTO example

Example 4-18 illustrates a SQL Server query in which the FOR XML AUTO mode is used. This query lists all employees in each department.

Example 4-18 SQL Server query using the FOR XML AUTO clause

```

SELECT  department.deptno,
        department.deptname,
        employee.empno,
        employee.firstnme,
        employee.lastname
FROM    dbo.department department,
        dbo.employee employee
WHERE   department.deptno = employee.workdept
ORDER BY department.deptno
FOR XML AUTO

```

Example 4-19 shows the partial XML output from Example 4-18:

Example 4-19 XML output for query using FOR XML AUTO

```

<department deptno="A00" deptname="SPIFFY COMPUTER SERVICE DIV.">
<employee empno="000010" firstnme="CHRISTINE" lastname="HAAS"/>
<employee empno="000110" firstnme="VINCENZO" lastname="LUCCHESSI"/>
<employee empno="000120" firstnme="SEAN" lastname="O&apos;CONNELL"/>
</department>

```

```

<department deptno="B01" deptname="PLANNING">
<employee empno="000020" firstnme="MICHAEL" lastname="THOMPSON"/>
</department>

<department deptno="C01" deptname="INFORMATION CENTER">
<employee empno="000030" firstnme="SALLY" lastname="KWAN"/>
<employee empno="000130" firstnme="DOLORES" lastname="QUINTANA"/>
<employee empno="000140" firstnme="HEATHER" lastname="NICHOLLS"/>
</department>

<department deptno="D11" deptname="MANUFACTURING SYSTEMS">
<employee empno="000150" firstnme="BRUCE" lastname="ADAMSON"/>
<employee empno="000160" firstnme="ELIZABETH" lastname="PIANKA"/>
<employee empno="000170" firstnme="MASATOSHI" lastname="YOSHIMURA"/>
<employee empno="000180" firstnme="MARILYN" lastname="SCOUTTEN"/>
<employee empno="000060" firstnme="IRVING" lastname="STERN"/>
<employee empno="000190" firstnme="JAMES" lastname="WALKER"/>
<employee empno="000200" firstnme="DAVID" lastname="BROWN"/>
<employee empno="000210" firstnme="WILLIAM" lastname="JONES"/>
<employee empno="000220" firstnme="JENNIFER" lastname="LUTZ"/>
</department>

```

Note how in the SELECT list in Example 4-18, the deptno and deptname labels identify the department table. Therefore, a <department> element is created and deptno and deptname are added as its attributes. Next, the empno, firstnme and lastname column names identify the employee table. An <employee> element is added as a sub-element of <department>, and the empno, firstnme, and lastname attributes are added to the <department> element.

To produce a similar result in DB2 UDB, a SELECT statement can be used as shown in Example 4-20:

Example 4-20 The DB2 UDB equivalent statement of Example 4-18

```

SELECT      xml2clob(XMLELEMENT(name "department",
                                XMLATTRIBUTES( d.deptno AS "deptno",
                                                d.deptname AS "deptname"),
                                XMLAGG (XMLELEMENT (name "employee",
                                                XMLATTRIBUTES( e.empno AS "empno",
                                                                e.firstnme AS "firstnme",
                                                                e.lastname AS "lastname")))))
FROM        enterprise.department d, enterprise.employee e
WHERE       d.deptno = e.workdept
GROUP BY    d.deptno, d.deptname

```

The key to performing this type conversion is to produce the same nesting as SQL Server. When various FOR XML clauses are translated with DB2 SQL/XML queries, XMLAGG() is used to produce the nesting. In the above example,

`XMLAGG()` is used to list all `<employee>` elements under a particular department. `XMLAGG()` resolves the *1:n* relationships in XML.

Example 4-21 shows a more complex example of using the `FOR XML AUTO` mode in SQL Server. In this example, the result set contains all employees in each department along with all projects assigned to each employee.

Example 4-21 Complex example of SQL Server's FOR XML AUTO clause

```
SELECT    department.deptno,
          department.deptname,
          employee.empno,
          employee.firstname,
          employee.lastname,
          project.projno,
          project.projname
FROM      dbo.department department,
          dbo.employee employee,
          dbo.project project
WHERE     department.deptno=employee.workdept AND
          employee.empno=project.respemp
ORDER BY  department.deptno
FOR XML AUTO
```

Example 4-22 shows the partial XML output from Example 4-21:

Example 4-22 XML output from Example 4-21

```
<department deptno="A00" deptname="SPIFFY COMPUTER SERVICE DIV.">
<employee empno="000010" firstname="CHRISTINE" lastname="HAAS">
<project projno="AD3100" projname="ADMIN SERVICES"/>
<project projno="MA2100" projname="WELD LINE AUTOMATION"/>
</employee>
</department>

<department deptno="B01" deptname="PLANNING">
<employee empno="000020" firstname="MICHAEL" lastname="THOMPSON">
<project projno="PL2100" projname="WELD LINE PLANNING"/></employee>
</department>

<department deptno="C01" deptname="INFORMATION CENTER">
<employee empno="000030" firstname="SALLY" lastname="KWAN">
<project projno="IF1000" projname="QUERY SERVICES"/>
<project projno="IF2000" projname="USER EDUCATION"/>
</employee>
</department>

<department deptno="D11" deptname="MANUFACTURING SYSTEMS">
<employee empno="000060" firstname="IRVING" lastname="STERN">
<project projno="MA2110" projname="W L PROGRAMMING"/>
</employee>
```

```

<employee empno="000150" firstnme="BRUCE" lastname="ADAMSON">
<project projno="MA2112" projname="W L ROBOT DESIGN"/>
</employee>
<employee empno="000160" firstnme="ELIZABETH" lastname="PIANKA">
<project projno="MA2113" projname="W L PROD CONT PROGS"/>
</employee>
<employee empno="000220" firstnme="JENNIFER" lastname="LUTZ">
<project projno="MA2111" projname="W L PROGRAM DESIGN"/>
</employee>
</department>

```

In this example, a third table, project, becomes part of the SELECT clause, as identified by the projno and projname columns in the SELECT list. Therefore, a <project> element is created and projno and projname attributes are included. In the previous example, <department> is the top level element. The <employee> element is nested under <department>. In this example, a new <project> element is introduced, and is nested under <employee>.

To produce a similar result in DB2 UDB, multiple sub-SELECT statements are used, as shown in Example 4-23:

Example 4-23 DB2 UDB equivalent statement of Example 4-21

```

SELECT      xml2clob(XMLELEMENT(name "department",
                                XMLATTRIBUTES ( w.deptno AS "deptno",
                                                  w.deptname AS "deptname"),
                                emp))
FROM
  ( SELECT    deptno,
              deptname,
              XMLAGG ( XMLELEMENT (name "employee",
                                XMLATTRIBUTES( v.empno AS "empno",
                                                  v.firstnme AS "firstnme",
                                                  v.lastname AS "lastname"),
                                emp))
    FROM
      ( SELECT    t.deptno, t.deptname, t.empno, t.firstnme, t.lastname,
                  XMLAGG (XMLELEMENT (name "project",
                                XMLATTRIBUTES( t.projno AS "projno",
                                                  t.projname AS "projname"))))
    FROM
      ( SELECT    department.deptno,
                  department.deptname,
                  employee.empno,
                  employee.firstnme,
                  employee.lastname,
                  project.projno,

```

```

        project.projname
FROM    enterprise.department department,
        enterprise.employee employee,
        enterprise.project project
WHERE   department.deptno=employee.workdept AND
        employee.empno=project.respemp
ORDER BY department.deptno, employee.empno, project.projno
) AS t (deptno, deptname, empno, firstnme, lastname, projno, projname)
GROUP BY deptno, deptname, empno, firstnme, lastname)
AS v (deptno, deptname, empno, firstnme, lastname, emp)
GROUP BY deptno, deptname
) AS w (deptno, deptname, emp)

```

As previously mentioned, the key to performing this type of conversion is to preserve the nesting produced by SQL Server. This conversion technique involves the use of multiple sub-SELECT statements. In order to mimic the hierarchical structure of the XML document, each sub-SELECT should correspond to a child layer within the XML document.

The first SELECT statement is constructed as usual:

```

SELECT   department.deptno,
        department.deptname,
        employee.empno,
        employee.firstnme,
        employee.lastname,
        project.projno,
        project.projname
FROM     enterprise.department department,
        enterprise.employee employee,
        enterprise.project project
WHERE    department.deptno=employee.workdept AND
        employee.empno=project.respemp
ORDER BY department.deptno, employee.empno, project.projno

```

A sub-SELECT with XMLAGG() is then constructed on top of that which produces the inner-most layer child element. In this example, it is the <project> element. XMLAGG() is always used together with the GROUP BY clause (see Example 4-24). In fact, this is where the hierarchical structure arises. In the SELECT clause, XMLAGG() is used to list all <project> elements. The elements specified in the GROUP BY clause belong to the parent elements of the <project> element. For example, since deptno, deptname, empno, firstnme, and lastname are part of the GROUP BY clause, they all belong to the <department> and <employee> elements.

Example 4-24 Add sub-SELECT to DB2 code

```

SELECT   t.deptno,
        t.deptname,

```

```

        t.empno,
        t.firstnme,
        t.lastname,
        XMLAGG (XMLELEMENT (name "project",
                           XMLATTRIBUTES ( t.projno AS "projno",
                                           t.projname AS "projname"))))
FROM
  (   SELECT   department.deptno,
              department.deptname,
              employee.empno,
              employee.firstnme,
              employee.lastname,
              project.projno,
              project.projname
        FROM     enterprise.department department,
              enterprise.employee employee,
              enterprise.project project
        WHERE    department.deptno=employee.workdept AND
              employee.empno=project.respemp
        ORDER BY department.deptno, employee.empno, project.projno
      ) AS t (deptno, deptname, empno, firstnme, lastname, projno, projname)

GROUP BY deptno, deptname, empno, firstnme, lastname

```

The next step is to navigate through the XML output and build another sub-SELECT, which corresponds to the <employee> element (Example 4-25).

Example 4-25 Add 2nd sub-SELECT to DB2 code

```

SELECT   deptno,
        deptname,
        XMLAGG (XMLELEMENT (name "employee",
                           XMLATTRIBUTES (v.empno AS "empno",
                                           v.firstnme AS "firstnme",
                                           v.lastname AS "lastname"),
                           emp ))
FROM
  (   SELECT   t.deptno,
              t.deptname,
              t.empno,
              t.firstnme,
              t.lastname,
              XMLAGG (XMLELEMENT (name "project",
                           XMLATTRIBUTES (t.projno AS "projno",
                                           t.projname AS "projname"))))
        FROM
  (   SELECT   department.deptno,
              department.deptname,
              employee.empno,

```

```

        employee.firstnme,
        employee.lastname,
        project.projno,
        project.projname
FROM      enterprise.department department,
        enterprise.employee employee,
        enterprise.project project
WHERE     department.deptno=employee.workdept AND
        employee.empno=project.respemp
ORDER BY  department.deptno, employee.empno, project.projno
) AS t (deptno, deptname, empno, firstnme, lastname, projno, projname)
GROUP BY deptno, deptname, empno, firstnme, lastname
) AS v (deptno, deptname, empno, firstnme, lastname, emp)
GROUP BY deptno, deptname

```

Finally the top element, the <department> element is built (not shown).

Another method of converting this example to DB2 UDB involves the use of a common table expression. Example 4-26 demonstrates how to nest the XMLAGG() function within a common table expression. The SELECT statement becomes much easier to construct because it reflects the natural hierarchy of the XML output.

Example 4-26 Converting FOR XML AUTO query using a common table expression

```

SELECT xml2clob (
    XMLELEMENT (name "department",
        XMLATTRIBUTES ( dd.deptno as "deptno",
                        dd.deptname as "deptname"),
        ( SELECT XMLAGG (
            XMLELEMENT (name "employee",
                XMLATTRIBUTES ( ee.empno as "empno",
                                ee.firstnme as "firstnme",
                                ee.lastname as "lastname"),
                ( SELECT XMLAGG (
                    XMLELEMENT (name "project",
                        XMLATTRIBUTES (
                            pp.projno as "projno",
                            pp.projname as "projname"))
                    FROM      db2admin.project pp
                    WHERE     pp.respemp = ee.empno)))
            FROM      db2admin.employee ee
            WHERE     ee.workdept = dd.deptno)))
        FROM      db2admin.department dd
        WHERE     dd.deptno IN (SELECT workdept FROM db2admin.employee)
    )

```

FOR XML EXPLICIT example

The FOR XML EXPLICIT clause returns a user-defined mapping of XML data. Example 4-27 shows a SQL Server query using the FOR XML EXPLICIT clause.

Example 4-27 FOR XML EXPLICIT SQL Server code

```
SELECT 1 as tag,
        NULL as parent,
        department.deptno as [department!1!deptno],
        NULL as [employee!2!empno]
FROM    department
UNION ALL
SELECT 2,
        1,
        department.deptno,
        employee.empno
FROM    department, employee
WHERE   employee.workdept = department.deptno
ORDER BY [department!1!deptno],[employee!2!empno]
FOR XML EXPLICIT
```

Example 4-28 shows the partial XML output of Example 4-27.

Example 4-28 Partial XML output of FOR XML EXPLICIT example

```
<department deptno="A00">
  <employee empno="000010" />
  <employee empno="000110" />
  <employee empno="000120" />
</department>
<department deptno="B01">
  <employee empno="000020" />
</department>
<department deptno="C01">
  <employee empno="000030" />
  <employee empno="000130" />
  <employee empno="000140" />
</department>
```

The key to performing this conversion is to understand the XML output produced. Thus, we use the same techniques covered in the previous examples to produce a similar result in DB2 UDB with SQL/XML. This result is shown in Example 4-29.

Example 4-29 DB2 UDB equivalent statement of Example 4-27

```
SELECT xm12clob(XMLELEMENT(name "department",
                           XMLATTRIBUTES(d.deptno AS "deptno"),
                           XMLAGG (XMLELEMENT (name "employee",
                                                XMLATTRIBUTES(e.empno AS "empno")))))
FROM    enterprise.department d, enterprise.employee e
```

```
WHERE      d.deptno = e.workdept
GROUP BY  d.deptno
```

4.10.2 OPENXML statement

OPENXML is a T-SQL keyword that provides a relational row set view over an in-memory XML document. OPENXML is a row set provider similar to a table or a view. OPENXML can be used in statements such as SELECT, INSERT, UPDATE, and DELETE. Instead of specifying a source table or view, OPENXML can be used. Example 4-30 shows a sample SQL Server OPENXML statement.

Example 4-30 A sample SQL Server OPENXML statement

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc = '
<root>
<department deptno="A00" deptname="SPIFFY COMPUTER SERVICE DIV.">
  <employee empno="000010" firstnme="CHRISTINE" lastname="HAAS">
    <project projno="OR1000" projname="OLAP REPORT"></project>
  </employee>
</department>
<department deptno="E21" deptname="SOFTWARE SUPPORT">
  <employee empno="000100" firstnme="THEODORE" lastname="SPENSER">
    <project projno="DM1000" projname="DATA MINING"></project>
  </employee>
</department>
</root>'

-- Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc

-- Execute an INSERT statement that uses the OPENXML row set provider.
INSERT INTO project (respemp, deptno, projno, projname)
SELECT *
FROM OPENXML (@idoc, '/root/department/employee/project',2)
WITH (empno char(6) '../@empno',
      deptno char(3) '../@deptno',
      projno char(6) '@projno',
      projname varchar(24) '@projname')
```

One way to convert OPENXML to DB2 UDB is to use the DB2 XML Extender. The DB2 XML Extender offers the capability to shred an XML document into a set of DB2 relational tables. A similar relational result set as the OPENXML row set can then be created by SELECTing from the relational tables. The steps for doing this with the DB2 XML Extender are now presented.

1. The hierarchical structure of the incoming XML data can be mapped to a set of DB2 UDB relational tables. For example, if the XML data is in a format similar to the following:

```
<department deptno="A00" deptname="SPIFFY COMPUTER SERVICE DIV.">
  <employee empno="000010" firstnme="CHRISTINE" lastname="HAAS">
    <project projno="OR1000" projname="OLAP REPORT"></project>
  </employee>
</department>
<department deptno="E21" deptname="SOFTWARE SUPPORT">
  <employee empno="000100" firstnme="THEODORE" lastname="SPENSER">
    <project projno="DM1000" projname="DATA MINING"></project>
  </employee>
</department>
```

Then, it can be mapped to three tables in DB2 UDB:

- a. department_nn table with deptno and deptname columns
- b. employee_nn table with empno, firstnme, lastname and deptno columns
- c. project_nn table with projno, projname and empno columns

It is not necessary to explicitly specify the referential integrity relationship between the tables. However, a deptno column in the employee table and empno column in the project table have been included so these tables could be joined together in a query.

2. After creating the relational tables in DB2 UDB, the relationship between them and the XML document is established in the DAD (Document Access Definition) file. A RDB (Relational Database) mapping is used in this example. The mapping between the table columns and the XML elements/attributes appear in the DAD file. In addition, the join conditions among the tables is also specified in the DAD file. Example 4-31 shows a sample DAD file.

Example 4-31 Sample DAD file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DAD PUBLIC "dadId" "dad.dtd">
<DAD>
  <dtdid>NewProject1.dtd</dtdid>
  <validation>NO</validation>
  <Xcollection>
    <prolog?>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE root PUBLIC "NewProject1Id" "NewProject1.dtd"
    </doctype>
    <root_node>
      <element_node name="root">
        <RDB_node>
          <table name="JZHANG.DEPARTMENT_NN"/>
          <table name="JZHANG.EMPLOYEE_NN"/>
          <table name="JZHANG.PROJECT_NN"/>
          <condition>
```



```

        JZHANG.DEPARTMENT_NN.DEPTNO=JZHANG.EMPLOYEE_NN.DEPTNO
        AND JZHANG.EMPLOYEE_NN.EMPNO=JZHANG.PROJECT_NN.EMPNO
    </condition>
</RDB_node>
<element_node name="department" multi_occurrence="YES">
    <attribute_node name="deptno">
        <RDB_node>
            <table name="JZHANG.DEPARTMENT_NN"/>
            <column name="DEPTNO" type="Character(3)"/>
        </RDB_node>
    </attribute_node>
    <attribute_node name="deptname">
        <RDB_node>
            <table name="JZHANG.DEPARTMENT_NN"/>
            <column name="DEPTNAME" type="VarChar(29)"/>
        </RDB_node>
    </attribute_node>
    <element_node name="employee">
        <attribute_node name="firstnme">
            <RDB_node>
                <table name="JZHANG.EMPLOYEE_NN"/>
                <column name="FIRSTNME" type="VarChar(12)"/>
            </RDB_node>
        </attribute_node>
        <attribute_node name="empno">
            <RDB_node>
                <table name="JZHANG.EMPLOYEE_NN"/>
                <column name="EMPNO" type="Character(6)"/>
            </RDB_node>
        </attribute_node>
        <attribute_node name="lastname">
            <RDB_node>
                <table name="JZHANG.EMPLOYEE_NN"/>
                <column name="LASTNAME" type="VarChar(15)"/>
            </RDB_node>
        </attribute_node>
        <element_node name="project">
            <attribute_node name="projno">
                <RDB_node>
                    <table name="JZHANG.PROJECT_NN"/>
                    <column name="PROJNO" type="Character(6)"/>
                </RDB_node>
            </attribute_node>
            <attribute_node name="projname">
                <RDB_node>
                    <table name="JZHANG.PROJECT_NN"/>
                    <column name="PROJNAME" type="VarChar(24)"/>
                </RDB_node>
            </attribute_node>
        </element_node>
    </element_node>

```

```

        </element_node>
    </element_node>
</element_node>
</element_node>
</root_node>
</Xcollection>
</DAD>

```

3. Based on the DAD mapping, the XML Extender's stored procedures can be invoked to shred the XML document into the DB2 UDB tables. The key when converting to DB2 UDB is to produce a relational data set that matches the OPENXML row set. To produce the same relational data set in DB2 UDB, it must be determined which tables, produced from the XML document shredding, should be queried. In this case, the following SELECT statement returns a similar relational set:

```

SELECT    d.deptno, e.empno, projno, projname
FROM      department_nn d, employee_nn e, project_nn p
WHERE     d.deptno=e.deptno and e.empno=p.empno

```

4. The PROJECT table in DB2 UDB can be populated by retrieving from the temporary tables. Thus, the SQL Server INSERT statement can be converted as:

```

INSERT INTO project (deptno, respemp, projno, projname)
SELECT *
FROM (
    SELECT d.deptno, e.empno, projno, projname
    FROM department_nn d, employee_nn e, project_nn p
    WHERE d.deptno = e.deptno AND e.empno=p.empno
)

```

Other XML integration possibilities

OPENXML is quite flexible in terms of exploring XML data as a row set. When specifying a rowpattern, OPENXML has the capability of specifying column data types in the returning rowset dynamically. The DB2 XML Extender is not as flexible as the OPENXML approach since there are several steps that must be taken to shred XML documents.

There is an alternative method for converting OPENXML to DB2 UDB. With WebSphere Information Integrator, an XML wrapper can be used to consume a XML file. XML nicknames can be created against the XML wrapper. The XML nicknames can then be used in regular SQL queries as a data source.

4.11 SQL limits

There are differences in SQL limits between SQL Server and DB2 UDB relating to identifier lengths and space/size limitations of various database entities and objects. While these differences do not typically cause major issues when porting from SQL Server to DB2 UDB, they can be time consuming to deal with because of the amount of effort required to rename existing objects and/or potentially re-architect some non-compliant components.

Refer to Appendix F, “SQL limits” on page 505 for a more detailed comparison of SQL limits in DB2 UDB and SQL Server.

If object identifiers need to be changed to meet DB2 UDB’s SQL limits, changes must also be reflected in SQL statements and/or scripts where an object is explicitly referred to by name, such as a user defined function. Objects that are very infrequently referred to by name from outside the database, such as triggers or indexes, typically do not require changes external to the database, aside from the object creation code.

Note: We recommend using the same object identifier names in SQL Server and DB2 UDB implementations of your application. This makes code maintenance and design easier.

The full list of DB2 UDB SQL Limits can be found in the DB2 UDB documentation *SQL Reference - Volume 1*, SC09-4844.

Planning for a conversion

Converting from SQL Server to DB2 UDB requires proper planning, whether you perform the conversion on your own or use IBM resources. Unexpected delays in project execution can be avoided by following well planned conversion procedures.

This chapter introduces the typical conversion process. The planning process, as well as the available tools and resources are also documented in this chapter. Detailed conversion information can be found in Chapter 7 through Chapter 10. The phases of the conversion discussed in this chapter include:

- ▶ **Preparation**
This includes project planning, education of database administrators and developers, and the setup of the environment.
- ▶ **Conversion**
This includes the conversion of the database structure, objects, and data, as well as the applications that connect to the database, and any maintenance/batch scripts.
- ▶ **Post-Conversion**
This includes performance tuning of the converted database and application.

5.1 Preparation

The conversion preparation phase includes all the activities that take place before setting up the system and converting the database, database objects, and data. The preparation phase is also a good time to consider items mentioned in 5.3, “Post-conversion” on page 132. There, we introduce other DB2 UDB offerings that can help meet the needs of your project. The major tasks involved in preparation include:

- ▶ Completing a porting assessment
- ▶ Understanding and selecting conversion tools
- ▶ Estimating the effort required
- ▶ Planning the conversion
- ▶ Becoming educated on DB2 UDB

Tip: Visit the Porting Zone Web site: <http://www.ibm.com/db2/porting>

5.1.1 Performing a porting assessment

Planning a port begins with an assessment of the current system and environment, as well as an understanding of the resources that can be utilized.

Architecture and environment profiling

An accurate profile of the system wide architecture is key to the success of the port. The assessment begins with collecting information about both the source and target environments. This detailed information is used to determine the scope of the conversion, and is the main input for the port planning process.

The following questions provide a sample of the considerations that require attention:

- ▶ What best characterizes the workload type (OLTP, OLAP/DSS)?
- ▶ Which database objects (number of tables, indexes, stored procedures, etc.) are used?
- ▶ What language (C, C++, VB, C#, Java, etc.) is the application written in?
- ▶ What are the characteristics of the SQL statements in the application?
- ▶ What is the client target operating system (Windows NT®, Windows XP, Browser Based, etc.) and the version/release/FixPak number?
- ▶ What is the server target hardware platform (IBM Series, HP, Sun, etc.)?
- ▶ What is the server target operating system (Windows, AIX 4.3, Linux, etc.) and version/release/FixPak number?

- What is the typical configuration of your database server (Number of boxes, number of CPUs, RAM, and disks, etc.)?

Often, you can retrieve this information by using scripts that query the system catalog tables, or by using utilities provided by SQL Server. You can also use the IBM DB2 Migration Toolkit (MTK) to retrieve structural and object information from a SQL Server database.

5.1.2 Understanding and selecting conversion tools

Although a conversion can be performed without the help of tools, IBM has created a tool that is specifically designed to make a conversion as simple as possible. The tool is introduced here and covered in detail in later chapters.

IBM DB2 Migration Toolkit

The *IBM DB2 Migration Toolkit* helps convert from SQL Server, Oracle, and Sybase databases to DB2 UDB databases on any supported DB2 UDB workstation platform. This tool can be used to generate DDL scripts that create database objects including tables, indexes, views, triggers, stored procedures, and user defined functions. It also aids in moving the data from the original system to the target DB2 UDB system. For instance, the MTK can either connect directly to the source system and extract the SQL Server database structure and database object definitions, or it can use a valid T-SQL script extracted by other tools.

Other porting tools

There are a number of third part conversion tools available to assist you in converting your database, application, and data from SQL Server to DB2 UDB. These tools and services are not provided by IBM.

- **SQLWays**

SQLWays of Ispirer Systems is product that converts tables, views and indexes. You can also transfer data based on the converted tables. The tool supports SQL Server 2000, 7.0 and 6.5.

- **SwisSQL**

The *SwisSQL - SQL Server to DB2 Edition* helps migrate SQL Server objects and T-SQL procedures, functions and triggers to DB2 UDB.

Modeling tools

There are a number of modeling tools that can capture the entity-relation (E-R) descriptions of your database. By capturing this information, the tool can generate the appropriate DB2 UDB object definition syntax. A few of the common modeling tools are:

- ▶ **Rational® Rose Family**
Development tools with a database design tool that allows database designers, business analysts, and developers to work together through a common language.
- ▶ **CA AllFusion ERwin Data Modeler**
A data modeling solution that helps create and maintain databases, data warehouses, and enterprise data models.
- ▶ **Embarcadero Technologies ER/Studio**
ER/Studio can reverse-engineer the complete schema of many database platforms by extracting object definitions and constructing a graphical data model. Other tools are available for application development (Rapid SQL and DBArtisan).
- ▶ **Borland Together**
Borland's enterprise development platform provides a suite of tools that enables development teams to build systems quickly and efficiently. Borland Together Control Center is an application development environment that encompasses application design, development, and deployment. Borland Together Edition for WebSphere Studio offers IBM-centric development teams a complete models-to-code solution. Borland Together Solo provides an enterprise class software development environment for small development teams.

5.1.3 Estimating the effort required

An accurate estimation of the effort required, resources needed, and total costs requires knowledge of the products, applications, and porting experience.

MTK can be used as an assessment tool to determine the complexity of the migration. Using MTK in this manner can highlight complex stored procedures or SQL statements that may require manual conversion effort. A working knowledge of the MTK is mandatory when using it in this manner.

The cost of the new database software and conversion tools should also be considered when estimating the project cost. MTK is provided free of charge. Contact your IBM Sales Representative for details about DB2 UDB pricing.

Training costs for DBAs and end-users should also be figured into your estimation. Finally, hardware procurement must be planned if your existing data server does not have the capacity to run the existing SQL Server installation, MTK, and DB2 UDB.

The IBM Software Migration Project Office (SMPO) can also provide porting estimates. Contact the SMPO at:

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

5.1.4 Planning the conversion

Each port is different, but there are general signs that can indicate the overall complexity and effort expected. For instance, in applications that frequently use stored procedures, the number and complexity of the stored procedures to be converted greatly affects the length of the port. The same applies to the use of special data types and large objects. Physical requirements (use of raw or formatted disk areas, spaces, nodes, etc.) may also represent a large amount of work, especially if the data grows significantly over time.

A project plan can be as simple as a spreadsheet that lists the main tasks of the conversion and some of the associated information for each task (start date, end date, elapsed time, task dependencies, who is assigned, etc.). There are also project planning tools (such as Microsoft Project, Primavera TeamPlay, and Primavera Enterprise) that are specifically designed to plan, and track the progress of the project. These tools let you assign tasks to specific people (or roles), establish dependencies among the various steps of the port (for instance, you cannot start testing until you move the database structure and the test data), and chart the original plan against what actually happens.

To help in the planning process, a porting and migration planning whitepaper has been developed. It can be downloaded from the following Web site:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0311simchuk>

By following the steps and suggestions in this whitepaper, you should be able to develop an accurate project plan.

5.1.5 Becoming educated on DB2 UDB

It is important that members of the conversion team have a good understanding of important DB2 UDB features, such as transaction management, locking, authorities, and memory management. Although most database vendors implement similar sets of database features, there are often substantial differences in how they behave.

The fastest way to prepare your technical staff for effectively working with DB2 UDB is through some form of education. For more information about the various training and certification options available, view the section entitled *Convert your skills* on the following Web site:

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

This Web site provides numerous references to sites within IBM that offer classroom training, online tutorials, and reference materials.

5.1.6 Environment preparation

Before starting the conversion, the target development environment should be set up. Ensure that all hardware and software prerequisites are met, that the network is properly configured, and that there is enough hardware to properly support the conversion and development.

In this step, you should complete the following tasks:

- ▶ Configure operating system
- ▶ Configure disk storage
- ▶ Install DB2 UDB
- ▶ Create a DB2 UDB instance
- ▶ Install and configure tools
- ▶ Configure the DB2 UDB Instance
- ▶ Configure the database environment and registry variables
- ▶ Test the environment with a sample application

For more information about these steps and others, refer to the porting zone Web site at:

<http://www.ibm.com/db2/porting/>

5.2 Conversion

The steps required to convert a database and application to DB2 UDB are introduced in this section. The methods employed can vary, but at a minimum, the following steps are required. This chapter introduces:

- ▶ Converting the database structure
- ▶ Converting database objects
- ▶ Modifying the application
- ▶ Modifying the database interface
- ▶ Migrating the data

For more information about these steps and others, refer to the porting zone Web site at:

<http://www.ibm.com/db2/porting/>

5.2.1 Converting the database structure

The first step in the conversion process involves moving or duplicating the structure of the source database into a DB2 UDB database. Before this can happen, differences between the source and destination (DB2 UDB) structures

must be addressed. These differences can result from different interpretation of SQL standards, or the addition and omission of particular functions. Many differences can be fixed by altering the existing syntax, but in some cases, custom functions must be added and the application must be modified.

The logical Entity-Relationship (E-R) model of the database describes the meaning of each entity, the relations that exist, and their attributes. SQL data definition language (DDL) statements can be generated to create a database based on the model. If the model already exists in a modeling tool, it is often possible to have the modeling tool generate a new set of DDL code that is specific to DB2 UDB. Otherwise, the DDL code from the current system must be extracted and converted into a form that is compatible with DB2 UDB. After the DDL is modified, it can be used to create a new database (tables, indexes, constraints, etc.) in DB2 UDB.

There are three approaches that are commonly used to move the structure of a database:

- ▶ **Manual methods:** Extract the existing structure, import it to DB2 UDB, and manually adjust for problems.
- ▶ **Model transport:** Generate DB2 UDB DDL from an existing database model.
- ▶ **Conversion tools:** Use a tool to extract the database structure, make any necessary conversions, then load it in a DB2 UDB database.

Manual methods

Many DBMSs come with a utility that extracts the database structure into a text file. This DDL can then be used to re-create database objects on a new database server. However, before the extracted DDL likely requires syntax changes before it can properly execute in DB2 UDB. These syntax changes requires manual effort.

Besides syntactical differences, there may also be changes needed for data types and other proprietary features used. It is often easiest to convert a subset of the source DDL to DB2 UDB, and examine any issues that arise.

Metadata transport

Many databases are designed using modeling tools. These tools allow the designer to describe the database structure in the form of logical *entities* and *relationships*. The modeling tool can then generate the physical database structure (DDL) from the model. If the system to be ported is designed (and maintained) using a modeling tool, converting the database structure to DB2 UDB can be as simple as having the modeling tool and generate DB2 UDB specific DDL for the model.

Conversion tools

The most popular means of converting a database structure is through the use of a conversion tool. These tools not only connect to and extract structural information from the source database, but they can also convert it into a format acceptable by the target database system. MTK can be used to perform the conversion using this method.

5.2.2 Converting database objects

Database objects (stored procedures, triggers, and user defined functions) contain application logic which is stored in the database. Unfortunately, most of these objects are written in a language that is specific to the source DBMS or in a higher-level language that must be compiled and bound to the target database.

Capturing the definition of database objects can often occur at the same time the database structure is captured, if the objects are written in an SQL-like procedural language and stored within the database. For those objects written in higher-level languages (Java, C, etc.), capture and import involves transferring the source files to the DB2 UDB system, making any necessary syntax changes, then using a DB2 UDB compatible compiler and binding mechanism.

Stored procedures, user defined functions, and triggers need to be converted manually unless the tool used to extract the objects understands the SQL languages of both the source DBMS and DB2 UDB. MTK is an example of a tool that can convert stored procedures, user defined functions, and triggers from various DBMSs to DB2 UDB.

The conversion of database objects requires testing of the resulting objects. This implies that test data should be available before to testing. Preliminary data migration effort is therefore required in order to complete this conversion step.

After the object conversion is completed, some adjustments may still be required. Issues such as identifier length may need to be addressed. This can be done manually or using a tool.

5.2.3 Porting additional database components and products

Many database applications, require additional capabilities on top of general database access. As part of the planning effort, you should consider the following issues and integrate additional educational development time and additional products into your project plan:

- ▶ Information integration issues
- ▶ Content management issues
- ▶ Business intelligence issues

- ▶ Web integration issues
- ▶ Database administration issues

Information integration

Will there be other data sources that must be incorporated in the application and will this occur at the application code level, or at the database level through the use of federation?

Content management

Will there be a need for complex data objects and/or structures that are more sophisticated than those usually stored in relational database data types? If so, what other tools and capabilities are needed and how will they be integrated?

Business intelligence

Is there a need to summarize and analyze the data within the database(s) that may require the use of OLAP tools and different structures of the data? If so, can this be done without impact on the production system, or is an additional copy of key data required which could require replication or other similar processes?

Web integration

If the current GUI is awkward or out-dated, or if a larger population needs access to the application, is it time to consider using a Web-based front-end to the application? If so, what development and runtime tools are needed? What education will be needed for the staff to be able to implement this technology?

Database administration

What backup and recovery plan will be used to ensure the proper level of safety while still allowing acceptable performance? Is replication needed, and if so, how will it be implemented? What is the fail over strategy (redundant hardware, hot nodes that can replace failed ones, other high availability issues)?

5.2.4 Modifying the application

Although converting the database structure and objects can be automated to some extent using conversion tools, application code changes mostly require manual effort. If all database interaction is restricted to a database access layer, then the scope and complexity of necessary changes is well defined and manageable. However, when database access is not isolated to a database access layer, that is, it is distributed throughout application code files, contained in stored procedures and/or triggers, or used in batch programs that interact with the database, then the effort required to convert and test the application code depends on how distributed the database access is and on the number of

database-specific statements in each application source file that require conversion.

Before converting an application, it is important to first convert the database structure (DDL) and database objects (stored procedures, triggers, user-defined functions). It is then useful to populate the database with test data so that the application code can be ported and tested incrementally.

Few tools are available to port the actual application code since much of the work is dependent on vendor-specific issues. These issues include adjustments to compensate for differing approaches to transaction handling, join syntax, use of special system tables, and use of internal registers and values. Often, proprietary functions used in the source DBMS need to be emulated in DB2 UDB, usually by creating a DB2 UDB user defined function and/or stored procedure with the same name as the proprietary one being ported. This way, any SQL statements in the application code that call the proprietary function in question do not need to be altered. Conversion tools such as MTK are equipped with some of the most commonly used vendor-specific functions and automatically create them in DB2 UDB during the conversion process.

Applications written in a high-level language with embedded-SQL can be converted automatically by extracting SQL statements from the source code, adjusting them with a conversion tools such as MTK, and then reinserting them into the source.

One issue when porting high-level language (C, C++, Java, COBOL, etc.) code involves compiler differences. Modifications to the application code may be required if a different compiler and/or object library are used in the DB2 UDB environment. This may result from using a different hardware or OS platform. It is important to fully debug and test such idiosyncrasies before moving a system into production.

For more information about various application development topics relating to DB2 UDB, visit the DB2 developerWorks Web site at:

<http://www.ibm.com/developerworks/db2/>

or the DB2 application development Web site at:

<http://www.ibm.com/software/data/db2/udb/ad/>

5.2.5 Modifying the database interface

Applications that connect to the source database using a standardized interface driver, such as ODBC and OLE-DB, usually require few changes to work with DB2 UDB. In most cases, simply providing the DB2 UDB supported driver for

these interfaces is enough for the application to be up and running with a DB2 UDB database.

There are certain circumstances where the DB2 UDB-supported driver for an interface does not implement or support one or more features specified in the interface standard. In these cases, you must take action to ensure that application functionality is preserved after the conversion. This usually involves changing application code to remove references to the unsupported functions and either replacing them with supported ones, or simulating them by other means.

Applications that use specialized or native database interfaces (e.g., SQL Server's MSS) will require application code changes. Such applications can be ported using DB2 UDB's native CLI interface, or by using a standardized interface such as ODBC.

DB2 UDB also provides a library of administrative functions which can be used to develop administrative applications that can administer DB2 UDB instances, backup and restore databases, import and export data, and perform operational and monitoring functions. These administrative functions can also be run from the DB2 UDB Command Line Processor (CLP), Control Center, and DB2 UDB scripts.

The following sections highlight some of the common interfaces used with DB2 UDB. These interfaces are described more fully in the DB2 UDB documentation *Application Development Guide: Programming Client Applications*, SC09-4826.

ADO and OLE DB

Microsoft's ActiveX Data Objects (ADO) interface provide a set of methods for accessing data from a wide variety of data sources including relational databases, HTML, video, text, and just about any other source of data. Access to the data is handled by ADO and is accessed through a service such as OLE DB or ODBC.

Microsoft .NET

.NET is Microsoft's development platform that competes with the Java J2EE standard. The .NET framework programming model enables developers to build Web-based applications, smart client applications, and XML Web services applications, which expose their functionality programmatically over a network using standard protocols such as SOAP and HTTP. More information about the .NET Framework program and DB2 UDB's support for .NET can be found on the DB2 UDB .NET Program Web page at:

<http://www.ibm.com/developerworks/db2/zones/vstudio/>

JDBC and SQLj

DB2 UDB provides several JDBC drivers to write dynamic SQL programs in Java. DB2 UDB provides support for the Type 2, and Type 3 driver, and, new to Version 8, the Type 4 drivers.

SQLj offers developers a way to write static SQL programs using Java. *SQLj* programs generally outperform their JDBC counterparts because the query access plans of executable statements are optimized before run-time.

More information about the Java supported interfaces can be found at the following links.

- ▶ DB2 UDB Java Application Development Web page at:
<http://www.ibm.com/developerworks/db2/zones/java/>
- ▶ The article *DB2 and Java: The Big Picture* posted on the DB2 developerWorks site also provides a good summary about the different Java options available in DB2 UDB. The article can be viewed at:
<http://www.ibm.com/developerworks/db2/zones/java/bigpicture.html>
- ▶ *Considering SQLj for Your DB2 V8 Java Applications* is a white paper that addresses access to relational data from Java. This paper can be view at:
<http://www.ibm.com/developerworks/db2/library/techarticle/0302tsui/0302tsui.html>

Embedded SQL (static and dynamic)

DB2 UDB gives programmers the option of writing applications with SQL statements directly embedded in the host language. The SQL statements provide an interface to the database while the host language provides facilities to perform the application logic. DB2 UDB supports several host languages including C/C++, FORTRAN, COBOL, and Java (SQLj). Programmers have the option of using static or dynamic SQL, depending on the needs of the application.

ODBC

Microsoft's ODBC standard provides a set of APIs for accessing a vendor's database. Vendors must supply their own driver that implements a subset of the API for accessing the database. The DB2 UDB CLI driver can be used on its own to access a DB2 UDB database or as an ODBC driver. DB2 UDB conforms to most of the Level 3 compliance level for ODBC.

Perl DBI

DBI is an open standard API that provides database access for client applications written in Perl. DBI defines a set of functions, variables, and conventions that provide a platform-independent database interface. The latest

DB2 UDB Perl driver and information about Perl can be found at the DB2 UDB Perl DBI support Web page at:

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

DB2 UDB CLI

DB2 UDB's CLI Driver implements most of the function set defined in the ODBC standard as well as additional functionality specific to DB2 UDB. This interface offers more functionality than the other interfaces. The latest information about developing applications with CLI can be found in the DB2 UDB documentation *Call Level Interface Guide and Reference - Volume 1*, SC09-4849, and *Volume 2*, SC09-4850.

Stored procedures

Another popular method of interfacing with the database is through stored procedures. Stored procedures can be written in DB2 UDB's SQL procedural language, or in an external programming language such as a supported Microsoft CLR language or Java. Restricting database access through stored procedures offers numerous benefits such as a reduction in network traffic (all processing takes place on the server), and providing an additional layer of isolation between the application code and business logic. An excellent reference for building SQL stored procedures can be found in:

- ▶ *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, and Windows, i5/OS, and z/OS*, ISBN 0131477005
- ▶ *DB2 Application Development Guide: Programming Server Applications*, SG09-4827

5.2.6 Migrating the data

You can move data between DBMSs using commercially available tools such as MTK, Ascential DataStage, and others.

In many cases, as the data is moved, it is also converted to a format that is compatible with the new DBMS (date and time data are a good example). This process can be quite lengthy, especially when there is a large amount of data. This makes it imperative to have the data conversions well-defined and tested.

In some cases, it is still necessary to perform customized conversions of specialized data, such as time series, and/or geospatial data. This can sometimes be accomplished through the creation of a small program or script.

5.3 Post-conversion

Several activities should occur after the database and application have been successfully converted to DB2 UDB, and preliminary testing completed. These activities include:

- ▶ Performance tuning
- ▶ Utilizing additional DB2 utilities
- ▶ Defining a maintenance strategy

We highly recommend performing these tasks to ensure user acceptance.

5.3.1 Performance tuning

Performance tuning is the continuous process of ensuring maximum system performance. The process includes making hardware configuration adjustments, operating system adjustments, application code optimizations, interface changes, database parameter changes, and query optimization.

While it is beyond the scope of this book to offer a full discussion of performance tuning, a list of performance tuning links and resources can be found at the following URL:

<http://www.ibm.com/developerworks/db2/zones/porting/tuning.html>

Methodology

The following steps work well when beginning any type of performance tuning activity:

- ▶ Define the objectives: What is wrong or what needs to be accomplished?
- ▶ Determine the information to analyze.
- ▶ Determine the monitor(s) to use.
- ▶ Test and obtain monitor data.
- ▶ Analyze the information.
- ▶ Determine the changes required.
- ▶ Implement changes (one at a time) and go back to step four.

By following this process, and carefully recording all settings, adjustments, and results, you should be able to achieve incremental improvements in performance, or determine that you have reached an optimal set of parameters. It is important to remember that performance tuning is a balancing act. You often must make trade-offs.

Performance Tuning Documentation

There is a rich source of information about performance tuning in the DB2 UDB documentation. The DB2 UDB documentation *Administration Guide: Performance V8*, SC09-4821, offers an extensive description of the tuning process (at various levels) and addresses many of the common areas where tuning can be most beneficial.

Other information sources

The DB2 developerWorks Web site contains a number of helpful articles that can help you increase the effectiveness of your use of DB2 UDB.

- ▶ *SQL Procedures Performance: Hints and Tips*
<http://www.ibm.com/developerworks/db2/library/techarticle/0306arocena/0306arocena.html>
- ▶ *SQL Access to DB2 Monitoring Data: Capturing Snapshots*
<http://www.ibm.com/developerworks/db2/library/techarticle/0305deroos/0305deroos.html>
- ▶ *Tuning DB2 Universal Database Using the Statement Event Monitor*
<http://www.ibm.com/developerworks/db2/library/techarticle/0303kolluru/0303kolluru.html>
- ▶ *When We Think That the Optimizer Doesn't Get It Right*
<http://www.ibm.com/developerworks/db2/library/techarticle/0302kuznetsov/0302kuznetsov.html>
- ▶ *Tuning DB2 SQL Access Paths*
<http://www.ibm.com/developerworks/db2/library/techarticle/0301mullins/0301mullins.html>

Useful tools

Part of performance tuning involves being able to monitor and measure the items to be analyzed. Here are some tools that may be helpful:

- ▶ **IBM DB2 Performance Expert:**
This IBM tool integrates performance monitoring, reporting, buffer pool analysis, and a Performance Warehouse function into one tool.
- ▶ **Configuration Advisor:**
This GUI wizard provides configuration recommendations based on user-supplied answers to system environment questions. This wizard can be accessed through Control Center.

- ▶ Design Advisor:

This GUI wizard provides data access object (index, MDC tables, MQTs) recommendations based on an input workload. This wizard eliminates the guesswork involved in deciding which objects to use.

5.3.2 Utilizing additional DB2 utilities

Other database administration tasks that should be considered include:

- ▶ Backup and recovery: To prevent the loss of data, you should have an adequate strategy for backing up the data in your databases and restoring it in the case of a failure or error. The more data you have, the longer and more sophisticated your strategy will become, especially if the database must always be online.
- ▶ Replication: The DB2 UDB replication feature enables you to transfer data between systems for the purpose of building new data stores, or for duplicating some or all, of the original data in another DBMS.
- ▶ High availability: High availability ensures that your data sources are always available for use (regardless of failures in hardware and software). This concept is closely related to backup and recovery, and is often implemented at the same time.
- ▶ Federation: Federation allows access to data in numerous, heterogeneous data stores from one query.

5.3.3 Defining a maintenance strategy

The key to keeping a system performing well is to keep data fragmentation to a minimum and keep database statistics up-to-date. Based on lessons learned from the development and performance tuning steps, recommendations can be made about how to best keep the database performing well. It is important that these details be communicated in the form of documentation to the end-user.

Keep in mind the following tasks:

- ▶ Examine needs for maintaining performance
- ▶ Establish and implement a **REORG** strategy
- ▶ Establish and implement a **RUNSTATS** strategy
- ▶ Establish and implement a **REBIND** strategy (if necessary)
- ▶ Establish and implement a Backup/Recovery strategy
- ▶ Establish and implement a High Availability strategy
- ▶ Test and refine all strategies as needed

5.4 Additional references

For more information about the DB2 UDB conversion process, as well as information about database administration, application development, and performance tuning activities, consult the following Web sites:

- ▶ IBM developerWorks
<http://www.ibm.com/developerworks/db2/>
- ▶ DB2 Porting Zone
<http://www.ibm.com/db2/porting>
- ▶ IBM Redbooks
<http://www.redbooks.ibm.com/>

The IBM DB2 Migration Toolkit for SQL Server

In this chapter, we introduce the IBM DB2 Migration Toolkit (MTK) for SQL Server. Specifically, the following topics are covered:

- ▶ MTK technical overview
- ▶ Set up MTK for migration
- ▶ Using MTK

MTK for SQL Server can be downloaded free of charge from the following Web site:

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

MTK can simplify your migration to DB2 Universal Database (UDB) and reduce time to migrate your database. It provides database administrators (DBAs) and application programmers with the tools needed to automate previously inefficient and costly migration tasks. You can eliminate human error, and cut back on person hours and other resources associated with traditional database migration by using MTK. MTK converts the following SQL Server source database objects into the equivalent DB2 UDB database objects:

- ▶ Data types
- ▶ Tables
- ▶ Columns

- ▶ Views
- ▶ Indexes
- ▶ Constraints
- ▶ Packages
- ▶ Stored procedures
- ▶ Functions
- ▶ Triggers
- ▶ Sequences

MTK enables the following tasks:

- ▶ Obtaining source database metadata (DDL) by extracting information from the source database system catalogs through (JDBC/ODBC).
- ▶ Obtaining source database metadata (DDL) by importing DDL scripts created by SQL Server or third-party tools.
- ▶ Automating the conversion of database object definitions, including stored procedures, user defined functions, triggers, packages, tables, views, indexes, and sequences.
- ▶ Deploying T-SQL and Java compatibility functions that permit the converted code to “behave” similarly to the source code.
- ▶ “On-the-fly” conversion of T-SQL statements using SQL Translator tool. This is also effective as a DB2 UDB SQL PL learning aid for SQL Server T-SQL developers.
- ▶ Viewing conversion information and messages.
- ▶ Deployment of the converted objects into a new or existing DB2 UDB database.
- ▶ Generating and running data movement (unload/load) scripts or performing the data movement online.
- ▶ Tracking the status of object conversions and data movement, including error messages, error location, and DDL change reports using the detailed migration log file and report.

6.1 MTK technical overview

For a complete MTK technical review, refer to the documentation of the downloaded version of your MTK. The details below are for MTK Version 1.3. Before installing MTK, verify that the hardware and software requirements provided are met.

6.1.1 Supported operating system and versions

MTK can run on the following operating systems and versions:

- ▶ Windows NT 4.0, Windows 2000, Windows XP professional
- ▶ Linux (tested on Mandrake Version 8.1 with pdksh installed)
- ▶ AIX 4.3.3.0 or later
- ▶ HP-UX (tested on Version 11i)
- ▶ Sun Solaris 5.7

6.1.2 Hardware requirements

The hardware requirements for installation of MTK are:

- ▶ 50 MB of disk space for installation, 5 MB per project and extra space to unload data.
- ▶ 128 MB minimum memory recommended.
- ▶ 300 MHZ or higher processor.

6.1.3 MTK software requirements

MTK supports several source database management systems. Previously, in order to deploy SQL stored procedures in DB2 UDB version 8.1 or earlier, Microsoft Visual C++ version 5 or later was required to compile the procedures. DB2 UDB v8.2 does not require a compiler anymore.

6.1.4 Where to install MTK

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

MTK stores the extracted data on the server where it is installed. Hence, there is a performance advantage to installing MTK where the source database resides, since it is faster to unload data locally than across a network.

6.2 MTK setup

MTK installation is simple and requires minimum preparation. This section highlights a typical MTK installation. To install MTK:

1. Download the latest IBM DB2 Migration Tool Kit for Windows from the following Web site:

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

At the time of writing, the current the MTK version is 1.3
(*db2mtk_V1_3_GA_win.exe*.)

2. Launch the MTK installation by double-clicking the downloaded file. This displays the *Welcome* screen (Figure 6-1). Click **Next** on the Welcome window to proceed with the installation.

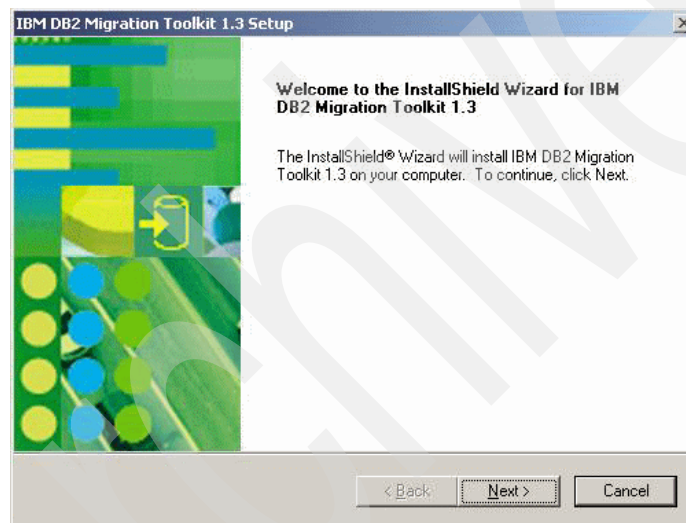


Figure 6-1 MTK Installation - Welcome screen

3. On the *License Agreement* screen, click **Yes** to accept the agreement (Figure 6-2).

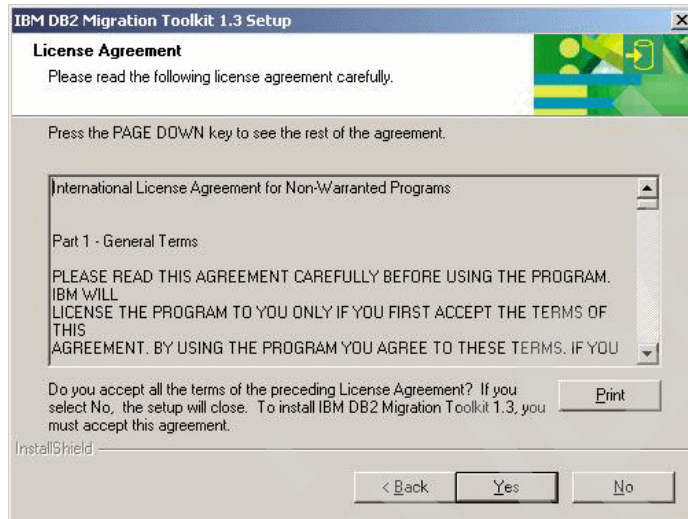


Figure 6-2 MTK Installation - Agreement screen

4. On the *Choose Destination Location* Screen, specify the **Folder** where you want to install the MTK (Figure 6-3).
Keep the default settings and click **Next** for this project.

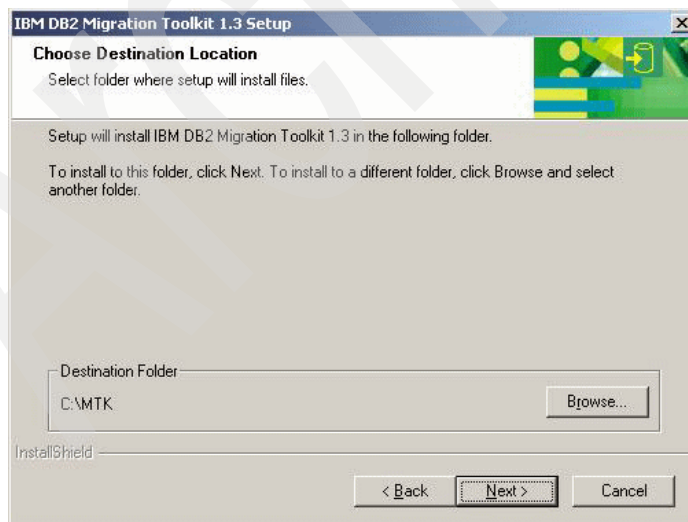


Figure 6-3 MTK Installation- Installation folder

5. The MTK begins installing all the required files. A progress screen, similar to Figure 6-4 is shown to indicate overall installation progress.

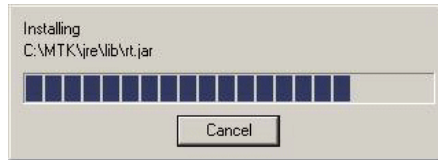


Figure 6-4 MTK Installation - Progress screen

6. The *InstallShield Wizard complete* screen (Figure 6-5) is displayed when installation has completed.

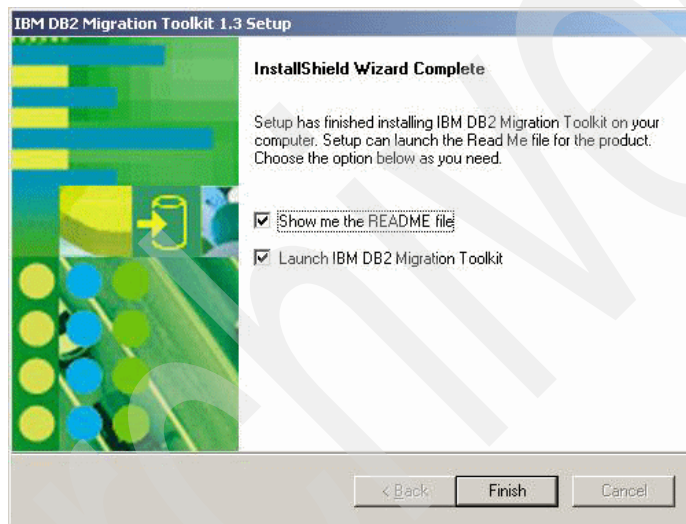


Figure 6-5 MTK Installation - Install Complete Screen

6.3 Using MTK

In this screen, we demonstrate how the MTK can be used to migrate a SQL Server database to DB2 UDB.

6.3.1 MTK GUI interface

The MTK main window (Figure 6-6) presents five tabs, each of which represents a specific task in the conversion process. The tabs are organized from left to right and are titled:

- Specify Source
- Convert
- Refine
- Generate Data Transfer Scripts
- Deploy to DB2

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

- **Application:** Allows you to set-up preferences, such as your favorite text editor.
- **Project:** Start a new project, open or modify an exiting project, import a SQL source file, or perform backup/restore function.
- **Tools:** Launch the SQL Translator, reports, and the MTK log.
- **Help:** MTK help information

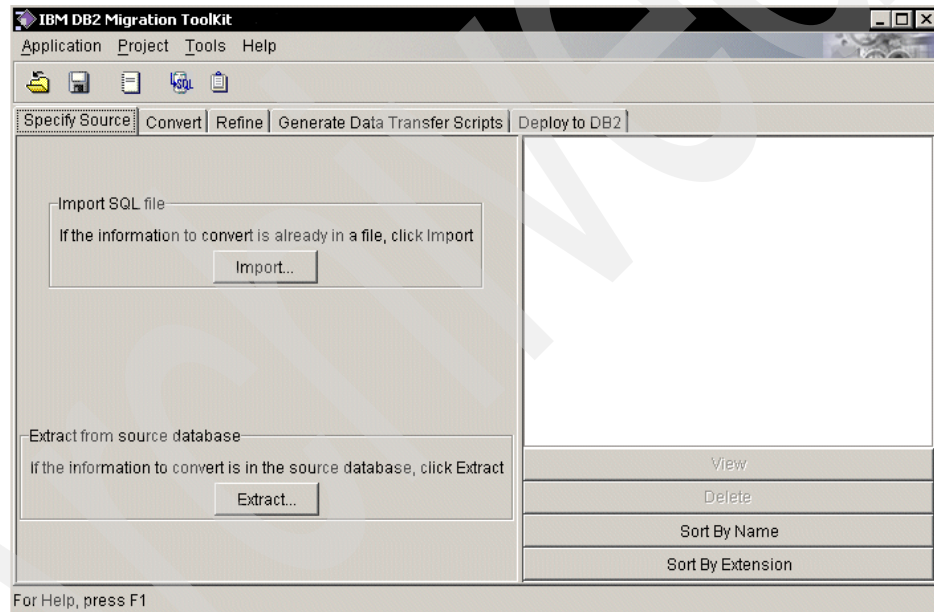


Figure 6-6 MTK Main window

6.3.2 Migration tasks

The five tabs in the MTK main window represent the five phases of the migration process. The following is an overview of each of these five tasks:

► Task 1: Specify Source

The *Specify Source* task (Figure 6-7) focuses on extracting or importing SQL Server database metadata (DDL) into the tool. The MTK then translates this

SQL Server DDL to the equivalent DB2 UDB DDL. Extraction requires a connection to the source database through ODBC or JDBC. Once the ODBC/JDBC connection is established to SQL Server, MTK will query the system catalog tables and extract the object definitions for use in the conversion process. *importing*, on the other hand, requires an existing file, or files, which contain database object DDL. The import task copies the DDL into the MTK project directory for use in the database structure conversion process. Using MTK to perform data movement is limited if the *importing* option is chosen.

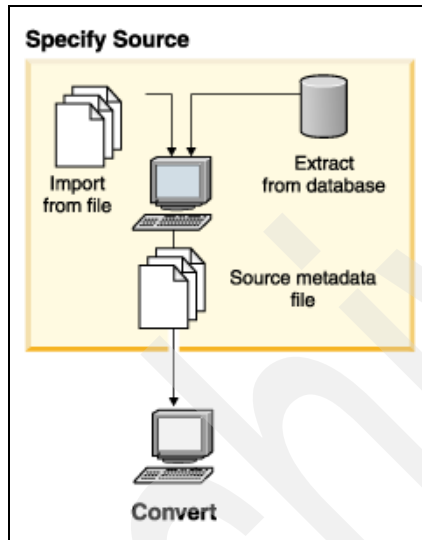


Figure 6-7 Specify source

► Task 2: Convert

During the *Convert* task (Figure 6-8), the user can choose to complete several optional tasks before the actual translation of the source DDL code. These tasks are:

- Selecting format options for the converted code. This includes keeping the source code as comments in the converted code; including the DROP statement before CREATE <object> statements, among others.
- Making changes to the default mapping between a source data type and its target DB2 UDB data type.

Once the optional tasks are completed, the user can click the Convert button and the source DDL statement is converted into DB2 UDB compatible DDL.

Each conversion generates two files:

- **.db2** file contains all of the source code converted to DB2 UDB code.

- **.rpt** file contains a log of MTK translation messages made during the conversion.

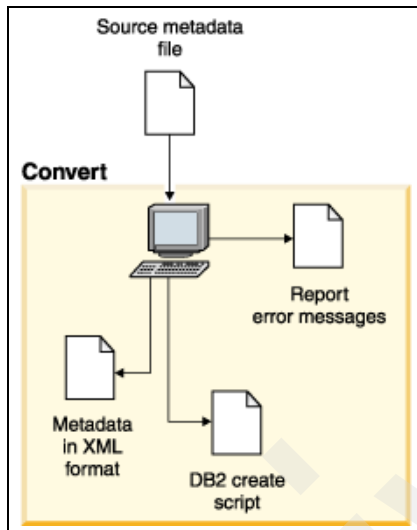


Figure 6-8 Convert

► Task 3: Refine

During the *Refine* task (Figure 6-9) the user may:

- Examine the results of the conversion.
- View various types of messages generated by the tool and, if necessary specify changes to be made to the converted DDL.

If any changes are made to the converted DDL in the Refine step, the user return to the Convert step to apply the changes.

Other, such as the SQL Translator, Log, and Reports to help you refine the conversion. After the DDL statements have been refined satisfactorily, the Generate Data Transfer Scripts step or the Deploy to DB2 step can be executed.

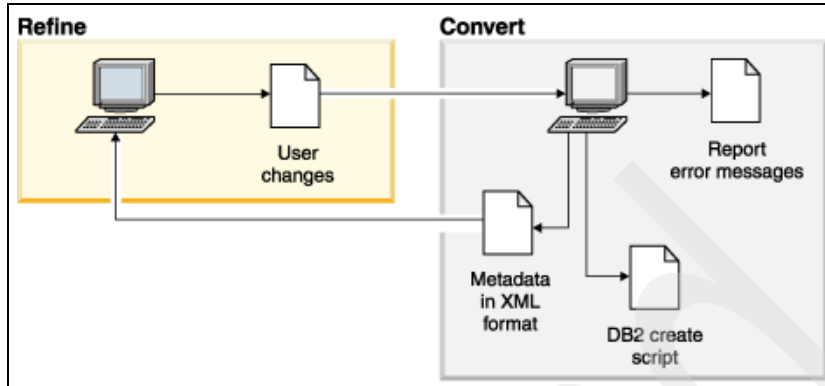


Figure 6-9 Refine

► Task 4: Generate data transfer scripts

In the *Generate data transfer scripts* task (Figure 6-10), scripts are generated that:

- Unload data from the source database.
- LOAD or IMPORT data into DB2 UDB.

Before creating the scripts, advanced options can be set that affect how the IMPORT or LOAD utilities operate. This allows the user to refine the LOAD or IMPORT specifications to correspond with the requirements of their data and environment.

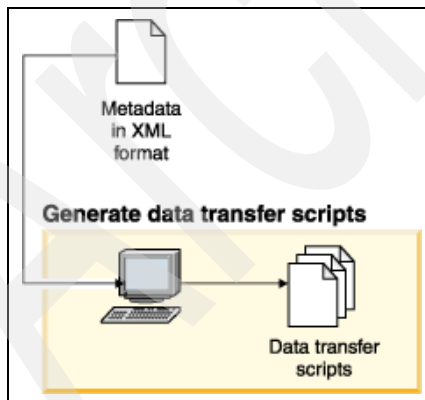


Figure 6-10 Generate data transfer script

► Task 5: Deploy to DB2

The *Deploy to DB2* task (Figure 6-11) is used to install database objects and *IMPORT/LOAD* data into the target DB2 UDB database. The various options in this task include:

- Choose to create a new database or install the objects in an existing database.
- Execute the DDL script to create the database objects.
- Extract data from the source database.
- *LOAD/IMPORT* the source data into the target DB2 UDB tables.

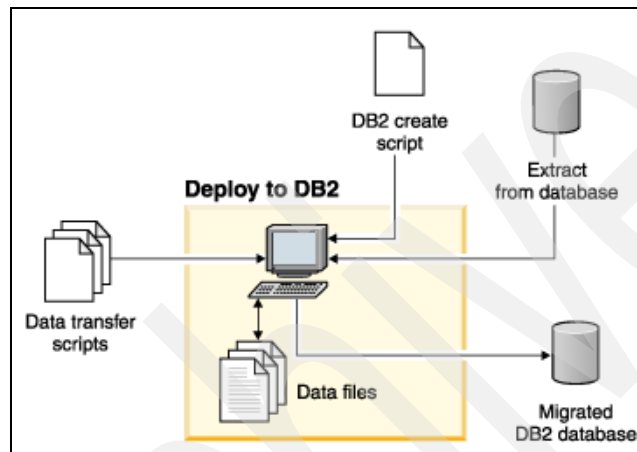


Figure 6-11 *Deploy to DB2*

An overview of all the tasks in the MTK conversion process is shown in Figure 6-12.

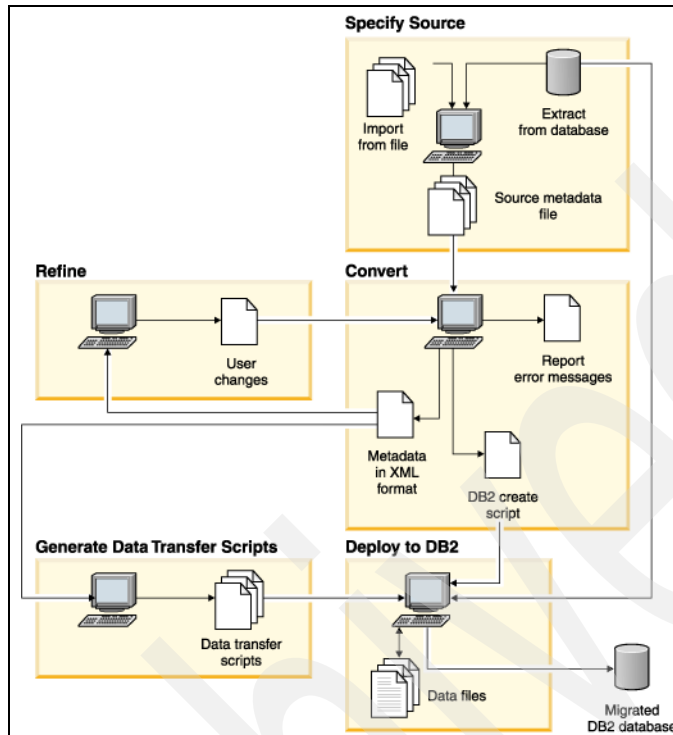


Figure 6-12 Overview of MTK conversion tasks

6.3.3 The MTK SQL Translator

The MTK SQL Translator (Figure 6-13) allows for the “on-the-fly” conversion of individual or compound SQL statements. The SQL Translator requires that all dependent objects referenced in the SQL statements are available to MTK. This may be accomplished in either of two ways:

- ▶ The current MTK project already contains all of the *converted* dependent objects (tables, views, etc.)
- ▶ The dependent objects *will be created* in the SQL Translator window by placing them before the SQL statements that reference them.

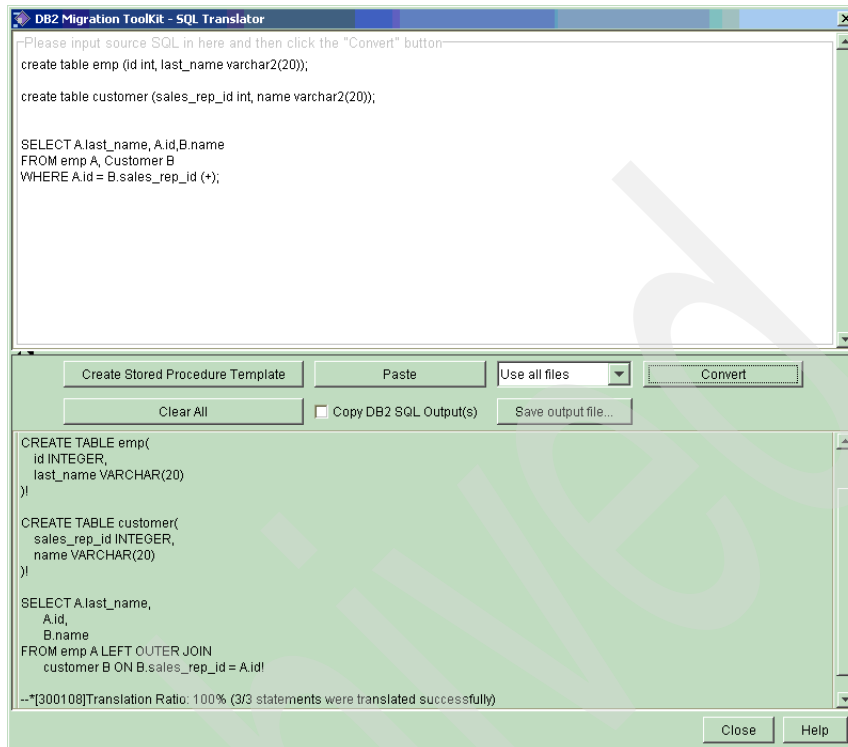


Figure 6-13 MTK SQL Translator

The SQL Translator can be used for application code translation as well. You can copy SQL statements from your application, paste it into the editing area of the window, and click **Convert** to convert the SQL statements into DB2 UDB supported syntax. You can then copy the converted code and paste it back to your application. This makes your conversion effort easier.

Database structure conversion

In this chapter we discuss the process of converting the database structure from SQL Server to DB2 UDB. We first examine the database structure differences between SQL Server and DB2 UDB. We then provide conversion examples using manual conversion methods and the IBM DB2 Migration Toolkit (MTK).

7.1 Databases

In both SQL Server and DB2 UDB, you can use a GUI tool or the command line to create a database. The `CREATE DATABASE` statement has syntax differences, thereby requiring modification if databases are being manually created using command line. MTK can automatically create the target database if it is installed on the target database server.

By default, the database is created in the default database path specified in the database manager configuration file (`DFTDBPATH` parameter). In Windows, this parameter specifies a drive. The following command creates a database called `REDBOOK` using all the default options:

```
CREATE DATABASE redbook
```

By default, DB2 UDB creates a database with three default table spaces, `SYSCATSPACE`, `TEMPSPACE1`, and `USERSPACE1`.

7.2 Table spaces

In DB2 UDB, data is physically stored in table spaces. The different types of table spaces are explained in 2.4.2, “DB2 UDB disk allocation” on page 18. If MTK is used for the conversion, it automatically creates the required table spaces. You have the option of using the MTK-generated table spaces, or modifying the table space syntax MTK generated. For example, you can modify a table space name generated by MTK to one of your choice. During a manual conversion effort, you will need to add table space create statements to the script if the default table spaces do not suffice.

DB2 UDB uses four types of table spaces depending on the data to be stored:

- ▶ `REGULAR` - used for tables, indexes, and system catalog tables.
- ▶ `SYSTEM TEMPORARY` - used by the database for sorts, joins and other operations.
- ▶ `USER TEMPORARY` - used for global declared temporary tables.
- ▶ `LARGE` - used to store Large Object Data (LOB). This can only be created using DMS table space.

The basic `CREATE TABLESPACE` statement syntax is shown in Example 7-1:

Example 7-1 DB2 UDB CREATE TABLESPACE statement

Syntax:

```
CREATE {REGULAR|SYSTEM TEMPORARY|USER TEMPORARY|LARGE} TABLESPACE <tablespace  
name> .....
```

Sample statement:

```
CREATE TABLESPACE myts1 MANAGED BY SYSTEM
  USING ('d:\sj_tbsp','e:\sj_tbsp')
  EXTENTSIZE 64
  PREFETCHSIZE 32
)
```

SQL Server uses a fixed page size of 8KB for all its tables, whereas DB2 UDB supports multiple page sizes: 4KB (default), 8KB, 16KB, and 32KB. If any row in a table in the source database has a length greater than 3,985 bytes, table space with an 8KB page size or larger is required in DB2 UDB.

To create a table space with an 8KB page size, a buffer pool with an 8KB page size must exist. Example 7-2 shows how a buffer pool and table space using an 8KB page size can be created:

Example 7-2 Creating a buffer pool and table space using an 8KB page size

```
CREATE BUFFERPOOL bp_8k IMMEDIATE SIZE 250 PAGESIZE 8 K;

CREATE REGULAR TABLESPACE userspace_8k PAGESIZE 8 K MANAGED BY
  SYSTEM USING ('C:\DB2\NODE0000\redbook\userspace_8k' )
  BUFFERPOOL bp_8k;
```

7.3 Tables

In SQL Server, the CREATE TABLE statement may include the database name, owner and table name. The database name is optional and defaults to the current database name.

In DB2 UDB, a fully qualified table name consists of the schema name and table name. In the following two examples:

```
DB2ADMIN.CLASS
ITS0.PRICE
```

DB2ADMIN and ITS0 are schema names, while CLASS and PRICE are table names.

If no schema is specified with the CREATE TABLE statement, the default schema is used (the ID of the user creating the table). Table 7-1 shows an example comparing the CREATE TABLE syntax between SQL Server and DB2 UDB:

Table 7-1 CREATE TABLE syntax - compare between SQL Server, DB2 UDB

SQL Server	DB2 UDB
<pre>CREATE TABLE [employee] ([empid] INT, [name] VARCHAR(40) NOT NULL, [job] VARCHAR(15) NOT NULL, [hire_date] DATETIME NOT NULL, [department] INT NULL, [basic salary] DECIMAL(8,2) NULL, [commission] DECIMAL(8,2) NULL)</pre>	<pre>CREATE TABLE employee (empid INT, name VARCHAR(40) NOT NULL, job VARCHAR(15) NOT NULL, hire_date TIMESTAMP NOT NULL, department INT, basic_salary DECIMAL(8,2), commission DECIMAL(8,2))</pre>

The DB2 UDB CREATE TABLE statement can specify different table spaces for indexes, tables, and LOBs, if DMS table spaces are used. Example 7-3 illustrates how to specify the desired table space in the CREATE TABLE statement, using the IN clause:

Example 7-3 DB2 UDB CREATE TABLE syntax

Syntax:

```
CREATE TABLE <tablename> (<column list>) IN <tablespace name for regular data>  
INDEX IN <tablespace name for index> LARGE IN <tablespace name for LOB data>
```

Sample statement:

```
CREATE TABLE itso.mytab1(  
    Col1 INT,  
    Col2 CHAR(3))  
IN mytbs1
```

7.3.1 Data types

While SQL Server and DB2 UDB implement many of the same data types, there are several differences between the built-in data types in both database systems. A data type mapping table is provided in Appendix B, “Data type mapping” on page 469 for your reference. MTK performs this data type mapping automatically, although it is possible to change the default mapping used by MTK.

7.3.2 IDENTITY

The SQL Server IDENTITY property can be specified in the table definition to provide system-generated values in sequence. DB2 UDB also provides an IDENTITY attribute that can be specified in the table definition. The syntax used in DB2 UDB differs from SQL Server.

DB2 UDB only allows a single column in a table to be defined as **IDENTITY**. If unique sequences are required for additional columns, sequence objects can be used. Unlike **IDENTITY** columns, sequence objects are separate database objects and are not defined in the table definition.

Table 7-2 shows an example comparing the **IDENTITY** column syntax differences between SQL Server and DB2 UDB:

Table 7-2 *IDENTITY column syntax - compare between SQL Server, DB2 UDB*

SQL Server	DB2 UDB
<pre>CREATE TABLE [employee] ([empid] INT IDENTITY, [name] VARCHAR(40) NOT NULL, [job] VARCHAR(15) NOT NULL, [hire_date] DATETIME NOT NULL, [department] INT NULL, [basic salary] DECIMAL(8,2) NULL, [commission] DECIMAL(8,2) NULL)</pre>	<pre>CREATE TABLE employee(empid INT GENERATED ALWAYS AS IDENTITY, name VARCHAR(40) NOT NULL, job VARCHAR(15) NOT NULL, hire_date TIMESTAMP NOT NULL, department INT, basic_salary DECIMAL(8,2), commission DECIMAL(8,2))</pre>

The **IDENTITY_VAL_LOCAL()** function can be used to retrieve the last generated identity value. Note that this function returns the last value used in the same unit of work.

For more information about generating unique values in DB2 UDB, refer to the following article:

<http://www.ibm.com/developerworks/db2/library/techarticle/0205pilaka/0205pilaka2.html>

7.3.3 Computed columns

SQL Server computed columns are not stored in the table, and as the name suggests, are calculated using the expression that defines the column value when needed. Indexes can be created on computed columns, but there are restrictions such as, the expression must be deterministic.

In contrast, DB2 UDB generated columns are defined in a base table where the stored value is computed using an expression. One or more generated columns can be added to a table. It is also possible to create non-unique indexes on a generated column.

Table 7-3 shows the differences in syntax between a SQL Server computed column and a DB2 UDB generated column.

Table 7-3 Computed column syntax - compare between SQL Server, DB2 UDB

SQL Server	DB2 UDB
<pre>CREATE TABLE [employee] ([empid] INT IDENTITY, [name] VARCHAR(40) NOT NULL, [job] VARCHAR(15) NOT NULL, [hire_date] DATETIME NOT NULL, [department] INT NULL, [basic_salary] DECIMAL(8,2) NULL, [commission] AS ([basic_salary] * 0.5))</pre>	<pre>CREATE TABLE employee(empid INT GENERATED ALWAYS AS IDENTITY, name VARCHAR(40) NOT NULL, job VARCHAR(15) NOT NULL, hire_date TIMESTAMP NOT NULL, department INT, basic_salary DECIMAL(8,2), commission GENERATED ALWAYS AS (basic_salary * 0.5))</pre>

7.3.4 Constraints

Both SQL Server and DB2 UDB support column constraints on tables.

Unique

There is no difference between unique constraints in SQL Server and DB2 UDB. No syntax changes are required when converting these type of constraints.

Primary key

The only difference in primary key constraint syntax is that DB2 UDB requires the NOT NULL keywords to be added. Table 7-4 shows an example highlighting the difference in syntax between SQL Server and DB2 UDB.

Table 7-4 Primary key syntax - compare between SQL Server and DB2 UDB

SQL Server	DB2 UDB
<pre>CREATE TABLE [employee] ([empid] INT IDENTITY PRIMARY KEY, [name] VARCHAR(40) NOT NULL, [job] VARCHAR(15) NOT NULL, [hire_date] DATETIME NOT NULL, [department] INT NULL, [basic_salary] DECIMAL(8,2) NULL, [commission] AS ([basic_salary] * 0.5))</pre>	<pre>CREATE TABLE employee(empid INT GENERATED ALWAYS AS IDENTITY NOT NULL PRIMARY KEY, name VARCHAR(40) NOT NULL, job VARCHAR(15) NOT NULL, hire_date TIMESTAMP NOT NULL, department INT, basic_salary DECIMAL(8,2), commission GENERATED ALWAYS AS (basic_salary * 0.5))</pre>

Check constraints

SQL Server and DB2 UDB support column level and table level check constraints. The syntax for these constraints is very similar. Formatting related

constraints are not natively supported in DB2 UDB; however, they can be simulated using a user-defined function (UDF). If the MTK is used to convert the database structure, it automatically creates the necessary UDF for you. Table 7-5 shows a SQL Server table definition with a CHECK constraint and the DB2 UDB implementation using a UDF that is created by MTK. For more details about the MS7.ISLIKE() function, refer to Appendix C, “Function mapping” on page 481.

Table 7-5 Check constraint syntax - compare between SQL Server, DB2 UDB

SQL Server	DB2 UDB
<pre>CREATE TABLE [employee] ([empid] INT IDENTITY CONSTRAINT [CK_emp_id] CHECK ([emp_id] LIKE '[A-Z][A-Z]-[0-9][0-9][0-9][0-9]'), [name] VARCHAR(40) NOT NULL, [job] VARCHAR(15) NOT NULL, [hire_date] DATETIME NOT NULL, [department] INT NULL, [basic salary] DECIMAL(8,2) NULL, [commission] AS ([basic salary] * 0.5))</pre>	<pre>CREATE TABLE employee(empid INT GENERATED ALWAYS AS IDENTITY CONSTRAINT CK_emp_id CHECK (MS7.isLike(CAST(emp_id AS CHAR(8)), '[A-Z][A-Z]-[0-9][0-9] [0-9][0-9]', '')= 1)), name VARCHAR(40) NOT NULL, job VARCHAR(15) NOT NULL, hire_date TIMESTAMP NOT NULL, department INT, basic_salary DECIMAL(8,2), commission GENERATED ALWAYS AS (basic_salary * 0.5))</pre>

Referential constraints

SQL Server referential constraints have two possible actions: CASCADE and NO ACTION (default).

DB2 UDB referential constraint definitions have four possible actions: NO ACTION (the default), RESTRICT, CASCADE, and SET NULL.

When converting to DB2 UDB, no change in syntax is required.

Note: DB2 UDB informational constraints are constraints on tables that are not enforced by the database manager, but can still be exploited by the optimizer. The attributes ENFORCED and NOT ENFORCED define whether the constraint is enforced by the database manager during data modification operations. NOT ENFORCED should only be specified if the table data is independently known to conform to the constraint. Furthermore the ENABLE QUERY OPTIMIZATION and DISABLE QUERY OPTIMIZATION attributes define whether the constraint can be used for query optimization under appropriate circumstances.

7.4 Indexes

SQL Server uses a B-tree data structure to store clustered and non-clustered indexes. However, non-clustered indexes are not ordered physically according to the index keys and the leaf nodes consist of index rows rather than data pages.

DB2 UDB creates indexes in a separate data structure that replicates the keys' values. A B+ tree data structure is used to store indexes. To maintain the cluster factor of a clustered index or improve it dynamically as data is inserted into the associated table, DB2 UDB attempts to insert new rows physically close to the rows with index key values in the same range.

In both SQL Server and DB2 UDB, only one clustered index per table is permitted. There is a small difference in syntax used to create the indexes. In DB2 UDB, the CLUSTERED clause proceeds the index definition, as the example in Table 7-6 shows.

Table 7-6 Clustered index syntax-compare between SQL Server and DB2 UDB

SQL Server	DB2 UDB
CREATE CLUSTERED INDEX [PK_author_id] ON [authors] ([author_id] ASC)	CREATE INDEX PK_author_id ON authors (author_id ASC) CLUSTER

In a DB2 UDB non-clustered index, the NONCLUSTERED clause is assumed by default. The example in Table 7-7 shows the difference in syntax for a non-clustered index between SQL Server and DB2 UDB.

Table 7-7 Non-clustered index syntax-compare between SQL Server, DB2 UDB

SQL Server	DB2 UDB
CREATE NONCLUSTERED INDEX [idx_authors] ON [authors] ([name] ASC, [firstname] ASC)	CREATE INDEX idx_authors ON authors (name ASC, firstname ASC)

Note: DB2 UDB indexes can be either type-1 or type-2. A type-1 index is only used to support DB2 UDB versions prior to V8. The primary advantages of type-2 indexes are to improve concurrency because the use of next-key locking is reduced to a minimum and an index can be created on columns that have a length greater than 255 bytes. A table must have only type-2 indexes before online table **reorg** and online table **load** commands can be issued against the table. They are also required for multidimensional clustering (MDC) tables. By default, all new indexes created in DB2 UDB V8 are type-2 indexes, except if a type-1 index already exists on a table.

7.5 Schemas

In SQL Server, a schema that can be thought of as a conceptual object containing definitions of tables, views, and permissions. The `CREATE SCHEMA` statement is used for providing a way to create tables and views and to grant permissions of these objects with a single statement. The created objects do not have to appear in logical order, except for views that reference other views. Example 7-4 shows a typical implementation of the `SCHEMA` statement in SQL Server.

Example 7-4 CREATE SCHEMA in SQL Server

```
CREATE SCHEMA AUTHORIZATION kishore
GRANT SELECT ON v1 TO PUBLIC
CREATE VIEW v1(c1) AS SELECT c1 FROM t1
CREATE TABLE t1(c1 INT)
```

A DB2 UDB schema is a logical object which can be used to group database objects together. The naming convention used for all database objects uses both the schema name and the object name (e.g., `SCHEMA.OBJECT`). The default schema for all users is their user name (e.g., `JDOE.TABLE`). The default schema is used to resolve all object references where no schema is used (e.g., for user `JDOE`, the statement `SELECT * FROM EMPLOYEE` would be equivalent to the statement `SELECT * FROM JDOE.EMPLOYEE`). The default schema can be changed for the duration of a database connection using the `SET SCHEMA` command.

You can use the DB2 UDB `CREATE SCHEMA` statement to create a schema, then create other objects under it. Example 7-5 shows a simple schema declaration and use in DB2 UDB.

Example 7-5 Schema declaration and use in DB2 UDB

```
CREATE SCHEMA billing AUTHORIZATION user1;
CREATE TABLE billing.orders (.....,.....)
```

or

```
CREATE SCHEMA billing AUTHORIZATION user1;
SET SCHEMA billing;
CREATE TABLE orders (.....,.....)
```

SQL Server `CREATE SCHEMA` statements therefore do not directly correspond to DB2 UDB `CREATE SCHEMA` statements. To convert them to DB2 UDB, create the corresponding object definitions first, then assign all required privileges on them. Example 7-6 shows the equivalent DB2 UDB statements of those in Example 7-4.

Example 7-6 DB2 UDB implementation of SQL Server's CREATE SCHEMA statement

```
CREATE TABLE kishore.t1(c1 INT);
CREATE VIEW kishore.v1(c1) AS SELECT c1 FROM t1;
GRANT SELECT ON kishore.v1 TO PUBLIC;
```

7.6 Views

There is little difference in view syntax between SQL Server and DB2 UDB. However, the SELECT statement the view is based on may require some changes. Table 7-8 shows an example where such a change is necessary. In this example, SQL Server column aliases must be changed to match DB2 UDB's column alias syntax.

Table 7-8 View syntax - compare between SQL Server and DB2 UDB

SQL Server	DB2 UDB
<pre>CREATE VIEW v_redbook AS SELECT title, Authors=authors.name, brand_name, price, release_date FROM authors, redbooks, brands WHERE authors.author_id = redbooks.author_id AND redbooks.brand_id = brands.brand_id</pre>	<pre>CREATE VIEW v_redbook AS SELECT title, authors.name AS Authors, brand_name, price, release_date FROM authors, redbooks, brands WHERE authors.author_id = redbooks.author_id AND redbooks.brand_id = brands.brand_id</pre>

7.7 Privileges

SQL Server restricts access to objects and commands based on a user's identity or group membership. Database permissions can be assigned directly to Windows NT and Windows 2000 users. The SQL statements GRANT and REVOKE authorize or prevent users from accessing database objects. Most of SQL Server's permissions can be mapped to DB2 UDB privileges.

In DB2 UDB, privileges are stored in the database catalog for each database. The following types of object privileges exist: database, schema, table space, table, view, nickname, server, package, index, routine and sequence. The table space USE privilege controls the table spaces users can create tables in. The GRANT and REVOKE statements can be executed to assign or to remove privileges

from a given user. Some privileges in DB2 UDB are not available in SQL Server, such as package and schema privileges.

SQL Server has a fixed server role called sysadmin with permissions to perform any activity. User accounts can become members of the sysadmin role. The built-in login system administrator (SA) is only provided for backward compatibility. In DB2 UDB, the System Administration (SYSADM) authority is the highest level of authority for an instance, and controls all instance objects.

One task involved in converting the database structure involves identifying the privileges required by each user ID that connects to the database. In SQL Server, database object privileges are granted to Roles and Users. Privilege information can be collected from the extracted SQL Server code or from Enterprise Manager. Table 7-9 shows a simple example outlining the Roles and Users defined for the REDBOOK database.

Table 7-9 SQL Server user list for the REDBOOK database

Roles	Users
librarian	user1, user2, user3, user4
developer	user5, user6, user7, user8
appuser	user10, user9, user18, user15

Once this list is prepared, the equivalent user IDs and groups can be created on the target DB2 UDB server, or wherever user authentication takes place.

Table 7-10 shows the object privileges required for the Roles/Users defined in Table 7-9.

Table 7-10 DB2 UDB object privilege list for the REDBOOK database

Roles/Users	Objects	Privileges
librarian	table1, view1	SELECT, INSERT, REFERENCE, etc.
user1	UDF1, stored_procedure1	EXECUTE
appuser	database	USE

Based on this information, the equivalent DB2 UDB GRANT statements can be issued to give the corresponding DB2 UDB user IDs the correct privileges.

```
GRANT SELECT, INSERT ON TABLE table1 TO librarian
...
GRANT EXECUTE ON PROCEDURE stored_procedure1 TO user1
```

7.8 Examples

In this section, we show the structure conversion of a sample SQL Server database, called REDBOOK, using MTK and a manual process. A SQL Server REDBOOK database DDL script is provided as additional material. Refer to Appendix G, “Additional material” on page 509 for download instructions. Run this script to create the REDBOOK database in your SQL Server environment.

7.8.1 Database structure conversion using MTK

Before beginning the conversion, an empty database called REDBOOK must be created in DB2 UDB:

- ▶ Open a DB2 Command window:
Start → Programs → IBM DB2 → Command Line Tools → Command Window
- ▶ Issue the following command:
db2 create database redbook

Launch MTK and create a project called RBOOK. Select the source REDBOOK database in SQL Server. In the *Extract* step, extract the database definition statements from SQL Server and save them to a file.

Example 7-7 shows a snippet of what is contained in this file.

Example 7-7 SQL Server REDBOOK database code extracted by MTK

```
-- Extractor has started
-- Version 1.3, build: 041209.0332
-- Wed Feb 16 16:49:19 PST 2005
-- Source JDBC Connection = sqlserver
-- UserID = kishore
-- include dependencies
use redbook
go

SETUSER 'dbo'
go

-- write user-defined types
exec sp_addtype [t_empid], "char(8)","NOT NULL"
go

...

CREATE TABLE [authors]
```



```

([author_id] [t_id] NOT NULL CONSTRAINT [CK_author_id] CHECK ([author_id]
like '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9]'),
[name] varchar(40) NOT NULL ,
[firstname] varchar(20) NOT NULL ,
[phone] char(12) DEFAULT ('n/a') NOT NULL ,
[address] varchar(40) NULL ,
[city] varchar(20) NULL ,
[state] char(2) NULL ,
[zip] char(5) NULL CHECK ([zip] like '[0-9][0-9][0-9][0-9][0-9]'),
[contract] bit NOT NULL )
go

CREATE CLUSTERED INDEX [PK_author_id]
ON [authors] ( [author_id] ASC )
go

CREATE NONCLUSTERED INDEX [idx_authors]
ON [authors] ( [name] ASC, [firstname] ASC )
go
ALTER TABLE [authors] ADD PRIMARY KEY ( [author_id] ASC )
go
...

```

In the *Convert* step, convert the SQL Server database object definitions into equivalent DB2 UDB object definitions. Save these converted object definitions in a different file. We recommend that you examine all the error/warning messages in the generated code and try to correct the errors.

Example 7-8 shows a snippet of the DB2 UDB object definitions generated by MTK.

Example 7-8 DB2 UDB object definitions generated by MTK

```

--* [300305] DBC Version [040625.0305]. Infos: Microsoft SQL Server, Case
sensitive
--| -- Extractor has started
--| -- Version 1.3, build: 041209.0332
--| -- Wed Feb 16 16:49:19 PST 2005
--| -- Source JDBC Connection = sqlserver
--| -- UserID = kishore
--| -- include dependencies
--|
--| use redbook
--| go
--|
--| SETUSER 'dbo'
--| go
--|

```

```

--| -- write user-defined types
--| exec sp_addtype [t_empid], "char(8)","NOT NULL"

CREATE DISTINCT TYPE t_empid AS CHAR(8) WITH COMPARISONS!

...

--| CREATE TABLE [authors]
--|   ([author_id] [t_id] NOT NULL CONSTRAINT [CK_author_id] CHECK
--|   ([author_id] like '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9]'),
--|   [name] varchar(40) NOT NULL ,
--|   [firstname] varchar(20) NOT NULL ,
--|   [phone] char(12) DEFAULT ('n/a') NOT NULL ,
--|   [address] varchar(40) NULL ,
--|   [city] varchar(20) NULL ,
--|   [state] char(2) NULL ,
--|   [zip] char(5) NULL CHECK ([zip] like '[0-9][0-9][0-9][0-9][0-9]'),
--|   [contract] bit NOT NULL )

CREATE TABLE authors (
  author_id t_id NOT NULL,
  name VARCHAR(40) NOT NULL,
  firstname VARCHAR(20) NOT NULL,
  phone CHAR(12) DEFAULT 'n/a' NOT NULL,
  address VARCHAR(40),
  city VARCHAR(20),
  state CHAR(2),
  zip CHAR(5),
  contract SMALLINT NOT NULL,
  CONSTRAINT CK_author_id
  CHECK (MS7.isLike(CAST(author_id AS VARCHAR(11)),
    '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]', '') = 1),
  CHECK (MS7.isLike(zip, '[0-9][0-9][0-9][0-9][0-9]', '') = 1))!

--|
--| go
--| CREATE CLUSTERED INDEX [PK_author_id]
--| ON [authors] ( [author_id] ASC )
CREATE INDEX PK_author_id ON authors (author_id ASC) CLUSTER!
--|
--| go
--|
--| CREATE NONCLUSTERED INDEX [idx_authors]
--| ON [authors] ( [name] ASC, [firstname] ASC )

CREATE INDEX idx_authors ON authors (name ASC, firstname ASC)!
--|
--| go
--|

```

```
--| CREATE NONCLUSTERED INDEX [idx_redbooks_author_id]
--| ON [authors] ( [author_id] ASC )

--* [300198] ""C:\MTK\projects\structure\structure.src""(42:27)-(42:50)
Translator Information: The object name has been changed to idx_redbooks_autho.
CREATE INDEX idx_redbooks_autho ON authors (author_id ASC)!
--|
--| go
--|
--| ALTER TABLE [authors] ADD PRIMARY KEY ( [author_id] ASC )

ALTER TABLE authors
ADD PRIMARY KEY (author_id)!
```

The changes made by MTK in this code include:

- ▶ Old code is left as comments.
- ▶ LIKE operator is changed to MS7.isLike() function.
- ▶ In CREATE INDEX statements, the CLUSTER keyword is moved to the end of the statement.
- ▶ The index called idx_redbooks_author_id is changed to idx_redbooks_autho.

Open the converted file, make any necessary corrections and re-save it.

In the *Deploy to DB2* step, specify the database named REDBOOK, whether it is a local or remote database, and the user ID and password required to connect. Select *Launch <DB2_objects_file_name> in the database* and click *Deploy*. This creates all the converted objects in the DB2 UDB REDBOOK database.

7.8.2 Manually converting database structure

The manual process of converting the database structure includes:

1. Extract object definitions from the source database and save them to a file.
2. Make changes to the extracted code to make it compatible with DB2 UDB syntax.
3. Create the target database in DB2 UDB, if the script does not already do so.
4. Execute the converted object definition code in a DB2 Command Window or Command Editor to create the objects in the DB2 UDB database.

Extract DDL from source database

To extract the object definitions manually from SQL Server, follow the steps below:

1. Launch the SQL Server Enterprise Manager.
2. Expand the tree in the left pane of the window until the available database names are visible.
3. Right-click the database you want to convert (for this example, it is the database called REDBOOK).
4. Select **All Tasks** → **Generate SQL Script...**
5. In the *General* tab, click **Show All**.
6. Select **All Tables**, **All Views** and **All user-defined data types**.
We are only performing a database structure conversion in this step, otherwise all the database objects could be selected.
7. In the *Formatting* tab, select **Generate the CREATE <object> command for each object**.
8. In the *Options* tab, select **Script Index** and **Script Primary Key...**
9. Click **OK**.
10. Save the results to a file. In this example, we call the file *redbook.src*. Make a copy of *redbook.src* called *redbook.db2*

Convert database definition script

Open the *redbook.db2* file in your favorite text editor:

- ▶ Remove all the square brackets from the object names.
- ▶ Remove the GO keyword and add a statement terminator character at the end of each SQL statement. We use exclamation symbol (!).

Note: You can use any terminator character at the end of your statement. This can be specified using `-td<terminate_character>` flag of CLP or in the provided text field in Command Editor.

- ▶ Add NOT NULL to column definitions wherever a primary key is defined.
- ▶ Convert CHECK constraints as required.
- ▶ Remove spaces from table, index, view, and column names.
- ▶ Check the length of constraint names. They should not exceed 18 characters. (Refer to Appendix F, “SQL limits” on page 505)

- ▶ Make appropriate changes in SELECT statements in VIEW definitions. More details about SQL statements can be found in Chapter 4, “SQL considerations” on page 65.
- ▶ Make appropriate changes in CREATE INDEX statements.

Create the DB2 UDB database

The extracted SQL Server DDL did not include a database creation statement. Create the REDBOOK database in DB2 UDB using the CLP with the following command:

```
CREATE DATABASE redbook
```

Alternatively, you can create the database using the Create Database wizard in Control Center.

Execute the converted DDL in DB2 command window

Execute the DB2 UDB compatible object definition script:

- ▶ Open a DB2 Command window:
Start → Programs → IBM DB2 → Command Line Tools → Command Window
 Or,
Start → Run, enter **db2cmd** and click **OK**
- ▶ Navigate the folder in which you have saved your script.
 - Connect to the REDBOOK database:
DB2 CONNECT TO redbook
 - Execute the script file using the following command:
db2 -td! -vf redbook.db2

Alternatively, you can load this script in Command Editor and execute it from there.

Data and script migration

In this chapter, we guide you through migrating data and scripts from SQL Server to DB2 UDB. You must ensure that data is migrated completely, consistently, and within a reasonable time frame. Data migration is not a trivial task. Proper planning is important.

Both source and target systems can be set up to run concurrently, keeping both data sources synchronized and minimizing downtime while bringing the target data source to production.

This chapter includes:

- ▶ Moving through the data migration process
 - Migrating a subset of data using the IBM DB2 Migration Toolkit (MTK)
 - Verifying referential integrity of data
- ▶ Using the DB2 UDB EXPORT, IMPORT, and LOAD utilities
- ▶ Using WebSphere Information Integrator
 - Access SQL Server data from DB2 UDB
 - Replicate data from SQL Server to DB2 UDB
- ▶ Using other tools
- ▶ Converting scripts

8.1 Introduction

Data migration is the process of moving data from a source database to a target database. In general, data must be retrieved from (also referred to as *extracted* or *exported*) the source database into one or more files, then imported or loaded from the files into the target database.

Modern DBMSs provide built-in utilities for moving data. In SQL Server, the Bulk Copy Program (BCP) and Data Transformation Services (DTS) are two such utilities for importing and exporting data from SQL Server databases. DB2 UDB provides the EXPORT utility for extracting data from a DB2 UDB database, and the LOAD and IMPORT utilities for importing data into a DB2 UDB databases. A combination of these utilities can be used to complete the data migration.

Data migration is often not trivial because of data type dissimilarities and the existence of large volumes of data. Other tools, such as MTK, can simplify the process of data migration. Some of these tools are listed in 8.5, “Using other tools” on page 182.

8.2 A typical data migration process

Data migration is usually performed quite late in the conversion process. However, subsets of the data should be migrated with the database structure, in order to verify that everything is ported correctly. It is not a good use of your time to spend hours extracting and loading all your data, only to find out there are errors and that you must reload all the data from the beginning.

Migrate subsets of the production data and perform some tests. These tests include performing simple comparisons against the source and target databases, such as row counts. You can also identify if any transformations or adjustments are required and implement them before migrating the full set of data.

Once you are completely satisfied with the results of your tests, you can migrate the full set of production data. If you have plans to run both systems in parallel and keep both systems synchronized or replicated, then *WebSphere Information Integrator Replication Edition* can be used.

After moving all the production data from the source to target system, correct any referential integrity issues, then perform a thorough verification of the migrated data against the source data for accuracy and completeness. Figure 8-1 illustrates this process.

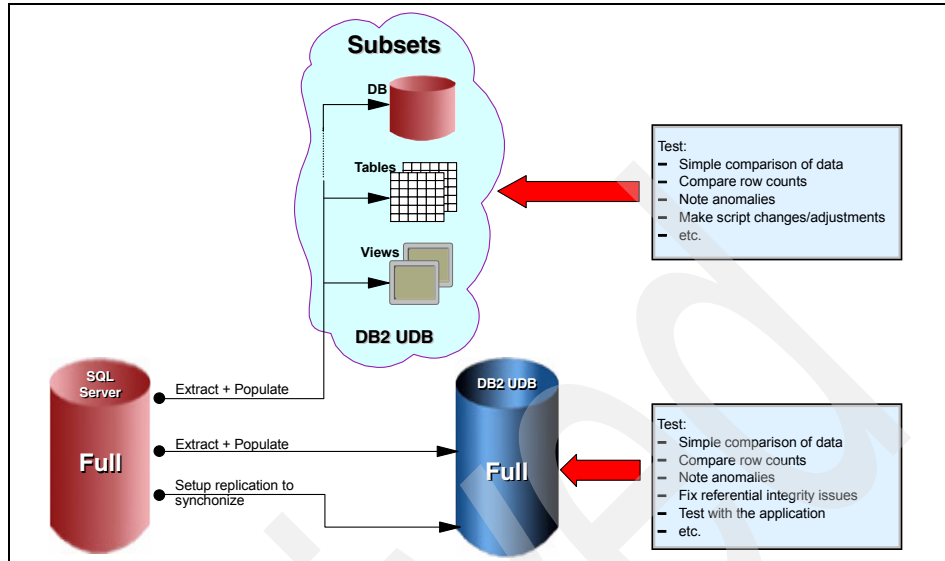


Figure 8-1 Suggested data migration process

8.2.1 Migrating a subset of data using MTK

In this section, we demonstrate how to migrate a subset of data from a sample SQL Server database called REDBOOK to DB2 UDB using MTK.

We assume that you have already created the equivalent REDBOOK database structure (refer to Chapter 7, "Database structure conversion" on page 151). Most importantly, you have already decided the data type mappings to use. The following steps outline how to migrate a subset of data using MTK:

1. Launch MTK.
2. In the project management window, enter a project name. For this example, enter REDBOOK.
3. Leave the project path and project description as the default values.
4. Choose **Microsoft SQL Server** as your source database, and **8.2 for Linux, UNIX, and Windows** as your target database, then click **OK**.
5. At the *Extract from source database* section in the *Specify Source* tab, click **Extract**.
6. A *Connect to Database* window appears. Fill in the corresponding JDBC/ODBC DSN Alias, and the user ID and password. Click **OK**.
7. Only select the tables in the REDBOOK database shown in Figure 8-2. Name the generated script redbook.src. Leave all other options as their default values.

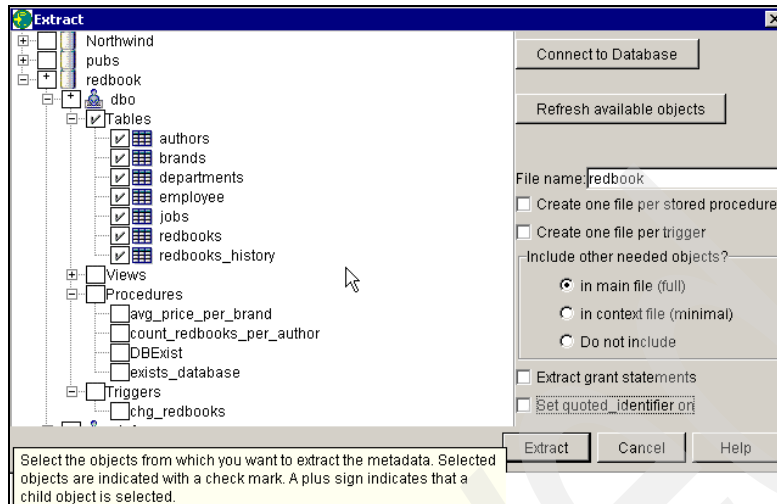


Figure 8-2 Extract table data in the REDBOOK database within SQL Server

8. Click **Extract**.
9. Select the *Convert* tab.
10. On the left pane, select **redbook.src**.
11. Click **Global Type Mapping**.
12. If your target database is not created to support double-byte character data (e.g., UTF-8), then the GRAPHIC and VARCHAR data types are unavailable. For this example, change the data type mapping of NCHAR(I), and NVARCHAR(I) to VARCHAR(I) instead. Figure 8-3 summarizes this mapping.

✓	NCHAR(128..max)(I)	✎	VARCHAR(I)
✓	NVARCHAR(I)	✎	VARCHAR(I)

Figure 8-3 Global Type Mapping summary

Note: GRAPHIC and VARCHAR are sequences of bytes that represent double-byte character data and are only available in unicode databases. DB2 UDB supports UTF-8 and UCS-2.

13. Leave all other options as their default values, and click **Convert**.
14. At the *Refine* tab, some translated information is shown. Error messages are placed here if MTK could not convert some of the database structure.
15. Click **Generate Data Transfer Scripts** tab (Figure 8-4).

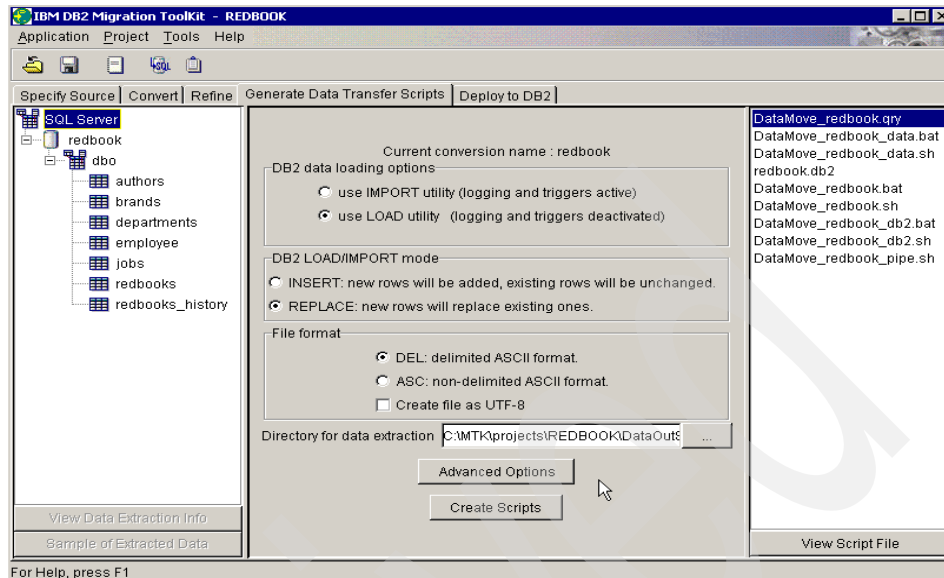


Figure 8-4 Generate Data Transfer Scripts in MTK

Note: To help you decide whether to use the **IMPORT** or **LOAD** utility in your environment, refer to DB2 UDB documentation “Appendix B” of *Data Movement Utilities Guide and Reference V8*, SC09-4830.

16. Select **Use LOAD utility** (logging and triggers are deactivated). Try to use **IMPORT** after this exercise and compare your results.
17. Click **Advanced Options**, and enter **20** in the *row count* field (Figure 8-5).

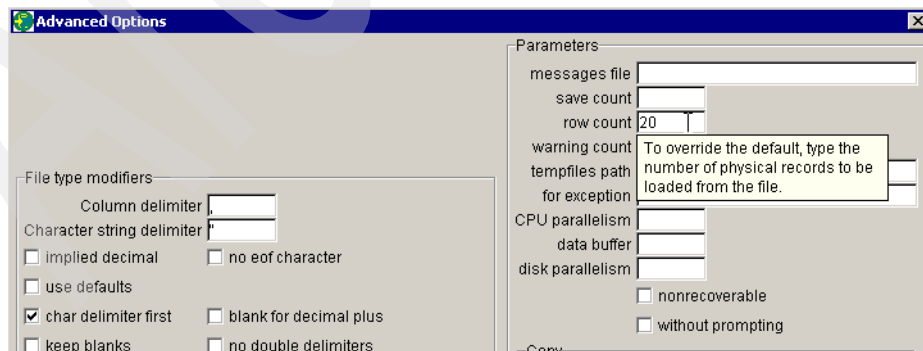


Figure 8-5 Migrating a subset of the database data using MTK

18. Click **OK** to exit the Advanced Options settings, then click **Create Scripts**.

Note: By indicating the row count value in the *Advance Option* settings, the LOAD commands generated will only load the indicated number of records in every table.

19. Deselect **Launch redbook.db2** in the database because we assume that the database structure has already been created in the DB2 UDB database.

20. Select **Extract and store data on this system** as well as **Load data to target database using generated scripts**.

You may want to extract the data, then load it, in two separate steps. Two step processing allows you to ensure the extracted data is good before loading it.

21. Click **Deploy**.

22. Your default Internet browser launches with the deployment results displayed.

23. Look at the deployment results and check if all the table records were successfully loaded into DB2 UDB.

Note: After loading subsets of data, you may want to pursue loading other subsets and continue assessing them. You can empty the tables quickly using the **LOAD** command with the **REPLACE** option with an empty input file.

Having loaded 20 or less rows of data, referential integrity issues may exist at this point. Refer to “Deactivating referential integrity checking” on page 175 for instructions to enable access to the data.

8.2.2 Verifying referential integrity

So far, we have used MTK to extract a subset of data from the SQL Server REDBOOK database and loaded it into DB2 UDB using the LOAD utility. You should perform some simple data comparison tests with the source data to ensure that data is migrated properly.

Note: The LOAD utility turns off constraint checking for self-referencing and dependent tables. This places the tables into *check pending* state. Once the data loading completes, MTK turns on the constraint checking for all the tables for which it was turned off.

Deactivating referential integrity checking

MTK automatically enables constraint checking by running the **SET INTEGRITY** command on all the relevant tables after LOAD operations. However, migrating just a portion of a database usually causes missing records to have referential integrity issues. You can deactivate referential integrity checking and access data immediately by using the **SET INTEGRITY** command:

```
SET INTEGRITY FOR <schema>.employees ALL FULL ACCESS IMMEDIATE UNCHECKED
```

Fixing referential integrity

At a later stage, ensure that all foreign key values are valid. If you choose to use the **LOAD** utility in MTK, it automatically checks referential integrity on all the loaded tables. Error reports are displayed if any violation is encountered. However, if you manually use the **EXPORT** and **LOAD** utilities, you need to activate integrity checking by manually running the **SET INTEGRITY** command.

For more information about the **SET INTEGRITY** command, refer to DB2 UDB documentation: *SQL Reference - Volume 2*, SC09-4845.

For more information about migrating data using the MTK, refer to the following article on the developerWorks Web site:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0411yip/>

Recommendation: Do not migrate the actual production database until you have first successfully migrated a subset of that database and tested it with your applications.

8.3 Using Export, Import, and Load

As previously shown, it is possible to customize scripts generated by MTK. MTK can even be used to create scripts that periodically refresh data in DB2 UDB database so that multiple data sources are synchronized.

SQL Server data movement tools

You can use the BCP command utility or DTS Import/Export wizard to extract data from SQL Server tables into text files, or directly to DB2 UDB via OLE DB or ODBC connectivity. If you export the data into text files, they must be in ANSI format (ASCII text), which is compatible with DB2 UDB's **IMPORT** and **LOAD** utilities. Once the data has been exported to files, you can use the **LOAD** or **IMPORT** utilities in DB2 UDB to populate the target database. For an example illustrating how to export data from a SQL Server table into a text file using BCP, refer to

Example 8-1. You can run BCP command utility from a Windows command prompt.

Example 8-1 Using the BCP utility to export data to a text file

BCP command:

```
C:\temp>BCP redbook.dbo.authors OUT authorsOUT.del -t , -r \n -e exportERR.txt
```

BCP command options:

-t , : column delimiter using comma
-r \n : row delimiter using next line

Output text file: [authorsOUT.del]

```
004-45-4108,Kaszinski,Santo,(588) 001-40,16 Ucptv Pkqx,Concord,Ne,,0  
028-14-6637,Filiberto Badertscher,Vincenzo,(102) 884-69,3411 Epsamx Tlejct,Trenton,Ne,11654,0  
...  
995-69-9813,Sari Berrocal,Erasmus,(525) 248-66,107 Qak Lt,Jefferson City,Mi,41107,0
```

DB2 UDB data movement utilities

DB2 UDB provides two utilities for loading data into a database, LOAD and IMPORT. You can access both utilities from Control Center, Command Editor, or the command line.

In general, the LOAD utility is faster than the IMPORT utility because it writes formatted pages directly into the database. The LOAD utility validates the uniqueness of the indexes, but it does not fire triggers or perform referential and constraint checking.

DB2 UDB LOAD utility

The LOAD utility is capable of efficiently importing large amounts of data into tables.

Example 8-2 shows the DB2 UDB equivalent operation as Example 8-1. The LOAD command options used in Example 8-2 are:

- ▶ **MODIFIED BY NOCHARDEL**: specifies that all bytes found between the column delimiters to be part of the column's data, without quotes (" ... ", " ... ").
- ▶ **REPLACE INTO**: deletes table contents before inserting data.

Example 8-2 Using the LOAD utility to import data from a text file

```
db2 => LOAD FROM authorsOUT.del OF DEL MODIFIED BY NOCHARDEL MESSAGES  
loadERR.txt REPLACE INTO stefan.authors
```

DB2 UDB IMPORT utility

The IMPORT utility inserts data from an input file into a table or a updatable view. If the table or view receiving the imported data already contains data, data can either be replaced or appended. Example 8-3 shows how the IMPORT utility can be used with a text file generated by BCP tool. The IMPORT command options used in the following example are:

- ▶ **MODIFIED BY NOCHARDEL**: Specifies that all bytes found between the column delimiters to be part of the column's data, without quotes (" ... ", " ... ").
- ▶ **REPLACE INTO**: Deletes table contents before inserting data.

Example 8-3 Using the IMPORT utility to import data from a text file

```
db2 => IMPORT FROM authorsOUT.del OF DEL MODIFIED BY NOCHARDEL REPLACE INTO  
      stefan.authors
```

For more information about the LOAD and IMPORT utilities, refer to the DB2 UDB documentation *Data Movement Utilities Guide and Reference*, SC09-4830.

8.4 Using WebSphere Information Integrator

WebSphere Information Integrator (WebSphere II) is an integration solution that provides a range of integration technologies such as data replication, data transformation, data and content federation, data event publishing, and enterprise search. This is a separate product from DB2 UDB, and is therefore installed separately. In the interest of migrating data over to DB2 UDB, we only focus on the heterogeneous data replication facility that this product offers.

WebSphere Information Integrator gives you the option of migrating the data in stages. Once WebSphere II is set up with nicknames for the SQL Server source tables, you can access them through DB2 UDB. This permits testing of the ported application without moving any data to DB2 UDB. Nevertheless, the ultimate goal is to move the data over to DB2 UDB. After that stage, you can continue to use WebSphere II to replicate data over to DB2 UDB and synchronize both databases until production moves over to the new system.

In this section, we provide a high level overview of how WebSphere II can be used to access SQL Server data from DB2 UDB, and how to replicate data over to DB2 UDB and synchronize the data in both databases.

8.4.1 Accessing SQL Server data from DB2 UDB

Note: WebSphere II is used in this context solely for migration efforts. However, using WebSphere II to access disparate data sources has a bigger objective - federation. With multiple heterogeneous data sources connected to WebSphere II, you can view all of them as though they are a single data source - simplifying data access, replication, enterprise search, and event publishing of heterogeneous data sources.

To enable a DB2 UDB database to access the sample REDBOOK database and tables defined in SQL Server, you need to perform the following steps using Control Center (CC), or Command Line Processor (CLP). Both methods follow the same general process.

CC can be used to quickly register SQL Server tables. The Create Nickname wizard can be launched by expanding the database of interest in CC. Right-click the **Nicknames** folder, then select the **Create...** options, as illustrated in Figure 8-6.

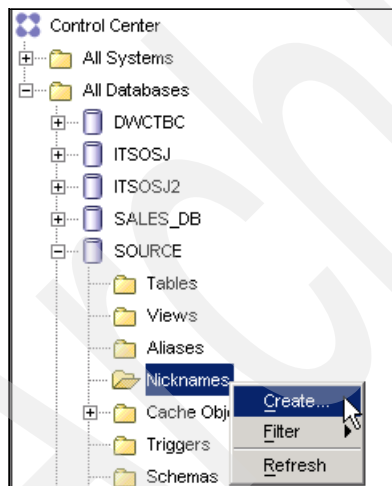


Figure 8-6 Creating nickname using Control Center

Using CLP, the equivalent steps need to be completed:

1. Create a wrapper

A wrapper is a library that allows access to a particular type of data source. It contains information about the remote data source (in this case SQL Server) characteristics and understands its capabilities.

Register a wrapper with the name “SQL Server” on the federated server by issuing the following statement:

```
CREATE WRAPPER SQLSERVER LIBRARY 'db2mssql3.dll'
```

Note: As an alternative, the ODBC wrapper ‘db2rcodbc.dll’ can be used.

2. Create a server.

A *server* represents a specific data source that is accessible through a wrapper. Each server can have its own set of server options. For example, the collating sequence of the SQL Server database can be specified. This affects the sort order used and helps determine if parts of a SQL statement should be pushed down and handled by the remote data source.

Register a server with the name REDBOOK that resides on SQL Server, using SQLSERVER as the wrapper name:

```
CREATE SERVER redbookSRV TYPE MSSQLSERVER VERSION '2000' WRAPPER  
"SQLSERVER" OPTIONS( ADD NODE 'sqlserver', DBNAME 'redbook');
```

For the NODE option, the value used should be the System DSN name for the SQL Server remote server being accessed. This value is case sensitive.

3. Create a user mapping.

A user mapping provides a mapping from a user ID that accesses the DB2 UDB server to a data source user ID and password that WebSphere II includes in connections to the remote data source on behalf of the DB2 UDB user.

The following example shows how to map a DB2 UDB server user ID to a SQL Server remote server user ID and password:

```
CREATE USER MAPPING FOR db2user1 SERVER redbooksSRV OPTIONS  
(REMOTE_AUTHID 'mssqluser1', REMOTE_PASSWORD 'password')
```

4. Create nicknames.

A nickname is a local alias of a database object (table, view, etc.) residing in SQL Server. Queries can reference nicknames as though they are local database objects in DB2 UDB.

To register a nickname for a SQL Server table named authors, the following statement can be issued:

```
CREATE NICKNAME asn.authors FOR redbook.dbo.authors
```

For more detailed instructions, refer to the WebSphere II documentation *IBM WebSphere Information Integrator Data Source Configuration Guide Version 8.2*.

Once you have successfully created nicknames for the REDBOOK database tables, you can issue DB2 UDB SQL statements that access data in the SQL Server REDBOOK database. The example in Table 8-1 illustrates how data that still resides in SQL Server can be accessed using a DB2 UDB query. The local schema name for the nicknames is ASN.

Table 8-1 Compare between query using nickname in SQL Server, DB2 UDB

SQL Server query	DB2 UDB query
<pre>SELECT 'Author'=RTRIM(name) + ', ' + firstname, 'Title'=title FROM redbook.dbo.authors AS A JOIN redbook.dbo.redbooks AS RB ON A.author_id=RB.author_id WHERE A.state <> 'CA' ORDER BY 1</pre>	<pre>SELECT RTRIM(name) CONCAT ', ' CONCAT firstname AUTHORS, title TITLE FROM asn.authors AS A INNER JOIN asn.redbooks AS RB ON A.author_id=RB.author_id WHERE A.state <> 'CA' ORDER BY 1</pre>

Note: Notice how the SQL statement using nicknames accesses the remote tables residing in SQL Server. The end user/application is unaware that they represent remote tables.

Figure 8-7 shows how a SQL Server table can be accessed from Control Center once a nickname for it has been setup.

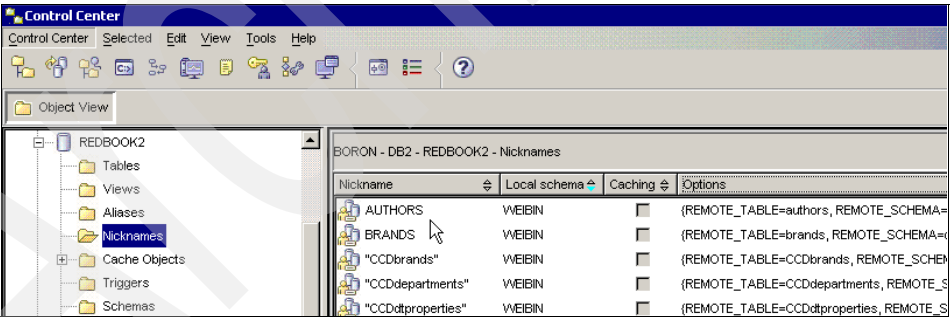


Figure 8-7 SQL Server tables accessed in Control Center through nicknames

8.4.2 Replicating data from SQL Server to DB2 UDB

After registering SQL Server tables and views as nicknames in DB2 UDB, you can begin to set up data replication and migrate data over to DB2 UDB. In this process, you should also specify the frequency that data updates in SQL Server

should be replicated to DB2 UDB. Once replication is setup, application testing can be performed with data that totally resides in DB2 UDB.

Some of the fundamental terminology of DB2 UDB SQL replication that you need to understand when using WebSphere II is listed below. For more information, refer to the DB2 UDB documentation *IBM WebSphere Information Integrator SQL Replication Guide and Reference Version 8.2, SC27-1121*.

Capture control server

The first step after setting up nicknames in DB2 UDB is to create a capture control server. The creation of this control server creates a capture program (CAPTURE) along with control tables and changed data (CD) staging tables. CAPTURE detects changes in the SQL Server data source and feeds those changes into CD staging tables. The CAPTURE, control tables, and the CD staging tables must reside in the source database. However, the SQL replication program interprets these tables as nicknames thus treating them as local DB2 UDB data sources.

Apply control server

Once the apply control server is created, the apply program (APPLY) is created along with its control tables. APPLY periodically (based on user specification) connects to the source database to read the changes from the CD staging tables. It then replicates those changes to the target tables. The apply control server and its objects do not need to reside on the target database server.

Subscription Set

A subscription set defines the relationship between the replication source database and a target database. A subscription set member defines a relationship between the replication source table and one or more target tables. It is best to have the target tables and the rest of the database objects pre-created before setting up replication.

The following steps summarize how to setup SQL replication between SQL Server and DB2 UDB.

1. Create replication control tables for the CAPTURE program.
2. Enable the source database for replication.
3. Register a replication source.
4. Create replication control tables for the APPLY program.
5. Create a subscription set and a subscription-set member.
6. Create an APPLY password file.
7. Start the APPLY program.

8.5 Using other tools

There are several other tools that can help you move data between different databases. The tools we have discussed so far are:

- ▶ IBM DB2 Migration Toolkit
- ▶ DB2 UDB's EXPORT, IMPORT, and LOAD utilities
- ▶ WebSphere Information Integrator

There are also number of third party data migration tools available to assist you in moving your database, application, and data from an existing DBMS to DB2 UDB.

- ▶ **Ispirer Systems**

Ispirer Systems offers SQLWays, a database and data migration tool

- ▶ **Ascential DataStage**

The DataStage product family is an extraction, transformation, and loading (ETL) solution with end-to-end metadata management and data quality assurance functions.

- ▶ **Data Junction**

DataJunction data migration tool provides assistance in moving data from source database to DB2 UDB. This tool accounts for data type differences, and can set various filters to dynamically modify target columns during the conversion process.

8.6 Converting scripts

Scripts provides a powerful method of automating repetitive tasks. If your application uses scripts to regularly populate the database with new or refreshed data, they will need to be converted to DB2 UDB.

To convert SQL scripts from SQL Server to DB2 UDB, the main task comprises making syntax changes. Refer to Chapter 4, "SQL considerations" on page 65, which helps you identify SQL syntax differences between SQL Server and DB2 UDB. Also, refer to 10.2, "Converting to DB2 UDB compatible SQL syntax" on page 226 that explains how to use MTK's SQL Translator to convert SQL Server syntax to DB2 UDB-compatible syntax.

In the remainder of this chapter, we discuss how to work with DB2 UDB SQL scripts.

Create a SQL script

Almost every wizard in DB2 UDB has a **Show Command** button that launches a pop-up window containing the command that is generated by the wizard. These

commands can be saved as SQL scripts and run at a later time, as illustrated in Figure 8-8.

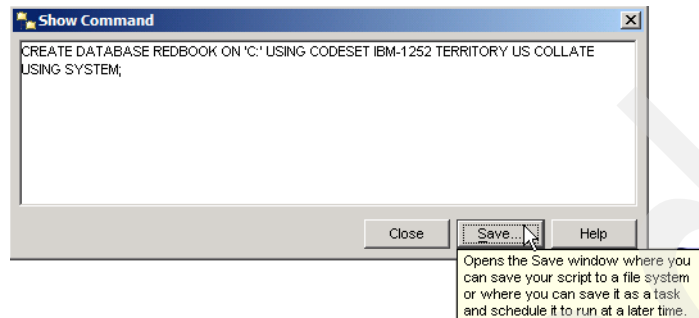


Figure 8-8 Show command pop-up window

You can also create a script using a text editor.

Executing a SQL Script

To run a SQL script from the command line, use one of the following commands:

- ▶ Use the following command if the termination character for the SQL statements in the file use the semicolon (;) character:
- ▶ If the SQL script uses a character other than a semicolon as the statement terminator, use the following command:

```
db2 -tvf filename
```

```
db2 -td<char> -vf filename
```

where <char> is the statement termination character.

SQL scripts can also be loaded or created in Task Center and scheduled to run at a specified time. Figure 8-9 shows how a script can be loaded into the task center.

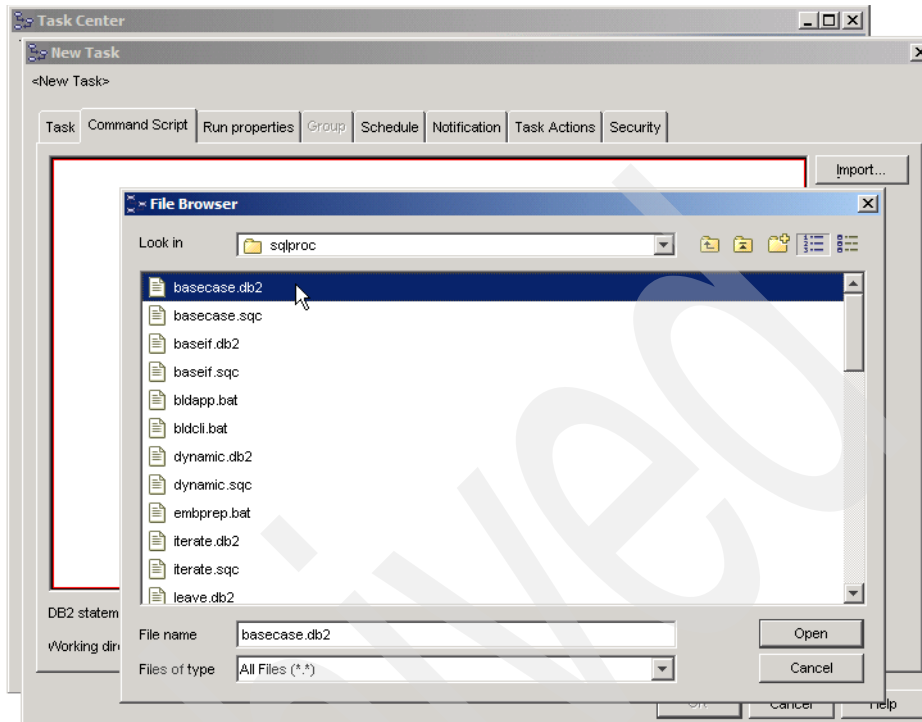


Figure 8-9 Importing SQL scripts into Task Center

Note: In DB2 UDB, only inline SQL can be used for SQL scripting.

For more information about scripting with DB2 UDB, consult the following articles:

<http://www.ibm.com/developerworks/db2/library/techarticle/0203yip/0203yip.html>

<http://www.ibm.com/developerworks/db2/library/techarticle/0211yip/0211yip.html>

<http://www.ibm.com/developerworks/db2/library/techarticle/0307fierros/0307fierros.html>

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0406fierros/index.html>

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0503melnik/index.html>

Converting database objects

This chapter highlights the key differences between database objects in SQL Server and DB2 UDB. The term *database object* is used to denote an object that resides on the database server and contains business logic and/or is used in an application programming capacity. Other database objects related to the logical design or schema of the database, such as tables and views, are covered in Chapter 7.

This chapter also provides examples that can serve as a reference during a conversion to DB2 UDB. The following database objects are covered:

- ▶ Temporary tables
- ▶ Stored procedures
- ▶ User defined functions
- ▶ Triggers

9.1 Temporary tables

Temporary tables are most often used to store temporary and/or intermediate results. They can significantly boost performance because of their reduced logging and locking requirements. Temporary tables are supported in both SQL Server and DB2 UDB, albeit with some differences.

SQL Server temporary tables

SQL Server temporary tables are stored in the tempdb database, which is re-created every time the server is restarted. Two types of temporary tables are supported, private and global. Private temporary tables are visible only in the current session (connection) and are denoted with a single number sign (#) preceding the table name. Global temporary tables are visible in all sessions and are denoted with double number sign (##) preceding the table name.

DB2 UDB temporary tables

In DB2 UDB, a declared global temporary table (DGTT) is accessible only by the connection in which it was created and is dropped when that connection to the database is severed or an explicit DROP statement is issued. The DECLARE GLOBAL TEMPORARY TABLE command is used to create new temporary tables. If the WITH REPLACE clause is included, an existing DGTT with the same name is dropped and replaced with the new table definition.

DGTTs are created in a USER TEMPORARY table space, which can be optionally specified using the IN clause of the DECLARE GLOBAL TEMPORARY TABLE statement. If this clause is not specified, a table space is chosen from the set of existing user temporary table spaces. The table space chosen must be one for which the user has USE privilege and must have sufficient space to contain the table. Locking is *NOT* performed on DGTTs. Modifications made to temporary tables during a transaction can optionally be logged or not. Example 9-1 shows a representative DGTT declaration in DB2 UDB.

Example 9-1 DB2 UDB temporary table definition

```
DECLARE GLOBAL TEMPORARY TABLE gbl_temp
LIKE employee
ON COMMIT DELETE ROWS
NOT LOGGED
IN usr_tbsp
!
```

In this example, a temporary table called gbl_temp is created with the same structure as the employee table. When a COMMIT statement is eventually issued, all rows in the table are deleted automatically. Further, no logging occurs when

any modifications are made. The table is created in the table space called `usr_tbsp`.

DGTTs must always be referenced using the `SESSION` schema. For instance, the temporary table created in Example 9-1 needs to be fully qualified with the `SESSION` schema when referenced in a SQL statement such as the following:

```
SELECT * FROM SESSION.gb1_temp
```

DB2 UDB also supports the use of indexes and statistics collection on DGTTs to improve performance. Undo logging is also supported to permit `ROLLBACKS`.

Note: In a default DB2 UDB installation, a user temporary table space is not created. Any attempt to create a temporary table therefore results in a `SQL0286N` error. A user temporary table space of sufficient size must be created prior to creating the temporary table. The user creating the temporary table must also have the `USE` privilege on the table space.

A DB2 UDB DGTT is semantically equivalent to a SQL Server private temporary table. Global temporary tables that are accessed by different database connections can be implemented in DB2 UDB using standard tables defined with the `NOT LOGGED INITIALLY` clause specified in the `CREATE TABLE` and/or `ALTER TABLE` statements. This clause provides the table with similar logging characteristics as temporary tables. Note that row and table level locking still occurs on these tables.

Attention: The `NOT LOGGED INITIALLY` condition must be re-activated after each transaction (`COMMIT` or `ROLLBACK` statement).

9.1.1 Private temporary tables

Example 9-2 shows a private temporary table definition in SQL Server.

Example 9-2 SQL Server private temporary table definition

```
CREATE TABLE #order (  
    prod_id INT NOT NULL PRIMARY KEY,  
    title_id CHAR(6) NOT NULL,  
    qty INT NOT NULL  
)  
GO
```

Example 9-3 shows the equivalent temporary table definition in DB2 UDB. The table is created in a table space called `temp4k`.

Example 9-3 Equivalent DB2 UDB temporary table definition

```
DECLARE GLOBAL TEMPORARY TABLE order (  
    prod_id  INT NOT NULL PRIMARY KEY,  
    title_id CHAR(6) NOT NULL,  
    qty      INT NOT NULL  
)  
IN temp4k  
!
```

9.1.2 Global temporary tables

Example 9-4 shows a global temporary table definition in SQL Server.

Example 9-4 SQL Server global temporary table definition

```
CREATE TABLE ##customer (  
    c_id  INT NOT NULL PRIMARY KEY,  
    name  VARCHAR(80) NOT NULL,  
    addr  VARCHAR(200) NOT NULL,  
    tel   CHAR(10) NOT NULL  
)  
GO
```

Example 9-5 shows the equivalent temporary table definition in DB2 UDB. The table is created using a standard DB2 UDB table defined with the ability to turn logging on and off.

Example 9-5 Equivalent DB2 UDB temporary table definition

```
CREATE TABLE customer (  
    c_id  INT NOT NULL PRIMARY KEY,  
    name  VARCHAR(80) NOT NULL,  
    addr  VARCHAR(200) NOT NULL,  
    tel   CHAR(10) NOT NULL  
)  
NOT LOGGED INITIALLY  
!
```

9.1.3 Additional information

For more information about temporary tables, refer to the DB2 UDB documentation DB2 UDB documentation *SQL Reference - Volume 1*, SC09-4844. The IBM Press book *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS*, ISBN 0131477005, also contains useful information about application development with temporary tables.

9.2 Stored procedures

Stored procedures are server-side database objects used to encapsulate frequently executed SQL statements with flow logic. Procedures are executed on the database server, which is often a much faster machine than the client. Stored procedures help reduce network traffic since only the original request and the final output need to be transmitted between the client and the server. Both SQL Server and DB2 UDB provide the ability to create customized stored procedures. Although stored procedures are supported in both platforms, implementation differences exist.

SQL Server stored procedures

SQL Server stored procedures are coded using T-SQL and stored in the database. The maximum number of parameters supported in a stored procedure is 1024. Nested stored procedures are supported up to a limit of 32 nesting levels. Procedure parameters can be declared with default values. Wildcard parameters for strings, using the percent (%) character, are also supported. Parameter values can be passed by explicitly naming the parameters or by providing all the parameter values anonymously but in correct positional order.

SQL Server extended stored procedures are routines written in a language such as C. SQL Server automatically loads and execute these routines just like a regular stored procedure. These procedures run directly in the address space of SQL Server. The DLLs are created using the Open Data Services (ODS) API.

SQL Server also supports temporary stored procedures. Temporary stored procedures are stored in the tempdb database, which is re-created each time the server is re-started. These objects do not exist after SQL Server is shut down. Similar to temporary tables, temporary stored procedures can be of two types, private and global. A private temporary stored procedure, identified by a single number sign (#) prefixing the procedure name, is only accessible to the connection that created it. A global temporary stored procedure, identified by a double number sign (##) prefixing the procedure name, is available for invocation by all connections.

DB2 UDB stored procedures

DB2 UDB stored procedures can be coded in the SQL PL language, or in compiled libraries using a third-generation language, including C, C++, COBOL, .NET common language runtime (CLR) languages, OLE, and Java. Stored procedures coded in a third generation language are known as *external* stored procedures. The focus of this section is SQL PL stored procedures, since SQL Server T-SQL stored procedures can usually be converted to SQL PL stored procedures in a straightforward manner.

The maximum number of supported parameters in a procedure is 32,767. Nested stored procedures are supported in SQL PL, Java and C procedures, up to a limit of 16 nesting levels. Parameter default values are not supported, nor are wildcard characters in parameters. All parameter values must be explicitly provided when the procedure is invoked. Multiple result sets (cursors) can be returned to the calling application. Stored procedures can run in the same address space as the database manager (instance) or in a separate space by specifying the clause `FENCED` or `NOT FENCED` in the `CREATE PROCEDURE` statement. Once a stored procedure is created, it can be invoked by executing the `CALL` statement from client applications, other stored procedures, triggers, user defined functions, dynamic compound statements and the DB2 command line processor (CLP).

Important: Running stored procedures in `NOT FENCED` mode that have not been adequately tested can compromise integrity of the server. Ensure that stored procedures are well-tested when defined as `NOT FENCED`.

In DB2 UDB, once the `CREATE PROCEDURE` statement is executed, procedural and SQL statements in the procedure are converted into a native representation which is stored in the database catalogs. When an SQL procedure is called, the native representation is loaded from the catalogs and the DB2 UDB engine executes the procedure.

9.2.1 Temporary stored procedures

There is no equivalent to SQL Server temporary stored procedures in DB2 UDB. Temporary stored procedures must be converted to DB2 UDB using standard SQL PL stored procedures. To simulate the “temporary” nature of these procedures in DB2 UDB, they should be explicitly dropped upon database shutdown.

9.2.2 Extended stored procedures

SQL Server extended stored procedures are equivalent to DB2 UDB external stored procedures. Generally, the business logic contained in the host language code will require few changes. However, changes to how the procedure is registered with the database as well as other host language interfaces changes are necessary.

9.2.3 Remote procedure calls

In SQL Server, calls can be made to stored procedures compiled on remote servers. For example, the call invokes procedure `proc1` in the master database on server `server1` for the user `dbo`:

```
exec server1.master.dbo.proc1
```

In DB2 UDB, it is not possible to invoke a procedure on a remote server or from a different database. To invoke a procedure in DB2 UDB, a connection must first be made to the database where the procedure was created. Additionally, the user invoking the procedure must also have the `EXECUTE` privilege on the procedure.

9.2.4 @@PROCID

The SQL Server function `@@PROCID` returns the stored procedure identifier (ID) of the current procedure. Example 9-6 shows a sample stored procedure that uses a `SELECT` statement to display the `@@PROCID` setting from inside the procedure.

Example 9-6 SQL Server stored procedure using the @@PROCID function

```
CREATE PROCEDURE testprocedure AS
SELECT @@PROCID AS 'ProcID'
GO
EXEC testprocedure
GO
```

To simulate this functionality in DB2 UDB, a user defined function can be created. Example 9-7 shows a Java user defined function that returns the ID of the stored procedure. The `CREATE FUNCTION` DDL code is also provided.

Example 9-7 DB2 UDB external Java UDF simulating the @@PROCID function

```
CREATE FUNCTION proc_id()
RETURNS INTEGER
LANGUAGE JAVA
PARAMETER STYLE DB2GENERAL
DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
DBINFO
EXTERNAL NAME 'proc_id!proc_id'
!

import COM.ibm.db2.app.*;
```

```

public class proc_id extends UDF
{
    public void proc_id(int procid_out) throws Exception
    {
        set(1, getDBprocid());
    }
}

```

9.2.5 Simple conversion example

Example 9-8 shows a simple SQL Server T-SQL stored procedure. The procedure returns the price of the most expensive book in the brand given as input to the procedure.

Example 9-8 Simple SQL Server stored procedure

```

CREATE PROCEDURE book_info
    @b_id int,
    @max_price money
AS
    SELECT @max_price = MAX(price)
    FROM redbooks
    WHERE brand_id = @b_id
GO

```

Example 9-9 shows the equivalent DB2 UDB SQL PL stored procedure. Notice how additional exception handlers have been defined in the converted procedure. This is due to the differences between how errors are handled in T-SQL and SQL PL. Refer to 4.9, “Transact-SQL to SQL PL translation” on page 96 for more information.

Example 9-9 Equivalent DB2 UDB stored procedure

```

CREATE PROCEDURE book_info ( IN v_b_id INTEGER,
                             OUT v_max_price DECIMAL(19,4))
LANGUAGE SQL
READS SQL DATA
BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';

    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET l_error = '00000';

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
        BEGIN
            SET l_error = SQLSTATE;

```

```

        IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
            RESIGNAL;
        END IF;
    END;

    SELECT ROUND(MAX(price), 4)
        INTO v_max_price
    FROM redbooks
    WHERE brand_id = v_b_id
END
!

```

9.2.6 Default parameter values

SQL Server permits the use of default stored procedure parameter values, as illustrated in Example 9-10. The procedure in this example returns the title and price of all Redbooks that cost less than *x* dollars, where *x* is a parameter to the procedure. If an amount is not specified when the procedure is invoked, a default value of twenty dollars is used.

Example 9-10 Using default parameter values in SQL Server stored procedures

```

CREATE PROC get_cheap_books
    @cost money = 20.00
AS
    SELECT title, price FROM redbooks WHERE price < @cost
GO

```

In DB2 UDB, default parameter values cannot be specified as part of the CREATE PROCEDURE statement. There are two options for converting this functionality to DB2 UDB:

- ▶ Ensure that all invocations of the procedure (in the application code and other database objects) always supply the correct default parameter value when one is required. This solution requires the management of specific default values outside of the stored procedure logic. That is, all places in the existing code that invoke the procedure must always specify the correct default value. When the default value changes, all the corresponding invocation statements must be updated with the new value.
- ▶ Ensure that all invocations of the procedure (in the application code and other database objects) invoke the procedure with a “dummy” value to indicate to the procedure that a default value defined within the procedure body should be used instead. Note that for this solution to be successful, a dummy value must be chosen that is unique from the range of expected real data values. Further, all invocations of the procedure (in the application code and other database objects) must be modified to invoke the procedure with the dummy

value specified in the procedure body. This solution allows default values to be stored and managed in one place in the stored procedure code, instead of multiple places in the application code. Example 9-11 shows how the procedure in Example 9-10 can be converted to DB2 UDB using the dummy value approach. In this example, the dummy value is represented by the number '-9999.99'.

Example 9-11 DB2 UDB stored procedure simulating default parameter values

```
CREATE PROCEDURE get_cheap_books (IN v_cost DECIMAL(19,4))
LANGUAGE SQL
READS SQL DATA
DYNAMIC RESULT SETS 1
BEGIN
    -- the real default value for the input parameter
    -- this variable is used in place of the input parameter
    DECLARE default_cost DECIMAL(19,4) DEFAULT 20.00;

    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';

    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET l_error = '00000';

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
    BEGIN
        SET l_error = SQLSTATE;
        IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
            RESIGNAL;
        END IF;
    END;

    -- only change the local default cost variable when the input parameter
    -- provided is NOT the dummy value
    IF (v_cost != -9999.99) THEN
        SET default_cost = v_cost;
    END IF;

    BEGIN
        -- cursor is declared using the locally declared cost value, NOT
        -- the input parameter
        DECLARE temp_cursor CURSOR WITH HOLD WITH RETURN TO CLIENT
            FOR SELECT title, price
                FROM redbooks
                WHERE price < default_cost;

        OPEN temp_cursor;
    END;
END
```


!

The following statement can be executed to invoke the procedure:

```
CALL get_cheap_books(-9999.99)
```

When invoked with the statement above, the dummy value is replaced with the real default value (20.00) defined in the procedure.

Wildcard in default parameters

SQL Server supports the use of wildcard characters in default parameters, as shown in Example 9-12. In this example, if no parameters are supplied when the procedure is invoked, all book titles and author names are listed for authors whose surname starts with the letter “K”. Otherwise, the procedure returns the list of book titles and author names of authors whose first and last names match those provided as input to the procedure.

Example 9-12 SQL Server stored procedure using default wildcard parameters

```
CREATE PROCEDURE get_author_info
    @lname varchar(30) = 'K%',
    @fname varchar(18) = '%'
AS
    SELECT a.name, a.firstname, r.title
    FROM authors a INNER JOIN redbooks r ON a.author_id = r.author_id
    WHERE a.firstname LIKE @fname AND a.name LIKE @lname
    GROUP BY a.name, a.firstname, r.title
GO
```

Example 9-13 shows the equivalent stored procedure in DB2 UDB. This procedure uses the dummy value approach to simulate the use of default parameters. The dummy value used in this example is the character string '!!!!!!!'.

Example 9-13 Simulating wildcard parameters in a DB2 UDB stored procedure

```
CREATE PROCEDURE get_author_info (IN lname VARCHAR(40),
                                  IN fname VARCHAR(20))
LANGUAGE SQL
READS SQL DATA
DYNAMIC RESULT SETS 1
BEGIN
    DECLARE lname_default VARCHAR(40) DEFAULT 'G%';
    DECLARE fname_default VARCHAR(20) DEFAULT '%';

    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';
```

```

DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET l_error = '00000';

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
BEGIN
    SET l_error = SQLSTATE;
    IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
        RESIGNAL;
    END IF;
END;

IF (lname != '!!!!!!!!') THEN
    SET lname_default = lname;
END IF;

IF (fname != '!!!!!!!!') THEN
    SET fname_default = fname;
END IF;

BEGIN
DECLARE temp_cursor CURSOR WITH HOLD WITH RETURN TO CLIENT
FOR    SELECT a.name, a.firstname, r.title
        FROM authors a INNER JOIN redbooks r ON a.author_id = r.author_id
        WHERE a.firstname LIKE fname_default AND a.name LIKE lname_default
        GROUP BY a.name, a.firstname, r.title;

    OPEN temp_cursor;
END;
!

```

9.2.7 Passing parameter values

In SQL Server, parameter values can be passed by explicitly naming the parameters or by providing all the parameter values anonymously but in correct positional order. Example 9-14 shows a sample stored procedure with default parameter values in SQL Server.

Example 9-14 SQL Server stored procedure with default parameters

```

CREATE PROCEDURE return_params
    @p0 int=NULL,    -- Defaults to NULL
    @p1 int=1,       -- Defaults to 1
    @p2 int=2,       -- Defaults to 2
    @p0_out int,
    @p1_out int,
    @p2_out int
AS

```

```

SET @p0_out = @p0
SET @p1_out = @p1
SET @p2_out = @p2
GO

```

Table 9-1 shows sample invocations of the procedure and the resulting output.

Table 9-1 Different invocation of SQL Server procedure shown in Example 9-14

Condition	Invocation Statement	Output
No parameters (all defaults)	EXEC return_params	NULL 1 2
All parameters (in order)	EXEC return_params 0, 10, 20	0 10 20
Two parameters out of order	EXEC return_params @p2=200, @p1=NULL	NULL NULL 200
Second parameter defaults, others by position	EXEC return_params 0, DEFAULT, 20	0 1 20

Example 9-15 shows the corresponding stored procedure in DB2 UDB. Note, in this example, the “dummy value” approach described in 9.2.6, “Default parameter values” on page 193, is used to simulate the use of default parameter values. The dummy value used for each parameter is the number ‘-9999999’.

Example 9-15 Equivalent DB2 UDB stored procedure simulating default parameters

```

CREATE PROCEDURE return_params (
    IN p1 INTEGER,
    IN p2 INTEGER,
    IN p3 INTEGER,
    OUT p1_out INTEGER,
    OUT p2_out INTEGER,
    OUT p3_out INTEGER)

LANGUAGE SQL
CONTAINS SQL
BEGIN
    -- the real default values for the input parameters
    -- these variables are used in place of the input parameters
    DECLARE default_p1 INTEGER DEFAULT NULL;
    DECLARE default_p2 INTEGER DEFAULT 1;
    DECLARE default_p3 INTEGER DEFAULT 2;

    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';

    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET l_error = '00000';

```

```

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
BEGIN
    SET l_error = SQLSTATE;
    IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
        RESIGNAL;
    END IF;
END;

-- only change the local default variables when the input parameters
-- provided are NOT the dummy value
IF (p1 != -9999999) THEN
    SET default_p1 = p1;
END IF;

IF (p2 != -9999999) THEN
    SET default_p2 = p2;
END IF;

IF (p3 != -9999999) THEN
    SET default_p3 = p3;
END IF;

SET p1_out = default_p1;
SET p2_out = default_p2;
SET p3_out = default_p3;
END
!
```

Table 9-2 shows the corresponding invocation scenarios of the procedure and the output.

Table 9-2 Different invocation of the equivalent DB2 UDB procedure

Condition	Invocation Statement	Output
No parameters (all defaults)	N/A	N/A
All parameters (in order)	CALL return_params(0, 10, 20, ?, ?, ?)	0 10 20
Two parameters out of order	N/A	N/A
Second parameter defaults, others by position	CALL return_params(0, -9999999, 20, ?, ?, ?)	0 1 20

As is evident from Table 9-2, all parameters must be supplied when invoking a stored procedure. Default parameter values are specified using the dummy value expected by the procedure.

Output parameters

In DB2 UDB, output parameters should be specified with a question mark (?) when invoked from Command Line Processor (CLP), or when a dynamic SQL statement is PREPARED. Example 9-16 shows a DB2 UDB procedure with two input parameters and one output parameter. The procedure accepts an author's first name and surname, and counts how many books were written by that author. The result is returned using an output parameter.

Example 9-16 DB2 UDB stored procedure with output parameters

```
CREATE PROCEDURE count_books (    IN p_name VARCHAR(40),
                                IN p_firstname VARCHAR(20),
                                OUT p_book_count INTEGER)

LANGUAGE SQL
READS SQL DATA
BEGIN
    SELECT COUNT(1) INTO p_book_count
    FROM redbooks
    WHERE author_id = (
        SELECT author_id
        FROM authors
        WHERE name = p_name AND firstname = p_firstname);

END
!
```

To invoke this procedure from CLP, the following command can be issued:

```
CALL count_books('Smith', 'John', ?)
```

A question mark (?) is used as a placeholder for the output parameter. In DB2 UDB, all parameters must be supplied when invoking a stored procedure.

Tip: The procedure in Example 9-16 is purely designed to demonstrate the use of output parameters. In reality, this procedure is very simple and would be better implemented as a user defined function, since it does not require the advanced constructs of SQL PL.

9.2.8 Returning a cursor to the caller

In SQL Server, output cursor parameters are used to pass a cursor that is local to a stored procedure back to the calling batch, stored procedure, or trigger. An example of such a procedure is shown in Example 9-17. In this procedure, a static, forward-only cursor, is defined to return the list of available redbook titles along with their book number. The cursor is declared and assigned to the cursor output parameter. The cursor is then explicitly opened before procedure execution terminates.

Example 9-17 SQL Server stored procedure returning a cursor to the caller

```
CREATE PROCEDURE titles_cursor
    @cur_titles CURSOR VARYING OUTPUT
AS
    SET @cur_titles = CURSOR FORWARD_ONLY STATIC FOR
        SELECT book_no, title FROM redbooks

    OPEN @titles_cursor
GO
```

To invoke the procedure above and receive and manipulate the output cursor, the following code can be used:

```
DECLARE @MyCursor CURSOR
EXEC titles_cursor @cur_titles = @MyCursor OUTPUT
WHILE (@@FETCH_STATUS = 0)
BEGIN
    FETCH NEXT FROM @MyCursor
END
CLOSE @MyCursor
DEALLOCATE @MyCursor
GO
```

In DB2 UDB, cursor result sets can also be passed back to the calling application, although through a different mechanism. To indicate that a cursor is to be returned to the calling application, the following three steps must be completed:

1. The `DYNAMIC RESULT SETS` clause in the `CREATE PROCEDURE` statement must specify the number of result sets that will be returned.
2. The cursor must be declared as `WITH RETURN`.
3. The cursor must be opened before being returned to the client application.

Example 9-18 shows the equivalent DB2 UDB stored procedure.

Example 9-18 Equivalent DB2 UDB stored procedure returning a cursor to the caller

```
CREATE PROCEDURE titles_cursor()
LANGUAGE SQL
READS SQL DATA
DYNAMIC RESULT SETS 1
BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';

    DECLARE titles_cursor CURSOR WITH HOLD WITH RETURN TO CLIENT
    FOR
        SELECT book_no, title
        FROM redbooks;
```

```

DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET l_error = '00000';

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
BEGIN
    SET l_error = SQLSTATE;
    IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
        RESIGNAL;
    END IF;
END;

OPEN titles_cursor;
END
!
```

Example 9-19 shows another DB2 UDB procedure that counts the number of book titles in the database. This procedure invokes the one in Example 9-18 and receives the cursor that is passed back to it. Note that Example 9-18 is purely for illustration purposes in order to demonstrate the use of returning result sets to the caller. There are more efficient ways to carry out the intended counting operation.

Example 9-19 A DB2 UDB stored procedure receiving a cursor result set

```

CREATE PROCEDURE count_titles (OUT p_counter INT)
LANGUAGE SQL
BEGIN
    DECLARE v_book_no CHAR(10);
    DECLARE v_title VARCHAR(80);
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE v_rs RESULT_SET_LOCATOR VARYING;

    CALL titles_cursor();
    ASSOCIATE RESULT SET LOCATOR (v_rs)
        WITH PROCEDURE titles_cursor;

    ALLOCATE c_rs CURSOR FOR RESULT SET v_rs;

    SET p_counter = 0;
    WHILE (SQLSTATE = '00000') DO
        SET p_counter = p_counter + 1;
        FETCH FROM c_rs INTO v_book_no, v_title;
    END WHILE;
END
!
```

9.2.9 Exception handling and transaction control

Several examples are now presented for converting T-SQL error handling and transaction statements to the equivalent constructs in SQL PL. Refer to 4.9, “Transact-SQL to SQL PL translation” on page 96 for a mapping of T-SQL statements to SQL PL and a detailed explanation about how T-SQL error handling constructs map to SQL PL.

@@ERROR

The @@ERROR function is used to check for a success or failure status after the evaluation of a statement. Example 9-20 shows a SQL Server stored procedure that checks the value of @@ERROR after performing an INSERT statement. If an error value is returned from the INSERT statement, the transaction is rolled back.

Example 9-20 SQL Server stored procedure referencing the @@ERROR function

```
CREATE PROCEDURE add_brand(  
    @b_id VARCHAR(6),  
    @b_name VARCHAR(20),  
    @b_desc VARCHAR(80))  
AS  
BEGIN TRAN  
    INSERT brands(brand_id, brand_name, brand_desc)  
    VALUES (@b_id, @b_name, @b_desc)  
  
    IF (@@ERROR <> 0)  
    BEGIN  
        ROLLBACK TRAN  
        RETURN 1  
    END  
  
    COMMIT TRAN  
    RETURN 0  
GO
```

The equivalent DB2 UDB stored procedure is shown in Example 9-21.

Example 9-21 Equivalent DB2 UDB stored procedure with exception handlers

```
CREATE PROCEDURE add_brand ( v_b_id INTEGER,  
                             v_b_name VARCHAR(20),  
                             v_b_desc VARCHAR(80) )  
  
LANGUAGE SQL  
MODIFIES SQL DATA  
BEGIN  
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';  
    DECLARE l_error CHAR(5) DEFAULT '00000';  
    DECLARE v_trancount INTEGER DEFAULT 0;
```



```

DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET l_error = '00000';

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
BEGIN
    SET l_error = SQLSTATE;
    IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
        RESIGNAL;
    END IF;
END;

SET v_trancount = v_trancount + 1;

SET l_error = '00000';

INSERT INTO brands (brand_id, brand_name, brand_desc)
VALUES (v_b_id,v_b_name,v_b_desc);

IF v_trancount = 0 THEN
    COMMIT;
END IF;

IF (l_error <> '00000') THEN
    ROLLBACK;
    SET v_trancount = 0;
    RETURN 1;
END IF;

IF v_trancount = 1 THEN
    COMMIT;
END IF;

IF v_trancount > 0 THEN
    SET v_trancount = v_trancount - 1;
END IF;

RETURN 0;
END
!
```

Notice how COMMIT statements are handled in the code. A local procedure variable, `v_trancount`, is used to keep track of the transaction nesting level (equivalent to @@TRANSCOUNT in T-SQL). A COMMIT statement is only issued when the value of `v_trancount` is zero or one. Otherwise, the value of `v_trancount` is decremented by one. Although this additional transaction logic is not really required for this specific procedure, it has been included to demonstrate how similar transaction control behavior can be simulated in SQL PL.

RAISERROR

A RAISERROR statement is equivalent to a PRINT statement, followed by an assignment to @@ERROR. Example 9-22 shows a SQL Server stored procedure using a RAISERROR statement. This procedure attempts to retrieve the price of the redbook with the book number provided as a procedure argument. If no book can be found with that number, an error is raised.

Example 9-22 SQL Server stored procedure using the RAISERROR statement

```
CREATE PROCEDURE get_redbook_price (
    @b_no char(10),
    @book_price money)
AS
    DECLARE @b_price money

    SELECT @b_price = price
    FROM redbooks r
    WHERE r.book_no = @b_no

    IF @@ROWCOUNT = 0
    BEGIN
        RAISERROR ('No redbook with this book number was found.', 16, 1)
        RETURN 0
    END

    SET @book_price = @b_price

    RETURN 1
GO
```

Example 9-23 shows the equivalent stored procedure in DB2 UDB.

Example 9-23 Equivalent DB2 UDB stored procedure simulating RAISERROR

```
CREATE PROCEDURE get_redbook_price ( IN v_b_no CHAR(10),
                                     OUT v_book_price DECIMAL(19,4))
LANGUAGE SQL
READS SQL DATA
BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';
    DECLARE v_price DECIMAL(19,4);
    DECLARE l_rowcount INTEGER;

    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET l_error = '00000';

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
```

```

BEGIN
    SET l_error = SQLSTATE;
    IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
        RESIGNAL;
    END IF;
END;

SELECT price
    INTO v_price
FROM redbooks r
WHERE r.book_no = v_b_no
FETCH FIRST 1 ROWS ONLY;

SELECT COUNT(*)
    INTO l_rowcount
FROM (
    SELECT price
    FROM redbooks r
    WHERE r.book_no = v_b_no) temp_table;

IF l_rowcount = 0 THEN
    -- Signalling an exception using RAISERROR and returning a value
    -- simultaneously is not possible. The RAISERROR statement is not
    -- translated.
    RETURN -9;

    -- The above RETURN statement can be replaced with a statement such as
    -- the following if it is desirable for an exception to be raised
    -- instead:
    -- SIGNAL SQLSTATE '50000' SET MESSAGE_TEXT = 'No title found.';
END IF;

SET v_book_price = v_price;
RETURN 1;
END
!
```

Notice how the RAISERROR statement is converted in the above stored procedure. The original T-SQL procedure returned a value as well as signalled an exception. Since both of these cannot happen simultaneously in SQL PL, an exception is not raised in the procedure. Rather, the original return value (-9) is returned to the calling application. However, if the desired effect in the procedure is to raise an exception, the RETURN statement can be replaced with a SIGNAL SQLSTATE statement.

Nested transactions and savepoints

Example 9-24 shows a SQL Server stored procedure using nested savepoints. This procedure assumes the following table definition:

```
CREATE TABLE nestlog (c1 INTEGER)
```

Example 9-24 SQL Server stored procedure with nested savepoints

```
-- Assumes the use of the following table definition
-- CREATE TABLE nestlog (c1 INTEGER)

CREATE PROCEDURE nested_tran
AS
    -- To start with, verify @@TRANCOUNT is 0
    SELECT @@TRANCOUNT
    BEGIN TRAN A
        -- Verify @@TRANCOUNT is 1
        SELECT @@TRANCOUNT

        INSERT INTO nestlog VALUES (1)

        SAVE TRAN B

        -- Verify @@TRANCOUNT is still 1. A savepoint does not affect it.
        SELECT @@TRANCOUNT

        INSERT INTO nestlog VALUES (2)

    ROLLBACK TRAN B

    -- @@TRANCOUNT is still 1 because the previous ROLLBACK
    -- affects just the savepoint, not the transaction
    SELECT @@TRANCOUNT

    INSERT INTO nestlog VALUES (3)

    ROLLBACK TRAN A

    -- This ROLLBACK succeeds, so @@TRANCOUNT is back to 0
    SELECT @@TRANCOUNT
GO
```

Example 9-25 shows the equivalent DB2 UDB stored procedure.

Example 9-25 Equivalent DB2 UDB stored procedure with nested savepoints

```
-- Assumes the use of the following table definition
-- CREATE TABLE nestlog (c1 INTEGER)
CREATE PROCEDURE nested_tran()
```

```

LANGUAGE SQL
t1: BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE l_error CHAR(5) DEFAULT '00000';
    DECLARE v_trancount INTEGER DEFAULT 0;

    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET l_error = '00000';

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
    BEGIN
        SET l_error = SQLSTATE;

        IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9' THEN
            RESIGNAL;
        END IF;
    END;

    -- for the sake of comparison, the initial value of @@TRANCOUNT is printed
    -- at the beginning of this procedure
    -- VALUES put_line(v_trancount);

    -- DB2 begins transactions implicitly
    -- Therefore, @@TRANCOUNT technically begins at 1
    SET v_trancount = v_trancount + 1;
    -- VALUES put_line(v_trancount);

    INSERT INTO nestlog(c1) VALUES (1);

    SAVEPOINT B ON ROLLBACK RETAIN CURSORS;
    -- A savepoint does not affect @@TRANCOUNT.
    -- VALUES put_line(v_trancount);
    INSERT INTO nestlog(c1) VALUES (2);

    ROLLBACK TO SAVEPOINT B;

    -- @@TRANCOUNT would still be 1 because ROLLBACK affects just the
    -- SAVEPOINT, NOT the entire transaction
    -- VALUES put_line(v_trancount);

    INSERT INTO nestlog(c1) VALUES (3);

    -- ROLLBACK un-does the entire transaction (i.e. right back to the BEGIN
    -- statement)
    ROLLBACK;

    -- A ROLLBACK if the outer transaction resets @@TRANCOUNT back to 0
    SET v_trancount = 0;
    -- VALUES put_line(v_trancount);

```

```
END t1
!  
!
```

The values of `v_trancount` can be written to a file using the user defined function called `put_line()`, which enables file output from pure SQL. This function is provided as additional material with this redbook. This function has similar functionality as the `PRINT` statement. It can be used as a tool for debugging stored procedures, but also allows any messages to be written to a specified file for other purposes as well. Calls to this function have been left as comments in the above code.

The function code sources and an installation guide can be found on the IBM Redbooks Web site. For more details, see Appendix G, “Additional material” on page 509.

9.2.10 Additional information

The IBM Press book *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS*, 0131477005 is the most comprehensive source for details and examples about using the SQL PL language.

The following articles on the DB2 developerWorks Web site are also helpful for improving the performance of stored procedures:

- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/0306arocena/0306arocena.html>
- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0501riela u/>
- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0406riela u/index.html>
- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0409riela u/index.html>

Additional information can also be found in the DB2 UDB documentation *Application Development Guide: Programming Server Applications*, SC09-4827.

9.3 User defined functions

User defined functions (UDFs) are server-side database objects that are commonly used to encapsulate simple, frequently used, business logic and database lookups. Functions can be inlined in SQL statements and shared by all

applications using a database. Both SQL Server and DB2 UDB provide the ability to create custom UDFs on top of the functions already available in each platform. Although UDFs are supported in both platforms, implementation differences exist.

SQL Server user defined functions

SQL Server supports two types of functions: scalar functions (functions that return a single scalar value such as INT or CHAR) as well as table functions (functions that return a table). Table functions can be further classified as inline functions or multi statement table-valued functions. If the RETURNS clause specifies TABLE with no additional table definition information, the function is an inline function and there should be a single SELECT statement as the body of the function. If the RETURNS clause uses the TABLE statement while specifying a table variable name, the function is a multi statement table-valued function. The RETURNS clause also lists the columns and data types for the table.

SQL Server UDFs cannot be used to perform a set of actions that modify the global database state. Further, the SQL statements inside a scalar-valued function cannot include any non-deterministic system functions.

DB2 UDB user defined functions

DB2 UDB supports three types of UDFs: SQL functions, external functions, and sourced functions. *SQL functions* are written in DB2 UDB's inline SQL PL language. *External functions* are registered in the database using the SQL CREATE FUNCTION statement, but refer to object code libraries written in an external programming language such as a .NET CLR language and Java. *Sourced functions* reference another built-in or user defined function. The focus of this section is SQL PL functions, since the majority of SQL Server UDFs can be converted as SQL PL UDFs.

SQL PL UDFs can be of two different types, scalar functions (return a scalar value, including NULL) and table functions (return a row or table of data). A scalar function's return type is specified in the RETURNS clause of the function. For table functions, the ROW or TABLE keyword is specified in the RETURNS clause. Functions can be deterministic or non-deterministic. Only table functions are able to modify the database.

9.3.1 Scalar functions

SQL Server scalar functions can be converted into inline SQL PL UDB scalar functions. Example 9-26 shows a SQL Server scalar UDF. This function returns the average price of Redbooks of the brand provided as input to the function.

Example 9-26 SQL Server user defined scalar function

```
CREATE FUNCTION avg_price(@brand_id int)
RETURNS money
AS
BEGIN
    RETURN ( SELECT avg(price)
              FROM redbooks
              WHERE brand_id = @brand_id)
END
GO
```

Example 9-27 shows the corresponding function definition in DB2 UDB.

Example 9-27 Equivalent DB2 UDB user defined scalar function

```
CREATE FUNCTION avg_price (v_brand_id INTEGER)
RETURNS DECIMAL(19,4)
LANGUAGE SQL
READS SQL DATA
RETURN ( SELECT AVG(price)
          FROM redbooks
          WHERE brand_id = v_brand_id)
!
```

9.3.2 Table functions

SQL Server ok table functions can be converted into DB2 UDB table functions. Example 9-28 shows a SQL Server table UDF. This function returns the table (a result set) of all Redbooks of the brand ID passed as a parameter to the procedure.

Example 9-28 SQL Server user defined table function

```
CREATE FUNCTION books_by_brand (@brand_id int)
RETURNS TABLE
AS
RETURN ( SELECT book_no, title, price
          FROM redbooks
          WHERE brand_id = @brand_id)
GO
```

Example 9-29 shows the corresponding table function in DB2 UDB.

Example 9-29 Equivalent DB2 UDB user defined table function

```
CREATE FUNCTION books_by_brand (v_brand_id INTEGER)
RETURNS TABLE(
```



```

        book_no CHAR(10),
        title VARCHAR(80),
        price DECIMAL(19,4))
LANGUAGE SQL
READS SQL DATA
BEGIN ATOMIC
    RETURN ( SELECT book_no,
                    title,
                    price
              FROM redbooks
              WHERE brand_id = v_brand_id );
END
!
```

To execute a table function in DB2 UDB, the **TABLE** keyword must be specified to indicate the function returns a table, for example:

```
SELECT * FROM TABLE(books_by_brand(32)) AS T
```

SQL Server multi statement table-valued functions have no direct equivalent in DB2 UDB. If the function does not contain references to table variables, it can be converted as a DB2 UDB table function. Otherwise, table variable logic would need to be simulated by other means. Usually, this can be accomplished with SQL statements that operate on a standard table. A SQL stored procedure or external UDF might also be a good candidates to simulate the behavior of a multi statement table-valued function.

9.3.3 Overcoming inline SQL PL limitations in user defined functions

Recall that DB2 UDB's procedural language for UDFs, inline SQL PL, is actually a subset of DB2's SQL PL procedural language. As such, certain SQL PL elements that are supported in SQL stored procedures are not supported in UDFs. Specifically, the following commonly used statements are not supported in DB2 UDB UDFs:

- ▶ Transactional statements (COMMIT and ROLLBACK)
- ▶ Dynamic SQL statements (PREPARE, EXECUTE, EXECUTE IMMEDIATE)
- ▶ Cursor declarations (DECLARE <cursor>, ALLOCATE CURSOR)
- ▶ Complex exception handling (DECLARE ... HANDLER, RESIGNAL)
- ▶ SQL constructs (SELECT INTO, LOOP, REPEAT)
- ▶ Table variables

To convert a SQL Server UDF using any of the above features, there are three options:

- ▶ Attempt to replace the unsupported statement with an equivalent statement or remove it entirely.

- Move the unsupported statement into a new stored procedure and invoke the stored procedure from the UDF.
- Convert the UDF itself as a stored procedure.

The first option is usually the easiest way to proceed. Sometimes, replacing an unsupported statement with a similar supported one, or removing it entirely is possible. For instance, the LOOP and REPEAT statements can usually be implemented using WHILE statement logic. An explicit CURSOR declaration can sometimes be replaced with an implicit cursor using the FOR <cursor_name> AS logic. The SELECT INTO statement can be replaced with a SET statement.

Example 9-30 shows a SQL Server UDF performing a variable assignment in a SELECT statement. This function returns the average price of Redbooks for a particular brand.

Example 9-30 SQL Server UDF performing SELECT list variable assignment

```
CREATE FUNCTION avg_price2(@brand_id int)
RETURNS money
AS
BEGIN
    DECLARE @avg money

    SELECT @avg = avg(price)
    FROM redbooks
    WHERE brand_id = @brand_id

    RETURN @avg
END
GO
```

Example 9-31 shows the corresponding DB2 UDB function. Since the SELECT INTO statement is not supported in inline SQL PL, the SET statement is used instead.

Example 9-31 Equivalent DB2 UDB user defined function using a SET statement

```
CREATE FUNCTION avg_price2 (v_brand_id INTEGER)
RETURNS DECIMAL(19,4)
LANGUAGE SQL
READS SQL DATA
BEGIN ATOMIC
    DECLARE v_avg DECIMAL(19,4);

    SET v_avg = ( SELECT AVG(price)
                  FROM redbooks
                  WHERE brand_id = v_brand_id);
```

```
    RETURN v_avg;
END
!
```

If it is not possible to replace or remove the unsupported statement, it can sometimes be moved into a SQL stored procedure, and the procedure in turn can be invoked by the UDF.

Example 9-32 shows a DB2 UDB table function that invokes a stored procedure to perform more complicated transaction handling logic using savepoints. The following table definition is assumed by the stored procedure:

```
CREATE TABLE result_tab (c1 INTEGER)
```

Example 9-32 DB2 UDB table function that invokes a stored procedure

```
-- Assumes the use of the following table definition:
-- CREATE TABLE result_tab (c1 INTEGER)
```

```
CREATE FUNCTION sproc_func()
RETURNS TABLE (c1 INTEGER)
LANGUAGE SQL
MODIFIES SQL DATA

f1: BEGIN ATOMIC
    CALL svpt_sp();
    RETURN (SELECT brand_id FROM redbooks);
END f1
!

CREATE PROCEDURE svpt_sp()
LANGUAGE SQL
MODIFIES SQL DATA
sp1: BEGIN
    SAVEPOINT save1 ON ROLLBACK RETAIN CURSORS;
    INSERT INTO result_tab (c1) VALUES (123);
    INSERT INTO result_tab (c1) VALUES (456);
    INSERT INTO result_tab (c1) VALUES (789);

    ROLLBACK TO SAVEPOINT save1;

    INSERT INTO result_tab (c1) VALUES (123);
END sp1
!
```

Attention: If a stored procedure is invoked by user defined function, a ROLLBACK or COMMIT statement is not allowed in the body of the procedure unless it is rolled back to a savepoint defined in the same procedure.

Note that when invoking stored procedures from UDFs, data access restrictions must be kept in mind. Both SQL stored procedures and UDFs have several data access options, specified using the CONTAINS SQL, READS SQL DATA, and MODIFIES SQL DATA clauses in the procedure/function definition. The CONTAINS SQL option indicates that only statements that do not read or modify data are allowed. The READS SQL DATA option indicates that only statements that do not modify SQL data (INSERT, UPDATE, DELETE) are allowed. The MODIFIES SQL DATA option is the least restrictive and indicates that any supported SQL statement can be executed. These access option, presented in increasing order of accessibility, cannot conflict with each other. That is, UDFs or stored procedures with lower data access level cannot invoke UDFs or SQL procedures with higher data access levels. For instance, a SQL UDF defined with the READS SQL DATA cannot invoke a stored procedure defined with MODIFIES SQL DATA. Similarly, a SQL stored procedure defined as CONTAINS SQL cannot execute a UDF defined with READS SQL DATA, and so on.

It is also important to be aware of read and write conflict errors that can occur when a SQL stored procedure is invoked by a UDF in DB2 UDB. A set of data integrity rules are enforced when a SQL stored procedure is invoked by a UDF. Essentially, DB2 UDB does not allow conflicting operations to occur when a stored procedure invoked by a UDF concurrently accesses the same table. If such a conflict occurs, an error (SQLCODE -746, SQLSTATE 57053) is returned to the statement that caused the conflict at runtime. It is therefore important to test all conditions to ensure that conflict errors do not occur at runtime.

9.3.4 Additional information

The IBM Press book *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS*, 0131477005 is the most comprehensive source for details and examples about using the SQL PL language, including inline SQL PL.

The following articles on the DB2 developerWorks Web site also provide useful information for user defined function development:

- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/0309greenstein/0309stolze.html>
- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0501nicholson/index.html>

- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/0303stolze/0303stolze.html>
- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0402greenstein/>

Additional information can also be found in the DB2 UDB documentation *Application Development Guide: Programming Server Applications*, SC09-4827.

9.4 Triggers

Triggers are server-side database objects that are commonly used to maintain complex data integrity rules, implement referential actions such as cascading DELETES, and maintain an audit record of changes. Triggers are supported in both SQL Server and DB2 UDB, albeit with some differences.

SQL Server triggers

SQL Server triggers are coded using T-SQL and are stored directly in the database. SQL Server provides two types of triggers, AFTER triggers and INSTEAD OF triggers. An AFTER trigger is executed once for each INSERT, UPDATE, or DELETE statement, regardless of the number of rows it affects. INSTEAD OF triggers replace the triggering SQL statement with the logic provided in the trigger body. Multiple AFTER triggers per table are permitted, but only one INSTEAD OF trigger is permitted per table. The system stored procedure `sp_settriggerorder()` can be used to set the first and last AFTER trigger to be executed; any other AFTER triggers are executed in random order.

SQL Server supports INSERT, UPDATE, and DELETE triggers. A trigger can be associated with more than one type of INSERT, UPDATE, and DELETE event. Triggers can be nested up to 32 levels. Direct or indirect recursion is available as a database configuration option.

DB2 UDB triggers

DB2 UDB triggers are coded using in-line SQL PL, defined with the CREATE TRIGGER statement, stored in the database, and compiled at run time with the SQL statements that are associated with the trigger. DB2 UDB supports INSERT, UPDATE, and DELETE triggers. A separate trigger must be defined for each of these operations.

Triggers can be activated before or after any table data is affected by the triggering SQL statement, by specifying the NO CASCADE BEFORE or AFTER clauses in the CREATE TRIGGER statement. If the NO CASCADE BEFORE clause is specified, the trigger logic is applied before any changes caused by the triggering SQL statement are applied to the database. This implies that the triggering SQL

statement will not cause other triggers to be activated, hence INSERT, UPDATE, and DELETE operations are not allowed in triggers defined with the NO CASCADE BEFORE clause.

Triggers can also be defined to fire every time a row in the target table is affected by the triggering SQL statement or fire once for each statement, no matter how many rows are affected by the triggering statement. Only AFTER triggers can be defined to fire once for each statement. Multiple triggers can be defined for the same event. The order in which the triggers are fired is the same as the order they were created in. The maximum depth of cascading triggers is 16 levels.

A trigger action is composed of one or more SQL procedural statements, which can contain a dynamic compound statement or any of the SQL control statements.

9.4.1 Accessing transition values in triggers

SQL Server maintains special pseudo tables called deleted and inserted that have the same set of columns as the underlying table being changed. A trigger has access to the before image and after image of the data via the special pseudo tables. The before and after values of specific columns can be checked and action taken depending on the values encountered. The inserted table can be referenced in a DELETE trigger and the deleted table can be referenced in an INSERT trigger. These tables are empty when the corresponding triggers are activated.

The DB2 UDB CREATE TRIGGER statement allows transition variables to be referenced using correlation names and provides referencing for transition tables by specifying table names. Correlation names identify a specific row in the set of rows affected by the triggering SQL statement, while table names identify the complete set of affected rows. Each row or set of rows affected by the triggering SQL statement is available to the triggered action by qualifying the desired columns with correlation names and table names.

9.4.2 Simple conversion example

Example 9-33 shows a straightforward SQL Server AFTER trigger. This trigger fires after an INSERT or UPDATE operation occurs on the redbooks table. This trigger enforces the business rule that an author can only write one redbook. Whenever a new record is inserted into the redbooks table, or a row in the redbooks table is updated with a new author_id value, the trigger will undo the insert/update if that author_id already exists somewhere in the table. A UNIQUE constraint on the author_id column could also be used instead of the trigger.

Example 9-33 SQL Server AFTER trigger

```
CREATE TRIGGER tr_ins_upd_brand
    ON redbooks
    FOR INSERT, UPDATE
AS
-- Has the author already written a book?
IF EXISTS
    (SELECT * FROM inserted WHERE inserted.author_id IN
      (SELECT author_id FROM redbooks) )
BEGIN
    RAISERROR('Author has already written a redbook.
              Statement will be aborted.', 16, 1)
    ROLLBACK TRAN
END
GO
```

Example 9-34 shows the equivalent the triggers in DB2 UDB.

Example 9-34 Equivalent DB2 UDB AFTER triggers

```
CREATE TRIGGER tr_upd_brand
    AFTER UPDATE ON redbooks
    REFERENCING OLD_TABLE AS deleted NEW_TABLE AS inserted
    FOR EACH STATEMENT
BEGIN ATOMIC
    IF EXISTS (
        SELECT * FROM inserted WHERE inserted.author_id IN (
            SELECT author_id FROM redbooks)) THEN

        SIGNAL SQLSTATE '50005'
        SET MESSAGE_TEXT = 'Author has already written a redbook. Statement will
                           be aborted.';

    END IF;
END
!
```

```
CREATE TRIGGER tr_ins_brand
    AFTER INSERT ON redbooks
    REFERENCING NEW_TABLE AS inserted
    FOR EACH STATEMENT
BEGIN ATOMIC
    IF EXISTS (
        SELECT * FROM inserted WHERE inserted.author_id IN (
            SELECT author_id FROM redbooks)) THEN

        SIGNAL SQLSTATE '50005'
        SET MESSAGE_TEXT = 'Author has already written a redbook. Statement will
                           be aborted.';
```

```
END IF;  
END  
!
```

Two DB2 UDB triggers are created: a trigger for the INSERT operation, and another for the UPDATE operation. Separate triggers must be defined for each type of triggering SQL statement in DB2 UDB.

It is also important to note that in DB2 UDB, a DELETE trigger cannot reference the NEW_TABLE transition table and an INSERT trigger cannot reference the OLD_TABLE transition table. To directly convert such functionality, references to the NEW_TABLE table in a DELETE trigger and the OLD_TABLE table in an INSERT trigger can be translated into a query that evaluates to an empty table (with the correct set of columns). In many cases, the statement containing such references is redundant and the trigger may be simplified by deleting the statement.

9.4.3 IF UPDATE(column)

The SQL Server IF UPDATE(column) condition can be used within a trigger to test if a particular column's value has been modified by the triggering SQL statement.

DB2 UDB supports similar functionality although using different syntax. To convert this condition to DB2 UDB, a WHEN (oldtable.column <> newtable.column) test can be used, or the column can be specified in the trigger's activation condition with the UPDATE OF <column> clause. Example 9-35 shows a SQL Server trigger where the IF UPDATE(column) function is used. This trigger fires after an update operation occurs on the authors table and the trigger logic only executes if the contract column is modified.

Example 9-35 SQL Server trigger using the IF UPDATE(column) function

```
CREATE TRIGGER tr_authors  
ON authors  
FOR UPDATE  
AS  
IF UPDATE(contract)  
-- executable-statements  
GO
```

Example 9-36 shows the equivalent trigger in DB2 UDB using the WHEN clause to simulate the IF UPDATE(column) function.

Example 9-36 Equivalent DB2 UDB trigger using a WHEN clause

```
CREATE TRIGGER tr_authors
```



```

    AFTER UPDATE ON authors
    REFERENCING OLD AS o NEW AS n
    FOR EACH ROW
    WHEN (o.contract <> n.contract)
BEGIN ATOMIC
    -- executable-statements
END
!
```

Example 9-37 shows another equivalent trigger in DB2 UDB using the UPDATE OF <column> syntax to simulate the IF UPDATE(column) function.

Example 9-37 Equivalent DB2 UDB trigger using the UPDATE OF <column> clause

```

CREATE TRIGGER tr_authors2
    AFTER UPDATE OF contract ON authors
    REFERENCING OLD AS o NEW AS n
    FOR EACH ROW
BEGIN ATOMIC
    -- executable-statements
END
!
```

9.4.4 Overcoming inline SQL PL limitations in triggers

Recall that DB2 UDB's procedural language for triggers, inline SQL PL, is actually a subset of DB2's SQL PL procedural language. As such, certain SQL PL elements that are supported in SQL stored procedures are not supported in triggers. Specifically, the following commonly used statements are not supported in DB2 UDB triggers:

- ▶ Transactional statements (COMMIT and ROLLBACK)
- ▶ Dynamic SQL statements (PREPARE, EXECUTE, EXECUTE IMMEDIATE)
- ▶ Cursor declarations (DECLARE <cursor>, ALLOCATE CURSOR)
- ▶ Complex exception handling (DECLARE ... HANDLER, RESIGNAL)
- ▶ SQL constructs (SELECT INTO, LOOP, REPEAT)

To convert a SQL Server trigger using any of the above features, there are two options:

- ▶ Attempt to replace the unsupported statement with an equivalent supported statement or remove it entirely.
- ▶ Move the unsupported statement into a new stored procedure and invoke the stored procedure from the trigger.

The first option is typically the easiest way to proceed. Sometimes, replacing an unsupported statement with a similar supported one, or removing it entirely is

possible. For instance, the LOOP and REPEAT statements can usually be implemented using WHILE statement logic. An explicit CURSOR declaration can sometimes be replaced with an implicit cursor using the FOR <cursor_name> AS logic. The SELECT INTO statement can be replaced with a SET statement.

Example 9-38 shows an example of a SQL Server trigger using an explicit cursor declaration. This trigger fires after an INSERT operation occurs on the titles table.

Example 9-38 SQL Server trigger with an explicit cursor declaration

```
CREATE TRIGGER tr_authors3
  ON authors
  FOR INSERT
AS
  DECLARE @rownum INT
  DECLARE @l_name VARCHAR(40)

  DECLARE c1 CURSOR FOR
    SELECT name FROM authors

  OPEN c1

  FETCH NEXT FROM c1
    INTO @l_name

  SET @rownum = 0
  WHILE @@FETCH_STATUS = 0
  BEGIN
    SET @rownum = @rownum + 1

    FETCH NEXT FROM c1
      INTO @l_name
  END

  CLOSE c1
  DEALLOCATE c1

  IF (@rownum > 5)
  BEGIN
    RAISERROR('Too many authors. Statement will be aborted.', 16, 1)
    ROLLBACK TRAN
  END

GO
```

Example 9-39 shows how this trigger can be converted to DB2 UDB by using an implicit cursor declaration. In this trigger, a NULL indicator variable is used to detect if the cursor does not return any data. If no data is returned, the v_rowcount

variable is not incremented. After the implicit cursor finishes moving through the result set, if the value of `v_rowcount` is greater than five, an exception is raised.

Example 9-39 Equivalent DB2 UDB trigger using an implicit cursor

```
CREATE TRIGGER tr_authors3
  NO CASCADE BEFORE INSERT ON authors
  FOR EACH ROW
  BEGIN ATOMIC
    DECLARE v_rowcount INTEGER;
    DECLARE l_name VARCHAR(40);
    DECLARE l_name_ind INTEGER;

    SET v_rowcount = 0;

    FOR c1 AS
      SELECT name, l AS null_ind FROM authors
    DO
      SET l_name = c1.name;
      SET l_name_ind = c1.null_ind;

      -- check to see if column value is actually a NULL value or
      -- the end of the cursor was reached (i.e. NO DATA FOUND)
      IF (l_name IS NULL AND l_name_ind IS NULL) THEN
        -- No data was returned, do nothing
      ELSE
        SET v_rowcount = v_rowcount + 1;
      END IF;
    END FOR;

    IF (v_rowcount > 5) THEN
      SIGNAL SQLSTATE '50000'
        SET MESSAGE_TEXT = 'Too many authors. Statement will be aborted.';
    END IF;
  END
!
```

Sometimes, it may not be possible to replace or remove the unsupported statement from the trigger in question. In such a case, it can be moved into a new SQL PL stored procedure, and the procedure can be invoked by the trigger.

Attention: If a procedure is invoked by trigger, a ROLLBACK or COMMIT statement is not allowed in the body of the procedure unless it is rolled back to a savepoint defined in the same procedure.

It is important to be aware of read and write conflict errors that can occur when a SQL stored procedure is invoked by a trigger in DB2 UDB. A set of data integrity

rules are enforced when a SQL stored procedure is invoked by a trigger. Essentially, DB2 UDB does not allow conflicting operations to occur when a stored procedure invoked by a trigger concurrently access the same table. If such a conflict occurs, an error (SQLCODE -746, SQLSTATE 57053) is returned to the statement that caused the conflict at runtime. It is therefore important to test all conditions to ensure that conflict errors do not occur at runtime.

9.4.5 INSTEAD OF triggers

SQL Server INSTEAD OF triggers override the triggering action with the logic specified in the trigger body. INSTEAD OF triggers can be defined on views, but AFTER triggers cannot. INSTEAD OF triggers can also be defined on tables. Only one INSTEAD OF can be defined for each triggering action (INSERT, UPDATE, and DELETE) on a table or view.

In DB2 UDB, INSTEAD OF triggers can only be defined on a view to perform an INSERT, UPDATE, or DELETE request on behalf of the view. When fired, the triggering SQL statement against the view gets replaced by the trigger logic, which performs the operation on behalf of the view. A DB2 UDB INSTEAD OF trigger is activated after the triggering statement is issued to the base view. Only one INSTEAD OF can be defined for each triggering action (INSERT, UPDATE, and DELETE) on a view.

Example 9-40 shows a SQL Server INSTEAD OF trigger defined on a view. This trigger fires when an INSERT statement is attempted against the view v_myView. Instead of inserting into the view, the logic contained in the trigger body is executed, namely inserting the first two columns of the rows that were attempted to be inserted into myView into the table myTable. The following view and table definitions are assumed by this example:

```
CREATE TABLE myTable (col1 INTEGER, col2 INTEGER)
CREATE VIEW myView AS SELECT col1, col2 FROM myTable
```

Example 9-40 SQL Server INSTEAD OF trigger defined on a sample view

```
-- The following definitions are assumed for this example
-- CREATE TABLE myTable (col1 INTEGER, col2 INTEGER)
-- CREATE VIEW myView AS SELECT col1, col2 FROM myTable

CREATE TRIGGER tr1 on v_myView
INSTEAD OF INSERT
AS
BEGIN
    INSERT INTO myTable
        SELECT col1, col2 FROM inserted
END
GO
```

Example 9-41 shows the equivalent DB2 UDB `INSTEAD OF` trigger.

Example 9-41 Equivalent DB2 UDB INSTEAD OF trigger

```
-- The following definitions are assumed for this example
-- CREATE TABLE myTable (col1 INTEGER, col2 INTEGER)
-- CREATE VIEW myView AS SELECT col1, col2 FROM myTable

CREATE TRIGGER tr1
  INSTEAD OF INSERT ON myView
  REFERENCING OLD AS o NEW AS n
  FOR EACH ROW
BEGIN ATOMIC
  INSERT INTO myTable
    VALUES(n.col1, n.col2)
END
!
```

SQL Server `INSTEAD OF` triggers that are defined on views can be converted as DB2 UDB `INSTEAD OF` triggers. SQL Server `INSTEAD OF` triggers that are defined on normal tables differ significantly in behavior than DB2 UDB `BEFORE` and `AFTER` triggers and a direct translation to DB2 UDB `INSTEAD OF` triggers would not be valid. The reason for this is that the SQL Server `INSTEAD OF` trigger action is executed instead of the triggering statement. In DB2 UDB `BEFORE` and `AFTER` triggers, both the triggering action *and* the triggering statements are executed eventually. Trigger redesign effort is required to convert this type of functionality to DB2 UDB if the same behavior is desired.

One method of performing this conversion involves renaming the original table, then creating a view with the same name as the original table. All the original statements in the application code referencing the table will now be referencing the view. The `INSTEAD OF` trigger can then be converted as a DB2 UDB `INSTEAD OF` trigger since it will operate on the newly created view.

9.4.6 Additional information

The IBM Press book *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS*, ISBN 0131477005, is the most comprehensive source for details and examples about using the SQL PL language, including inline SQL PL.

The following articles on the DB2 developerWorks Web site also provide useful information for trigger development:

- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/0308bhoga1/0308bhoga1.html>

- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/0211yip/0211yip.html>
- ▶ <http://www.ibm.com/developerworks/db2/library/techarticle/0210rielau/0210rielau.html>

Additional information can also be found in the DB2 UDB documentation
Application Development Guide: Programming Server Applications, SC09-4827.

Application conversion considerations

This chapter provides guidance for modifying SQL Server applications so they can be run with DB2 UDB.

In particular, the following topics are covered:

- ▶ Converting to DB2 UDB compatible SQL syntax
- ▶ Identifying transaction differences
- ▶ Identifying locking and lock escalation differences
- ▶ Identifying isolation level differences
- ▶ Identifying and modifying database interfaces
- ▶ Integrating DB2 UDB in Integration Development Environment (IDE)
- ▶ Approaching packaged application migration

10.1 Introduction

Modifying an application to run on DB2 UDB comprises:

- ▶ Converting existing SQL statements to DB2 UDB-compatible SQL.
- ▶ Identifying isolation level and locking requirements of the application.
- ▶ Making any necessary database Interface changes.

Since SQL Server and DB2 UDB do not implement the exact same components of the ANSI SQL standard, some of the SQL statements in the application may require alternation when converted to DB2 UDB. Any SQL Server proprietary SQL syntax or functions requires attention when converting to DB2 UDB. Usually this functionality can be simulated using DB2 UDB compatible features.

Aside from identifying SQL language differences, you also need to understand the transaction, locking, and isolation level differences between SQL Server and DB2 UDB. These differences influence how the application should be coded, and may require you to make some adjustments in order for it to run as expected.

The database interface being used to communicate with SQL Server can usually also be used with DB2 UDB, except in the case of proprietary interfaces, which are usually converted to use DB2's Call Level Interface (CLI) interface. When converting the application, you must determine which interface to use with DB2 UDB, configure the appropriate interface drivers, and modify the application for any function call differences, unsupported functions, and data type differences.

When converting a packaged application like SAP or Siebel, there are certain conversion limitations you must also be aware of and take into account, in order to maintain support from the vendor.

10.2 Converting to DB2 UDB compatible SQL syntax

Once the database conversion is complete, changes may be required in your application in order to be compatible with the DB2 UDB database. Some SQL statements in your application might use proprietary SQL Server features and/or syntax. MTK's SQL translator can help translate these SQL statements.

10.2.1 Using MTK's SQL Translator

To open the SQL Translator window, launch the MTK and open an existing project. You can create a new one if one does not exist already. Select **SQL Translator** from *Tools* menu, as shown Figure 10-1.

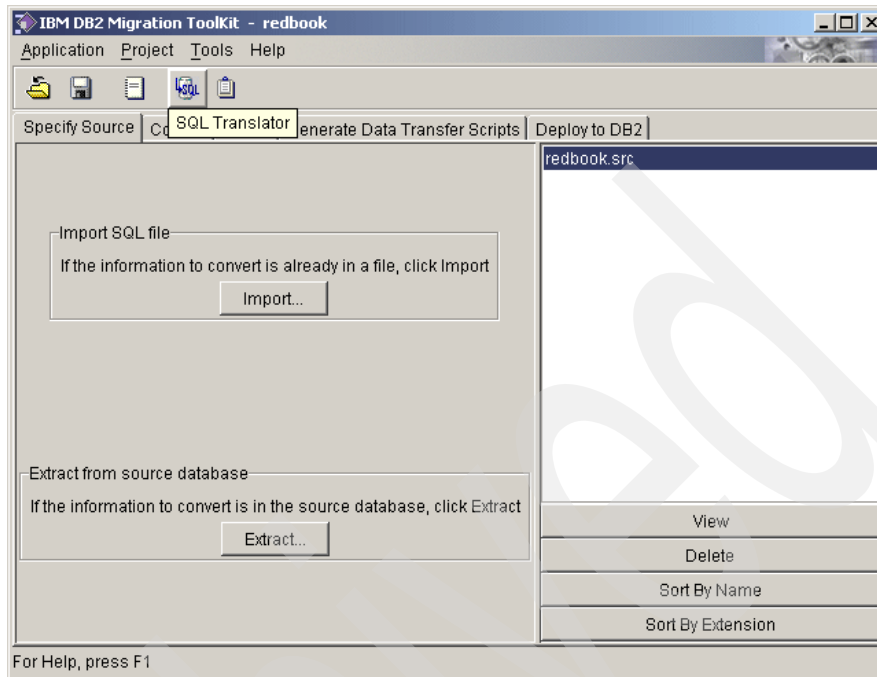


Figure 10-1 Opening the SQL Translator in MTK

Enter a SQL Server SQL statement in the white area of SQL Translator window (see Figure 10-2). Click the **Convert** button to convert the statement to DB2 UDB compatible syntax. Note that the MTK has some limitations and cannot convert all possible SQL Server syntax. Refer to the MTK documentation for more information about its limitations.

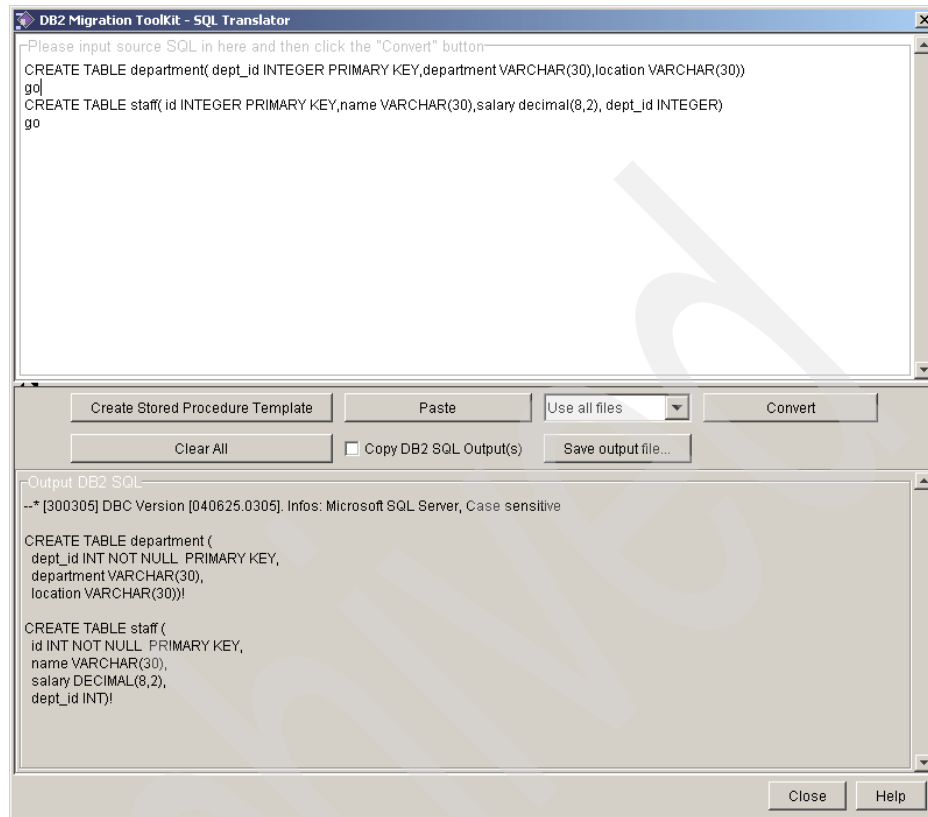


Figure 10-2 SQL Translator after converting a SQL Server SQL statement

10.2.2 Manual conversion

It is also possible to manually convert SQL statements in the application. This is usually the preferred approach when the changes required are very simple or few in number. Refer to Chapter 4, “SQL considerations” on page 65 for a comparison of SQL language syntax and semantics differences between SQL Server and DB2 UDB.

10.3 Identifying transaction differences

In SQL Server, explicit transactions are indicated using the key words BEGIN TRANSACTION, SAVE TRANSACTION, COMMIT TRANSACTION, and ROLLBACK TRANSACTION.

An implicit transaction is automatically started when those explicit key words are omitted. Such transactions need to be explicitly completed with a COMMIT or ROLLBACK statement.

In DB2 UDB, every SQL statement is part of a transaction. A group of SQL statements can be combined into a single executable block. This is known as *compound SQL*. In addition to reducing database manager overhead, compound SQL reduces the number of requests that are transmitted across the network for remote clients.

There are two types of compound SQL statements in DB2 UDB:

► ATOMIC

If one substatement ends in an error, the entire block is considered to have ended in an error. Any changes made to the database within the block are rolled back.

► NOT ATOMIC

All substatements within a block are executed regardless of whether or not the preceding substatement completed successfully. The group of statements can only be rolled back if the unit of work containing the NOT ATOMIC compound SQL is rolled back.

10.4 Identifying locking and lock escalation differences

SQL Server allows a range of database objects to be locked, from the entire database to a range of key values. Locks are primarily obtained as the result of data read and update requests, but can also be obtained for administrative reasons, such as during table reorganizations or index rebuilds.

DB2 UDB supports different levels of locking, which are similar to the SQL Server levels. Table 10-1 shows a comparison of SQL Server and DB2 UDB locking levels.

Table 10-1 Comparison of SQL Server and DB2 UDB locking levels

SQL Server	DB2 UDB
Database	Database
N/A	Table space
Table	Table
Row	Row
Index	Index

SQL Server	DB2 UDB
Page	N/A
Key	N/A
Range of keys	N/A

10.4.1 Types of locks

SQL Server supports three different types of base locks:

- ▶ Shared lock
- ▶ Update lock
- ▶ Exclusive lock

In addition, there are intent shared locks, intent exclusive locks, and intent update locks. SQL Server uses three additional modes of locking: schema stability locks, schema modification locks, and bulk update locks.

The type of lock chosen depends on the type of SQL statement issued and what isolation level is set. In general, every SQL statement that modifies data causes an exclusive lock to be requested on the corresponding database object. Shared locks are set when an SQL statement reads an object. Update locks are set when an application is using an UPDATE CURSOR.

DB2 UDB has a complete set of locks to ensure data integrity and provide access flexibility. Table 10-2 shows a complete list of DB2 UDB locks.

Table 10-2 A complete list of DB2 UDB locks

Lock type	Objects affected	Description
IN (Intent None)	Table spaces, blocks, tables	The lock owner can read any data in the object, including uncommitted data, but cannot update any of it. Other concurrent applications can read or update the table.
IS (Intent Share)	Table spaces, blocks, tables	The lock owner can read data in the locked table, but cannot update this data. Other applications can read or update the table.

Lock type	Objects affected	Description
NS (Next Key Share)	Rows	The lock owner and all concurrent applications can read, but not update, the locked row. This lock is acquired on rows of a table, instead of an S lock, where the isolation level of the application is either RS or CS. NS lock mode is not used for next-key locking. It is used instead of S mode during CS and RS scans to minimize the impact of next-key locking on these scans.
S (Share)	Rows, blocks, tables	The lock owner and all concurrent applications can read, but not update, the locked data.
IX (Intent Exclusive)	Table spaces, blocks, tables	The lock owner and concurrent applications can read and update data. Other concurrent applications can both read and update the table.
SIX (Share with Intent Exclusive)	Tables, blocks	The lock owner can read and update data. Other concurrent applications can read the table.
U (Update)	Rows, blocks, tables	The lock owner can update data. Other units of work can read the data in the locked object, but cannot update it.
NW (Next Key Weak Exclusive)	Rows	When a row is inserted into an index, an NW lock is acquired on the next row. For type 2 indexes, this occurs only if the next row is currently locked by an RR scan. The lock owner can read, but not update the locked row. This lock mode is similar to an X lock, except that it is also compatible with W and NS locks.
X (Exclusive)	Rows, blocks, tables, buffer pools	The lock owner can both read and update data in the locked object. Only uncommitted read applications can access the locked object.

Lock type	Objects affected	Description
W (Weak Exclusive)	Rows	This lock is acquired on the row when a row is inserted into a table that does not have type 2 indexes defined. The lock owner can change the locked row. To determine if a duplicate value has been committed when a duplicate value is found, this lock is also used during insertion into a unique index. This lock is similar to an X lock, except that it is compatible with the NW lock. Only uncommitted read applications can access the locked row.
Z (Super Exclusive)	Table spaces, tables	This lock is acquired on a table in certain conditions, such as when the table is altered or dropped, an index on the table is created or dropped, or for some types of table reorganization. No other concurrent application can read or update the table.

Explicitly setting the locking mode

Although locks are set implicitly when modifying a row, a table or database can be explicitly locked. The reasons for locking an entire table could be for a bulk data load or for administrative purposes.

The LOCK TABLE command syntax is shown in Figure 10-3.

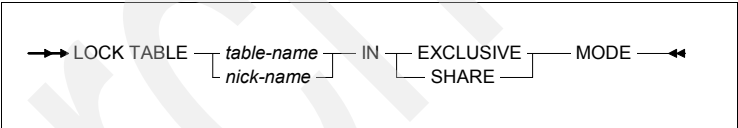


Figure 10-3 DB2 UDB LOCK TABLE syntax

An entire database can be locked explicitly when the database connection is established. DB2 UDB enables a database to be locked in either SHARE or EXCLUSIVE mode:

SHARE	Another user can connect the database, but is not allowed to lock the database in exclusive mode. This is the default setting when a lock mode is not specified.
EXCLUSIVE	The database is locked exclusively, which means that no other user can access the database.

The database connection syntax is shown in Figure 10-4.

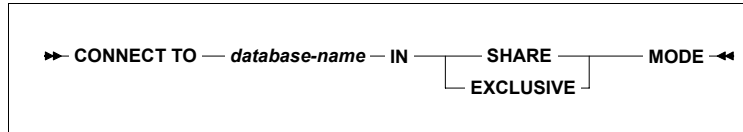


Figure 10-4 `CONNECT` database statement in DB2 UDB with lock mode

10.4.2 Lock escalation

Lock escalation is an internal mechanism that reduces the number of locks held on rows of a single table. The database may escalate many row locks to a table lock, or for multidimensional clustering (MDC) tables, from many row locks to a block lock. Lock escalation occurs when applications hold too many locks of any type. Lock escalation can also occur when a specific database agent exceeds its allocation of the lock list. Such escalation is handled internally; the only externally detectable result might be a reduction in concurrent access on one or more tables.

In an appropriately configured database, lock escalation occurs infrequently. For example, lock escalation might occur when an application designer creates an index on a large table to increase performance and concurrency, but a transaction accesses most of the records in the table. In this case, because the database manager cannot predict how much of the table should be locked, it locks each record individually instead of locking only the entire table. In this case, the DBA might consult with the application designer and recommend a `LOCK TABLE` statement for this transaction.

10.4.3 Deadlock

Deadlock occurs when two or more transactions wait indefinitely for each others locks, until an external event causes one or both of them to *rollback*. DB2 UDB uses a deadlock detection mechanism to check the system for deadlocks and selects one of the deadlocked transactions to be rolled back. An error code is returned to the application in this case.

A deadlock event can be recorded using the *deadlock event monitor*. This monitor can capture additional information, such as the SQL statement that 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 held them.

Tips for minimizing or resolving deadlocks include:

- ▶ Connections should process their own set of rows to help avoid lock incompatibilities or waits.
- ▶ Transactions should access tables in the same order.
- ▶ In some cases, try to ensure that a lock time out occurs first. Set the value of the database configuration parameter LOCKTIMEOUT to be lower than the value of DLCHKTIME.

For more information about diagnosing locking problems, refer to article *Diagnosing and Resolving Lock Problems with DB2 Universal Database* available at:

<http://www.ibm.com/developerworks/db2/library/techarticle/0310wilkins/0310wilkins.html>

More information can also be found in the DB2 UDB documentation *Application Development Guide: Programming Client Applications*, SC09-4826.

10.5 Identifying isolation level differences

The isolation level controls when and what kind of locks are obtained on a database object such as a row, page, or table. DB2 UDB supports the SQL ANSI isolation levels.

Table 10-3 shows a comparison of SQL Server and DB2 UDB isolation levels.

Table 10-3 A comparison of SQL Server and DB2 UDB isolation levels

SQL Server	DB2 UDB
Read Committed (default)	Cursor Stability (CS, default)
Serializable	Repeatable Read (RR)
Repeatable Read	Read Stability (RS)
Uncommitted Read	Uncommitted Read (UR)

In the following sections, we provide more detailed descriptions of the isolation levels in DB2 UDB.

Default isolation level

SQL Server's default isolation level is Read Committed. DB2 UDB's default isolation level is Cursor Stability. This ensures that any row that is changed by another application cannot be read until it is committed by that application.

How to set an isolation level

SQL Server uses its own method of changing between isolation levels (SET TRANSACTION ISOLATION LEVEL). DB2 UDB offers different techniques for changing the isolation level.

Changing the isolation level for an individual SQL statement

In DB2 UDB, to set the isolation level for a single SQL statement, the WITH clause can be added at the end of a SELECT SQL statement. Example 10-1 shows the WITH clause syntax.

Example 10-1 Changing the isolation level for an individual SQL statement

```
SELECT <columns>  
FROM <tables> ... WITH {UR|RR|CS|RS}
```

Changing the isolation level for a session

To change the isolation level for the duration of a entire session, the SET CURRENT ISOLATION statement can be used. This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Example 10-2 shows the syntax SET CURRENT ISOLATION statement.

Example 10-2 Changing the isolation levels for the session

```
SET CURRENT ISOLATION={UR|RR|CS|RS}
```

Changing the isolation level in packages

When a database package is created, it must eventually be bound against a database. When using the BIND utility, you can also set the default isolation level for this package. All SQL statements in the package will run with the isolation level specified, except those that explicitly specify their own isolation level. Example 10-3 demonstrates the impact of using the ISOLATION option when binding a package.

Example 10-3 Changing the isolation level in DB2 UDB package

```
C:\temp\samples\c>db2 bind utilemb.bnd ISOLATION UR
```

```
LINE      MESSAGES FOR utilemb.bnd
```

```
-----  
SQL0061W  The binder is in progress.  
SQL0091N  Binding was ended with "0" errors and "0" warnings.
```

```
C:\temp\samples\c>db2expln -database sample -schema emma -package utilemb -t
```

```
DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002  
Licensed Material - Program Property of IBM  
IBM DB2 Universal Database SQL Explain Tool
```

```
***** PACKAGE *****
```

```
Package Name = "EMMA"."UTILEMB"  Version = ""
```

```
Prep Date = 2005/02/14
```

```
Prep Time = 19:00:51
```

```
Bind Timestamp = 2005-02-14-19.01.24.103000
```

```
Isolation Level           = Uncommitted Read
```

```
Blocking                   = Block Unambiguous Cursors
```

```
Query Optimization Class = 5
```

```
Partition Parallel         = No
```

```
Intra-Partition Parallel = No
```

```
SQL Path                   = "SYSIBM", "SYSFUN", "SYSPROC", "EMMA"
```

```
No static sections qualify from package.
```

10.5.1 Repeatable Read (RR)

This isolation level ensures that:

- ▶ Any row read during a unit of work is not changed by other application processes until the unit of work is complete. Rows are read in the same unit of work as the corresponding cursor OPEN statement. Use of the optional WITH RELEASE clause on the cursor CLOSE statement means that any guarantees against non-repeatable reads and phantom reads no longer apply to previously accessed rows if the cursor is re-opened.
- ▶ Any row changed by another application process cannot be read until it is committed by that application process.

The Repeatable Read level does not allow phantom rows to be viewed. See the following section for more information about phantom rows

In addition to obtaining any exclusive locks, an application process using at the RR level acquires at least share locks on *all* the rows it references. Furthermore, locking is performed so that the application process is completely isolated from the effects of concurrent application processes.

10.5.2 Read Stability (RS)

Like the Repeatable Read level, the Read Stability level ensures that:

- ▶ Any row read during a unit of work is not changed by other application processes until the unit of work is complete. The rows are read in the same unit of work as the corresponding cursoriness statement. Use of the optional WITH RELEASE clause on the cursor CLOSE statement means that any guarantees against non-repeatable reads no longer apply to any previously accessed rows if the cursor is re-opened.
- ▶ Any row changed by another application process cannot be read until it is committed by that application process.

Unlike Repeatable Read, Read Stability does not completely isolate the application process from the effects of concurrent application processes. At the RS level, application processes that issue the same query more than once might see additional rows caused by other application processes adding new information to the database. These additional rows are called *phantom rows*.

For example, a phantom row can occur in the following situation:

1. Application process P1 reads the set of rows *n* that satisfy some search condition.
2. Application process P2 then inserts one or more rows that satisfy the search condition and commits those new inserts.
3. P1 reads the set of rows again with the same search condition and obtains both the original rows and the rows inserted by P2.

In addition to any exclusive locks, an application process running at the RS isolation level acquires at least share locks on all the *qualifying* rows.

10.5.3 Cursor Stability (CS)

Like the Repeatable Read level, the Cursor Stability level ensures that any row that is changed by another application process cannot be read until it is committed by that application process.

Unlike Repeatable Read, Cursor Stability only ensures that the current row of every UPDATE cursor is not changed by other application processes. Therefore, the rows that were read during a unit of work can be changed by other application processes.

In addition to any exclusive locks, an application process running at the CS isolation level acquires at least a share lock on the *current* row of every cursor.

10.5.4 Uncommitted Read (UR)

For SELECT INTO and FETCH with a read-only cursor, full-select in an INSERT statement, row full-select in an UPDATE statement, or scalar full-select (wherever it is used), the Uncommitted Read isolation level allows:

- ▶ Any row read during a unit of work to be changed by other application processes.
- ▶ Any row changed by another application process to be read, even if the change has not been committed by that application process.

For other operations, rules associated with the CS level apply.

10.6 Identifying and modifying database interfaces

Applications written to run with SQL Server or DB2 UDB can use a variety of interfaces to communicate with the database. The following applications and interfaces are supported by SQL Server:

- ▶ .NET applications:
SQL Server .NET data provider, OLE DB .NET data provider, ODBC .NET data provider (ADO.NET interfaces)
- ▶ Visual Basic applications:
*Data Access Objects (DAO), *Remote Data Objects (RDO), and ActiveX Data Objects (ADO), Open Database Connectivity (ODBC)
- ▶ Visual C++ applications:
**DAO, ODBC, and Object Linking and Embedding Data Base (OLE DB)
- ▶ Visual J++ applications:
Java Database Connectivity (JDBC)
- ▶ C Applications:
DB-Library API, Microsoft Embedded SQL for the C development environment (ESQL/C), ODBC, OLE DB

- Database administration applications:

Windows Management Instrument (WMI) provider

* will be deprecated and is suggested to be changed to ADO

** will be deprecated and is suggested to be changed to OLE DB or ODBC

DB2 UDB provides the *DB2 .NET Data Provider*, which allows .NET applications to access DB2 UDB database directly. OLE DB .NET and ODBC .NET data providers can also be used for .NET applications for direct access, but not as favorably as with the DB2 .NET data provider because they do not have the functionality or performance. Visual Basic applications, Visual C++ applications, and applications that conform to DAO and RDO, can access DB2 UDB using OLE DB, an ODBC bridge, or the IBM OLE DB Provider for DB2. For Java applications, JDBC or embedded SQL for Java (SQLJ) can be used.

The main task involved in converting the application interfaces involves ensuring that all function calls used and supported by the SQL Server interface driver are supported by the DB2 UDB driver. In most cases, they are and no changes are required. In some cases an unsupported function call may need to be removed or simulated using other methods. Any proprietary interfaces used, such as the DB-Library API, require changes to work with DB2 UDB. Usually, DB2 UDB's Call Level Interface (CLI) is used in these situations.

The following sections discuss the different DB2 UDB data providers and key considerations when using them.

10.6.1 ActiveX Data Objects .NET (ADO.NET) interfaces

ADO.NET is a suite of data access technologies included in the .NET Framework class libraries. As shown in the Figure 10-5, there are three providers that ADO.NET applications can use to access DB2 UDB:

- DB2 .NET Data Provider
- OLE DB .NET Data Provider
- ODBC .NET Data Provider

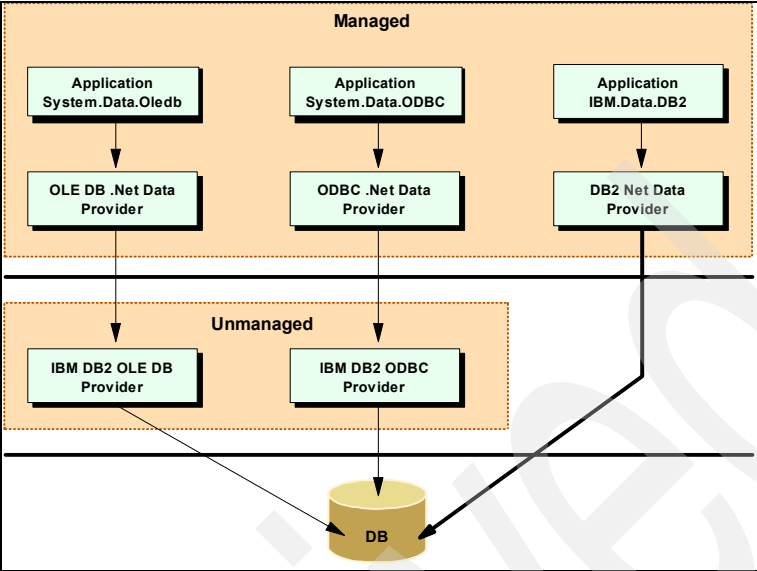


Figure 10-5 Providers in .NET to connect to DB2 UDB

The OLE DB .NET and ODBC .NET data providers are *bridge providers*. These type of providers enable ADO.NET access to any data source using existing data access technologies such as OLE DB and ODBC. The OLE DB .NET Data Provider uses the OLE DB provider whereas the ODBC .NET Data Provider uses the ODBC driver.

SQL Server's SQL .NET provider and DB2 UDB's .NET provider are *native providers* that communicate with only one specific data source. Native providers always yield better performance than bridge providers.

DB2 .NET data provider

Table 10-4 compares the SQL .NET and DB2 .NET provider functions at the code level. Ensure the appropriate changes are made in your application when porting to DB2 UDB.

Table 10-4 .NET provider code-level comparison

Object	SQL .NET Provider	DB2 .NET Provider
Namespace	System.Data.SqlClient	IBM.Data.DB2
Connection object (used to create connection with DB)	SqlConnection	DB2Connection

Object	SQL .NET Provider	DB2 .NET Provider
Command object (used to execute command)	SqlCommand	DB2Command
Data reader object (used to read retrieved data)	SqlDataReader	DB2DataReader
DataAdapter	SqlDataAdapter	DB2DataAdapter
Parameter	SqlParameter	DB2Parameter

If you installed Visual Studio .NET after DB2 UDB, you must register the DB2 Visual Studio add-in before being able to leverage all the integrated features in the Visual Studio integrated development environment (IDE).

To register the add-in, execute the following command from a command window:

```
C:\%DB2HOME%\SQLLIB\BIN\db2vsregister.bat
```

To use the add-in in an existing project, you must add a reference to the data provider library in your project.

1. Click **Project** menu → **Add Reference**
2. At the .NET tab, search for **IBM.Data.DB2.dll** in the component name column
3. Highlight it and click **Select**
4. Click **OK** to add the reference

In each class, add a reference to the DB2 name space:

```
[C#]: using IBM.Data.DB2;
[Visual Basic]: Imports IBM.Data.DB2
```

Replace the following object references in your application code to the DB2 UDB supported object references:

- ▶ SqlConnection → DB2Connection
- ▶ SqlDataReader → DB2DataReader
- ▶ SqlCommand → DB2Command
- ▶ SqlDataAdapter → DB2DataAdapter

Also change the data type enumerations to use the DB2 UDB supported types, for example:

- ▶ SqlDbType.Int → DB2Type.Int32
- ▶ SqlDbType.NVarChar → DB2Type.String
- ▶ SqlDbType.DateTime → DB2Type.DateTime

► SqlDbType.Money → DB2Type.Decimal

For a complete list, refer to DB2 .NET Data Provider reference documentation, available at:

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.dndp.doc/htm/fr1rfIBMDDataDB2Hierarchy.htm>

OLE DB .NET and ODBC .NET data providers

As illustrated in Figure 10-5 on page 240, the OLE DB .NET and ODBC .NET data providers require an extra layer, which can affect performance. There are different unmanaged providers available. The ones provided by IBM are the IBM OLE DB and IBM ODBC data providers. Example 10-4 compares the typical connection syntax for each provider.

Example 10-4 Comparison of typical connection syntax using OLE DB and ODBC

Microsoft OLE DB:

```
OleDbConnection con = new OleDbConnection("Provider=SQLOLEDB;  
Data source=localhost;Integrated Security=SSPI;Initial Catalog=redbook")
```

IBM OLE DB:

```
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;  
Data Source=REDBOOK;UID=db2user;" + PWD=password");
```

Microsoft ODBC:

```
OdbcConnection con = new OdbcConnection("Driver={SQL Server};  
Server=localhost;Trusted_Connection=yes;Database=redbook");
```

IBM ODBC:

```
OdbcConnection con = new  
OdbcConnection("DSN=redbook;UID=userid;PWD=password");
```

For Visual Basic applications, the connection parameters are the same as those in Example 10-4. These providers do not require many changes in the application code when converting to DB2 UDB, but you should investigate any additional usage restrictions they may have. Refer to the DB2 UDB documentation *Application Development Guide: Programming Client Applications*, SC09-4826, for more information.

10.6.2 ActiveX Data Object (ADO)

Generally, if the OLE DB or ODBC interfaces are used, fewer application code changes are required than if a native driver is used. Converting from Microsoft OLE DB to IBM OLE DB requires some change, but converting from Microsoft ODBC to IBM ODBC is relatively straightforward and few changes are required.

Object Linking and Embedding Database (OLE DB)

The IBM OLE DB Provider for DB2 allows DB2 UDB to act as a resource manager for the OLE DB provider. Example 10-5 shows a comparison of a typical connection string between the SQL Server OLE DB driver and the DB2 UDB OLE DB driver.

Example 10-5 Converting an OLE DB connection string from SQL Server to DB2 UDB

Microsoft OLE DB for SQL Server:

```
"Provider=SQLOLEDB;Data Source=serverName;Initial Catalog=REDBOOK;User ID=userid;Password=password;"
```

IBM OLE DB for DB2 UDB:

```
"Provider=IBMDADB2;Data Source=REDBOOK;UID=userid;PWD=password");
```

Table 10-5 provides a mapping between the OLE DB data types used in SQL Server and DB2 UDB.

Table 10-5 Mapping of OLE DB data types between SQL Server and DB2 UDB

DB2 UDB Data Type (DT)	Supported OLE DB DT	OLE DB DT	SqlServer DT
SMALLINT	DBTYPE_I2	DBTYPE_UI1	smallint
SMALLINT	DBTYPE_I3		tinyint
INTEGER	DBTYPE_I4		Int
BIGINT	DBTYPE_I8		bigint
REAL	DBTYPE_R4		Float
FLOAT/DOUBLE	DBTYPE_R8	DBTYPE_I1 DBTYPE_UI2 DBTYPE_UI4 DBTYPE_UI8 DBTYPE_DECIMAL DBTYPE_CY	Real
DEC (p,s)	DBTYPE_NUMERIC(p,s)		numeric
DEC (3,0)	DBTYPE_NUMERIC (3,s)		numeric(3, 0)
DEC (5,0)	DBTYPE_NUMERIC (5,s)		numeric(5,0)
DEC (10,0)	DBTYPE_NUMERIC (10,s)		numeric(10,0)
DEC (20,0)	DBTYPE_NUMERIC (20,s)		numeric(20,0)
DEC (p,s)	DBTYPE_NUMERIC (p,s)		decimal
DEC (19,4)	DBTYPE_NUMERIC (19,4)		money
DATE	DBTYPE_DBDATE		datetime
TIME	DBTYPE_DBTIME		
TIMESTAMP	DBTYPE_DBTIMESTAMP	DBTYPE_DATE	datetime
TIMESTAMP	DBTYPE_DBTIMESTAMP		datetime
CHAR(N)	DBTYPE_STR	DBTYPE_BOOL DBTYPE_GUID	char
VARCHAR(N)	DBTYPE_STR		varchar
LONG VARCHAR	DBTYPE_STR		text
CLOB(N)	DBTYPE_STR		text
CHAR(1) FOR BIT DATA	DBTYPE_BYTES		Bit
CHAR(13) FOR BIT DATA	DBTYPE_BYTES		uniqueidentif
CHAR(N) FOR BIT DATA	DBTYPE_BYTES		ier
VARCHAR(N) FOR BIT DATA	DBTYPE_BYTES		binary
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES		varbinary
BLOB(N)	DBTYPE_BYTES		
GRAPHIC(N)	DBTYPE_WSTR	DBTYPE_BSTR	image
GRAPHIC(N)	DBTYPE_WSTR		nchar
VARGRAPHIC(N)	DBTYPE_WSTR	DBTYPE_BSTR DBTYPE_VARIANT	nchar
VARGRAPHIC(N)	DBTYPE_WSTR		nvarchar
VARGRAPHIC(N)	DBTYPE_WSTR	DBTYPE_BSTR	nvarchar
LONG GRAPHIC	DBTYPE_WSTR		nvarchar(4000)
DBCLOB(N)	DBTYPE_WSTR		ntext

The IBM OLE DB data provider supports a large number of components, interfaces, and properties. The list of non-supported interfaces is shown in Table 10-6.

Table 10-6 IBM OLE DB Driver non-supported components and interfaces

Components	Interfaces
Command DataSource	ICommandPersist IConnectionPoint IDBAsynchNotify (consumer) IDBAsynchStatus IDBConnectionPointContainer
RowSet	IDBDataSourceAdmin IChapteredRowset IDBAsynchStatus IParentRowset IRowsetChapterMember IRowsetFind IRowsetIndex IRowsetView
Session	IAAlterIndex IAAlterTable IIndexDefinition ITableDefinition ITableDefinitionWithConstraints
View Objects	ITransactionObject ITransactionOptions IViewChapter IViewFilter IViewRowset IViewSort

ODBC and CLI

DB2 UDB's Open Database Connectivity (ODBC)/Call Level Interface (CLI) driver (*db2cli.dll*) is based on Microsoft's ODBC specification and the International Standard for SQL/CLI. ODBC applications can be converted to DB2 UDB without using any ODBC driver manager by linking your application with the DB2 UDB CLI driver. When this driver is loaded by the Microsoft ODBC driver manager, it acts as an ODBC driver.

Refer to Figure 10-6 for an illustration of the ODBC and CLI environment.

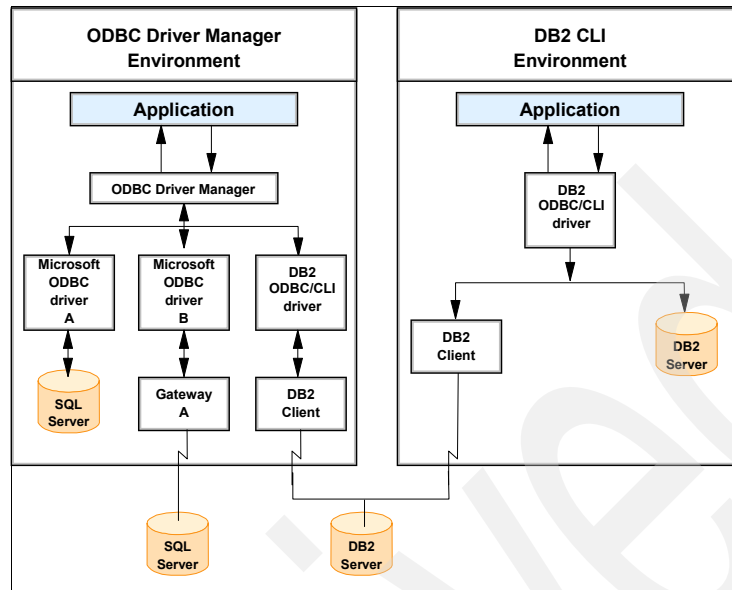


Figure 10-6 ODBC Driver Manager environment versus DB2 CLI environment

If the DB2 ODBC driver, going through the ODBC Driver Manager, is used, all the functions based on ODBC V3.51 are supported. DB2-specific functions cannot be accessed in an ODBC environment unless an escape clause is used.

Applications using DB2 CLI link directly to the DB2 CLI library. DB2 CLI includes support for many ODBC and ISO SQL/CLI functions, as well as DB2 specific functions. Table 10-7 shows a comparison of available functions between the DB2 ODBC driver and the DB2 CLI driver.

Table 10-7 DB2 CLI and ODBC function mapping

ODBC 3.51 Features	DB2 ODBC Driver	DB2 CLI Driver
Core Level Functions	All	All
Level 1 Functions	All	All
Level 2 Functions	All	All, except for SQLDrivers()

ODBC 3.51 Features	DB2 ODBC Driver	DB2 CLI Driver
Additional DB2 CLI Functions	All functions can be accessed by dynamically loading the DB2 CLI library	"SQLSetConnectAttr() "SQLGetEnvAttr() "SQLSetEnvAttr() "SQLSetColAttributes() "SQLGetSQLCA() "SQLBindFileToCol() "SQLBindFileToParam() "SQLExtendedBind() "SQLExtendedPrepare() "SQLGetLength() "SQLGetPosition() "SQLGetSubString()
Return codes	All listed for DB2 CLI Driver.	"SQL_SUCCESS "SQL_SUCCESS_WITH_INFO "SQL_STILL_EXECUTING "SQL_NEED_DATA "SQL_NO_DATA_FOUND "SQL_ERROR "SQL_INVALID_HANDLE
SQLSTATES	Mapped to X/Open SQLSTATES with additional IBM(R) SQLSTATES, with the exception of the ODBC type 08 S01.	Mapped to X/Open SQLSTATES with additional IBM SQLSTATES
Multiple connections per application	Supported	Supported

ODBC 3.51 Features	DB2 ODBC Driver	DB2 CLI Driver
SQL Data Types	All listed for DB2 CLI Driver.	"SQL_BIGINT "SQL_BINARY "SQL_BIT "SQL_BLOB "SQL_BLOB_LOCATOR "SQL_CHAR "SQL_CLOB "SQL_CLOB_LOCATOR "SQL_DBCLOB "SQL_DBCLOB_LOCATOR "SQL_DECIMAL "SQL_DOUBLE "SQL_FLOAT "SQL_GRAPHIC "SQL_INTEGER "SQL_LONG "SQL_LONGVARIABLE "SQL_LONGVARIABLE "SQL_LONGVARIABLEGRAPHIC "SQL_NUMERIC "SQL_REAL "SQL_SHORT "SQL_SMALLINT "SQL_TINYINT "SQL_TYPE_DATE "SQL_TYPE_TIME "SQL_TYPE_TIMESTAMP "SQL_VARIABLE "SQL_VARIABLE "SQL_VARIABLEGRAPHIC "SQL_WCHAR
C Data Types	All listed for DB2 CLI Driver.	"SQL_C_BINARY "SQL_C_BIT "SQL_C_BLOB_LOCATOR "SQL_C_CHAR "SQL_C_CLOB_LOCATOR "SQL_C_DATE "SQL_C_DBCHAR "SQL_C_DBCLOB_LOCATOR "SQL_C_DOUBLE "SQL_C_FLOAT "SQL_C_LONG "SQL_C_SHORT "SQL_C_TIME "SQL_C_TIMESTAMP "SQL_C_TINYINT "SQL_C_SBIGINT "SQL_C_UBIGINT "SQL_C_NUMERIC* "SQL_C_WCHAR

10.6.3 Java Database Connectivity (JDBC)

DB2 UDB implements two standards-based database interfaces for Java applications: JDBC and embedded SQL for Java (SQLJ).

In DB2 UDB Version 8, a new *Universal JDBC Driver* is provided. This driver is completely written in Java. The driver library, **db2jcc.jar**, can be found in the C:\DB2_HOME\SQLLIB\JAVA directory. This driver supports both JDBC and SQLJ APIs in a single implementation. Both JDBC and SQLJ can interoperate in the same application.

If your application is written in Java and uses JDBC to communicate with the database, it is straightforward to convert to DB2 UDB. The JDBC API is well defined and is database independent. For instance, the database connection logic is encapsulated in standard J2EE Data Source objects. The changes required include the JDBC driver file (which comes with DB2 UDB), the database connection string, as well as syntax for SQL statements.

Java applications can access DB2 UDB in one of the following ways:

- ▶ DB2 Server
 - Stored procedures (JDBC or SQLJ)
 - User-defined functions (JDBC or SQLJ)
- ▶ Internet browser
 - Applets based on JDBC (JDBC)
- ▶ J2EE Application Servers (such as WebSphere Application Server)
 - Java ServerPages (JSPs) (JDBC)
 - Servlets (SQLJ or JDBC)
 - Enterprise JavaBeans (EJBs) (SQLJ or JDBC)

For more information about Java application development, refer to the DB2 UDB Java Web site at:

<http://www.ibm.com/software/data/db2/udb/ad/v8/java/>

10.6.4 Embedded SQL for C (ESQL/C)

SQL Server provides a precompiler for embedded SQL (C) applications. It translates embedded SQL statements to the appropriate DB-Library API function calls.

DB2 UDB also supports embedded SQL programming for C applications; however, there are some differences between ESQL/C for SQL Server and DB2

UDB. This section highlights the steps needed to convert a C application with embedded SQL to DB2 UDB.

Connecting to a database

The **CONNECT TO** statement, along with the server name, database name, user name, and password, can be used to connect to a SQL Server database. If the server name is not specified, the local server is assumed. The password is optional and the user name can be replaced by \$integrated to use Windows Authentication.

The connection syntax in DB2 UDB is slightly different. The database name specification is mandatory, but the user name and password are optional. Example 10-6 shows a comparison between SQL Server and DB2 UDB **CONNECT TO** syntax. Depending on the authentication type set at the DB2 UDB instance, authentication location may vary when the user name is not provided. More information about this topic can be found in DB2 UDB documentation *Administration Guide: Implementation*, SC09-4820.

Example 10-6 CONNECT TO syntax using ESQL/C in SQL Server and DB2 UDB

```
// SQL Server CONNECT TO Syntax Examples:
EXEC SQL CONNECT TO servername.dbname USER "userid";
EXEC SQL CONNECT TO servername.dbname USER "userid.password";
EXEC SQL CONNECT TO servername.dbname USER $integrated;

// DB2 UDB CONNECT TO Syntax Examples:
EXEC SQL CONNECT TO dbname;
EXEC SQL CONNECT TO dbname USER userid USING password;
```

More information about the **CONNECT TO** command can be found in the DB2 UDB documentation *SQL Reference - Volume 2*, SC09-4845.

Host variable declaration

C application variables referenced in ESQL/C SQL statements are called *host variables*. Host variables allow an application to pass input data to and receive output data from the database. After the application is precompiled, host variables are treated as any other C variable by the compiler.

Host variables must be declared in C, not in Transact-SQL or SQL PL. Declared variables should be compatible with DB2 UDB data types, compatible with the DB2 UDB precompiler, as well as compatible with the C compiler. Appendix B, “Data type mapping” on page 469 provides a mapping of SQL data types to C data types.

Host variables in the C application must be declared in a special declaration section in order for the precompiler to identify the host variables and their data types. Example 10-7 shows a sample host variable declaration.

Example 10-7 Sample ESQL/C DECLARE SECTION in a C application

```
[..]  
EXEC SQL INCLUDE SQLCA ;  
EXEC SQL BEGIN DECLARE SECTION;  
    CHAR firstname[20] = {'\0'};  
    sqlint32 ret_code = 0;  
EXEC SQL END DECLARE SECTION;  
[..]
```

Error messages and warnings

The *SQL Communications Area* (SQLCA) is a collection of variables that is updated after the execution of every SQL statement. SQL Server uses the SQLCA data structure to trap and return run-time error information to the C application using embedded SQL.

When you pre-compile an E/SQL program, the compiler inserts host language variable declarations in place of the INCLUDE SQLCA statement. After executing a SQL statement, the database system fills in both SQLCODE and SQLSTATE with values.

Both SQLCODE and SQLSTATE indicate success or failure statement execution; however, SQLSTATE provides common error codes across the IBM relational database products. SQLSTATE also conforms to the ISO/ANSI SQL92 and FIPS 127-2 standard.

Note that if SQLCODE has a value that is less than 0, it means that an error has occurred and the statement has not been processed. If SQLCODE has a value greater than 0, it means a warning has been issued, but the statement was still processed.

Example 10-8 shows how to declare SQLCODE and SQLSTATE in a DB2 UDB ESQL/C application. The following PRECOMPILE command sets the LANGLEVEL precompile option to SQL92E:

```
PRECOMPILE applicationCode.sqc BINDFILE LANGLEVEL SQL92E
```

Example 10-8 Declare SQLCODE and SQLSTATE in ESQL/C code for DB2 UDB

```
EXEC SQL BEGIN DECLARE SECTION;  
    char      SQLSTATE[6]  
    sqlint32  SQLCODE;
```

EXEC SQL END DECLARE SECTION;

If neither of these variables is specified, the SQLCODE declaration is assumed during the precompile step. When using the LANGLEVEL option, the INCLUDE SQLCA statement should not be specified.

More information about this topic can be found in the DB2 UDB documentation *Application Development Guide: Programming Client Applications*, SC09-4826.

Building C-application with ESQL/C on DB2 UDB

Figure 10-7 depicts the procedure of building an ESQL/C application with DB2 UDB.

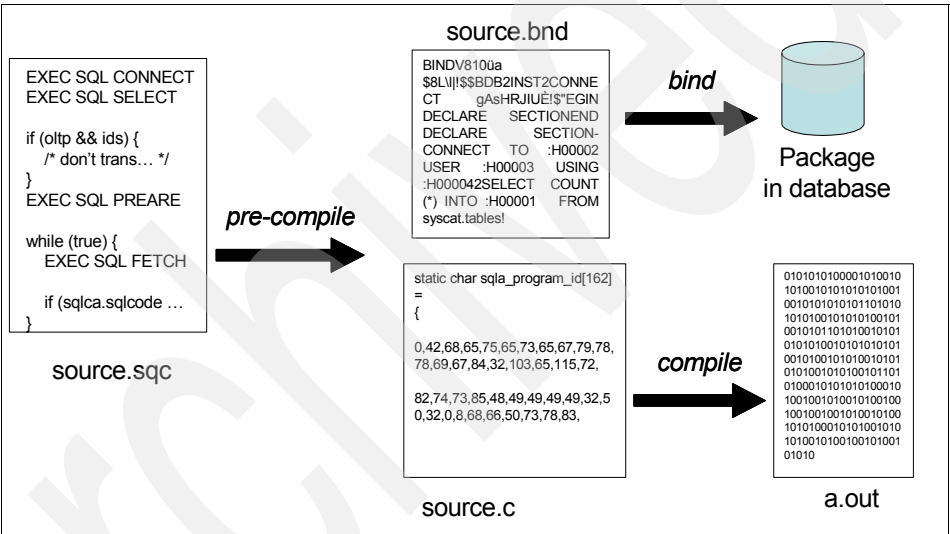


Figure 10-7 Procedure to build ESQL/C application with DB2 UDB

The precompiler (**PRECOMPILE/REP** command) is needed to convert SQL statements into DB2 run-time API calls. It also creates the information the database manager needs to process the SQL statements against the database. After precompiling, the bind files must be bound to the database to generate a package containing the access plans for the SQL statements. After precompiling and binding, the resulting C file can be compiled into an executable or library.

DB2 UDB provides sample build scripts for precompiling, compiling, and linking programs with ESQL/C. These are located in the `C:\DB2_HOME\SQLLIB\SAMPLES\C` directory, along with sample programs.

For more information about building C applications, refer to the DB2 UDB documentation:

- ▶ *Application Development Guide: Building an Running Applications*, SC09-4825
- ▶ *Application Development Guide: Programming Client Applications*, SC09-4826

10.7 Integrating DB2 UDB in Integration Development Environment (IDE)

DB2 UDB comes with several integration packages that enable you to extend your existing development environment. In this section, we describe the add-ins for Microsoft Visual Studio .NET and IBM WebSphere Studio.

10.7.1 Microsoft .NET add-in (Visual Studio)

The DB2 UDB V8.2 Add-In for Visual Studio .NET provides two key application development capabilities, namely, developing server-side schema and logic and developing client-side or middle tier ADO.NET application components using rapid application development (RAD) features.

The DB2 .NET Add-In is part of the DB2 Application Development Client for Windows, Version 8.1.2 or later. If you installed Visual Studio .NET before you installed DB2 UDB, the DB2 Development Add-In is registered automatically for you. If you installed Visual Studio .NET after you installed DB2, or if you modified your installation, you must manually register the Add-Ins. To register the Add-In for Visual Studio .NET:

1. Exit Visual Studio .NET.
2. From any command window, run **db2vsrgx.bat**.
3. Start Visual Studio .NET.

Some of the main integration items in the .NET platform for the Visual Studio .NET IDE include:

- ▶ **DB2 .NET Managed Provider**
The native DB2 .NET managed provider enables application developers to build and run applications using the standard ADO.NET programming model. This native provider provides an improved level of support over the ODBC and OLE DB generic managed providers for DB2 UDB.
- ▶ **Solution Explorer**
The DB2 UDB database project templates allows for script-based

development as part of the standard set of Visual Studio solutions supporting multi-configuration, source control management, project and project item build order, as well as editing and debugging of SQL scripts using the built-in editor and debugger.

► **Server Explorer**

The IBM Explorer provides the same look and feel as the Visual Studio server explorer enabling catalog access and rapid .NET application development using drag and drop functionality.

► **SQL Wizards**

The rich set of easy-to-use wizards allows for the creation of new server side tables, views, indexes, triggers, procedures, and functions from either the Solution Explorer or the Server Explorer.

► **SQL Editor**

The built-in Visual Studio text editor has been extended to provide native support for DB2 SQL scripts having *intellisense* and syntax colorization as well as advanced script options.

► **SQL Debugger**

The built-in Visual Studio debugger has been extended to provide debugging support for DB2 SQL routines allowing for source level debugging of cross-platform SQL stored procedures as part of the DB2 database project.

► **Dynamic Help**

The Visual Studio context sensitive help has been extended to provide online help for the DB2 .NET managed provider and the Visual Studio AD tools.

► **DB2 Tools Toolbar**

The various DB2 development and administration centers may be launched directly from the Visual Studio IDE using the DB2 Tools toolbar.

Tutorials

There are several tutorial available about DB2 UDB application development with Visual Studio .NET:

- The tutorial *ADO.NET data adapters using DB2 UDB V8.2 Procedures* walks you through the steps required to build an ADO.NET application that uses SQL stored procedures to SELECT, INSERT, UPDATE, and DELETE rows from database tables and views.

<http://www.ibm.com/developerworks/edu/dm-dw-dm-0408a1azzawe-i.html>

- The tutorial *DB2 Data Bound ASP.NET Form using VS.NET* demonstrates how to use the DB2 Development Add-In for Visual Studio .NET to build a rich ASP.NET WebForm in C# using design-time and run-time data binding.

<http://www.ibm.com/developerworks/edu/dm-dw-dm-0309a1azzawe-i.html>

- ▶ The tutorial *Binding DB2 Stored Procedures to Visual Basic WinForms* demonstrates the use of the Visual Basic WinForms project template and the IBM DB2 Development Add-In to create a client application that accesses stored procedures and tables residing in a DB2 UDB for z/OS database. This tutorial also applies to DB2 on distributed platforms.

<http://www.ibm.com/developerworks/edu/dm-dw-dm-0306alazzawe1-i.html>

- ▶ The tutorial *Developing a VB.NET Federated Application for Microsoft Access* demonstrates the use of the ODBC wrapper technology available through WebSphere Information Integrator and the DB2 Development Add-In to create a client application that accesses tables residing in a Microsoft Access database.

<http://www.ibm.com/developerworks/edu/dm-dw-dm-0307alazzawe-i.html>

- ▶ The tutorial *DB2 z/OS SQL procedures in VS.NET* demonstrates the use of the advanced features of the IBM DB2 Development Add-In for Visual Studio .NET to create and deploy SQL stored procedures from a development environment to a production DB2 for z/OS database environment.

<http://www.ibm.com/developerworks/edu/dm-dw-dm-0403alazzawe-i.html>

Additional information

For more information and the latest technical articles about DB2 UDB application development in the Visual Studio .NET environment, refer to the Visual Studio .NET zone on the DB2 developerWorks Web site at:

<http://www.ibm.com/developerWorks/db2/zones/vstudio/>

10.7.2 IBM WebSphere Studio Application Developer

Using Development Center, you can write stored procedures in Java and SQL. Development Center is also available as a WebSphere Studio Application Developer plug-in (see Figure 10-8).

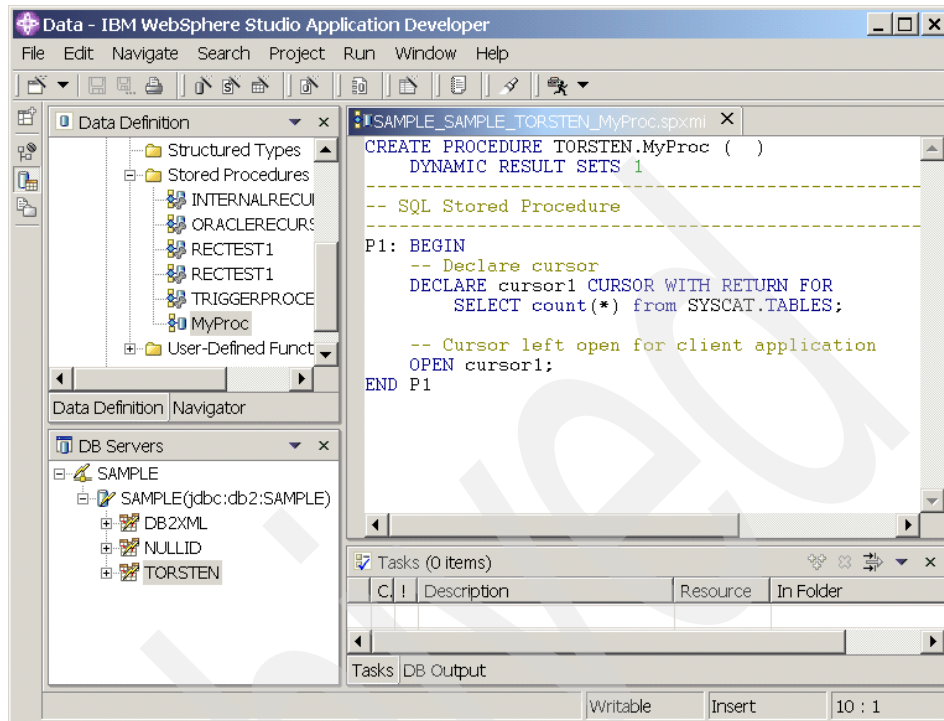


Figure 10-8 Development Center integrated into WebSphere

10.8 Approaching packaged application migration

For packaged applications such as Siebel or SAP, vendors deliver their application and database based on different databases, including DB2 UDB. The migration process in this case comprises the following steps:

- ▶ Check software and hardware availability and compatibility
- ▶ Educate developers and administrators about DB2 UDB
- ▶ Analyze customized changes required in application and database
- ▶ Set up the target environment
- ▶ Implement customized changes
- ▶ Test data migration and customized changes
- ▶ Roll-out to production

To maintain the support from the vendor, you must meet their prescribed migration plan and process. In this section, we highlight the conversion approach for SAP and Siebel environments.

10.8.1 SAP

SAP applications are designed in a layered structure which allows customers to use business applications independent from their database platform. Within this structure, SAP's application servers contain a software layer, called "Database Support Layer" (DBSL). Above this, no database dependencies exist. Simply exchanging the DBSL and migrating the database to DB2 UDB results in a stable and well running system. A learning curve to master any newly introduced technology should also be taken into account after the migration.

When migrating SAP databases, you must consider SAP's specific tables and data types, as well as SAP proprietary data compression methods. Therefore, databases can only be migrated using tools provided by SAP. In SAP terminology, a migration from Oracle, Informix®, or SQL Server to DB2 UDB is called a "heterogeneous system copy" or "migration".

Solutions from SAP are delivered with a broad range of service offerings, produced by the SAP Active Global Support Organization. One offering in the context of heterogeneous system copy to DB2 UDB is SAP's OSDB Migration Check.

Because of the complexity of the SAP ecosystem, we can only provide a brief overview of the required migration steps.

Details about SAP Migration Planning can be found at

<http://service.sap.com/osdbmigration>

Documentation to perform SAP migrations can be found at

<http://service.sap.com/instguides>

Migration requirements

For production database migration, SAP requires customers to order the SAP OSDB Migration Check for the system to be supported. To receive support during production system migration, this service is mandatory. For Test and Development Systems, as well as evaluation of Production System Migration, this service is not needed. The SAP OSDB Migration Check is fee-based and provides three specific service sessions which are designed to complement the migration project:

- ▶ Remote Project Audit session
- ▶ Analysis session
- ▶ Verification session

All of these sessions are delivered by certified consultants through remote connections, which must be in place for SAP customers anyway. The delivery of

the individual sessions takes place at key phases in the migration project. The Remote Project Audit session lasts no more than half a day, while the Analysis and Verification sessions have a normal duration of one day.

Another core requirement of any SAP migration is that a consultant holding SAP's special certification for OS/DB migration is on site and available as the primary contact for SAP's support. Certification can be obtained through SAP's training workshop TADM70.

To order the new CD media, customers must change their SAP software license to DB2 UDB to reflect the new target database platform. With this change, customers are able to order the new installation kit for their SAP software on DB2 UDB, and a special OS/DB Migration Kit.

To perform a migration, a migration key is needed. This key is provided online via SAP's support Web site. The key is mandatory for the data export step and data import step using SAP's tools.

The migration implementation

The central tool in an SAP migration is the **R3load** tool. It performs the export of data to disk and the import from disk into the new database.

The following steps are necessary on the source system:

1. Migration Preparation: "Clean Up" the SAP database (deletion of so-called QCM-Tables)
2. Generation of R3load-Control Files: Creation of specific Control Files to perform a migration.
3. Generation of Templates for Database Sizes: Calculation of target database size
4. Generation of R3load-Command Files: The database is exported in parallel. Each R3load run needs a control file.
5. Export the database.

The following steps must be completed on the target server machine:

1. Create the new SAP Instance
2. Obtain the migration key
3. Create the DB2 UDB Instance and update Instance parameters
4. Create the DB2 UDB database and update Database parameters
5. Import the data into the database
6. Post-migration activities

The migration project plan

To successfully perform a migration, you need to follow a well-defined process using the steps shown in Figure 10-9. Therefore, SAP demands a detailed project plan for each migration project. This plan is best designed with the help of an SAP Migration Consultant. The plan is reviewed during the Project Audit Session to ensure a successful migration.

A database migration consists many interdependent tasks. Therefore, we recommend that you begin planning three to four months in advance.

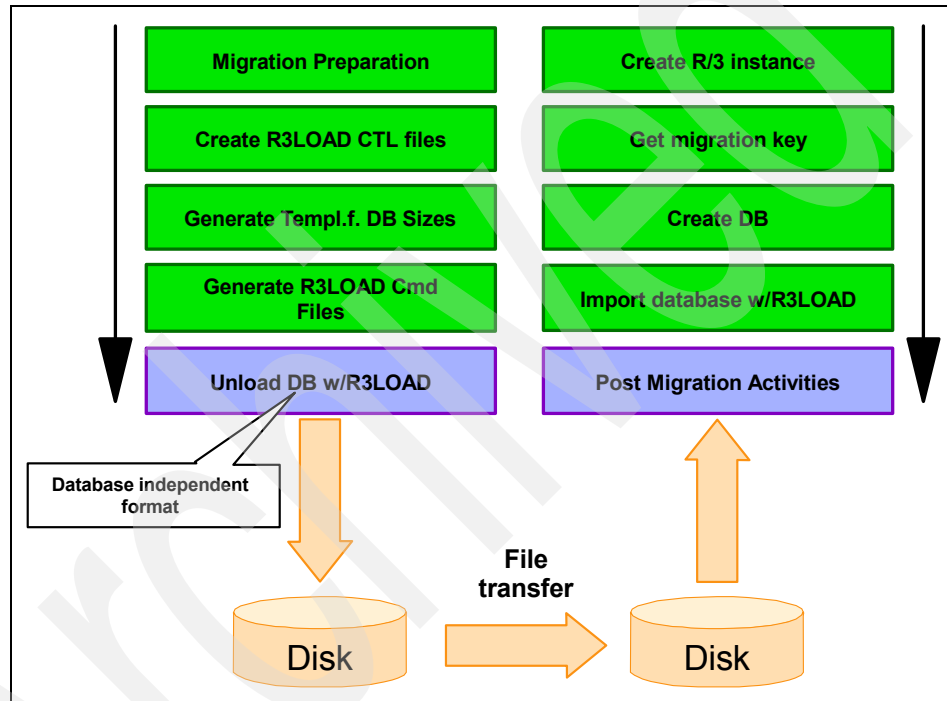


Figure 10-9 SAP migration project plan

Migration test and check

Before the final production database migration, customers must perform at least one migration test run, including thorough functional tests. This helps understand the timing required for the migration. It also helps gain confidence in the new environment, by reviewing the integration of all systems components and external interfaces in a development environment.

Since the final result of the migration is used by SAP's end-users, acceptance from end-users should be ensured by using migration sign-off criteria defined by them.

Final migration and verification

The final migration must be scheduled during weekends as SAP system downtime is needed to unload data in the source database and load it into DB2 UDB. For large database migrations, longer periods of time may be required.

10.8.2 Siebel

Siebel solutions are implemented in a layered architecture with database independence. Thus, the Siebel application layer and business logic layer are not affected by a database migration. Figure 10-10 shows the typical migration process of a Siebel System.

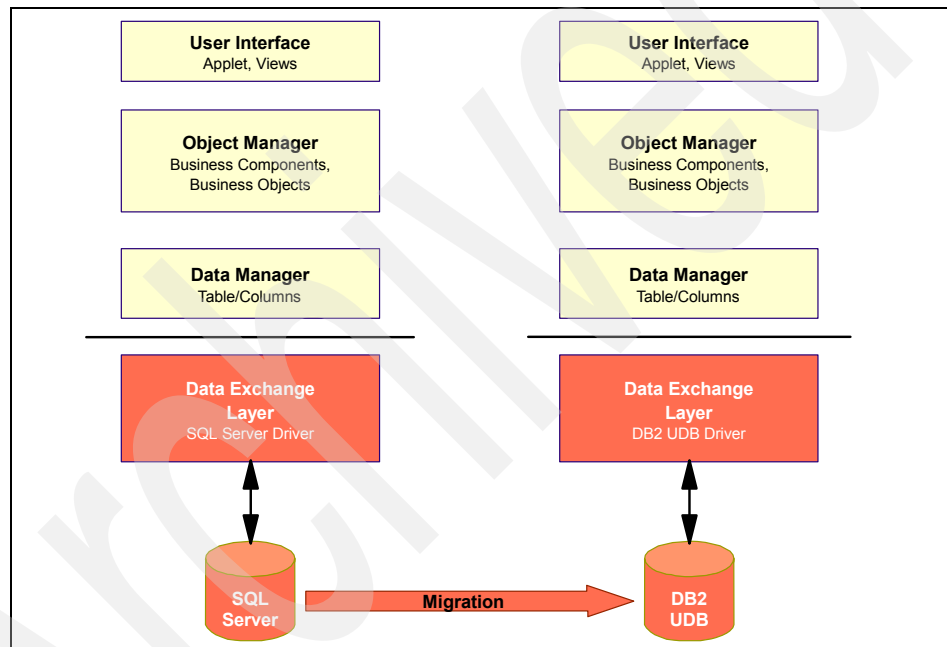


Figure 10-10 Siebel migration process

We recommend that you involve a Siebel or certified partner in the migration planning for a successful migration.

For additional information about Siebel migrations, refer to the IBM Redbook *Migrating Siebel Databases from DB2/Oracle for NT to DB2 for OS/390*, SG24-6236.

Planning tasks

Before performing a migration, there are several planning tasks that need to be completed. It is important to migrate a version of Siebel on a source system to the same level of Siebel on a target system. Each version of Siebel may have different database tables and columns.

If you need to migrate to a later version of Siebel, you must upgrade the existing implementation to that level first.

Ensure that the existing installation has all the required FixPacks applied for the version of Siebel you want to install. These can be found in Siebel's *Product Availability Matrix*.

Estimate the size of the target database server and the size needed for a data migration.

Identify personnel with database and Siebel customization skills. Analyze the amount of customization and plan work-arounds, especially where SQL Server proprietary features are used.

Once these tasks have been carried out, proceed to plan the data migration in more detail.

Setting up the target system

To set up the environment, use the recommendations from Siebel.

Data migration

Siebel delivers two utilities, **Dataexp** and **Dataimp**, which are often used by Siebel services personnel to move data. The **Dataexp** utility exports data from tables into a data file, which is independent of the database implementation. **Dataimp** takes this file and loads the contents into a target database.

Having a database independent data file means that **Dataimp** can be used to insert the data into a different target. This is exactly how Siebel handles distributing the seed and base repository data for different operating systems.

The **Dataexp** and **Dataimp** utilities are the ideal way of migrating Siebel data with minimal coding and administration. Running **Dataimp** for very large volumes is not advisable due to the slower INSERT process. Alternatively, you can use the DB2 UDB LOAD utility, which may improve data loading times.

Migrating additional programs

In a typical Siebel installation, there are additional programs such as batch jobs, which are not provided by Siebel. These programs should be migrated using the process outlined in this book.

Testing

Once the data migration process is completed, run various tests against the old system and the new system ensure that they operate the same way and that the same data is returned for test queries.

Additional information

Additional information about Siebel application and database migration is available in:

- ▶ Redbooks:
 - *Siebel 2000 Database Implementation on OS/390 Using NT Siebel Servers*, SG24-5953
 - *Migrating Siebel Database from DB2 NT to DB2 S/390®*, SG24-6236
- ▶ Web site:
<http://www.ibm.com/software/data/partners/aelpartners/siebel/>

Performing administrative tasks in DB2 UDB

Learning database administration in DB2 UDB is another critical task as you transition from SQL Server. You should familiarize yourself with how to administer instances, databases, table spaces, and buffer pools. You can utilize DB2 UDB's advanced database management features to configure backup and recovery, high availability, and utilities operations in your environment and provide high performance and an available database system.

This chapter discusses basic and intermediate database administration tasks that include:

- ▶ Working with instances
- ▶ Working with databases
- ▶ Managing data storage (table spaces and containers)
- ▶ Working with memory usage (buffer pools)
- ▶ Automating administrative tasks with Task Center
- ▶ Backing up, recovering, and administering logs
- ▶ Managing high availability
- ▶ Performing REORG and RUNSTATS

11.1 Working with instances

Working with DB2 UDB instances is equivalent to working with separate installations of SQL Server. In SQL Server, if you need two instances on the same machine, you must install SQL Server twice. If there are more than one instance on one or more machines, you must set up a linked server definition in SQL Server to allow it to execute commands across them.

DB2 UDB can have multiple instances defined on a machine with just one DB2 UDB installation. Each instance of DB2 UDB maintains its own databases. You can create or drop instances using only the command line. If you run **db2setup** to install DB2 UDB, an instance is created by default. The default instance names are:

- ▶ **db2inst1** on Linux and UNIX
- ▶ **DB2** on Windows

If you want to add an instance that resides on another DB2 UDB server, then you need register that instance in similar fashion as a linked server definition in SQL Server.

11.1.1 Create, drop, and list instances

Creating and dropping instances requires local administrator privileges on Windows and root access on Linux and UNIX. For every instance, you need to provide a user ID to own the processes to be started. On Linux and UNIX, a special fenced user ID must be created. This user ID is used to own processes related to external stored procedures and functions. By default, the user ID **db2fenc1** is created during installation.

The following commands can be used for creating, dropping, and listing instances:

Windows commands:

```
x:\DB2ICRT <instance name>  
x:\DB2IDROP <instance name>  
x:\DB2ILIST
```

Linux and UNIX commands:

```
<install_path>/instance/db2icrt -u <fenced userid> <instance name>  
<install_path>/instance/db2idrop <instance name>  
<install_path>/instance/db2ilist
```

For more information about the syntax usage of the **DB2ICRT**, **DB2IDROP**, and **DB2ILIST** command, refer to DB2 UDB documentation *Command Reference*, SC09-4828.

11.1.2 Attaching / switching instances

An administrator that wants to perform instance-level commands must attach to the instance first. The instance can be remote. Use the **ATTACH** command:

```
db2 ATTACH TO nodename
```

Where *nodename* is the name of the instance.

To attach to an instance using Control Center:

1. Expand the object tree until you see the *Instances* folder.
2. Right-click the instance you want to attach to.
3. In the Attach-DB2 dialog, enter the appropriate ID and password, and click **OK**.

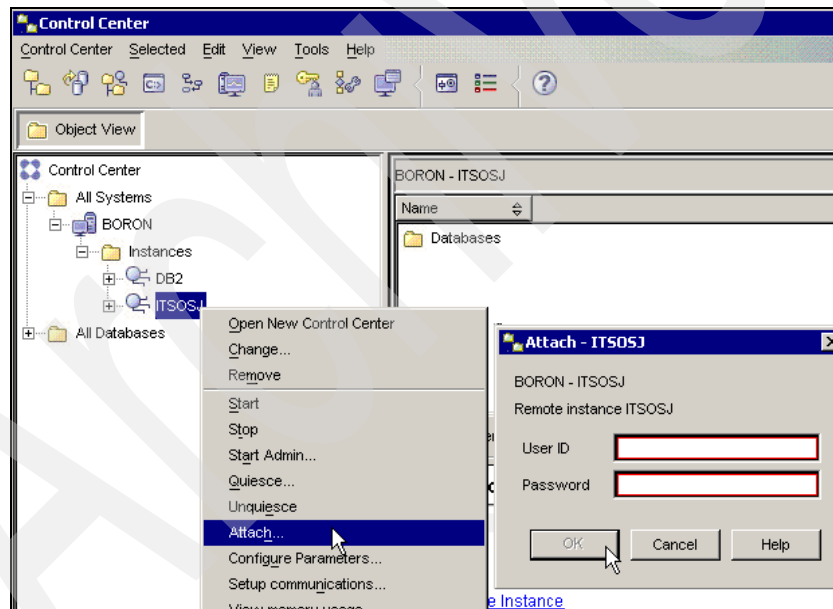


Figure 11-1 Attaching to an instance using Control Center

In CLP or Command Editor, instance-level commands are executed against the current instance, specified by the *DB2INSTANCE* environment variable. This instance, to which you are attaching, can be the default instance, another instance on the same machine, or an instance on a remote machine. Only one instance attachment can be in effect at a time. After performing maintenance

activities for the instance, you can then detach from it by issuing the **DETACH** command

Alternatively, on the Linux and UNIX platforms, simply log into the system using the instance owner's user ID, or source the instance's *db2profile* file, to switch instances. Logging into the instance owner or sourcing the *db2profile* file, sets up all the paths required to access the instance. On Windows, simply set the environment variable *DB2INSTANCE* to the desired instance name, or use the `SET DB2INSTANCE` command.

11.1.3 Start, stop, and quiesce instances

In SQL Server, you can start, stop, and pause the instance in the Windows background process. DB UDB's **quiesce** command is similar to SQL Server's **pause**, but differs in some respects.

In DB2 UDB, the **DB2START** command starts the current database manager instance background processes on a single database partition or on all the database partitions defined in a partitioned database environment. It needs to be started at the server before you can connect to a database, precompile an application, or bind a package to a database.

The **DB2STOP** command terminates the DB2 instance. However, the instance will not stop unless all active processes are terminated. If you try to issue a **DB2STOP** command under this condition, you receive the following error:

```
02/02/2004 18:02:28 0 0 SQL1025N The database manager was not stopped
because databases are still active.
```

You can force off or kill all active processes on the instance by using the **DB2STOP FORCE** command at the command line.

The **QUIESCE** command forces all users off the instance, and restricts new users from connecting to it, to enable system users to perform administrative and maintenance activities. This is similar to pausing a database in SQL Server, but in that it does force all users off.

To start, stop, and quiesce an instance using Control Center, refer to Figure 11-1 on page 265, where you can see the corresponding actions after right-clicking an instance in the object view.

11.1.4 Configure instances

In SQL Server, instance properties can be accessed from the Enterprise Manager by following these steps:

1. Expand the object tree until you see the *Instances* folder.

2. Right-click the selected instance whose parameters you want to modify.
3. Select **Configure Parameters**.

The equivalent commands that can be executed in CLP is:

```
GET DBM CFG (To change view all parameters and their values)
UPDATE DBM CFG USING <parameter> <value> (To change a parameter value)
```

11.2 Working with databases

In SQL Server, an instance can contain several databases. Databases in this instance can interact with each other. The MASTER and TEMPDB are databases that are shared by other databases and objects in the instance. In DB2 UDB databases are completely independent units. They do not share metadata, temporary data, or user data. All DB2 UDB database entries are stored in the database directory at the instance level.

Despite these differences, basic administration of both database management systems is quite similar. With the GUI tools available on both SQL Server and DB2 UDB, Enterprise Manager and Control Center, administrative tasks such as creating, dropping, and listing databases are fairly intuitive. The following subtopics focus on frequently used administrative tasks that are sufficient to take you through the migration process.

11.2.1 Create, drop, and list databases

Creating a simple database in SQL Server and DB2 UDB only requires you to specify a new database name. If you want to tailor the database to your requirements, SQL Server allows you to specify data files (containers), filegroups (table spaces), and transaction logs when you launch the create database tool.

DB2 UDB provides you with three wizards to help you create a database that includes specific container and table space specifications.

To create a database from the Control Center (Figure 11-2):

- ▶ Expand the object tree until you find the **All Databases** folder.
- ▶ Right-click **All Databases** folder, and select **Create Database** → **Standard** or With **Automatic Maintenance** from the next level menu.
- ▶ Proceed through with the wizard and override the database default properties if desired.

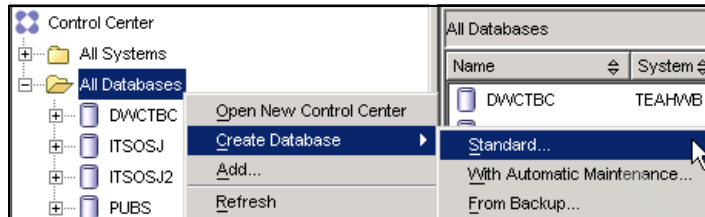


Figure 11-2 Creating database in Control Center

The following list summarizes the functionality of the three create database wizards and their descriptions:

- ▶ **Create Database Wizard (Standard):**
This selection helps you create and customize a new database.
- ▶ **Create Database With Automatic Maintenance:**
This selection helps you create a new database, configure it for optimal performance, turn on automatic maintenance, and configure notification by e-mail or pager if the database needs attention.
- ▶ **Create Database From Backup:**
This selection helps you create a new database from a backup image.

In the last step, the wizard provides a *Show Command* button (see Figure 11-3) UDB which can be used to display the DB2 command.

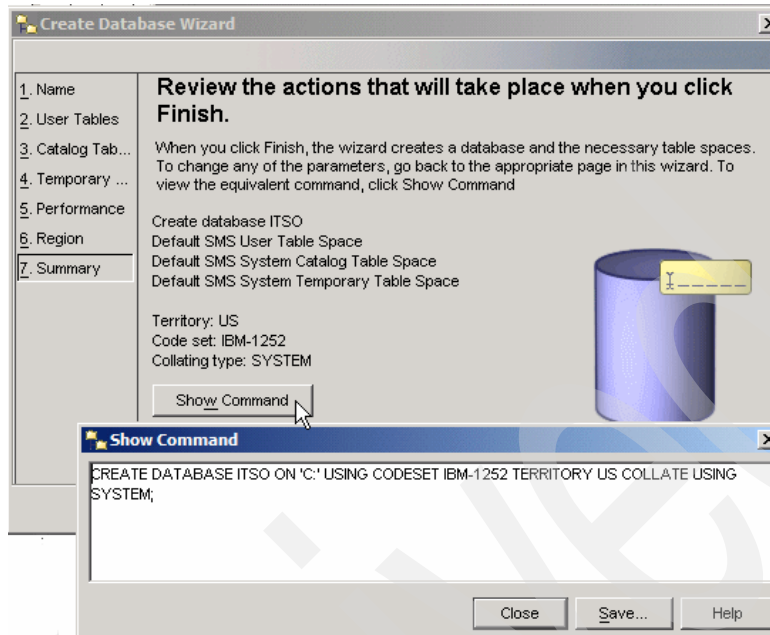


Figure 11-3 Show Command from the standard create database wizard

Transaction logging is configured once you have successfully created a database. To configure it, right-click the newly created database, and select **Configure Database Logging** (Figure 11-4) to launch a wizard.

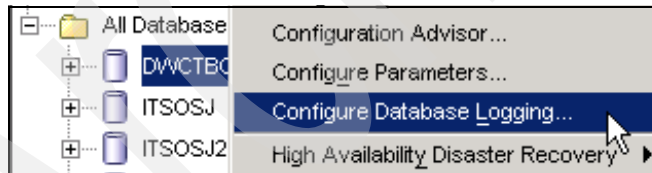


Figure 11-4 Configure database logging after creation of database

Refer to 11.3.3, “Transaction logging” on page 289 for more information about administering the transaction logs.

Dropping a database is straightforward. You need to first ensure that all connections to the database are closed and that the database is not active.

To drop a database from the Control Center, you must:

1. Expand the object tree until you see the *Databases* folder

2. Right-click the database you want to drop, and select **Drop** from the pop-up menu.
3. Click the Confirmation box, and click **OK**.

At CLP or Command Editor, use the following simplified commands:

CREATE DATABASE <database-alias> to create a database
DROP DATABASE <database-alias> to drop a database
LIST DATABASE DIRECTORY to list cataloged databases

11.2.2 Activate and terminate databases

By default, as soon as you have created a database in SQL Server, your database is brought online. When it is online (active), memory resources are obtained and connections are allowed. To bring it offline, you need to take it offline explicitly.

A database in DB2 UDB is offline (inactive) when it is first created and no connections are established. You can either explicitly activate a database using the **ACTIVATE** command, or wait until the first connection is made to a database. Database global memory is allocated when a database is active. This allocation of memory can be quite large because it includes the database's primary cache, the buffer pools.

If you explicitly activate a database, you also have to explicitly **DEACTIVATE** the database to release the memory. Issuing a **DB2STOP** command, which stops the instance, will deactivate all databases under that instance, as well as free up memory held by those databases.

While activating a database is not required, it is recommended for production databases. The command syntax to explicitly activate and deactivate a databases is:

ACTIVATE DATABASE database-alias [**USER** username [**USING** password]]
DEACTIVATE DATABASE database-alias [**USER** username [**USING** password]]

11.2.3 Connect, disconnect, and quiesce databases

To connect, disconnect, or quiesce database from the Control Center, expand the object tree until you find the database that you want to use to perform any of these operations. Right-click that database and select the **Connect**, **Disconnect**, or **Quiesce** menu option (see Figure 11-5).

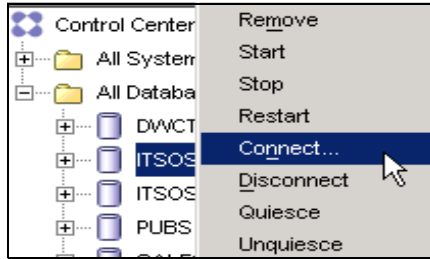


Figure 11-5 Connect, disconnect, quiesce, and unquiesce menu options

To connect to a database using Command Editor or CLP, use the **CONNECT TO** command. To connect using an explicit user ID and password, use the **USER** and **USING** options.

To disconnect from a database, issue the **CONNECT RESET** or the **TERMINATE** command. **TERMINATE** also results in the termination of the command line processor's back-end process.

Use the **QUIESCE** command to immediately force all users off a database and only allow administrative users to connect.

11.2.4 Configure databases

To configure a database in SQL Server via Enterprise Manager, expand the object tree until you see the desired database. Right-click the database you want to configure, and select **Properties** from the menu.

To configure a database in DB2 UDB using Control Center, expand the object tree until you see the desired database. Right-click that database, and select **Configure Parameters** (Figure 11-6) from the menu.

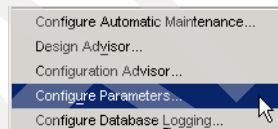


Figure 11-6 Configuring database parameters in Control Center

If you need to configure database parameters from the CLP, or Command Editor, use the following commands:

To display current values of the database configuration parameters:

```
GET DB CFG FOR <dbname>
```

To change a value of a database configuration parameter:

```
UPDATE DB CFG FOR <dbname> USING <parameter> <value>
```

As you can see from Figure 11-6, there are a few other options available to configure database parameters:

- ▶ **Configuration Advisor:**
Use this advisor to produce recommended configuration parameter values.
- ▶ **Configure Automatic Maintenance:**
Use this wizard to create a strategy for automating database maintenance activities.
- ▶ **Configure Database Logging:**
Refer to 11.3.3, “Transaction logging” on page 289.

11.2.5 List and force off applications

You can find out who, and which application is currently connected to a database by using the **LIST APPLICATION** command. You can also force off users prior to any required maintenance work. Applications that crashed while running, but still holding on to database resources can also be forced off.

To force an application off a database from the Control Center, do the following:

- ▶ Expand the object tree until you see the *Databases* folder
- ▶ Right-click the database you want to drop, and select **Application** from the pop-up menu.
- ▶ Select the application that you want to force off the database.
- ▶ Click **Force**.

At CLP or Command Editor, issue the following command (simplified for illustration):

LIST APPLICATION to list all the applications connected to current database.

FORCE APPLICATIONS {ALL | (app-handle)} to force off applications from a database.

11.3 Managing database storage

In this section we look at two areas of storage used in DB2 UDB: database disk management and log file management. We discuss how to set up and manage storage the of data, as well as log storage.

11.3.1 Table spaces and containers

User data in a DB2 UDB database is managed using table spaces and containers. Table spaces can be of two types: System Managed Space (SMS) or Database Managed Space (DMS). An SMS table space is created using directory containers. The DMS table space is created using either file containers or device containers such as a raw device. A container can be a directory name, a device name, or a file name.

A table space is a storage structure containing tables, indexes, large objects, and long data. They enable you to assign the location of database and table data directly into containers. This enables flexible configuration, which can also result in improved performance.

When a database is created, three default table spaces are created by default, as depicted in Figure 11-7. The DBA can customized how these table spaces are created. The following types of table spaces can be created:

- ▶ **Catalog table space:** One catalog table space that contains all of the system catalog tables for the database. This table space is called SYSCATSPACE, and it cannot be dropped.
- ▶ **User table space:** One or more user table spaces that contain all user-defined tables. By default, one table space, USERSPACE1, is created.
- ▶ **User temporary table space:** One or more temporary table spaces that store globally declared temporary tables. This type of table space must be manually created before any global declared temporary tables can be created
- ▶ **System temporary table space:** Used by the database manager to perform operations such as sorts or joins. A database must have at least one system temporary table space; by default, one system temporary table space called TEMPSPACE1 is created at database creation time.

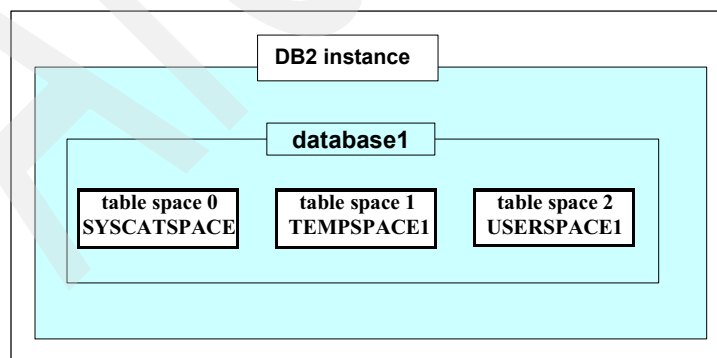


Figure 11-7 Default Table Spaces

SMS table spaces

System Managed Space (SMS) table spaces store data in operating system files. The data in the table spaces is striped by extent across all the containers in the system. An *extent* is a group of consecutive pages defined for each table space. Each table in a table space is given its own file name that is used by all containers. The file extension denotes the type of the data stored in the file. To keep space evenly used across containers in the table space, the starting extents for tables are placed in a round-robin fashion across all containers. Such distribution of extents is particularly important if the database contains many small tables.

Note: SMS table spaces can take advantage of operating system file system prefetching and caching.

In an SMS table space, space for tables is allocated dynamically. This expansion proceeds a single page at a time by default. However, in certain work loads such as ones with bulk inserts, you can improve performance with the **db2empfa** tool which instructs DB2 UDB to expand the table space in groups of pages or extents. The **db2empfa** tool is located in the `\\SQLLIB\\bin` directory. When you run the **db2empfa** tool, the `MULTIPAGE_ALLOC` database configuration parameter must be set to YES.

Important: When you create an SMS table space, you must specify the number of containers that you want because you cannot add or delete containers after an SMS table space is created. However, you can always increase the size of the underlying file system using operating system utilities. You can only add containers to an SMS table space when you add a new partition in a partitioned database environment, or perform a redirected restore from a database backup.

DMS table spaces

With Database Managed Space (DMS) table spaces, the database manager controls the storage space. A list of devices or files is selected to belong to a table space when the DMS table space is defined. The space on those devices or files is managed by the DB2 database manager. As with SMS table spaces and containers, DMS table spaces and the database manager use striping (by extent) to ensure an even distribution of data across all containers.

DMS table spaces differ from SMS table spaces in that the entire space is allocated when the table space is created versus not allocated when needed.

Also, the placement of data can differ between the two types of table spaces. For example, consider the need for efficient table scans: It is important that the pages

in an extent are physically contiguous. With SMS, the file system of the operating system decides where each logical file page is physically placed. The pages might, or might not, be allocated contiguously, depending on the level of other activity on the file system and the algorithm used to determine placement. With DMS, however, the database manager can ensure the pages are physically contiguous, because it interfaces with the disk directly.

DMS file containers do not take advantage of operating system file system prefetching and caching.

Recommendations

- ▶ Unlike SMS table spaces, the containers that make up a DMS table space do not need to be of equal size. However, we recommend using equal container sizes.
- ▶ When working with DMS table spaces, you should consider associating each container to a different physical disk. This allows for larger table space capacity and the ability to take advantage of parallel I/O operations.

Although the default table spaces are created upon database creation, you can specify the types of containers to be used for these table spaces. Table 11-1 shows the configuration recommendations each default table space.

Table 11-1 Recommendations for default table spaces

Table space	Usage	Preferred type
SYSCATSPACE	Used to store the system catalog tables for the database. These tables contain data about the database structures: where they are and how they are used.	SMS.
TEMPSPACE1	Used by the system temporary space is needed (e.g., sorts or joins).	SMS.
USERSPACE1	Used to store user data tables and indexes.	Must be DMS if you are partitioning data separate from the indexes.

Comparison of SMS and DMS table spaces

There are a number of trade-offs to consider when determining which type of table space you should use to store your data.

- ▶ Advantages of an SMS table space:
 - Space is not allocated by the system until it is required.
 - Easier to maintain.

- Takes advantage of operating system file caching
- ▶ Advantages of a DMS table space:
 - The size of a table space can be increased by adding or extending containers, using the `ALTER TABLESPACE` statement. Existing data can be automatically rebalanced across the new set of containers to retain optimal I/O efficiency.
 - A table can be split across multiple table spaces, based on the type of data being stored:
 - Long field and LOB data
 - Indexes
 - Regular table data
 - Improved performance, depending on the workload.

Extent size

The *extent size* of table space determines the number of pages of table data that will be written to a container before data will be written to the next container. When selecting an extent size, you should consider:

- ▶ The size and type of tables in the table space:

Space in DMS table spaces is allocated to a table one extent at a time. As the table is populated and an extent becomes full, a new extent is allocated. Space in SMS table spaces is allocated also one extent at a time, but only if multi page file allocation is enabled.
- ▶ The type of access to the tables:

If access to the tables includes many queries or transactions that process large quantities of data, prefetching data from the tables might provide significant performance benefits.
- ▶ The minimum number of extents required:

If there is not enough space in the containers for five extents, the table space is not created.

Creating and altering table spaces and containers

The syntax and process to create table spaces is similar, whether using DMS or SMS.

The basic parameters needed to explicitly create a table space are the table space name, how it is managed, and the container or containers. Optionally, you can specify the use of the table space, such as regular or user-temporary. You

can also optionally specify page size, extent size, prefetch size, and buffer pool name. In DB2 UDB, sizes can be entered in four different units (all Integers):

- ▶ Pages
- ▶ Kilobytes (K)
- ▶ Megabytes (M)
- ▶ Gigabytes (G)

Important: An SMS table space is full as soon as any one of its containers is full. Therefore, it is important to have the same amount of space available to each container.

In the following example we show how to create a DMS table space using a file container.

To create a new table space, perform the following steps.

1. Open Control Center and expand the database you are creating a new table space for. Right-click **Table Spaces** folder and select **Create....** The Create Table Space Wizard dialog box opens.
2. Enter the table space name. In Figure 11-8, we use the name `tablespace1`. You can optionally enter a comment. Click **Next**.

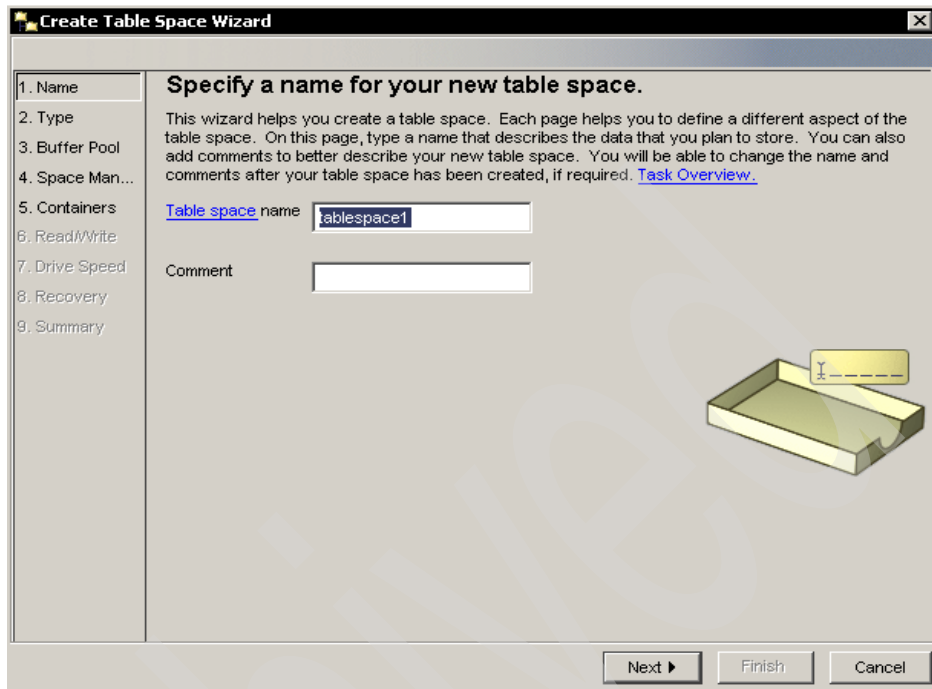


Figure 11-8 Create Table Space Wizard

3. Specify the type of table space to create. As shown in Figure 11-9, we have chosen **Regular**, because we will be storing table data in this table space.

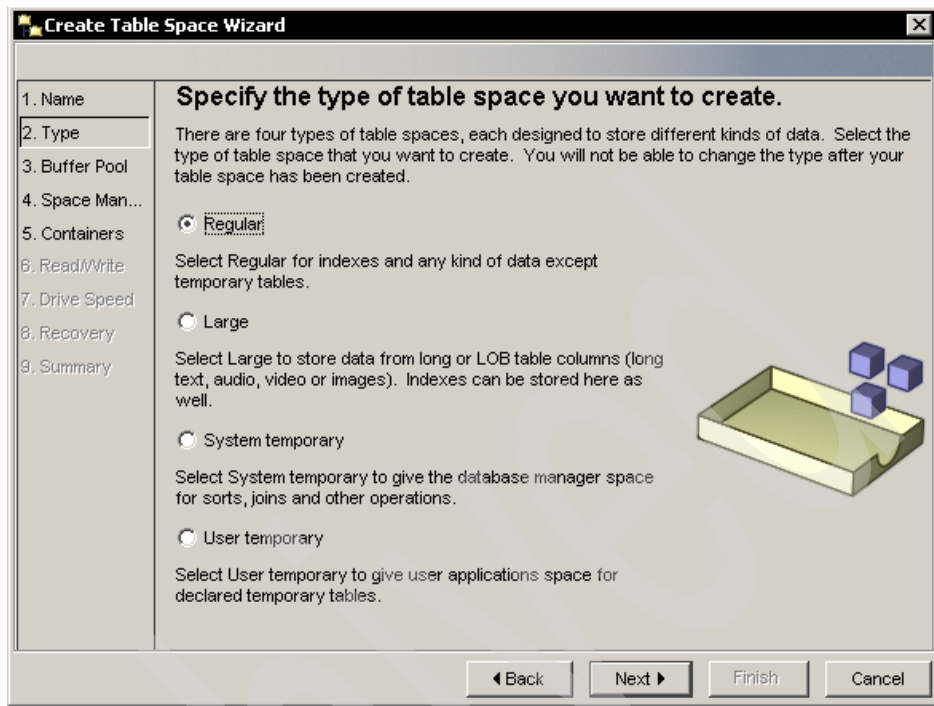


Figure 11-9 Create Table Space Wizard: Type of table space

4. Select the buffer pool the table space will use. In our example we have used a buffer pool called `buffer_pool` (Figure 11-10). Buffer pools are covered in 11.4, “Working with buffer pools” on page 294

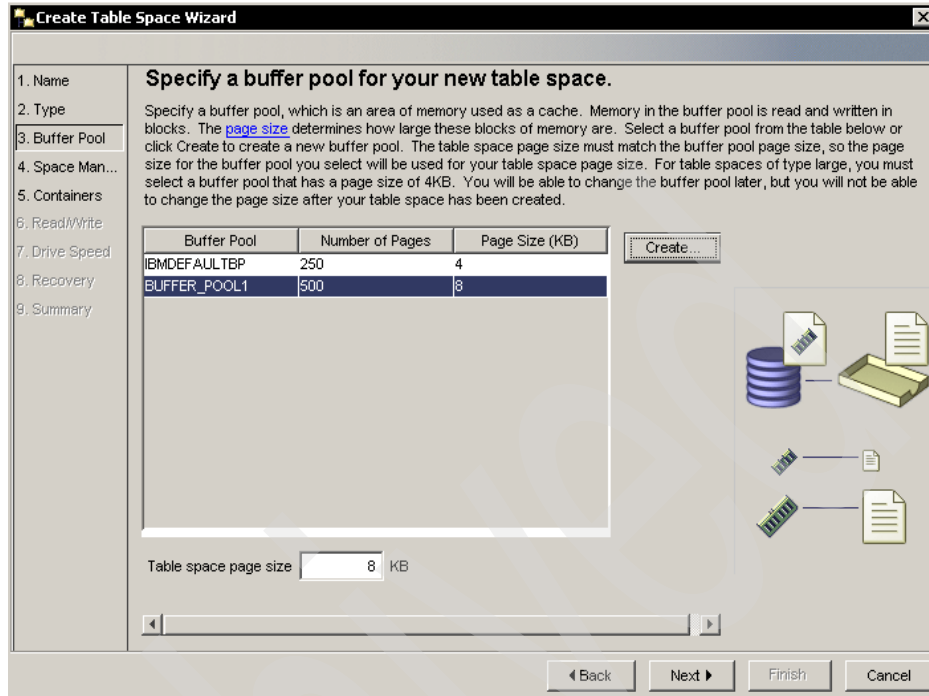


Figure 11-10 Create Table Space Wizard: select buffer pool

5. Select the type of table space to create. We create a DMS table space. Click **Next**.
6. Specify the container(s) to be used. In Figure 11-11, we use a file as the container. Click the **Add** button, browse to the location where the container will reside, give the container a name and specify the size of the container.

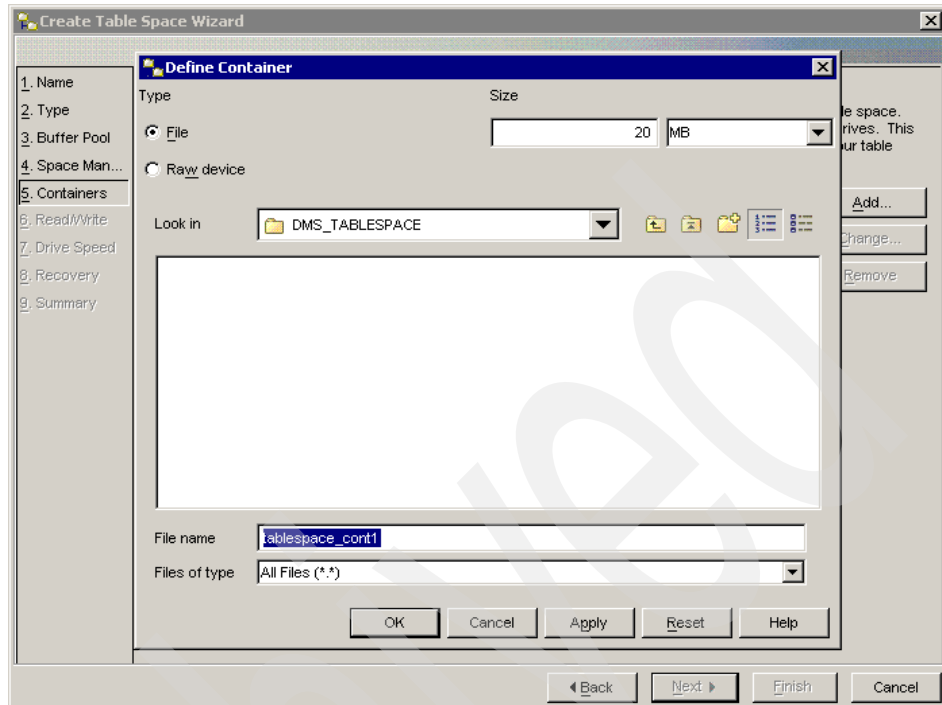


Figure 11-11 Create table space wizard: container options

7. Next three screens in the wizard allow you to set setting of the extent size, prefetch size, disk speed parameters and dropped table recovery. Dropped table recovery, as the name suggests, allows you to recover a table that has been dropped. This option can only be enabled on Regular table spaces.

Figure 11-12 shows the wizard summary screen with the options used for creating the table space. Selecting the **Show SQL** button shows the syntax of the command for creating the table space (shown in Example 11-1).

Example 11-1 Create table space command

```
CREATE REGULAR TABLESPACE TABLESPACE1 PAGESIZE 8K MANAGED BY DATABASE USING (
FILE 'C:\DMS_TABLESPACE\tablespace1_cont1' 6400 ) EXTENTSIZE 32 OVERHEAD 10.5
PREFETCHSIZE 32 TRANSFERRATE 0.14 BUFFERPOOL BUFFER_POOL1 DROPPED TABLE
RECOVERY OFF;
```

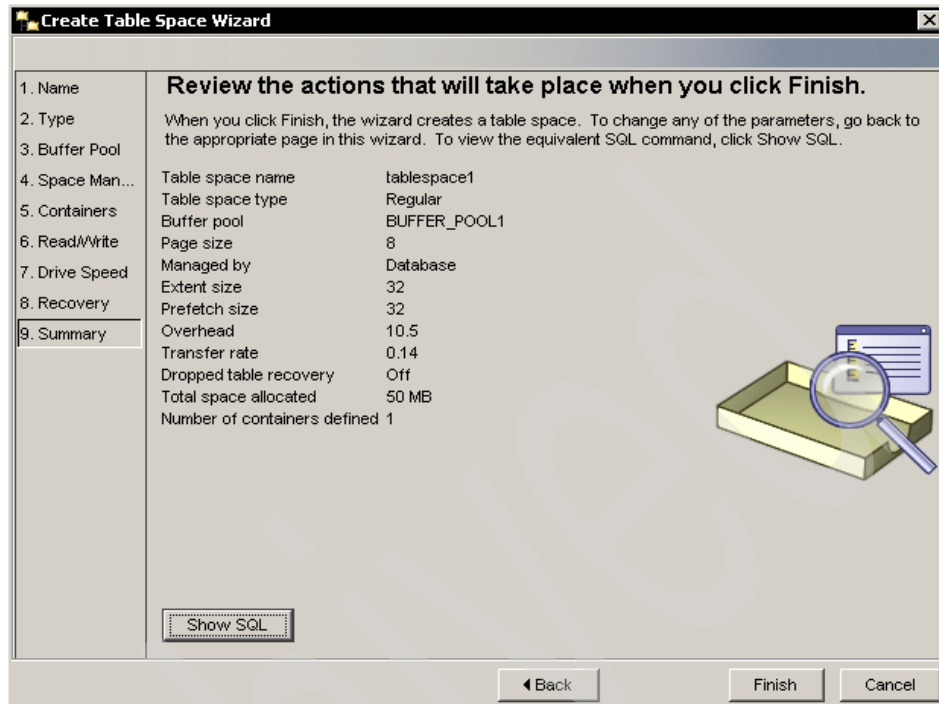


Figure 11-12 Create table space: summary

Use the `ALTER TABLESPACE` statement to modify an existing table space. You can make the following changes:

- ▶ Add a container to, or drop a container from, a DMS table space.
- ▶ Modify the size of a container in a DMS table space.
- ▶ Modify the following parameters: `PREFETCHSIZE`, `BUFFERPOOL`, `OVERHEAD`, or `TRANSFERRATE`.
- ▶ Modify the file system caching policy for a table space.

When new containers are added to a table space or existing containers are extended, a rebalance of the table space data might occur.

Adding a container that is smaller than existing containers results in a uneven distribution of data. This can cause parallel I/O operations (such as prefetching data) to perform less efficiently than they otherwise would on containers of equal size. Also remember that you cannot add containers to SMS table spaces except by using redirected restore.

With a DMS table space, you can drop a container from the table space or reduce the size of a container. Dropping or reducing a container is only allowed if the number of extents being dropped by the operation is less than or equal to the number of free extents above the high-water mark in the table space. This is necessary, because page numbers cannot be changed by the operation, and therefore, all extents up to and including the high-water mark must sit in the same logical position within the table space. Therefore, the resulting table space must have enough space to hold all of the data up to and including the high-water mark. In the situation where there is not enough free space, you will receive an error immediately upon execution of the statement.

The high-water mark is the page number of the highest allocated page in the table space. For example, a table space has 1000 pages and an extent size of 10, resulting in 100 extents. If the forty-second extent is the highest allocated extent in the table space, that means that the high-water mark is $42 * 10 = 420$ pages. This is not the same as used pages, because some of the extents below the high-water mark might have been freed up such that they are available for reuse.

In Figure 11-13 we add another container to the table space `tablespace1` by right-clicking the table space name and selecting the **Alter Table Space** option. Click the **Containers** tab, and you can then select the directory where the new container should be created. We labeled the container `tablespace1_cont2` in Figure 11-13.

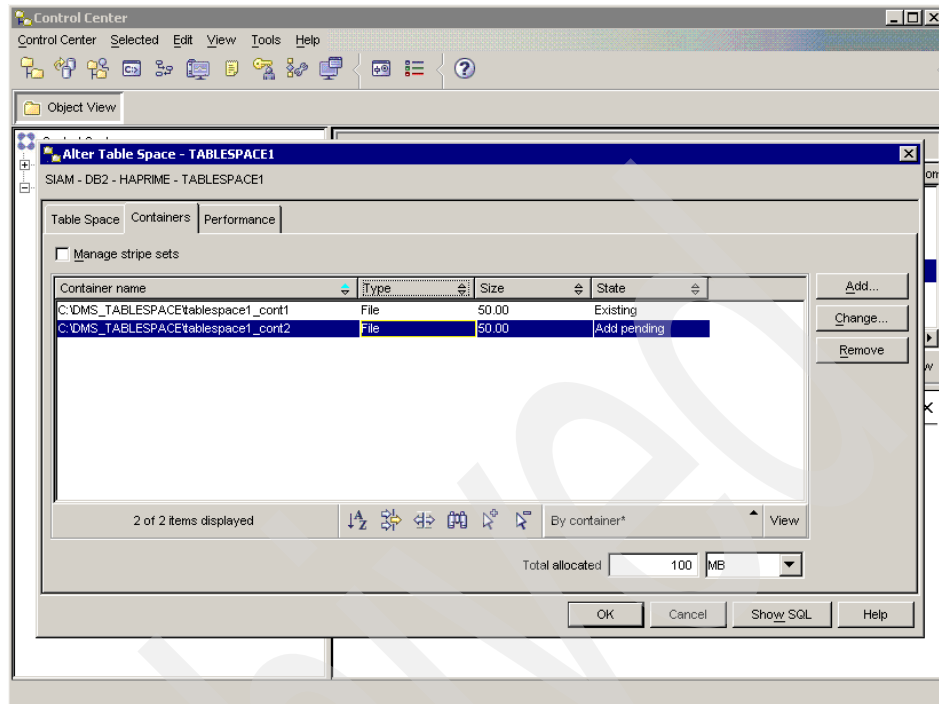


Figure 11-13 Add table space container wizard

Example 11-2 shows the command used to alter the table space when you select the **Show SQL** button in the wizard.

Example 11-2 Alter table space command

```
ALTER TABLESPACE TABLESPACE1 ADD ( FILE 'C:\DMS_TABLESPACE\tablespace1_cont2'
6400 );
```

11.3.2 Monitoring table space and container storage

Two methods are available to help monitor how much disk space is being used by the database objects. You can use either the Storage Management tool or a DB2 UDB command. We discuss both topics in the following sections.

The Storage Management tool

The Storage Management tool provides you with the ability to monitor database storage, including table spaces, containers, tables, and indexes.

Using the tool, you can:

- ▶ Display table space container statistics including size, total pages, and usable pages.
- ▶ Display table statistics such as number of rows and table space name.
- ▶ Set thresholds for data skew, space usage, and index cluster ratio. If a target object exceeds a specified threshold, the icons beside the object and its parent object in the Storage Management view are marked with a warning flag, indicated by a yellow triangle, or an alarm flag, indicated by a red circle.
- ▶ Take snapshots of storage objects. When a table space snapshot is taken, statistical information is collected from the system catalogs and database monitor for tables, indexes, and containers defined under the scope of the given table space. This information can be used for trend analysis.

Note: In order to see accurate storage information for databases in the Storage Management view, you must have up-to-date table statistics.

When a database or database partition group snapshot is taken, statistical information is collected for all the table spaces defined in the given database or database partition group. To launch the tool for an entire database, right-click the database name in Control Center and select **Manage Storage**. Figure 11-14 shows the Storage Management interface. In this example, table space TP1ACCTD is using 93% of its storage. From the Storage Management tool you can add additional containers to increase the storage. You can also review historical usage data relating to each table space.

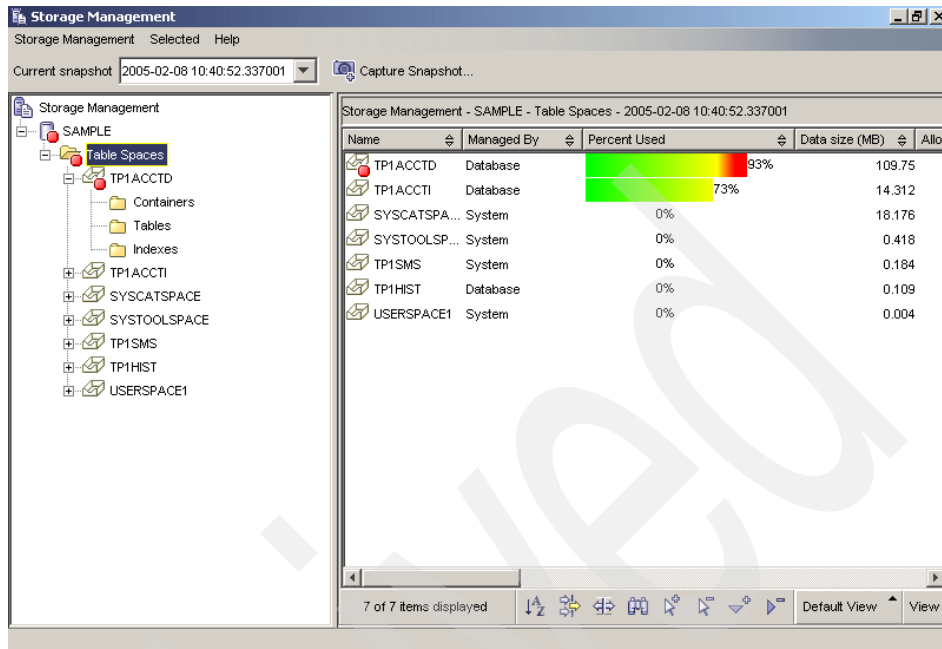


Figure 11-14 Storage management interface

Note: Typically, you will not want to set up thresholds for SMS table spaces, because the space will always be viewed as full, and the *Percent Used* indicator will show 100%. This occurs because the space grows with usage. It is still helpful to monitor SMS spaces because charts will show growth in the actual size over time and provide you with a helpful view of overall data usage.

Monitoring table space storage from the command line

You can use the DB2 UDB command **list tablespaces show detail** to view the table space type, page allocation and usage, extent size, page size, number of containers for all table spaces, and the table space state. A database connection is required to do so. Keep in mind that if you are using a SMS table space, all of the allocated pages are shown as used, and the free page value is not applicable. If you just want summary information, you can also use the **list tablespaces** command without the **show detail** option.

Example 11-3 shows detailed output from the **list tablespaces show detail** command.

Example 11-3 List table spaces show detail command

```
C:\>db2 list tablespaces show detail
```

Tablespaces for Current Database

Tablespace ID	= 0
Name	= SYSCATSPACE
Type	= System managed space
Contents	= Any data
State	= 0x0000
Detailed explanation:	
Normal	
Total pages	= 4338
Useable pages	= 4338
Used pages	= 4338
Free pages	= Not applicable
High water mark (pages)	= Not applicable
Page size (bytes)	= 4096
Extent size (pages)	= 32
Prefetch size (pages)	= 16
Number of containers	= 1
Tablespace ID	= 1
Name	= TEMPSPACE1
Type	= System managed space
Contents	= System Temporary data
State	= 0x0000
Detailed explanation:	
Normal	
Total pages	= 1
Useable pages	= 1
Used pages	= 1
Free pages	= Not applicable
High water mark (pages)	= Not applicable
Page size (bytes)	= 4096
Extent size (pages)	= 32
Prefetch size (pages)	= 16
Number of containers	= 1
Tablespace ID	= 2
Name	= USERSPACE1
Type	= System managed space
Contents	= Any data
State	= 0x0000
Detailed explanation:	
Normal	
Total pages	= 407
Useable pages	= 407
Used pages	= 407
Free pages	= Not applicable
High water mark (pages)	= Not applicable

Page size (bytes)	= 4096
Extent size (pages)	= 32
Prefetch size (pages)	= 16
Number of containers	= 1
Tablespace ID	= 3
Name	= TABLESPACE1
Type	= Database managed space
Contents	= Any data
State	= 0x0000
Detailed explanation:	
Normal	
Total pages	= 6400
Useable pages	= 6368
Used pages	= 96
Free pages	= 6272
High water mark (pages)	= 96
Page size (bytes)	= 8192
Extent size (pages)	= 32
Prefetch size (pages)	= 32
Number of containers	= 1
Minimum recovery time	= 2005-02-07-23.37.05.000000

In Example 11-3, the details for four table spaces are shown starting with the table space with ID of 0. Three default table spaces are shown. They are SYSCATSPACE, for system objects, TEMPSPACE1, for temporary system use, and USERSPACE1, for user data. All three table spaces are defined as SMS. SYSTOOLSPACE is a table space used for automatic statistics collection. TABLESPACE1 is DMS and holds table data.

To obtain more information about a table space, you can use the DB2 UDB command **list tablespace containers for *tablespace_number* show detail**. Example 11-4 shows the command executed against table space 3.

Example 11-4 List table space show detail for a specific table space

```
C:\>db2 list tablespace containers for 3 show detail
```

Tablespace Containers for Tablespace 3

Container ID	= 0
Name	= C:\DMS_TABLESPACE\tablespace1_cont1
Type	= File
Total pages	= 6400
Useable pages	= 6368
Accessible	= Yes
Container ID	= 1
Name	= C:\DMS_TABLESPACE\tablespace1_cont2

Type	= File
Total pages	= 6400
Useable pages	= 6368
Accessible	= Yes

Table space states

The state of the table space indicates its current status. A state is represented by a hexadecimal value. The state of a table space is composed of the hexadecimal sum of its state values. For example, if the state is “backup pending”, the value is 0x0020. The **db2tbs** (get tablespace state) command can be used to obtain the table space state associated with a given hexadecimal value. Example 11-5 shows the definition of each hexadecimal state value.

Example 11-5 DB2 UDB table space states

0x0	Normal
0x1	Quiesced: SHARE
0x2	Quiesced: UPDATE
0x4	Quiesced: EXCLUSIVE
0x8	Load pending
0x10	Delete pending
0x20	Backup pending
0x40	Roll forward in progress
0x80	Roll forward pending
0x100	Restore pending
0x100	Recovery pending (not used)
0x200	Disable pending
0x400	Reorg in progress
0x800	Backup in progress
0x1000	Storage must be defined
0x2000	Restore in progress
0x4000	Offline and not accessible
0x8000	Drop pending
0x20000000	Storage may be defined
0x40000000	StorDef is in 'final' state
0x80000000	StorDef was changed prior to rollforward
0x100000000	DMS rebalancer is active
0x200000000	TBS deletion in progress
0x400000000	TBS creation in progress
0x8	For service use only

11.3.3 Transaction logging

Transaction logging occurs at the database level in DB2 UDB and can be configured differently for each database in an instance. There are two types of logging, *circular* and *archival*.

SQL Server uses a combination of a physical log and virtual log for database recovery. The physical log contains an image of the page prior to change, and the logical log records the transaction.

DB2 UDB implements write-ahead logging in which the changed data is always written to the log files before the change is committed. DB2 UDB logs changes in its log files, including the old version of the data.

Managing logs

The number of logs to configure, their sizes, and other parameters affecting logging are set using database configuration parameters. DB2 UDB supports configurations that can dynamically extend the logs (using secondary logs) when primary log space is exhausted. DB2 UDB, by default, stores log files in the directory specified by the database configuration parameter `LOGPATH`. To change the log file path, the database configuration parameter `NEWLOGPATH` parameter should be updated. Log files can be stored as system files or a raw devices. You can control the number and size of the DB2 log files with the `LOGPRIMARY`, `LOGSECOND`, and `LOGFILSIZ` database parameters. Consult the DB2 UDB documentation *Administration Guide: Performance*, SC09-4821, for more information.

The DB2 UDB transaction log is an important part of the backup and recovery processes. The transaction log records all committed changes to the database and is used to bring the database to a consistent point after a database crash. It can also be used to restore a database to a recent state after media failure or application failure.

A total size of 256 GB is supported for logging. In addition to the primary logs, DB2 UDB also uses a number (between 0 and 126) of secondary logs as specified in the `LOGSECOND` parameter. Secondary logs do not occupy permanent file space, but are allocated one at a time as needed and deallocated as not needed. Secondary logs are used when the primary logs fill up and more log space is required. *Infinite active logging* is also new in DB2 UDB Version 8. Its allows an active unit of work to span the primary logs and archive logs effectively, allowing a transaction to use an infinite number of log files. Without infinite active log enabled, the log records for a unit of work must fit in the primary log space. Infinite active logging is enabled by setting the `LOGSECOND` parameter to -1. Infinite active logging can be used to support environments with large jobs that require more log space than you would normally allocate to the primary logs.

Another important configuration parameter, `BLK_LOG_DSK_FUL`, enables you to specify that DB2 UDB should not fail when running applications on a disk full condition from the active log path. When you enable this option, DB2 will retry every five minutes, enabling you to resolve the disk full situation and allowing the applications to complete.

The amount of space (in bytes) required for primary log files is:

$$(\text{LOGPRIMARY} * (\text{LOGFILSIZ} + 2) * 4096) + 8192$$

The amount of space (in bytes) required for primary and secondary log files is:

$$(\text{LOGPRIMARY} + \text{LOGSECOND}) * (\text{LOGFILSIZ} + 2) * 4096 + 8192$$

You should monitor the log storage directory to ensure that you have enough file system space to hold the logs.

Logging modes

DB2 UDB has two main logging modes: *non-recoverable*, and *recoverable*.

By default, when a database is created, the logging mode used is *non-recoverable*, also known as *circular logging*. When circular logging is used, the log files are reused and overwritten when more space is needed. This type of logging is similar to setting an SQL Server database to use the “simple” logging model.

The implications of this are that, with the exception of a system crash (where DB2 UDB will perform crash recovery), you must recover your database from a full offline backup. You cannot roll forward completed transactions that are contained in archive logs, but are not part of the offline backup. Circular logging is used to protect against power failure and by applications to roll back uncommitted changes if a processing error occurs. When this type of logging is enabled, both the LOGARCHMETH1 and LOGARCHMETH2 parameters are set to OFF. Only offline, full database backups can be performed with this type of logging.

The *recoverable* logging mode is used for databases that require recovery at a transaction level. With this mode, logs are not reused, but are maintained until they are archived. This type of logging provides all the protection of circular logging plus additional recovery benefits. The online and archived logs can be used, along with the last full backup, to recover a database to the state before a media or application failure. A database backup can be performed in online or offline mode and can be a full database backup or a selected table space backup. Recovery of a database can be performed to the last current committed transaction before the failure or to some specific point in time using the ROLLFORWARD utility. Also, with this type of logging, recovery can be performed on a table space basis. This method of logging is similar to the ‘Full’ logging model used in SQL Server. There are several options that affect how logs are archived with this mode. You can archive logs to disk, tape, use tape management software such as IBM Tivoli® Storage Manager, use third-party APIs, or write your own user exit routine.

Attention: If both LOGARCHMETH1 and LOGARCHMETH2 are specified, each log file will be archived in two different locations.

DB2 UDB supports log mirroring which is enabled by setting the MIRRORLOGPATH database configuration parameter.

To configure database logging in Control Center, perform the following steps:

1. Right-click the database to be configured, and select **Configure Database Logging**. You will be presented with a wizard that, gathers the required information, as shown in Figure 11-15.
2. Select the type of logging to be used. In this example, the current database SAMPLE, has circular logging enabled. We are going to change this to archive logging (recoverable).

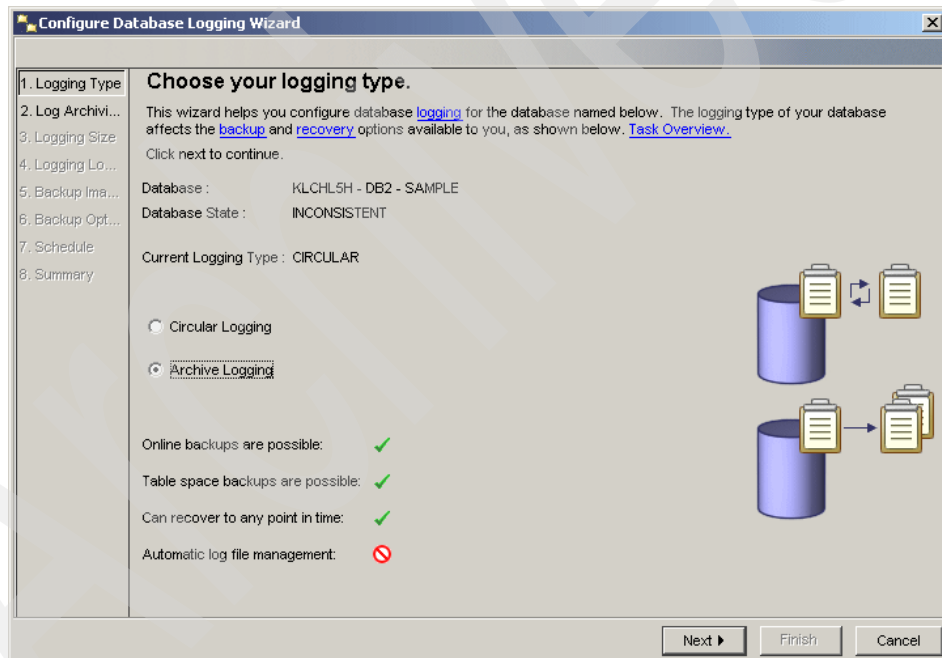


Figure 11-15 Configure Database Logging Wizard

3. Choose the location and size of your logs and mirrors (if using mirrors), the location of your backup, and any performance options. In this example, we are archiving our logs to the file system (DISK), with a primary archive path of C:\archive_log, a mirror log path of C:\mirror_logs, a failure archive path of

H:\SD-5D13\Alain\archive_logs, with ten primary and five secondary logs, each 1000 4k pages. See Figure 11-16.

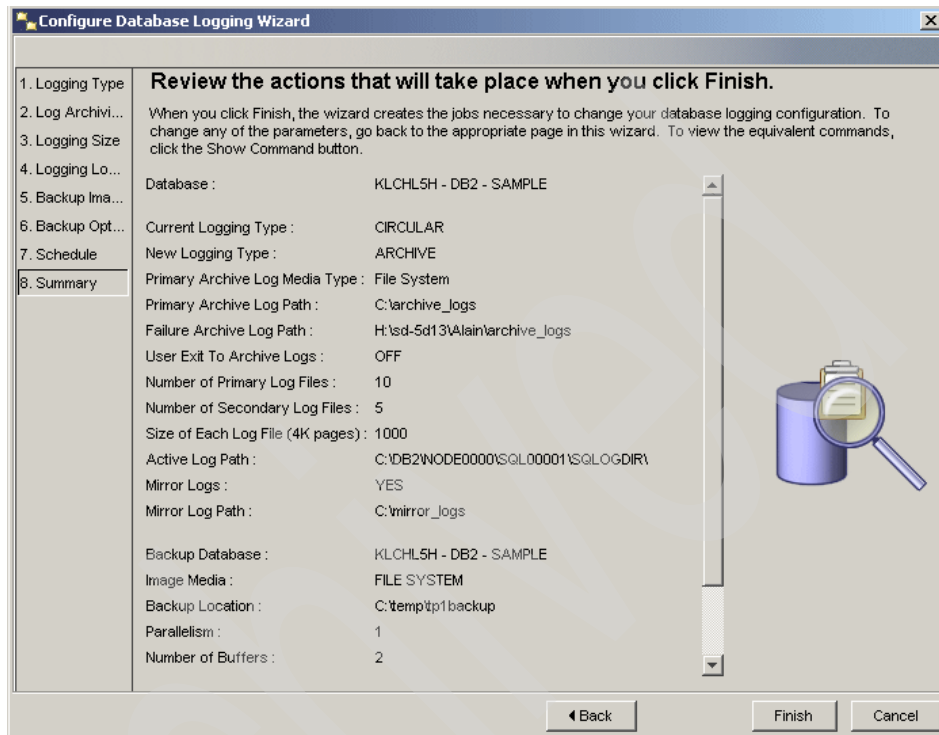


Figure 11-16 Configure Database Logging Wizard - Summary screen

Example 11-6 shows the commands generated by the wizard. First, we connect to the database, issue a quiesce to stop activity, and then update the database configuration to reflect the new archiving method and path. When changing the type of logging, you must first backup the database. The wizard also automatically issues the backup command. Note that the database will be taken offline for the backup, because circular logging does not allow online backups.

Example 11-6 Configure DB2 UDB database logging script

```
CONNECT TO SAMPLE;
QUIESCE DATABASE IMMEDIATE FORCE CONNECTIONS;
UNQUIESCE DATABASE;
CONNECT RESET;
UPDATE DB CFG FOR SAMPLE USING logarchmeth1 "DISK:C:\archive_logs" failarchpath
"H:\sd-5d13\Alain\archive_logs" logprimary 10 logsecond 5 logfilsiz 1000
mirrorlogpath C:\mirror_logs;
```

```

BACKUP DATABASE SAMPLE TO "C:\temp\tp1backup" WITH 2 BUFFERS BUFFER 1024
PARALLELISM 1 WITHOUT PROMPTING;
CONNECT TO SAMPLE;
QUIESCE DATABASE IMMEDIATE FORCE CONNECTIONS;
UNQUIESCE DATABASE;
CONNECT RESET;
UPDATE DB CFG FOR SAMPLE USING logarchmeth1 "DISK:C:\archive_logs" failarchpath
"H:\sd-5d13\Alain\archive_logs" logprimary 10 logsecond 5 logfilsiz 1000
mirrorlogpath C:\mirror_logs;
BACKUP DATABASE SAMPLE TO "C:\temp\tp1backup" WITH 2 BUFFERS BUFFER 1024
PARALLELISM 1 WITHOUT PROMPTING;

```

Important: In DB2 UDB Version 8.2, DB2 UD no longer requires the use of a user exit to archive logs. Also, the LOGARCHMETH1 and LOGARCHMETH2 parameters replace the LOGRETAIN and USEREXIT parameters that were used prior to V8.2. LOGRETAIN and USEREXIT are still available to maintain backward compatibility.

11.4 Working with buffer pools

Buffer pools are important memory components and are main elements that affect database performance. Every database in DB2 UDB must have at least one buffer pool defined. A buffer pool is memory used to cache table and index data pages as they are being read from disk or being modified. Because memory access is much faster than disk access, the less often the database manager needs to read from or write to a disk, the better the performance. Large objects and long field data does not go into buffer pools. One buffer pool, IBMDEFAULTBP with 4Kb page size is created by default when the **CREATE DATABASE** command is executed.

After converting all database objects, applications and data, you are probably expected to tune your database to meet your business requirement. Working with buffer pools is one of the necessary tasks in performance tuning, which is covered in Chapter 12, “Post-conversion tuning considerations” on page 339. In this section, we cover are just how to create, drop, list, and alter buffer pools.

Create, alter and drop buffer pools

Before you create a new buffer pool, answer the following questions:

- What buffer pool name will you use? Issue the following query:

```
SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
```

get the list of buffer pool names that already exist in the database. The name must not begin with the characters “SYS” or “IBM”.

- ▶ Is the buffer pool to be created immediately, or created the next time that the database is deactivated and reactivated?
- ▶ What values do you want to associate with the parameters controlling the size of the buffer pool, including the page size and the total size of the buffer pool?
- ▶ Do you want to use extended storage, block-based support, or neither? Using block-based buffer pools improves performance for sequential prefetching. If extended storage is enabled, it can be used as a secondary cache for pages that are evicted from the buffer pool. For more information, refer to DB2 UDB documentation *Administration Guide: Performance*, SC09-4821.

Important: The page sizes specified for your table spaces should determine the page sizes that you choose for your buffer pools. The choice of page size used for a buffer pool is important, because you cannot alter the page size after you create a buffer pool. A table space of a chosen page cannot be created until at least one buffer pool, using that same page size, exists.

Memory is allocated to a buffer pool when a database is activated or when the first application connects to the database. You can change buffer pool allocations while DB2 UDB is running. If you use the **IMMEDIATE** keyword when you use the **ALTER BUFFERPOOL** command to increase the size of the buffer pool, memory is allocated as soon as you enter the command if the memory is available. If the memory is not available, the change occurs when all applications are disconnected and the database is reactivated. If you decrease the size of the buffer pool, memory is deallocated at commit time. When all applications are disconnected, the buffer pool memory is deallocated.

To create a buffer pool, you can use either the **CREATE BUFFERPOOL** command or Control Center.

To create a buffer pool in Control Center:

1. Expand the object tree until you see the buffer pool folder for the specific database.
2. Right-click the *Buffer Pool* folder and select **Create...**
3. Enter the properties of the new buffer pool and click **OK**.
4. Click the **Show SQL** button to display the assigned DB2 UDB command see(Figure 11-17).

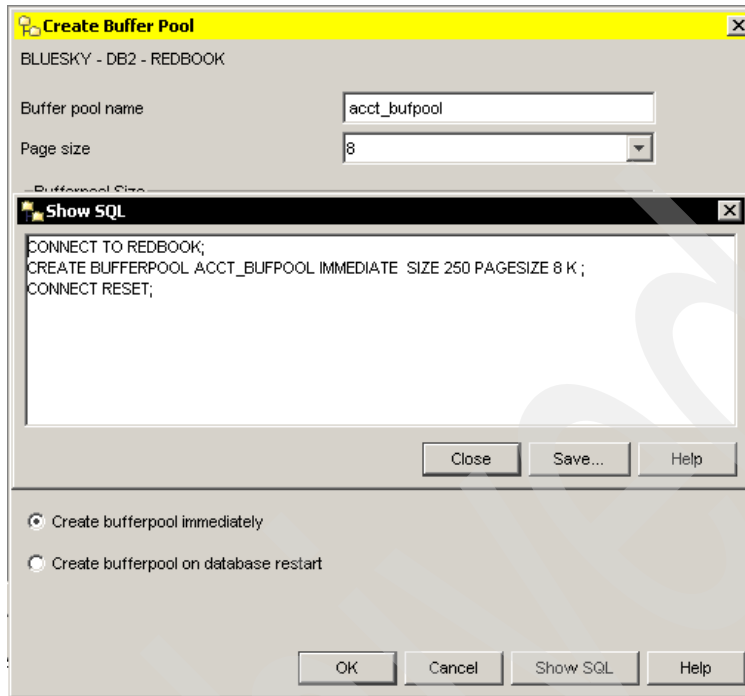


Figure 11-17 Create buffer pool window

After a buffer pool has been created, it can be associated with one or more table spaces during creation of table spaces.

Buffer pools can be resized to accommodate your changing memory requirements. To resize the buffer pool, we performed the following steps:

- ▶ Expand the object tree until you see the buffer pool folder for the specific database.
- ▶ At the right of the object view, right-click the specific buffer pool you want to alter, and select **Alter** at the pop-up menu.

If you want to drop a buffer pool, the steps are similar to altering a buffer pool, but instead select **Drop** at the pop-up menu.

11.5 Task Center and the DB2 Tools Catalog

In this section, we discuss how to use the DB2 Task Center to create and schedule tasks. The DB2 Administration Server and Tools Catalog database are the prerequisite for using the Task Center and most GUI tools. We discuss the

DB2 Administration Server and Tools Catalog database in preparation for enabling the Task Center and Scheduler.

11.5.1 DB2 Administration Server and Tools Catalog Database

The DB2 Administration Server (DAS) is a process used to perform administration tasks on DB2 UDB servers. You must have a running DAS if you want to use scheduling facilities provided by the Task Center, or if you want to administrate a DB2 UDB server remotely using the various GUI tools.DB2. The DAS supports the Control Center and Configuration Assistant when working on the following administration tasks:

- ▶ Enabling remote administration of DB2 servers
- ▶ Providing the facility for job management, including the ability to schedule both DB2 UDB and operating system command scripts
- ▶ Defining the scheduling of jobs, viewing the results of completed jobs, and performing other administrative tasks against jobs located either remotely or locally to the DAS, using the Task Center
- ▶ Providing a means for discovering information about the configuration of DB2 instances, databases, and other DB2 administration servers in conjunction with the DB2 Discovery utility

Task information includes the commands to be run, schedules, notifications, and completion actions associated with the task. The task execution results are stored in a DB2 IDB database called the *Tools Catalog* or *Tools Database*. The Tools Catalog can be created during installation of DB2 UDB. If you did not create the Tools Catalog database during installation, it can be created and activated through Control Center or CLP. If using Control Center, select **Tools** → **Tools Settings** → **Scheduler Settings** to create a new Tools Catalog, as shown in Figure 11-18.

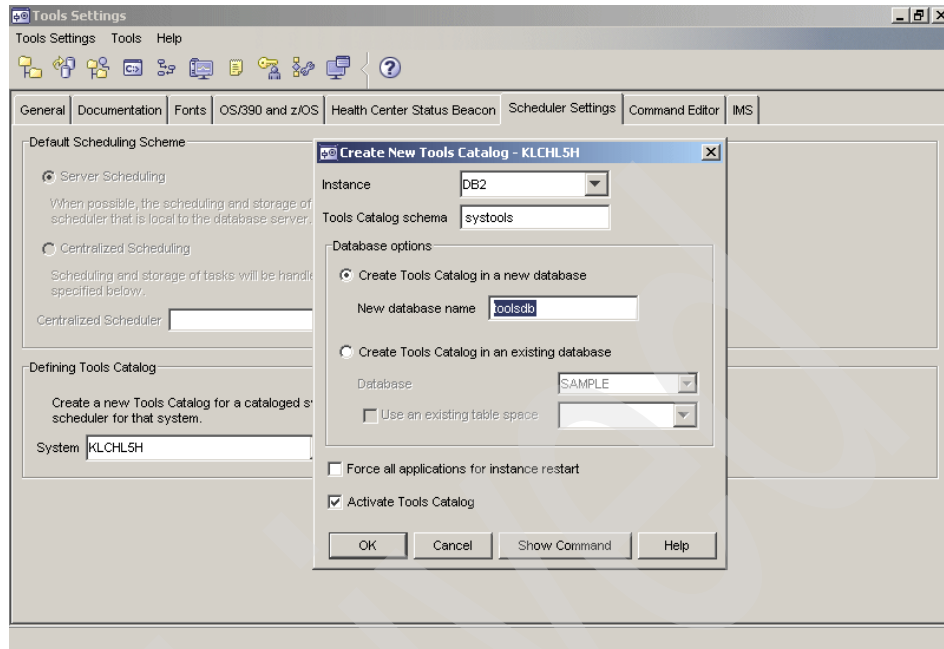


Figure 11-18 Creating the Tools Catalog from the Control Center

To create the Tools Catalog using CLP, use the **CREATE TOOLS CATALOG** command. Typical usage of the command would be:

```
db2 create tools catalog systools create new database toolsdb
```

Note: The *SYSTOOLS* and *TOOLSDB* parameters listed in the command example above are user provided. We have used these particular names in order to show the syntax.

To summarize, the tools catalog database contains task information created by the Task Center and Control Center. These tasks are run by the scheduler on the server where the DAS resides.

11.5.2 Task Center and Scheduler

After the DAS has started and the Tools Catalog is ready, you can begin scheduling tasks in Task Center. You can use the Task Center to run tasks, either immediately or according to a schedule, and to notify people about the status of completed and failed tasks. A task is a script, together with the associated success conditions, schedules, and notifications. You can create a new task in

the Task Center, create a script in another tool and save it to Task Center, import an existing script, or save the options from a DB2 dialog or wizard such as the Load wizard in Task Center. A script can contain DB2 commands, SQL statements, or operating system commands. Task Center can be used to do the following:

- ▶ Schedule the task.
- ▶ Specify success and failure conditions.
- ▶ Specify actions that should be performed when the task completes successfully or when it fails.
- ▶ Specify e-mail addresses (including pagers) that should be notified when the task completes successfully or when it fails.

You can specify conditional coding by creating task actions. Each task action consists of a task and the action that should be performed by the task. For example, task one can have the following task actions:

- ▶ If task one is successful, task action A enables the schedule for task two.
- ▶ If task one fails, task action B runs task three.

The following steps guide you through creating a task, scheduling the task and then viewing historical information about the task execution. In the example we create a backup job using Task Center. We also demonstrate creating a backup job using the “Backup Wizard” in Control Center.

1. Launch Task Center from either Control Center or from a Command Line with the **db2tc** command. Select the Scheduler System. This is the system where the task is scheduled on and can be the local system or a remote one. Right-click in the main window and select **New**. You can also select **Task** → **New** from the main menu.
2. Enter information such as the name of the task, type of script, a brief description, and the category of task, as shown in Figure 11-19. The type of scripts that can be run are DB2 UDB command scripts, operating system scripts, MVS™ and JCL scripts for running on host database systems such as z/OS, and Group Tasks, which allows several tasks to be grouped and run together. Categories of task can also be created. The Run System where the task will actually run must be specified. The Instance or partition that the script applies to must also be specified.

The screenshot shows the 'New Task' dialog box with the following fields and values:

Field	Value
Name	backup1
Type	DB2 command script
Description	sample database backup
Task category	Backups
Run system	KLCHL5H
DB2 instance and partition	DB2 0

Buttons at the bottom: OK, Cancel, Help.

Figure 11-19 Entering task information in Task Center

3. Input the script for the task. In the example in Figure 11-20, we have specified a database backup. By clicking the **Import** button you will be able to import any scripts previously created or ported from SQL Server.

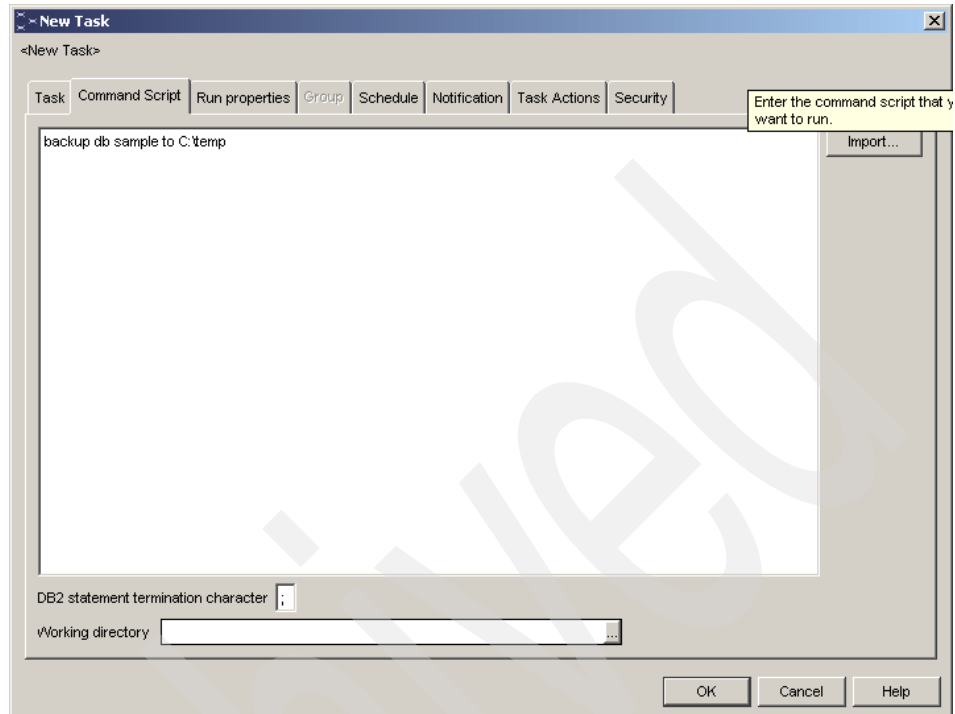


Figure 11-20 Entering script information in Task Center

4. Create success or failure creations for the task. You can define the specific success code for your task and choose whether DB2 UDB should stop execution if failure occurs during task execution. In Figure 11-21, we have chosen to stop execution when failure happens.

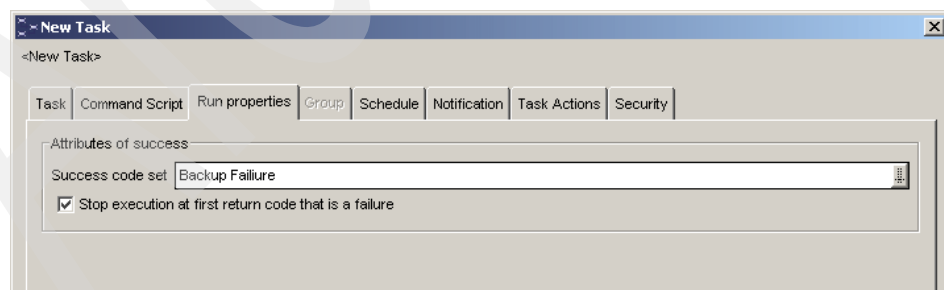


Figure 11-21 Entering task run properties

5. Schedule the task to run at a specific time, only once or a repeating schedule. In Figure 11-22, we specified that the backup job should run every two days at a specific time. If you want to save the task schedule for use with other tasks

at a later date, select the **Save List of Schedules** button. Then the next time you create a task that you want to run at the same time, select the **Use Saved Schedule** radio button and select the saved schedule from the list.

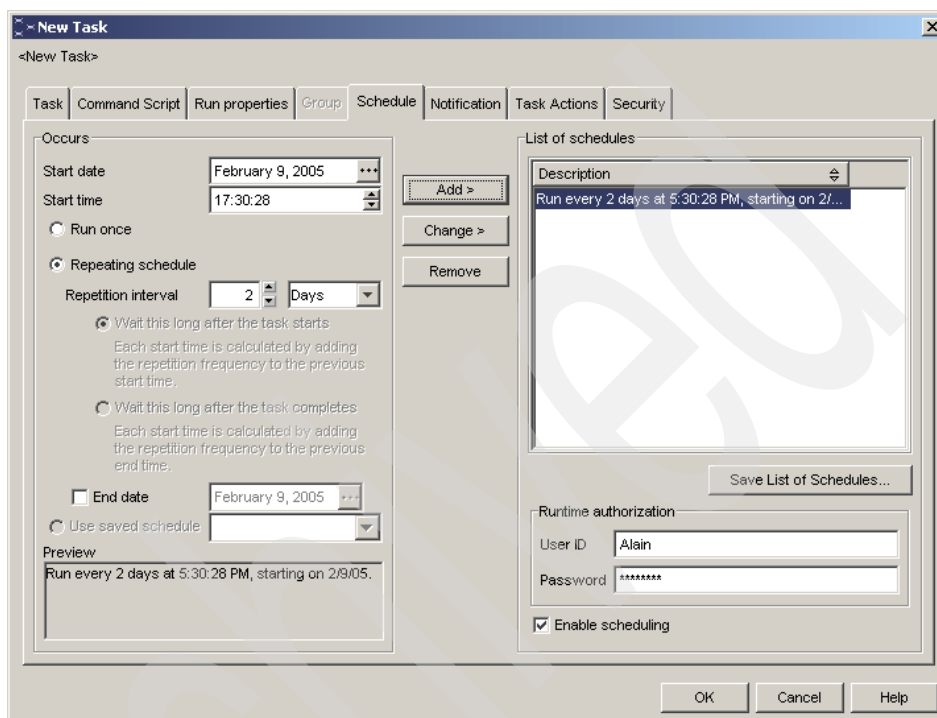


Figure 11-22 Scheduling the task in Task Center

6. Specify the task notification information. The notification can be sent to contact or contact group, by e-mail or pager, or recorded in Journal as a message entry. Success or failure condition can be specified. You also can specify multiple notifications for different conditions for a task. In Figure 11-23, an e-mail notification is configured.

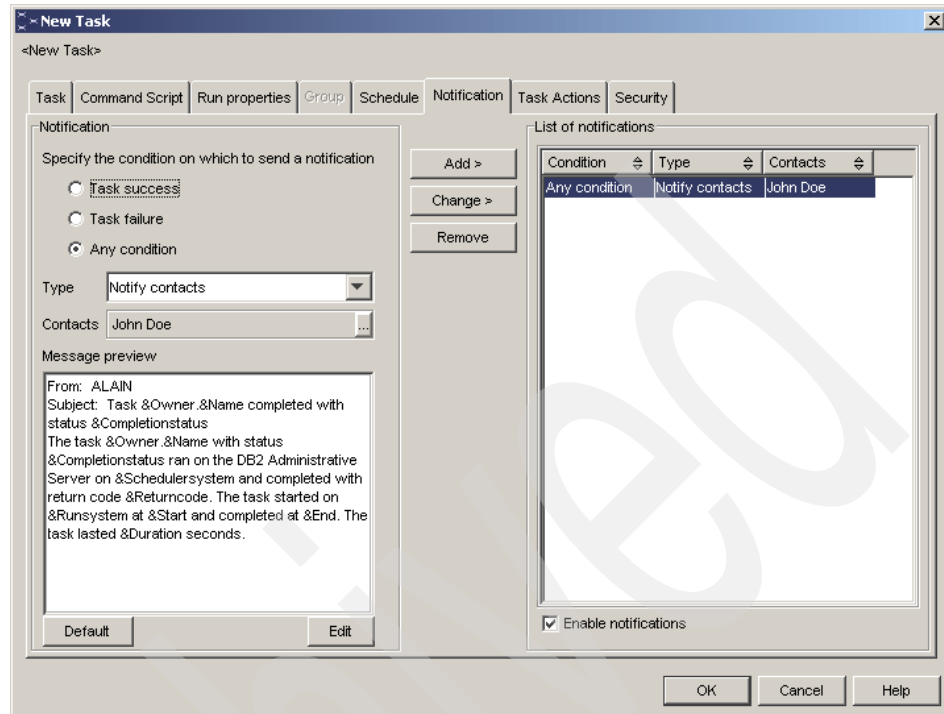


Figure 11-23 Entering notification information in Task Center

7. In addition to the notification setting, you can also specify follow-up actions under the *Task Actions* tab, as shown in Figure 11-24. For example, you can specify a *Task Failure* condition so if a task fails, you can run another task to remedy the condition, or to disable another task to avoid unnecessary damage.

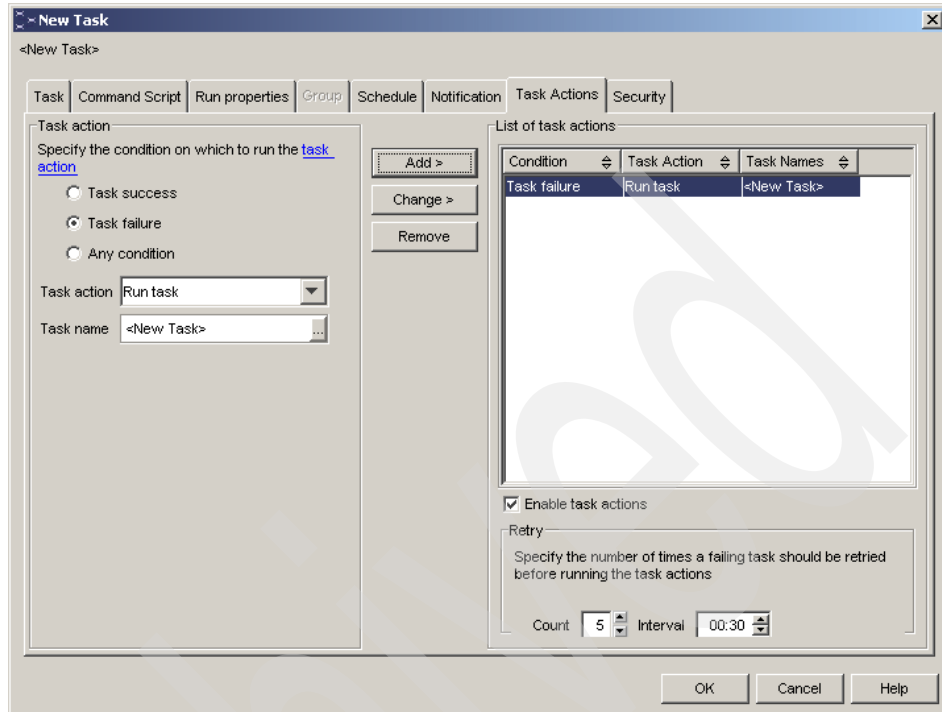


Figure 11-24 Specifying task action in Task Center

8. You can also specify access privileges to the task (Figure 11-25), such as read, run, or execute for different users and groups by clicking the **Security** tab. The final step is to submit the task to Task Center by clicking **OK**.

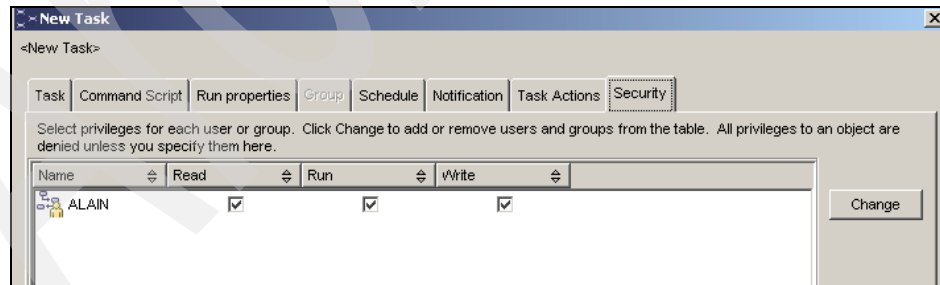


Figure 11-25 Specifying task security in Task Center

Saving tasks from wizards

You also have the option of saving and scheduling tasks from any of the wizards. Figure 11-26 shows an online backup task that we want to save to Task Center.

We have specified that the backup job will run every Monday at 11:00 p.m. The options *Save task only*, which will save the task to Task Center but not schedule it, and *Save and run task now* which executes the task and saves it to Task Center are also available.

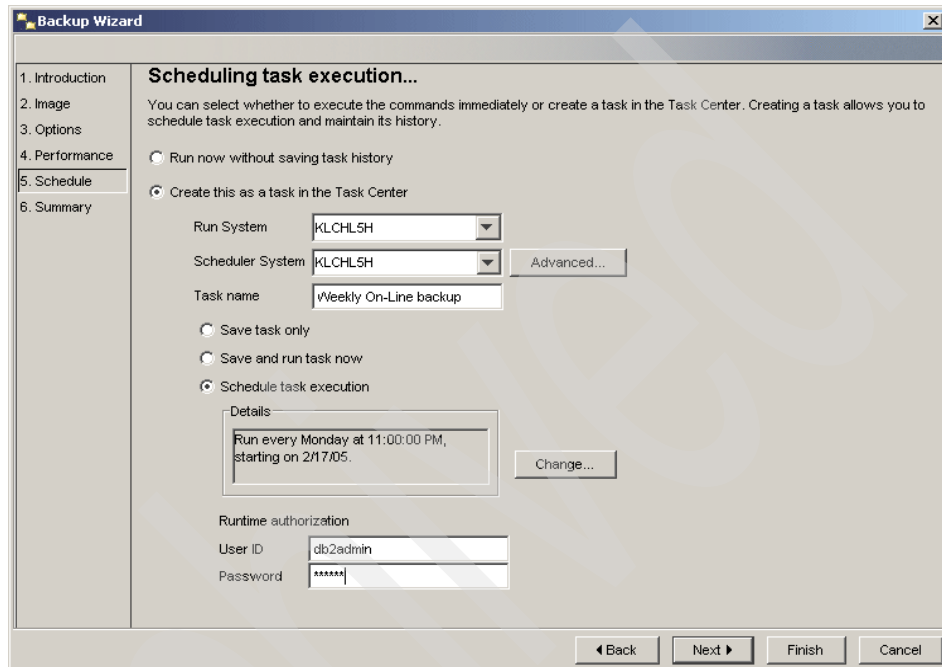


Figure 11-26 Saving and scheduling a task from the Backup Wizard

11.6 Backup, recovery, and log administration

Databases should to be backed up from time in case of system and media failures. Both SQL Server and DB2 UDB can perform full or incremental backups while the database is online. Both can also perform online restoration. Offline backups and restoration capabilities are also provided. DB2 UDB has more advanced backup and recovery features which we highlight in the subsequent sections. These features can help a DBA automate the backup/recovery tasks and increase productivity.

SQL Server backup and restore commands can be used on databases, transaction logs, database files, and file groups. Incremental backup and recovery include the changes from the last full backup. This type of backup is known as *differential* backup. Point in time recovery is achieved by restoring a

backup image and applying the transaction logs to recover the transactions. The set of backup history tables is maintained in the `msdb` database.

DB2 UDB, allows backup and restore commands on database and table spaces. Archived log files can be backed up as ordinary operating system files, and/or can be included into a backup image during online backup operations. DB2 UDB has two types of incremental backups - *incremental* (same as differential in SQL Server), and *delta* (changes from last backup of any type). Point in time recovery is achieved using the **rollforward** command. DB2 UDB has a **recover** command that combines **restore** and **rollforward** commands into a single step and uses the information in the recovery history file to determine which backup images and log files to use.

Both SQL Server and DB2 UDB are capable of performing backup restore operations in parallel and API support is also available. The following is a list of some of the functions that are new to SQL Server administrators are described in more detail in subsequent sections to help you enhance your database backup/recovery tasks:

- ▶ Automatic backup maintenance
- ▶ Backup using throttling mode
- ▶ Log file inclusion in backup images
- ▶ Backup compression
- ▶ Automatic log file management
- ▶ Undo management

Note: DB2 UDB now automatically self tunes its backup and restore operations by choosing the number of buffers, buffer size and parallelism settings based on the amount of memory available, number of processors available and database configuration, if those parameters are not supplied with the **backup/restore** command.

11.6.1 Automatic backup maintenance

Automatic maintenance is a time saving feature. DB2 UDB is equipped with this feature for policy-based backup, reorg and statistics gathering. Here we focus on how you could save time using automatic backup maintenance.

The goal of this feature is to simplify database backup management tasks by always ensuring that a recent full backup of the database is performed as needed. DB2 UDB determines the need to perform a backup operation based on one or more of the following metrics:

- ▶ If you have never completed a full database backup

- ▶ The time elapsed since the last full backup is more than a specified number of hours
- ▶ The transaction log space consumed since the last backup is more than a specified number of 4 KB pages (in archive logging mode only).

If you created the database without configuring automatic maintenance, you can access the *Configure Automatic Maintenance* wizard either from the Control Center or the Health Center.

To launch the wizard from the Control Center, expand the object tree until you see the desired database, then right-click the database and select **Configure Automatic Maintenance**.

Through the wizard (Figure 11-27), you can:

- ▶ Enable or disable customization of your backup maintenance activity
- ▶ Specify the online and offline maintenance windows
- ▶ Manage notification contact list
- ▶ Specify whether to send notification and/or automate backup operations
- ▶ Specify backup criteria, location, and mode

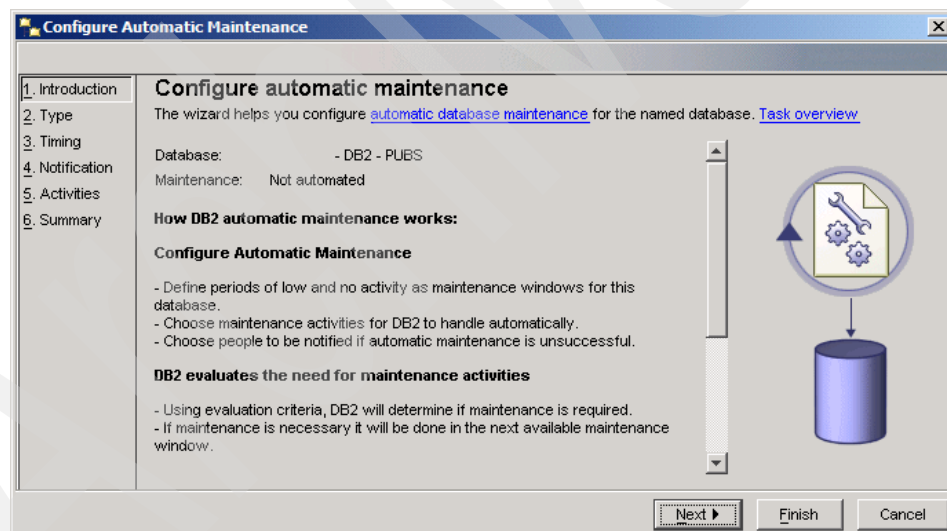


Figure 11-27 *Configure Automatic Maintenance Wizard*

If the database is enabled for rollforward recovery, then automatic database backup can be enabled for either online or offline backup. Otherwise, only offline backup is available. Automatic database backup supports disk, tape, Tivoli Storage Manager (TSM), and vendor DLL media types.

If backup to disk is selected, the automatic backup feature will regularly delete backup images from the directory specified in the Configure Automatic Maintenance wizard. Only the most recent backup image is guaranteed to be available at any given time. We recommend that this directory be kept exclusive for the automatic backup feature and not be used to store other backup images.

The automatic database backup feature can be enabled or disabled by using the `AUTO_DB_BACKUP` and `AUTO_MAINT` database configuration parameters.

11.6.2 Backup using throttling mode

Throttling is a usability advancement to backup, rebalance, and runstats utilities on DB2 UDB V8.2. This feature is unavailable in SQL Server. Throttling for backup operation enables:

- ▶ The DBA to reduce the impact of running resource-intensive backup utilities on operational workload, in this case, the backup operation.
- ▶ Priority settings to allow DB2 UDB to regulate the performance impact of the backup operation.

You can enable throttling when going through the backup wizard (Figure 11-28):

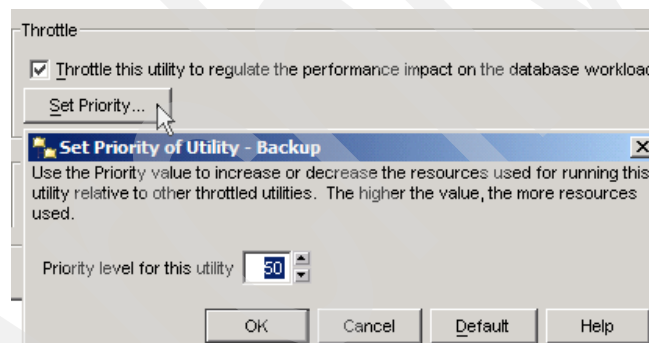


Figure 11-28 Enable throttling and set priority for backup utility

Priority can be any number between 1 and 100, with 1 representing the lowest priority, and 100 representing the highest priority.

11.6.3 Log file inclusion in backup images

DB2 UDB transaction log files are essential for restoring and recovering a database or table space to a consistent point in time.

For SQL Server, and DB2 UDB prior to Version 8.2, the backup and restore utilities requires shipping the backup image and corresponding log files as

separate objects to the disaster recovery sites. This risked losing the log files required for recovery.

The new feature in DB2 UDB Version 8.2 enables you to include the logs as part of the online backup. This way, you can ship the backup images with the log files to ensure that the backup will be restorable if archived logs are misplaced. This feature supports all types of online backups such as database, table space, incremental, and compressed. To enable it in Control Center, select the option available in the *Options* page of the Backup Wizard as shown in Figure 11-29.

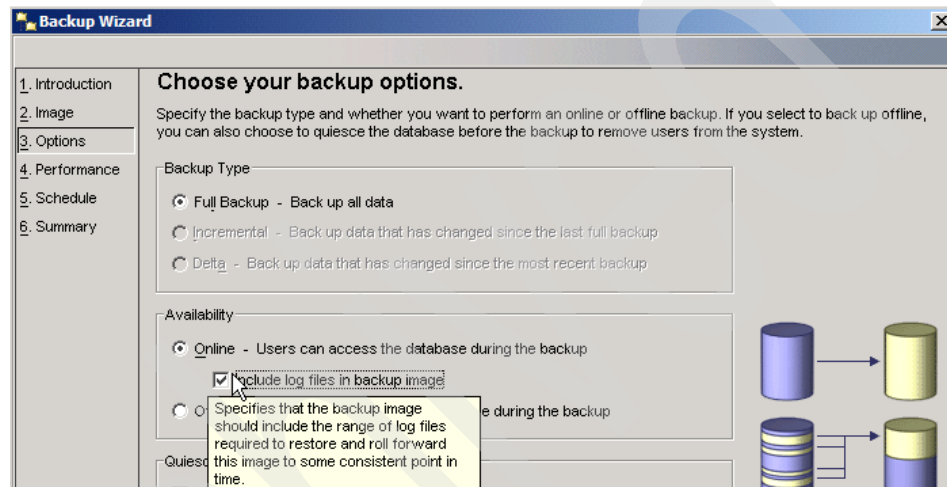


Figure 11-29 Include log files in backup images during online backup operation

If this option is enabled, DB2 UDB will truncate the tail log file and close it after the table space or database is backed up. All logs that are needed to restore the backup and roll forward to the time corresponding to the end of the backup are placed in the backup image. DB2 UDB backs up log files in parallel while backing up the data. If database backup compression is specified, the log files are compressed as well.

When restoring the database, specify the directory where you want DB2 UDB to place the log files, as shown in Figure 11-30.

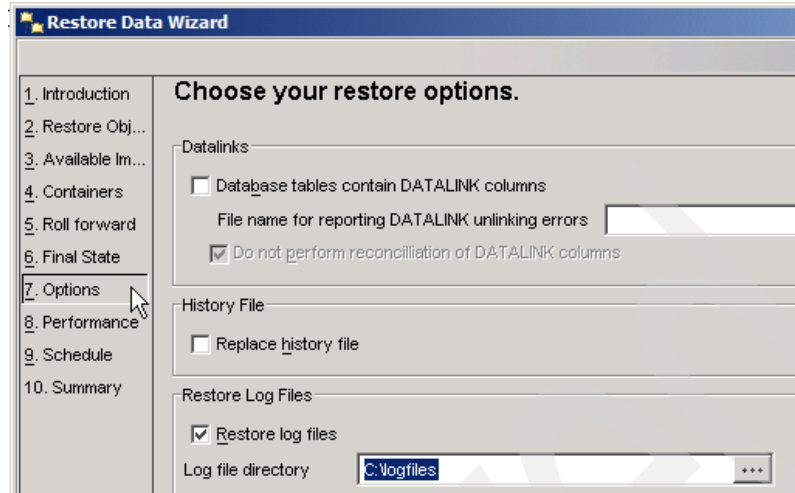


Figure 11-30 Restore log files into a specified directory from a backup image

During an automatic incremental restore operation, only the logs included in the target image of the restore operation are retrieved. Any logs that are included in intermediate images referenced during the incremental restore process are not extracted from those backup images. During a manual incremental restore, if you specify a log target directory when restoring a backup image that includes log files, the log files in that backup image are restored.

Note: This feature is only available on single-partition databases. Also, when logs are included in an online backup image, the resulting image cannot be restored on releases of DB2 UDB prior to Version 8.2.

11.6.4 Backup compression

DB2 UDB backups can be compressed automatically. This feature uses the built-in Lempel-Ziv (LZ) compression or a user-supplied compression library to compress the database backup image. It can significantly reduce backup storage costs. Heterogeneous restore is supported.

Use this feature by enabling it in the *Option* tab of the backup wizard, or in the **backup** command with the COMPRESS parameter. The **restore** utility recognizes the

compressed database and automatically decompresses the backup. No additional parameter is needed.

If you use a user-provided compression library, the library is stored in the backup image by default and is used on restore. The command syntax is:

```
DB2 BACKUP DATABASE sample COMPRESS COMPLIB libname COMPROPTS options  
DB2 RESTORE DATABASE sample COMPLIB libname COMPROPTS options
```

In these examples, COMPROPTS is used to pass the options to the compression library.

Use of the compression feature may result in additional CPU overhead due to the compression computations. However, the media I/O time is decreased because the image size is smaller. The overall backup/restore performance impact depends on whether CPU or media I/O is a bottleneck of the system.

11.6.5 Automated log file management

In DB2 UDB, you no longer need to customize userexits or code your own log management routine to archive and delete the DB2 UDB logs. DB2 UDB uses a log manager, *db2logmgr*, which manages the use, archive, retrieval, and deletion of log files with minimal user interaction. You can archive the logs to TSM, tape, disks, or third-party vendor device. You can specify multiple archive targets for extra redundancy or change the archive targets online.

To set up this feature, update the database configuration parameters based on your needs. The database parameters used by the log manager for controlling archive locations and handling archival problems are:

- ▶ Managing log archive locations:
 - *LOGARCHMETH1*: to specify the primary log archive. The option syntax is:
 - DISK: <path>
 - TSM: [management class name]
 - Vendor: <vendor library>
 - USEREXIT
 - LOGRETAIN
 - *LOGARCHMETH2*: optional log archival target, same syntax as *LOGARCHMETH1*.
 - *LOGARCHOPT1*: for specifying optional archiving parameters
 - *LOGARCHOPT2*: for specifying optional archiving parameters

For example:

```
DB2 UPDATE DB CFG FOR sample USING LOGARCHMETH1 DISK:d:/db2logsbk
DB2 UPDATE DB CFG FOR sample USING LOGARCHMETH1 TSM
DB2 UPDATE DB CFG FOR sample USING LOGARCHOPT1 -SERVERNAME=ITSOTSM1
```

Note: The LOGRETAIN and USEREXIT are deprecated starting in Db2 UDB V8.2.

- Handling log archival problems:
 - FAILARCHPATH: Specifies the path to be used when any primary targets cannot be archived. The media for FAILARCHPATH can only be disk. This parameter can be set dynamically after a problem with a primary target. Any pending-retry log archival requests are switched to use this path.
 - NUMARCHRETRY: The number of retry attempts on primary target(s) before archiving to FAILARCHPATH. The default value is 5.
 - ARCHRETRYDELAY: The number of seconds between retry attempts. The default value is 20 seconds.

Note: USEREXIT is still back-supported as another log management option.

In the **PRUNE HISTORY** command, the **AND DELETE** option is used to prune the unnecessary logs.

```
>>-PRUNE----->
>--+HISTORY--timestamp--+-----+--+-----+--+<
|                               '-WITH FORCE OPTION-'  '-AND DELETE-' |
|'-LOGFILE PRIOR TO--log-file-name-----' |
```

When the history file entry is removed, the associated log archives are physically deleted. This option is especially useful for ensuring that archive storage space is recovered when log archives are no longer needed.

Note: If you are archiving logs via a user exit program or in-house developed routine, the logs cannot be deleted using this option.

11.6.6 Undo management

Human error that causes data corruption in databases is an increasing problem. SQL Server uses point in time recovery with backup files, and delayed application of log records to a standby database. When an unwanted update occurs, the data administrator can restore the database up to the point before the data corruption. Or, if the problem is detected within the delay time, the original

data can be restored from the standby database. The disadvantage of restoring a database to a specific point in time to fix this error is that all tables in that database are affected. They are all recovered to the same point in time.

DB2 UDB provides a feature to address this problem: *undo management*. In DB2 UDB, it is possible to re-create mistakenly dropped tables. If you foresee that your environment is at risk of users or applications corrupting the data and you required a solution that fixes the problem to the specific granularity of the erroneously changed data, consider using undo management.

With the optional *DB2 Recovery Expert for Multiplatforms* product, you can use sophisticated undo management features that are based on mining archived and active logs. DB2 Recovery Expert gives you the option to reverse changes that were made incorrectly. It also supports autonomic recovery management. Refer to the product Web site to get the latest information about his product:

<http://www.ibm.com/software/data/db2imstools/db2tools/db2re/>

11.7 High availability

As you may recall from the section 2.10, “High availability” on page 30, DB2 UDB has a variety of mechanisms for maintaining the databases availability. In this section, we explore several of these mechanisms, including high availability disaster recovery (HADR) and mirroring.

11.7.1 Failover clustering

Failover clustering or HA clustering is the most common method of providing system availability. DB2 UDB supports failover clustering with Microsoft Cluster Server (MSCS) on Windows NT Server, Windows 2000 Server, and Windows Server 2003 operating systems. It also has support for HACMP™ on AIX, Veritas Cluster Server and Tivoli System Automation.

11.7.2 HADR

High Availability Disaster Recovery (HADR) is a feature introduced with DB2 UDB Version 8.2. HADR is a database replication feature that provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the *primary*, to a target database, called the *standby*.

DB2 UDB HADR applications can only access the primary database. Updates to the standby database occur by rolling forward the log data generated on the

primary database and shipped to the standby database. HADR can be set up using Control Center or from the command line.

The following example shows how HADR can be set up and configured using Control Center.

Before setting up HADR, you must configure your primary and standby environments first:

- ▶ Configure your primary database to use archive logging. This is a requirement for HADR.
- ▶ Create and configure a standby instance with TCP/IP communications enabled. This is a standard database instance created using the **db2icrt** command. You do not need to create a database for this instance; one will be created for you as part of the HADR configuration wizard. We have used the default instance name (DB2) for our standby server.

To set up and configure HADR:

1. In Control Center, select the database you want to configure by right-clicking, and then selecting **High Availability Disaster Recovery** → **Set Up**. The HADR introduction window opens. Click **Next**.
2. Identify the primary instance and database name, as shown in Figure 11-31. Here the primary database is called HAPRIME. Click **Next**.

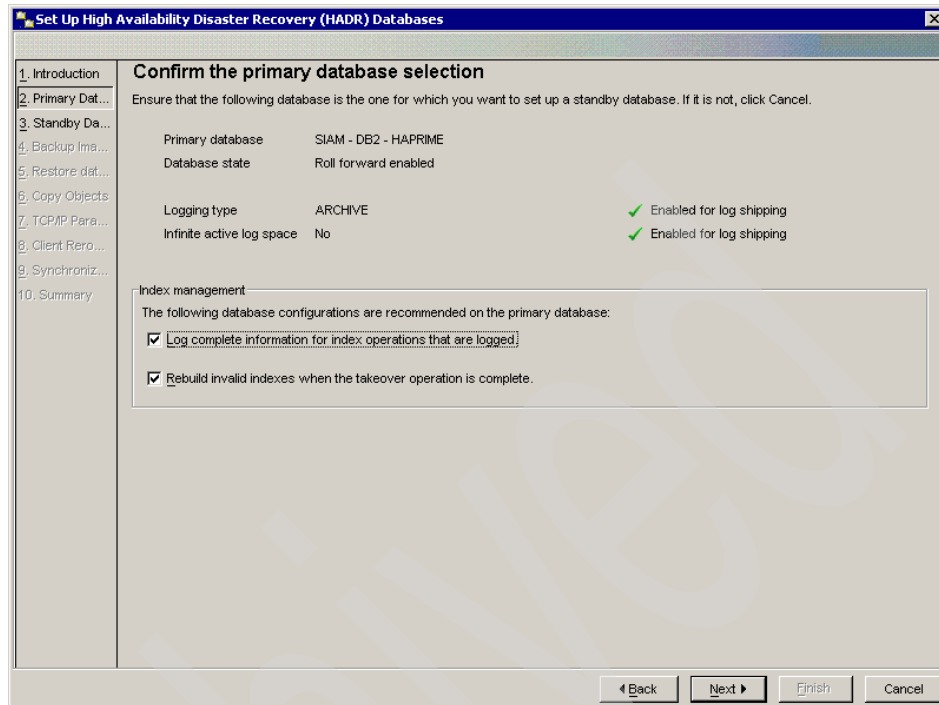


Figure 11-31 HADR Wizard: Confirm primary database selection

3. Identify a standby instance and database population method. In the example, shown in Figure 11-32, the standby instance is DB2HA. You can create and populate the standby database from a backup image of the primary, through another database that already exists, or using a split image. In our example, we will restore the standby database through a backup image. Click **Next**.

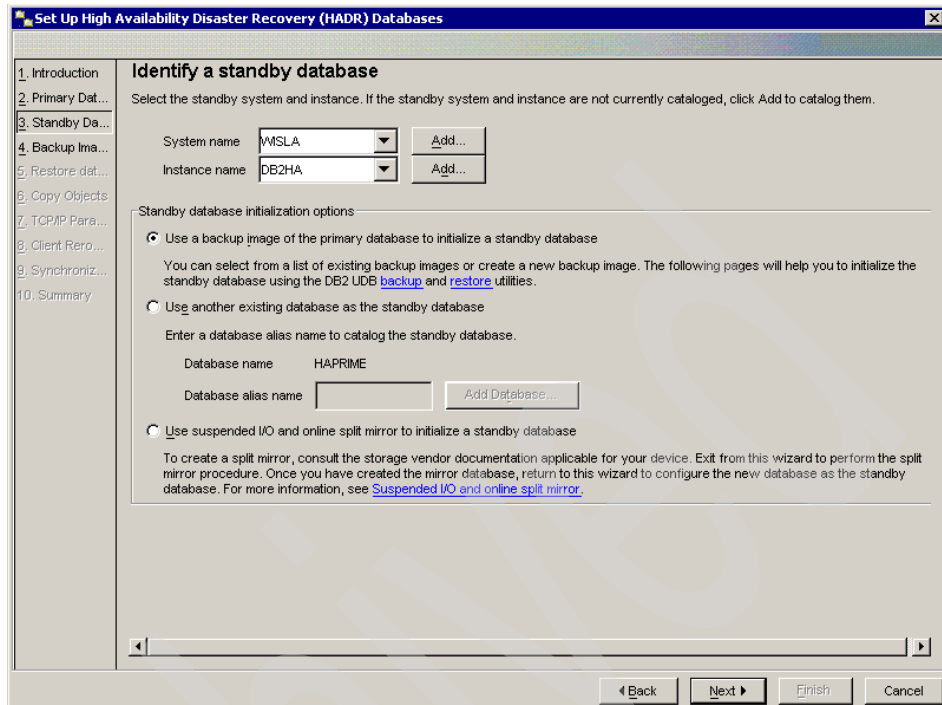


Figure 11-32 HADR Wizard: Select a standby database and initialization method

4. Identify the backup image to be used and select further backup options. The wizard will probe the history file for a list of valid backups. As shown in Figure 11-33, we select the backup taken at 2:34:54 p.m. on 2/04/05. Click **Next**.

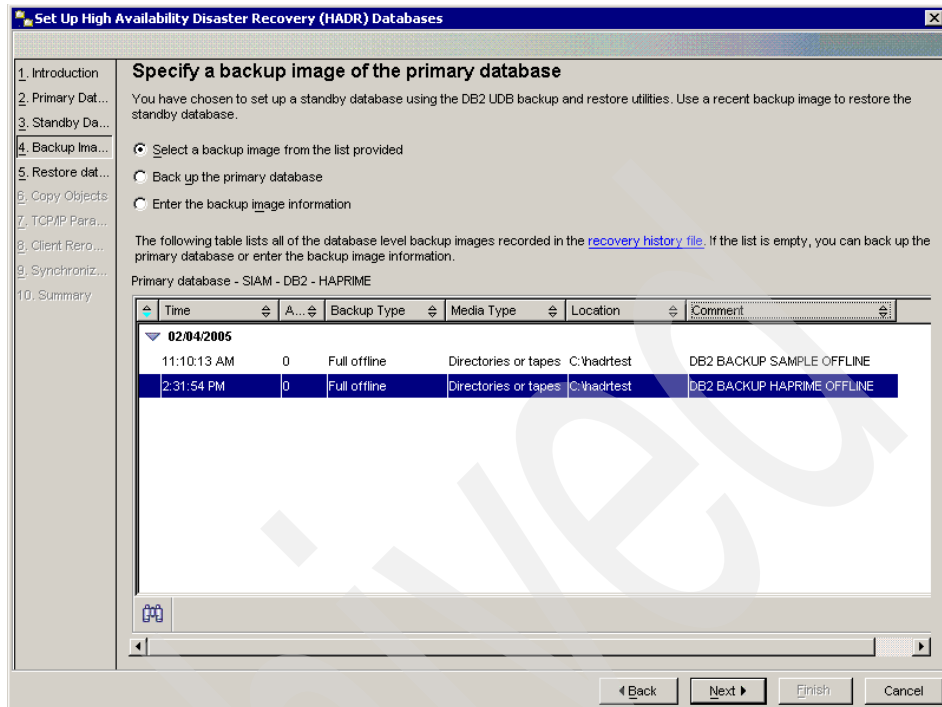


Figure 11-33 HADR Wizard: Identify a backup image to populate standby database

- Restore the database on the standby system. Select a database alias and how the backup image is copied to the standby system for restoration. The database on the standby system is also cataloged on the primary system. You will also notice in Figure 11-34 that the HADR wizard process can ship the primary database backup to the standby server in preparation for restoration. You can copy it manually. Once you have made your selections, click **Next**.

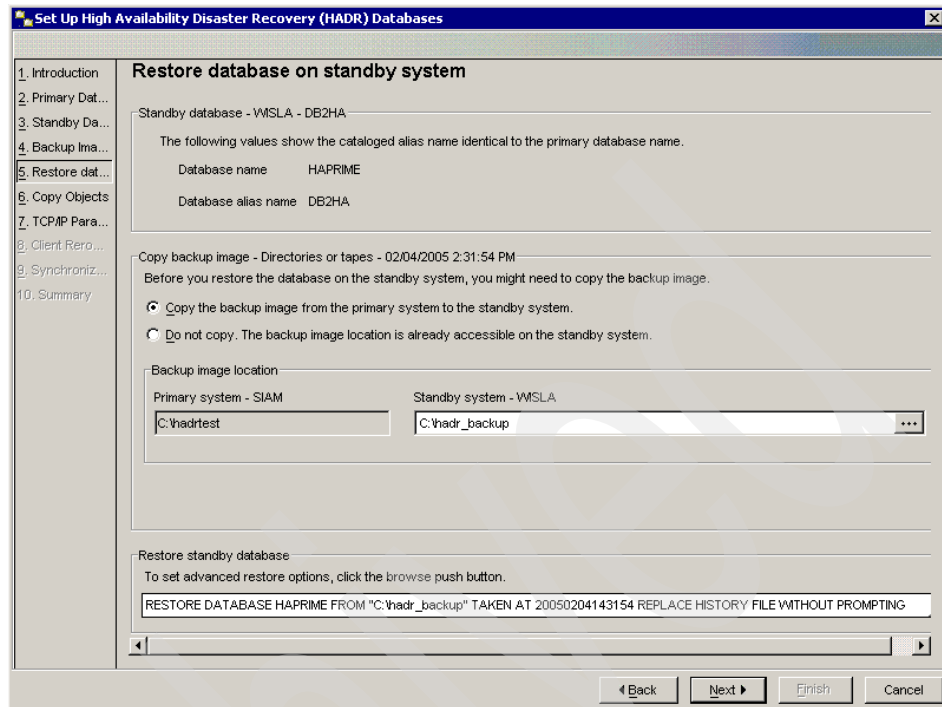


Figure 11-34 HADR Wizard: Restore the database on the standby system

- Optionally specify to copy objects that are stored externally (such as stored procedures and UDFs) and are not part of the database backup image. This step enables you to specify the objects to be copied from the primary system to the standby system (Figure 11-35). Click **Next**.

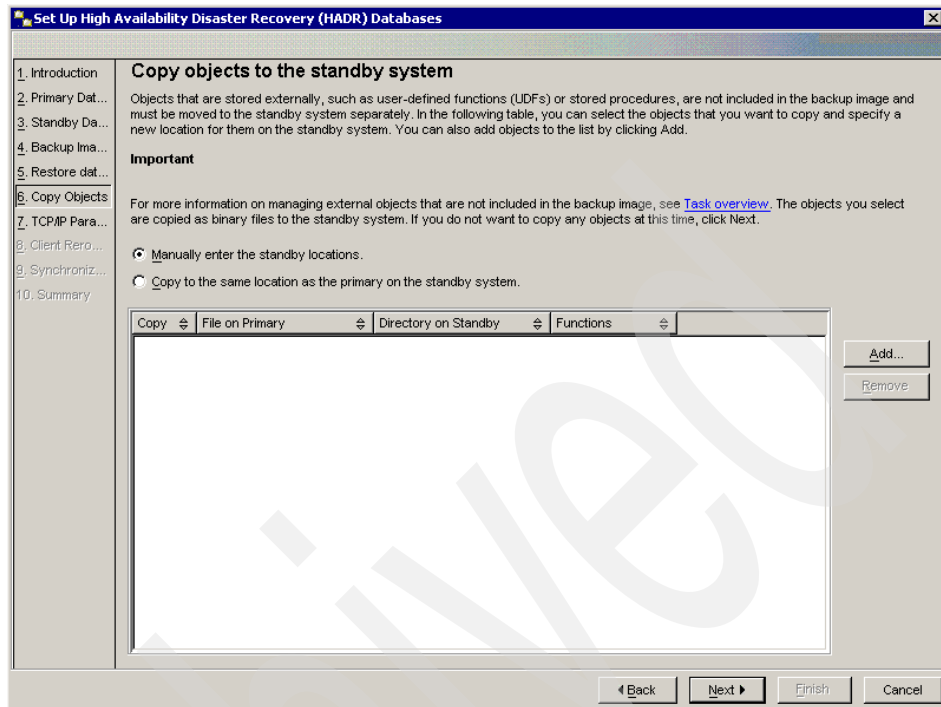


Figure 11-35 HADR Wizard: Copy objects to the standby system

7. Primary and standby systems communicate through TCP/IP. In this example we configure the TCP/IP communications parameters for the primary and standby. Figure 11-36 shows that you can specify the service name and port number for primary and standby systems. Click **View Service File** to view the service names and port numbers that are already defined in the operating system service file. Click **Next**.

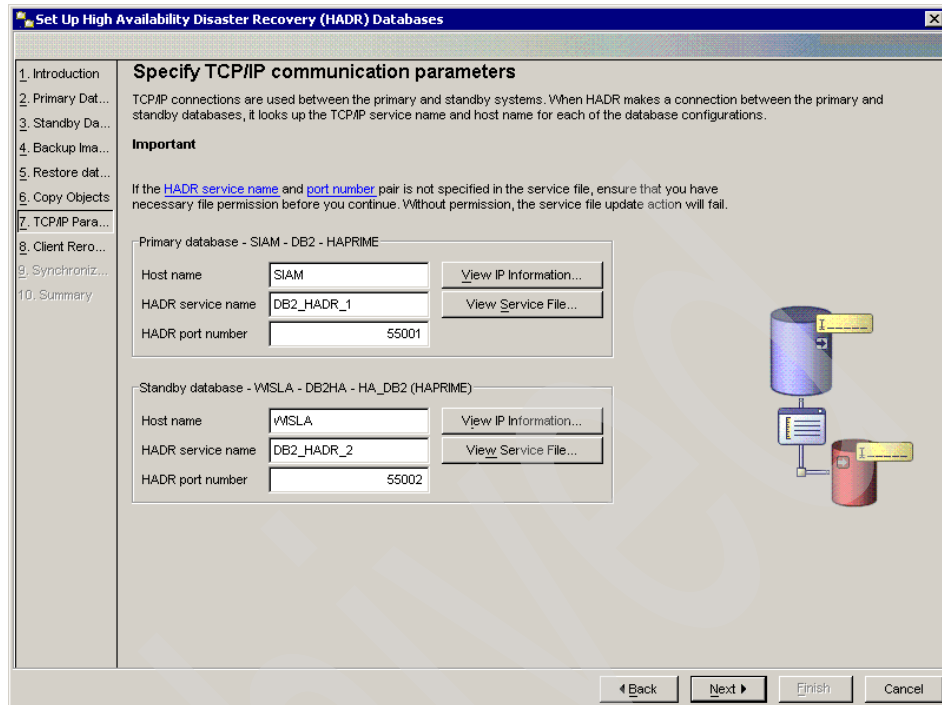


Figure 11-36 HADR Wizard: Configure Communications

8. DB2 UDB provides automatic client re-route so in the event that the primary database becomes unavailable, client connections are automatically re-routed to the standby. Figure 11-37, we have accepted the defaults for this option. Click **Next**.

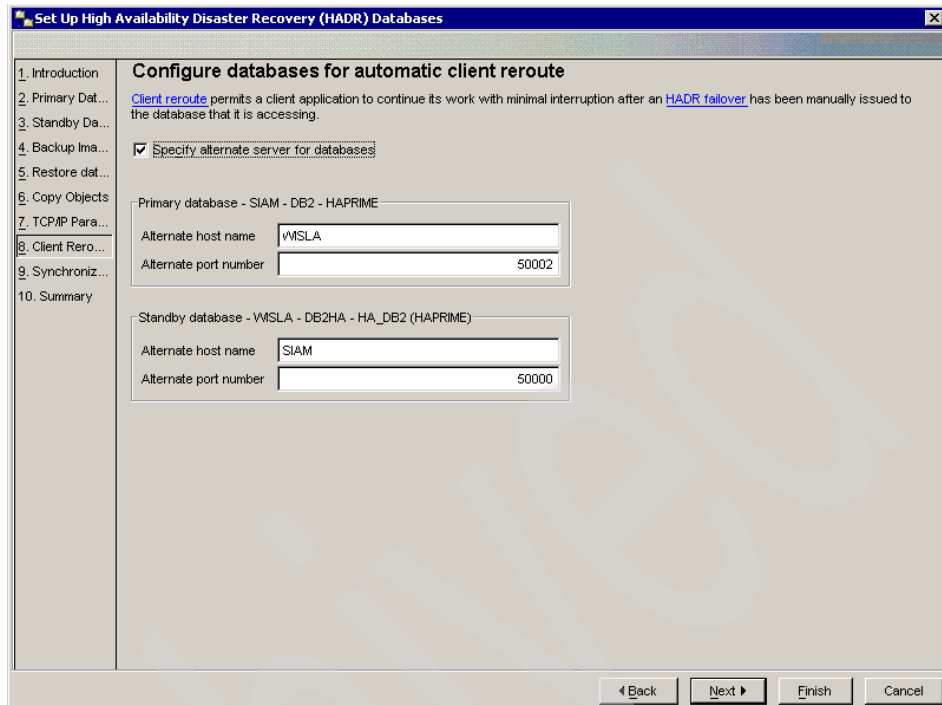


Figure 11-37 HADR Wizard: Configure automatic client re-route

9. Specify the synchronization mode for peer state log writing. The synchronization mode specifies the log write behavior between primary and standby when they are in peer state (when log pages are being shipped to the standby). There are three synchronization modes:
 - Synchronous (zero data loss): In this mode, DB2 UDB issues a commit acknowledgement to an application only when the transaction is written to disk on both primary and standby. Therefore, no transaction is lost, even if both primary and standby failed.
 - Near synchronous: In this mode, the log write at the primary and the send to the standby are performed in parallel. A commit operation succeeds when the log data is written to disk at the primary and it has been received by the standby. This is a no transaction loss in single site failure scenario.
 - Asynchronous: In this mode, a commit operation succeeds when the log data is written to disk at the primary and has been sent to the standby. The log data transmission is guaranteed by TCP/IP sockets. There is no acknowledgement of log data transmission between primary and standby.

In Figure 11-38, we have selected Near Synchronous which is also the default. Click **Next**.

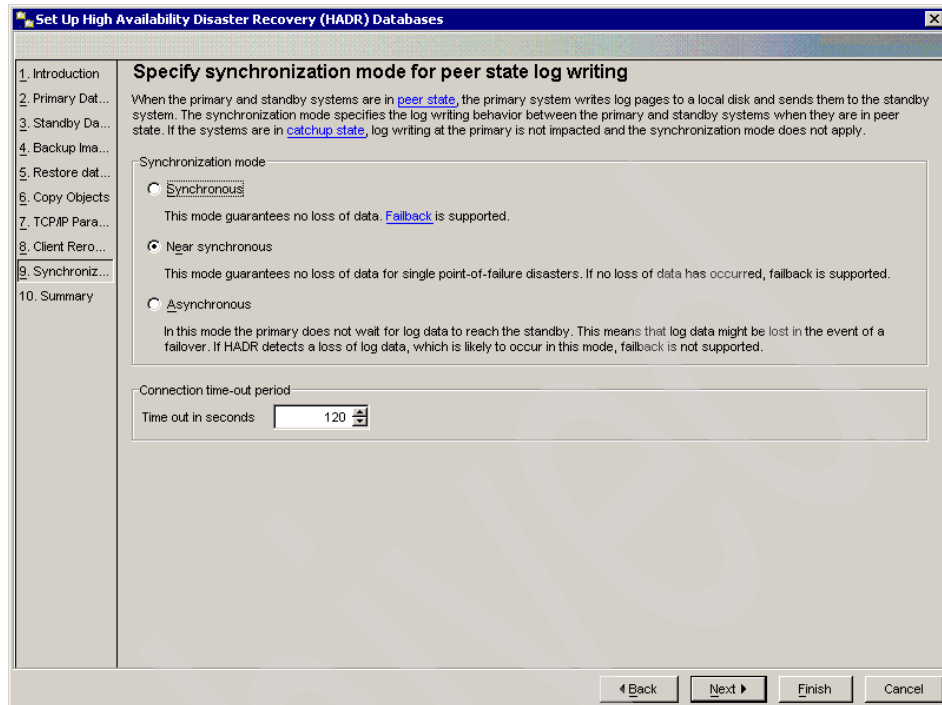


Figure 11-38 HADR Wizard: Selecting synchronization mode

10. A summary of the HADR setup information is presented. As you see in Figure 11-39, you have the option to setup HADR immediately or at a later time.

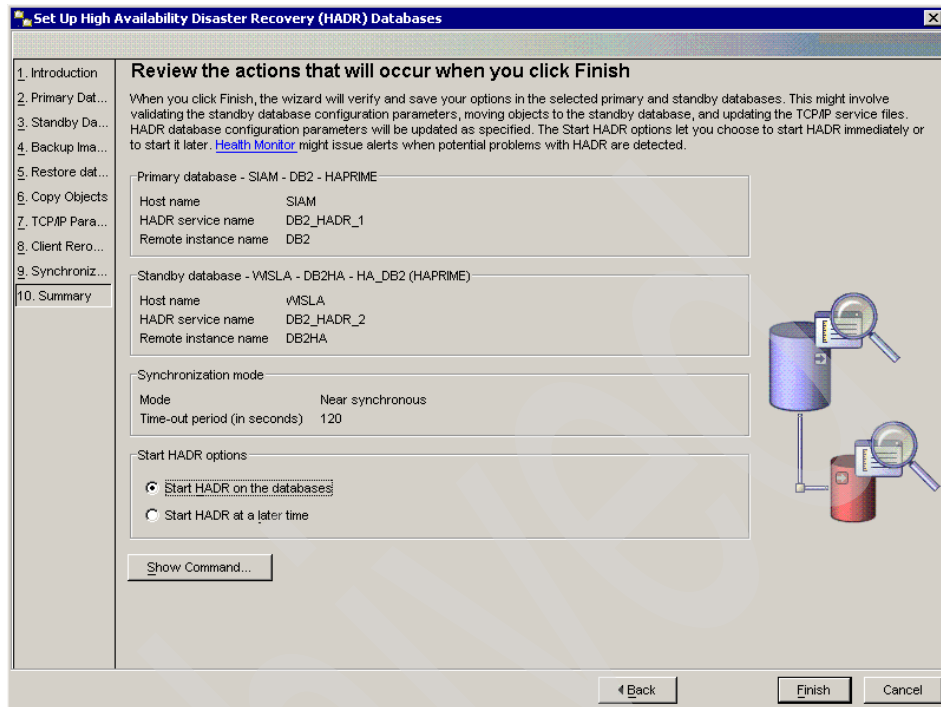


Figure 11-39 HADR Wizard: Summary of information

Selecting the **Show Command** button will show the script used to setup HADR (Example 11-7). This script can be invoked from the command line to setup HADR without the use of the GUI.

Example 11-7 HADR setup script

```
-- Update configuration parameters on primary database - SIAM - DB2 - HAPRIME
--
UPDATE DB CFG FOR DB HAPRIME USING LOGINDEXBUILD ON
UPDATE DB CFG FOR DB HAPRIME USING INDEXREC RESTART
--
-- Copy backup images from primary to standby system.
--
-- Location on primary system : C:\hadrtest
-- Location on standby system : C:\hadr_backup
--
-- Restore database on standby system - WISLA - DB2HA - HA_DB2 (HAPRIME)
--
RESTORE DATABASE HAPRIME FROM "C:\hadr_backup" TAKEN AT 20050204143154 REPLACE
HISTORY FILE WITHOUT PROMPTING
--
```

```

-- Configure databases for client reroute - SIAM - DB2 - HAPRIME
--
UPDATE ALTERNATE SERVER FOR DATABASE HAPRIME USING HOSTNAME WISLA PORT 50002
--
-- Configure databases for client reroute - WISLA - DB2HA - HA_DB2 (HAPRIME)
--
UPDATE ALTERNATE SERVER FOR DATABASE HAPRIME USING HOSTNAME SIAM PORT 50000
--
-- Update service file on primary system - SIAM
-- Service name : DB2_HADR_1
-- Port number : 55001
-- Service name : DB2_HADR_2
-- Port number : 55002
--
-- Update service file on standby system - WISLA
-- Service name : DB2_HADR_1
-- Port number : 55001
-- Service name : DB2_HADR_2
-- Port number : 55002
--
-- Update HADR configuration parameters on primary database - SIAM - DB2 -
HAPRIME
--
UPDATE DB CFG FOR HAPRIME USING HADR_LOCAL_HOST SIAM
UPDATE DB CFG FOR HAPRIME USING HADR_LOCAL_SVC DB2_HADR_1
UPDATE DB CFG FOR HAPRIME USING HADR_REMOTE_HOST WISLA
UPDATE DB CFG FOR HAPRIME USING HADR_REMOTE_SVC DB2_HADR_2
UPDATE DB CFG FOR HAPRIME USING HADR_REMOTE_INST DB2HA
UPDATE DB CFG FOR HAPRIME USING HADR_SYNCMODE NEARSYNC
UPDATE DB CFG FOR HAPRIME USING HADR_TIMEOUT 120
CONNECT TO HAPRIME
QUIESCE DATABASE IMMEDIATE FORCE CONNECTIONS
UNQUIESCE DATABASE
CONNECT RESET
--
-- Update HADR configuration parameters on standby database - WISLA - DB2HA -
HA_DB2 (HAPRIME)
--
UPDATE DB CFG FOR HAPRIME USING HADR_LOCAL_HOST WISLA
UPDATE DB CFG FOR HAPRIME USING HADR_LOCAL_SVC DB2_HADR_2
UPDATE DB CFG FOR HAPRIME USING HADR_REMOTE_HOST SIAM
UPDATE DB CFG FOR HAPRIME USING HADR_REMOTE_SVC DB2_HADR_1
UPDATE DB CFG FOR HAPRIME USING HADR_REMOTE_INST DB2
UPDATE DB CFG FOR HAPRIME USING HADR_SYNCMODE NEARSYNC
UPDATE DB CFG FOR HAPRIME USING HADR_TIMEOUT 120
--
-- Start HADR on standby database - WISLA - DB2HA - HA_DB2 (HAPRIME)
--
DEACTIVATE DATABASE HAPRIME

```

```

START HADR ON DATABASE HAPRIME AS STANDBY
--
-- Start HADR on primary database - SIAM - DB2 - HAPRIME
--
DEACTIVATE DATABASE HAPRIME
START HADR ON DATABASE HAPRIME AS PRIMARY
;

```

11. Select **Finish** from the Summary screen of the wizard. A progress window opens and indicates successful completion of the HADR setup steps.

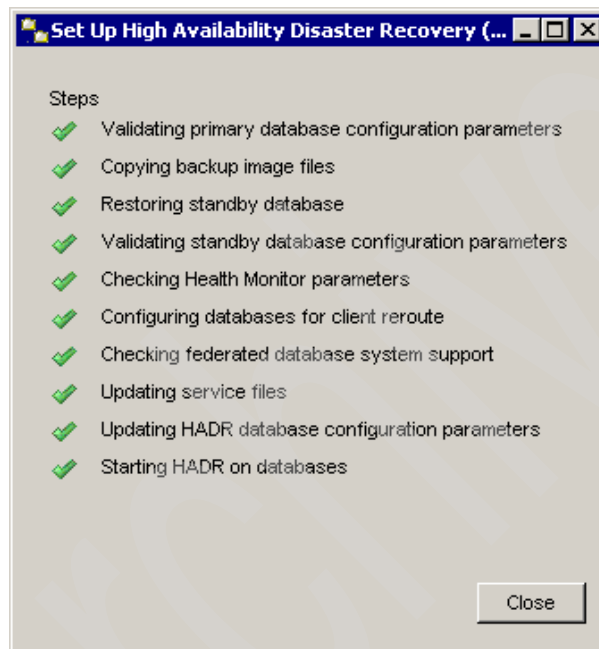


Figure 11-40 HADR Wizard: HADR setup confirmation dialog

After HADR is set up, you can monitor HADR by right-clicking the database from Control Center and selecting **High Availability Disaster Recovery- Manage**. In Figure 11-41, the primary and standby databases are enabled for HADR. You can also stop HADR or issue a takeover from this window. Issuing a takeover means that you switch roles between primary and standby databases.

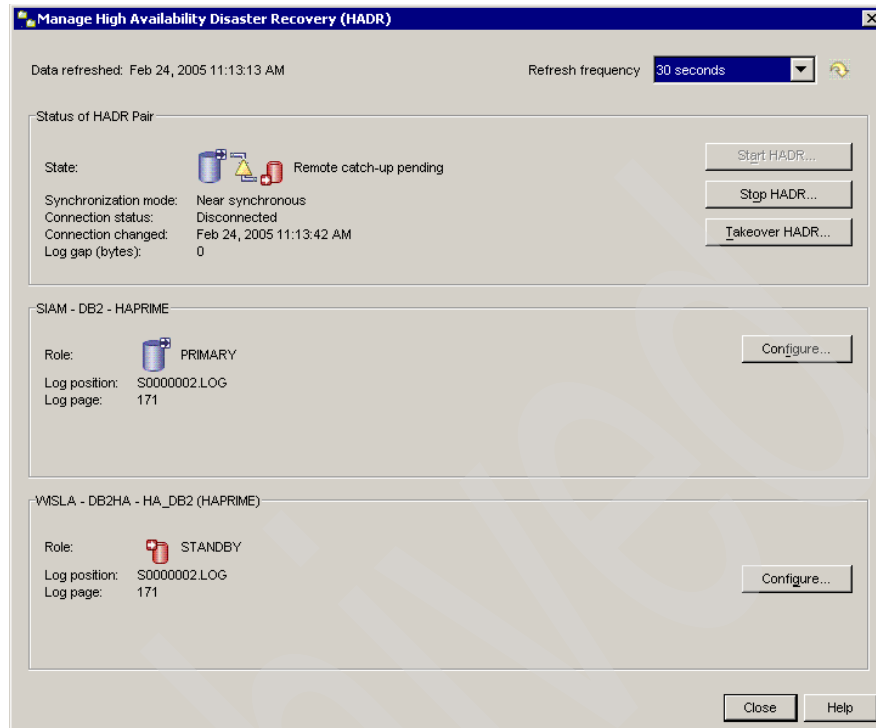


Figure 11-41 Manage HADR

Automatic client reroute

Uninterrupted service is the goal of high availability. The automatic client reroute feature enables client applications to recover from a loss of communication with the server and to continue working with minimal interruption. To enable DB2 UDB to reroute client applications, an alternate database location must first be specified at the server. This information then is passed to the client at database connection time. If the communication is lost, the DB2 client will use the information to route the application to the alternate database. Automatic client reroute is only supported using the TCP/IP protocol.

You can specify an alternate location using CLP by invoking an application programming interface (API), or when adding a database using Control Center or the advanced view of the Configuration Assistant. More information about the automatic client reroute feature can be found in the DB2 UDB documentation: *Data Recovery and High Availability Guide and Reference* SC09-4831.

Index logging

The index logging level is controlled by the database configuration parameter `LOGINDEXBUILD` and the `LOG INDEX BUILD` option in **CREATE TABLE** and **ALTER TABLE** commands. DB2 UDB performs different levels of logging when it creates, re-creates, or reorganizes indexes based on the setting of these parameters. `LOGINDEXBUILD` can be set to `ON` or `OFF`.

The index logging provides recoverability but can have performance trade-off. In the HADR environment we recommend that you set the `LOGINDEXBUILD` database configuration parameter to `ON` to ensure that complete information is logged for index creation, re-creation, and reorganization.

More information about Index Logging can be found in the DB2 UDB documentation *Data Recovery and High Availability Guide and Reference* SC09-4831.

11.7.3 Log mirroring

DB2 UDB supports log mirroring at the database level. Mirroring log files helps protect a database from accidental deletion of an active log and data corruption caused by hardware failure. The DB2 configuration parameter, `MIRRORLOGPATH`, specifies a secondary path for the database to manage copies of the active log, mirroring the volumes on which the logs are stored. The `MIRRORLOGPATH` configuration parameter allows the database to write an identical second copy of log files to a different path. We recommend that you place the secondary log path on a physically separate disk (preferably one that is also on a different disk controller). That way, the disk controller cannot be a single point of failure.

When `MIRRORLOGPATH` is first enabled, it is not actually used until the next database startup. This is similar to the `NEWLOGPATH` configuration parameter. If there is an error writing to either the active log path or the mirror log path, the database will mark the failing path as bad, write a message to the administration notification log, and write subsequent log records to the remaining good log path only. DB2 UDB does not attempt to use the bad path again until the current log file is completed. When DB2 UDB needs to open the next log file, it verifies that this path is valid, and if so, begins to use it. If not, DB2 UDB does not attempt to use the path again until the next log file is accessed for the first time. There is no attempt to synchronize the log paths, but DB2 UDB retains information about access errors so that the correct paths are used when log files are archived. If a failure occurs while writing to the remaining good path, the database shuts down.

11.7.4 Replication

DB2 UDB supports both SQL replication and queue replication to provide high availability. These functions maintain logically consistent copies of database tables at multiple locations. In addition, they provide flexibility and complex functionality, such as support for column and row filtering, data transformation, and updates to any copy of a table, and they can be used in partitioned database environments. DB2 UDB offers two choices for replication, using SQL and using queues. The queue version of replication is targeted more to the needs of high availability. DB2 UDB embeds WebSphere MQ technology for guaranteed message delivery.

For more information, see the documentation *WebSphere Information Integrator Replication and Event Publishing Guide and Reference*, SC18-7568.

Replication versus HADR

Figure 11-42 shows a comparison of HADR and queue-based replication. Each method offers advantages, so choose the method that best fits your requirements. If you require zero loss replication, HADR is the best solution. However, if you only require read or read-write access to the secondary, queue replication is the better solution.

Functionality	HADR	Q-Replication
Synchronous "zero-loss" replication*	Yes	No
Asynchronous replication	Yes	Yes
Replicate entire database*	Yes	No ⁽¹⁾
Replicate a subset of database tables	No	Yes
Replicate a subset of data	No	Yes
Cross-OS support	No	Yes
Two-way replication	No	Yes
Multiple targets (standby)	No	Yes
Read/Write on targets (standby)	No	Yes
Automatic Client Reroute	Yes	Yes
Monitoring	Yes	Yes
Requires staging tables	No	No
Requires installation and set-up of MQ	No	Yes
Network compression and encryption	No	Yes
DPF Support	No	Yes

¹While Q-Replication can replicate all the data, each table needs to be replicated individually and other database objects (like table spaces) are not kept in sync. Also, DDL is not replicated.

*Required for "true high availability"

Figure 11-42 HADR versus queue replication

11.7.5 Online split mirror and suspended I/O support

You can create a split mirror image of your database as a snapshot, standby, or backup image. Typically, this is achieved with the use of disk hardware or third-party packages that create a split mirror or snapshot of the database.

A split mirror is an 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. Disk mirroring is the process of writing all of the data to two separate hard disks, one being a mirror copy of the other. Splitting a mirror is the process of separating the primary and secondary copies of the database.

If you would rather not back up a large database using the DB2 UD *backup* utility, you can make copies from a mirrored image by using suspended I/O and the split mirror function. This approach also eliminates backup operation overhead from the production machine and is a fast way to clone systems. Use the **db2inidb** command in conjunction with the **suspend** and **resume** commands to do this.

The **SNAPSHOT** option specifies that the mirrored database will be initialized as a clone of the primary database. You can then use this clone for testing or as a point-in-time failover environment. When you use the **STANDBY** option, the database will be placed in rollforward pending state. New logs from the primary database can be fetched and applied to the standby database. The standby database can then be used in place of the primary database if it goes down. The **MIRROR** option specifies that the mirrored database is to be used as a backup image which can be used to restore the primary database.

To clone a database using split mirror, perform the following steps in CLPs:

1. Suspend I/O on the primary database:
db2 set write suspend for database
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.
3. Resume I/O on the primary database:
db2 set write resume for database
4. Catalog the mirrored database on the secondary system.

Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the **db2relocatedb** utility or the **RELOCATE USING** option of the **db2inidb** command to accomplish this.

5. Start the database instance on the secondary system:

```
db2start
```

6. Initialize the mirrored database on the secondary system:

```
db2inidb database_alias as snapshot
```

11.8 REORG and RUNSTATS

In this section we describe the considerable performance benefit gained from regular use of REORG and RUNSTATS.

11.8.1 Database reorganization

After many changes to table data, logically sequential data might reside on non-sequential physical data pages, meaning that the database manager must perform additional read operations to access data. In such a case, you might consider reorganizing the table and to reclaim space by using of the **REORG** command. You can reorganize the system catalog tables and the database tables. REORG is similar to shrinking a database in SQL Server.

Consider the following factors, which might indicate that table reorganization is needed:

- ▶ A high volume of insert, update, and delete activity on tables accessed by queries.
- ▶ Significant changes in the performance of queries that use an index with a high cluster ratio.
- ▶ Executing **runstats** to refresh statistical information does not improve performance.
- ▶ The **reorgchk** command indicates a need to reorganize tables.
- ▶ The trade-off between the cost of increasing degradation of query performance and the cost of reorganizing your table, which includes the CPU time, the elapsed time, and the reduced concurrency resulting from the **reorg** utility locking the table until the reorganization is complete.

To reduce the need for reorganizing a table, perform these tasks after you create the table:

- ▶ Alter the table and specify a value for the PCTFREE parameter.
- ▶ Create clustering index with PCTFREE on the index.
- ▶ Sort the data.
- ▶ Load the data.

A table with a clustering index and the setting of PCTFREE on the table helps preserve the original sorted order. If enough space is allowed in table pages, new data can be inserted on the correct pages to maintain the clustering characteristics of the index. As more data is inserted, and the pages of the table become full, records are appended to the end of the table so that the table gradually becomes unclustered.

Similarly, as tables are updated, index performance degrades in the following ways:

- ▶ Fragmentation of leaf pages increases I/O costs, because more leaf pages must be read to fetch table pages.
- ▶ The physical index page order no longer matches the sequence of keys on those pages, which is referred to as a badly clustered index.
- ▶ When leaf pages are badly clustered, sequential prefetching is inefficient and results in more I/O waits.
- ▶ The index develops more than its maximally efficient number of levels.

If you the MINPCTUSED parameter is set during index creation, the database server automatically merges index leaf pages if a key is deleted and the free space is less than the specified percentage. This process is called *online index defragmentation*. To restore index clustering, free space, and reduce leaf levels, you can use one of the following methods:

- ▶ Drop and recreate the index.
- ▶ Use the **reorg indexes** command to reorganize indexes online. You might choose this method in a production environment because it allows users to read from and write to the table while its indexes are being rebuilt.
- ▶ Use the **reorg table** command with options that allow you to reorganize both the table and its indexes offline.

Tip: Creating multidimensional clustering (MDC) tables might reduce the need to reorganize tables. For MDC tables, clustering is maintained on the columns provided arguments to the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. However, **reorgchk** might recommend reorganization of an MDC table if it detects that there are too many unused blocks or that blocks should be compacted.

Assessing the need for reorganization

The **reorgchk** command returns statistical information about data organization and can advise you about whether particular tables need to be reorganized. Running specific queries against the catalog statistics tables at regular intervals or specific times can also provide a performance history that enables you to spot

trends that might have wider implications for performance. The **reorgchk** command can also be used to update table and index statistics in the catalogs. Example 11-8 shows the output of a **reorgchk** on all tables (*table all* option). Note that the output has been truncated. Asterisks (*) in the output of the REORG column indicate that reorganization is needed, based on three separate formulas, which are labelled F1, F2, and F3. You will see in the example that SYSIBM.SYSINDEXES has asterisks for F1 and F3, thus a reorganization is needed (highlighted with bold typeface).

Example 11-8 reorgchk sample

```
C:\db2 reorgchk on table all
Doing RUNSTATS ....
```

Table statistics:

F1: 100 * OVERFLOW / CARD < 5

F2: 100 * (Effective Space Utilization of Data Pages) > 70

F3: 100 * (Required Pages / Total Pages) > 80

SCHEMA	NAME	CARD	OV	NP	FP	ACTBLK	TSIZE	F1	F2

Table: GB071538.ACCT									
GB071538	ACCT	1000000	0	27792	27792	-	1.10e+008	0	98
100	---								
Table: GB071538.BRANCH									
GB071538	BRANCH	100	0	4	4	-	11000	0	91
100	---								
Table: GB071538.HISTORY									
GB071538	HISTORY	-	-	-	-	-	-	-	-
-	---								
Table: GB071538.TELLER									
GB071538	TELLER	1000	0	29	29	-	110000	0	97
100	---								
Table: SYSIBM.SYSFUNCMAPPINGS									
SYSIBM	SYSFUNCMAPPINGS	-	-	-	-	-	-	-	-
-	---								
Table: SYSIBM.SYSHIERARCHIES									
SYSIBM	SYSHIERARCHIES	-	-	-	-	-	-	-	-
-	---								
Table: SYSIBM.SYSINDEXAUTH									
SYSIBM	SYSINDEXAUTH	26	0	1	1	-	1742	0	-
100	---								
Table: SYSIBM.SYSINDEXCOLUSE									
SYSIBM	SYSINDEXCOLUSE	556	0	7	7	-	26688	0	100
100	---								
Table: SYSIBM.SYSINDEXES									

SYSIBM	SYSINDEXES	208	102	18	35	-	179088	49	100
51	*-*								

CLUSTERRATIO or normalized CLUSTERFACTOR (F4) will indicate REORG is necessary for indexes that are not in the same sequence as the base table. When multiple indexes are defined on a table, one or more indexes may be flagged as needing REORG. Specify the most important index for REORG sequencing.

Tables defined using the ORGANIZE BY clause and the corresponding dimension indexes have a '*' suffix to their names. The cardinality of a dimension index is equal to the Active blocks statistic of the table.

For more information about reorganization of indexes, refer to DB2 UDB documentation: *Administration Guide: Performance*, SC09-4821.

Table reorganization

The **reorg table** command is used to reorganize a table. It can also reorganize an index at the same time.

Figure 11-43 shows an example of reorganizing the STATE table in the sample database. This example illustrates and outlines table reorganization, meaning that users have access to the table while the utility is running. DB2 UDB does this by reorganizing data in place, one block at a time. To access the **reorg** utility in Control Center, right-click the desired table, and select **Reorganize Table**. Note that in the following example, we reorganize the table's indexes at the same time.

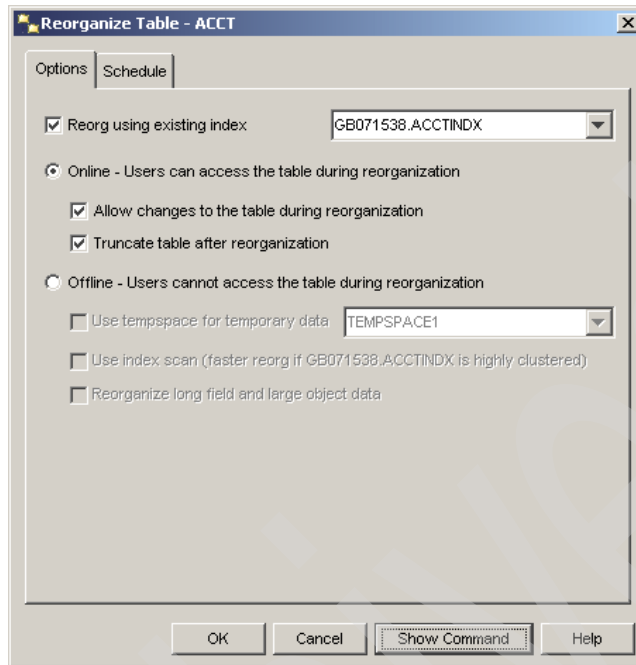


Figure 11-43 Reorganizing a table in Control Center

Example 11-9 shows the command used to run **reorg** from CLP.

Example 11-9 Table reorganization command

```
REORG TABLE GB071538.ACCT INDEX GB071538.ACCTINDX INPLACE ALLOW WRITE ACCESS
START ;
```

Index reorganization

DB2 UDB V8.2, now has the ability to read and update a table and its existing indexes during an index reorganization. For more information about index reorganization, see 12.5.1, “Indexing” on page 364.

Automatic reorganization

You can also configure automatic reorganization using the Automatic Maintenance wizard from Control Center or Health Center. We take a closer look at Automatic Maintenance in 12.3, “Configuring automatic maintenance” on page 346.

11.8.2 Database statistics

Both SQL Server and DB2 use a *cost-based optimizer*. That means statistical information, such as the number of rows in a table or the structure of an index, is used by the optimizer to calculate the cost of a query plan. It is up to the database administrator to keep these statistics as current as possible. Without the correct statistical information, the optimizer is not able to calculate an optimal query plan, which can result in degraded performance.

The DB2 UDB command for generating and updating statistics is **runstats**. By contrast SQL Server has two commands, **create statistics** and **update statistics** and a system stored procedure **sp_createstats**. Like the **update statistics** command used in SQL Server, you should run **runstats** periodically. The right moment to execute a **runstats** depends on the dynamic nature of the data. If the content of a table is changing constantly, you should update the statistics more often. If table content is static, a single **runstats** is usually sufficient.

The **runstats** command can collect table, column, and index information. This is reflected in the command syntax shown in Example 11-10.

Example 11-10 RUNSTATS syntax

```
db2inst2:/home/db2inst2> db2 ? runstats
RUNSTATS ON TABLE table-name [USE PROFILE | statistics-options]

statistics-options:
[table-object-options] [ALLOW {WRITE | READ} ACCESS]
[table-sampling-options] [profile-options] [UTIL_IMPACT_PRIORITY [priority]]

table-object-options:
[{FOR index-clause | [column-stats-clause] [AND index-clause]]}

table-sampling-options:
[TABLESAMPLE {BERNOULLI | SYSTEM} (numeric-literal)
[REPEATABLE (integer-literal)]]

profile-options:
[{SET PROFILE [NONE | ONLY] | UPDATE PROFILE [ONLY]]}

index-clause:
[[SAMPLED] DETAILED] {INDEXES | INDEX} {ALL | index-name [{,index-name}...]}

columns-stats-clause:
[ON {{ALL | KEY} COLUMNS [AND COLUMNS (column-option [{,column-option}...])]} |
COLUMNS (column-option [{,column-option}...])}] [distribution-clause]

column-option:
```

```
{column-name [LIKE STATISTICS] | (column-name [{,column-name}...])}

distribution-clause:
WITH DISTRIBUTION [ON {{ALL | KEY} COLUMNS [AND COLUMNS (dist-column
[{,dist-column}...])} | COLUMNS (dist-column [{,dist-column}...]) }}
[DEFAULT [NUM_FREQVALUES number] [NUM_QUANTILES number]]

dist-column:
{column-name | (column-name [{,column-name}...])} [NUM_FREQVALUES number]
[NUM_QUANTILES number]
```

The online help of the **runstats** command demonstrates that there are a lot more options you can use compared to SQL Server's **update statistics** command. We do not cover all of these in this book. Instead, we compare a few of the more frequently used options. Table 11-2 provides a basic comparison of **update statistics** and **runstats** commands.

Table 11-2 Basic statistics command - compare between SQL Server, DB2 UDB

SQL Server	DB2 UDB
UPDATE STATISTICS <i>tablename</i> WITH FULLSCAN, ALL	RUNSTATS ON TABLE <i>schema.tabname</i> AND INDEXES ALL
UPDATE STATISTICS <i>tablename</i> WITH FULLSCAN, COLUMNS Then run: UPDATE STATISTICS <i>tablename</i> WITH FULLSCAN, INDEX (Before these commands are ran you would need to run CREATE STATISTICS to generate statistics for the columns	RUNSTATS ON TABLE <i>schema.tabname</i> WITH DISTRIBUTION ON ALL COLUMNS AND DETAILED INDEXES ALL

You can also use Control Center to update statistics. Right-click the desired table and select **Run Statistics**. As shown in Figure 11-44, you can adjust all the statistics settings.

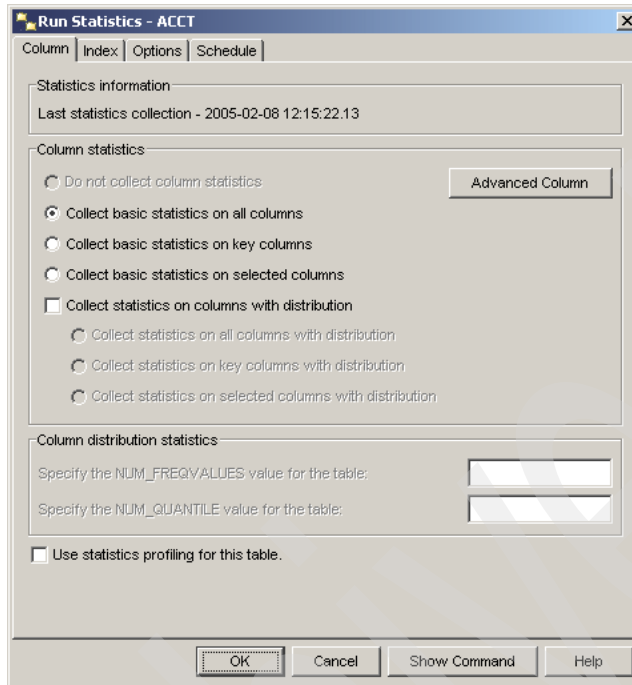


Figure 11-44 Executing RUNSTATS using Control Center

Example 11-11 shows a sample query meant to demonstrate the impact of statistics collection.

Example 11-11 Query used in RUNSTATS demonstration

```

WITH
  REACH (SOURCE, DESTINATION, COST, STOPS) AS
    ( SELECT SOURCE, DESTINATION, COST, CAST( 0 AS SMALLINT )
      FROM FLIGHTS
      WHERE SOURCE = 'Paris'
    UNION ALL
      SELECT R.SOURCE, F.DESTINATION, CAST(R.COST+F.COST AS SMALLINT ),
        CAST(R.STOPS+ 1 AS SMALLINT )
      FROM REACH R, FLIGHTS F
      WHERE R.DESTINATION=F.SOURCE
        AND R.STOPS < 5
    )
SELECT DESTINATION, COST, STOPS FROM REACH;

```

The query plans (shown in Figure 11-45) are taken before and after a **runstats** run. The query plans are generated with the Visual Explain tool (see 3.1.4,

“Visual Explain” on page 35). Notice in the following example how the total cost of the query decreased from 42.61 to 28.72 timerons after **runstats** updated the statistics, although the access path to the data remains the same.

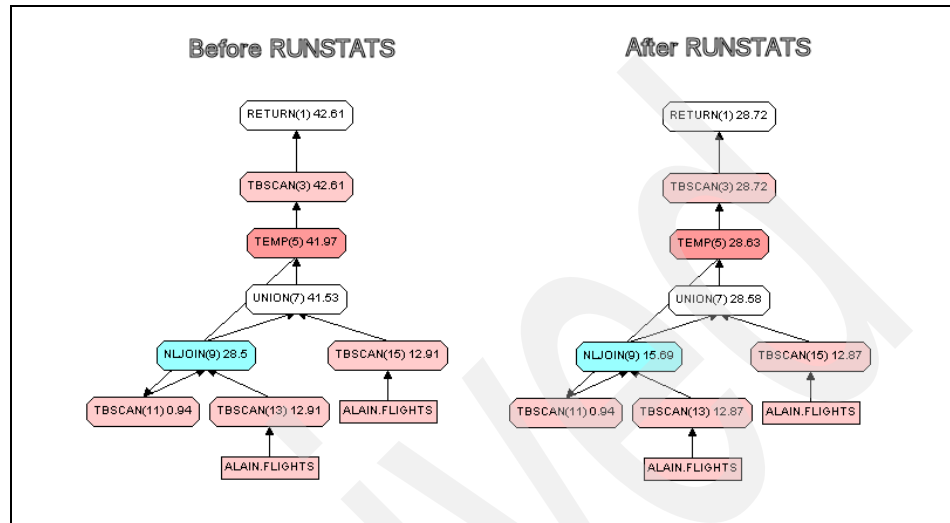


Figure 11-45 Query plans before and after a **RUNSTATS** run

We take a closer look at Visual Explain in 12.7.1, “Optimizer analysis” on page 383.

Automatic database statistics

Tables considered for automatic statistics collection can be configured the Automatic Maintenance wizard from the Control Center or Health Center. We take a closer look at Automatic Maintenance in.

Post-conversion tuning considerations

Once the existing database and application have both been successfully converted to DB2 UDB, you should begin to consider how the system can be tuned to achieve maximum performance in the DB2 UDB environment. While full coverage of performance tuning is beyond the scope of this book, we focus on several areas that will immediately begin to reap the benefits of tuning.

DB2 UDB should never be deployed in a production environment using the default factory settings. These settings likely do not take into account the characteristics of your specific environment, and therefore the application and DB2 UDB may not perform optimally and be utilized to their capacity. Fortunately, DB2 UDB ships with a collection of GUI tools and wizards that greatly simplify the process of performance tuning.

In this chapter, we highlight the important areas where initial performance tuning efforts should be directed. Some of the GUI tools that eliminate the guesswork involved in performance tuning are presented. We also describe some of the advanced performance-oriented access methods available in DB2 UDB that your environment may benefit from.

12.1 Performance tuning

Tuning a server instance and a database should ideally begin as part of the installation and development before the data is laid out on disk and before the very first lines of code are written. However, this part of the development process is unfortunately too often omitted, or left until after the system is operational and problems have already surfaced. The following discussion of performance and tuning is to give you a basic introduction to performance tuning in DB2 UDB. We also highly recommend reading the IBM Redbook *DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432. This book offers a thorough review of performance tuning in DB2 UDB.

Regardless of the database vendor, the discipline of performance tuning is the same. It begins with defining specific and measurable performance objectives, measuring them, and then making adjustments. The emphasis in this chapter is on DB2 UDB features that you will use throughout the life of the server and for which planning is needed. The following are the key areas that can be controlled during a tuning effort:

- ▶ *Application design*: Moving an existing design from one environment to another without adapting it is generally not recommended.
- ▶ *Database design*: Using the features of the database software to implement a logical design (tables, columns) and physical design (and thus placement of tables and indexes).
- ▶ *Server architecture*: Using the features and parameters of your server to the best effect.

There are three basic areas in DB2 UDB that can be tuned: the instance, the database, and the SQL queries being submitted. Performance tuning for the instance is based on optimizing operating system resource usage. Tuning the database is somewhat based on data layout, table schema, and indexes. Each database can be tuned to use a specific part of the resources allocated to the instance. Tuning SQL queries involve writing SQL queries with performance in mind and taking advantage of advanced database objects that may be used to improve performance.

12.2 Quick-start tips for performance tuning

After installing DB2 UDB, the following steps can be taken to begin an initial tuning of the DB2 UDB environment:

- ▶ Use the Configuration Advisor to recommend reasonable beginning default values for system configuration parameters. The Configuration Advisor recommendations are based on your unique hardware environment. Gather

information about the hardware at your site so that you can answer the wizard questions. You can apply the recommended configuration parameter settings immediately or let the wizard create a script based on your answers and run the script later.

- Use the Design Advisor to find out what indexes, materialized query tables, multidimensional clustering tables, and database partitions may improve query performance.

12.2.1 Design Advisor

The DB2 Design Advisor is a tool that can help you significantly improve your workload performance. The task of selecting which indexes, materialized query tables (MQTs), clustering dimensions, or partitions to create for a complex workload can be quite daunting. The Design Advisor identifies all of the objects needed to improve the performance of your workload. Given a set of SQL statements in a workload, the Design Advisor will generate recommendations for indexes, MQTs, conversion to multidimensional clustering (MDC) tables, and deletion of indexes and MQTs unused by the specified workload. You can have the Design Advisor implement some or all of these recommendations immediately or schedule them for a later time.

Using either the Design Advisor GUI or the command line tool, the Design Advisor can help simplify database design and workload performance tuning. While designing your database, use the Design Advisor to generate design alternatives in a test environment. You can also use it to evaluate indexes, MQTs, MDC tables, or partitioning strategies that have been generated manually.

After your database is set up, use the Design Advisor to improve the performance of a particular statement or workload. You can also improve general database performance, using the performance of a sample workload as a gauge. When creating a sample workload, we suggest that you create a workload based on the most frequently executed queries, which you can identify by using the Activity Monitor.

Launching the Design Advisor in a GUI environment

To access the Design Advisor, select the desired database from the object hierarchy in Control Center, right-click, and select **Design Advisor**. The advisor 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 current catalog statistics. There is an option to collect the required basic statistics in the tool, however, this increases the total calculation time. Figure 12-1 presents a sample Design Advisor window.

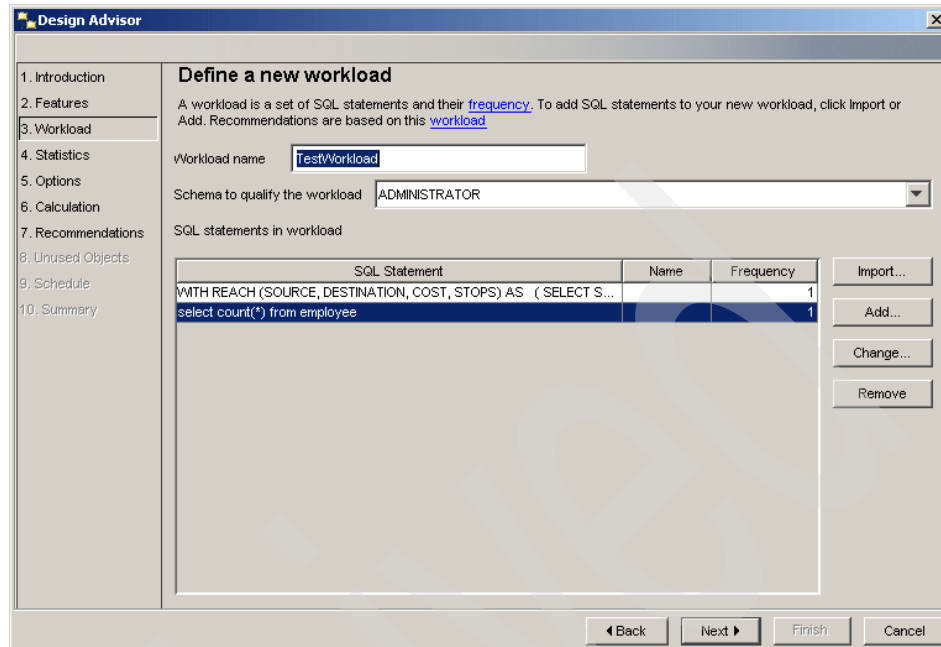


Figure 12-1 Design Advisor

The detailed information about the Design Advisor can be found in the following IBM 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

12.2.2 Configuration Advisor

The **Configuration Advisor** is a GUI tool that can be helpful in tuning an initial DB2 UDB configuration. The wizard requests information about the database, its data, and the environment, and then recommends new configuration parameter values.

To invoke this wizard from Control Center, expand the object tree, right-click the target database and select **Configuration Advisor**. The wizard collects information including the percentage of memory dedicated to DB2 UDB, the 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 immediately or to save them as a task in

the Task Center for later execution, (shown in Figure 12-2). The Configuration Advisor result window is shown in Figure 12-3.

Configuration Advisor

1. Introduction
2. Server
3. Workload
4. Transactions
5. Priority
6. Populated
7. Connections
8. Isolation
9. Schedule
10. Results

Scheduling task execution...

You can select whether to execute the commands immediately or create a task in the Task Center. Creating a task allows you to schedule task execution and maintain its history.

☐ Run now without saving task history

☒ Create this as a task in the Task Center

Run System: WIN2KADVSERVER

Scheduler System: WIN2KADVSERVER Advanced...

Task name: Configuration Advisor - 3/10/05

☐ Save task only

☐ Save and run task now

☒ Schedule task execution

Details:
Run once, on 3/10/05 at 11:00:00 PM. Change...

Runtime authorization

User ID: db2admin

Password: *****

Back Next Finish Cancel

Figure 12-2 Scheduling Configuration Advisor recommendations

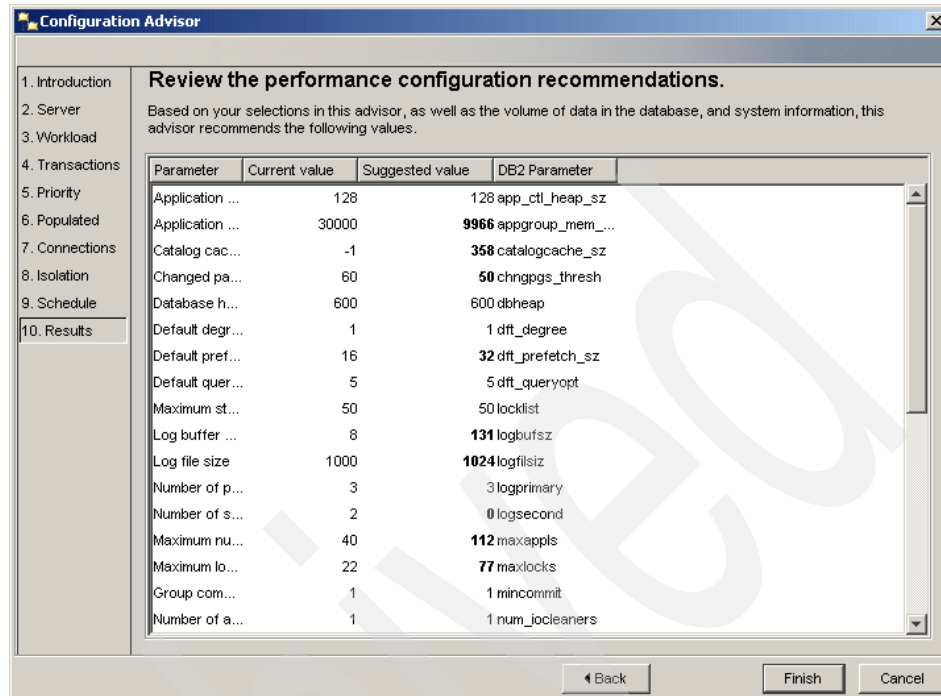


Figure 12-3 Configuration Advisor recommendations

Initial configuration recommendations can also be acquired using the **AUTOCONFIGURE** command in Command Line Processor (CLP). Example 12-1 shows a sample execution of this command.

Example 12-1 Sample AUTOCONFIGURE command in CLP

```
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 12-1 lists the **AUTOCONFIGURE** command parameters.

Table 12-1 Parameters for the AUTOCONFIGURE command

Parameter	Possible Values	Explanation
mem_percent	1-100 default: 80	Percentage of memory to dedicate to DB2 UDB
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?

12.2.3 ACTIVATE DATABASE command

Use the **ACTIVATE DATABASE** command to start databases and allocate the memory required immediately. In a partitioned database environment, this command activates the database on all partitions and avoids the startup time required to initialize the database when the first application connects.

Note: If you use the **ACTIVATE DATABASE** command, you must shut down the database with the **DEACTIVATE DATABASE** command. The last application that disconnects from the database does not shut it down.

12.2.4 RUNSTATS and REORG

Database statistics and data organization have a critical impact on database performance. Statistics are used by the optimizer to make access path decisions for data retrieval. The **runstats** utility is used to update these statistics and should be run on a regular basis. **runstats** is one of the most important utilities for maintaining optimal system performance and should be one of the first things to run if performance suddenly degrades. You can also have DB2 UDB perform **runstats** automatically if automatic maintenance has been configured (see the following section for configuring automatic maintenance).

Proper data organization is also critical for good performance. Data rows and/or indexes that are physically out of order usually results in additional I/O and thus additional response time. The **reorg** utility is used to reorganize table and index data and should be run whenever a large amount of table data has been modified. It is also possible to enable automatic maintenance and have the system run **reorg** when necessary (see the following section for configuring automatic maintenance).

For more information about the **runstats** and **reorg** utilities, see 11.8, “REORG and RUNSTATS” on page 330.

12.3 Configuring automatic maintenance

DB2 UDB provides automatic maintenance capabilities for performing database backups, keeping statistics current, and reorganizing tables and indexes. Enablement of the automatic maintenance features is controlled by setting the automatic maintenance database configuration parameters. These parameters are a set of hierarchical switches that allow for simplicity and flexibility in managing the enablement of these features. Automatic maintenance can also be configured using Control Center.

Note: The first time the Configure Automatic Maintenance wizard is run, the *Retrieving Policies* phase might take some time as new tables are being created to store control information. If you do not want to wait, you can click **Close** because the tables are created in the background. You can then continue with the wizard when the table creation process has completed.

Automatic database backup

Automatic database backup provides users with a solution that ensures that their database is backed up properly and regularly, without needing to worry about when to back up or having any knowledge of the **backup** command.

When using automatic database backup, the need to perform a backup operation is based on one or more of the following criteria:

- ▶ A full database backup has never been completed.
- ▶ The time elapsed since the last full backup is more than a specified number of hours.
- ▶ The transaction log space consumed since the last backup is more than a specified number of 4 KB pages (in archive logging mode only).

Important: When backup to disk is selected, the automatic backup feature regularly deletes backup images from the directory specified in the Configure Automatic Maintenance wizard. Only the most recent backup image is guaranteed to be available at any given time. We recommend that this directory be kept exclusive to the automatic backup feature and not be used to store other backup images.

The automatic database backup feature can be enabled or disabled using the `AUTO_DB_BACKUP` and `AUTO_MAINT` database configuration parameters.

Using the Configure Automatic Maintenance wizard in Control Center or Health Center, you can configure the requested time or number of log pages between backups, the backup media, and whether an online or offline backup is used.

Automatic database statistics

Automatic statistics collection attempts to improve the performance of the database by keeping up-to-date statistics. Automatic statistics profiling advises when and how to collect table statistics by detecting outdated, missing, and incorrectly specified statistics and by generating statistical profiles based on query feedback. Automatic statistics collection works by determining the minimum set of statistics that give the optimal performance improvement. The decision to collect or update statistics is taken by observing and learning how often tables are modified and how much the table statistics have changed. The automatic statistics collection algorithm learns over time how fast the statistics change on a per table basis and internally schedules **runstats** execution accordingly.

Normal database maintenance activities, such as **runstats**, **reorg**, or altering or dropping a table, are not affected by the enablement of this feature. The automatic statistics collection feature can be enabled or disabled by using the `AUTO_RUNSTATS`, `AUTO_TBL_MAINT`, and `AUTO_MAINTDATABASE` configuration parameters.

Tables considered for automatic statistics collection are configurable using the Automatic Maintenance wizard from Control Center or Health Center.

Automatic table and index reorganization

Automatic reorganization determines the need for reorganizing tables and indexes by using the **reorgchk** formulas. It periodically evaluates tables that have had their statistics updated to see if reorganization is required. If so, it internally schedules a reorganization for the table. Any applications that are connected to the database are not able to have write access to a table while it is being reorganized.

The automatic reorganization feature can be enabled or disabled by using the **AUTO_REORG**, **AUTO_TBL_MAINT**, and **AUTO_MAINT** database configuration parameters.

Automatic reorganization can be configured using the automatic maintenance wizard from Control Center or Health Center.

Automation example

The following example uses a wizard available from Control Center to configure automatic maintenance. You can use this wizard to enable and disable maintenance.

In Control Center, right-click the target database and select **Configure Automatic Maintenance**. Select **Change Automation Settings** in the Select Automatic Maintenance Type window. This window also displays the current settings for maintenance, which are all turned off in this example. Next, specify when maintenance activities can be run. You can specify both online and offline windows. We define an online window of 00-4 a.m., 7 days a week, and an offline window on the fifteenth day of the month for one hour. In the next window, add names to the notification list for health messages.

Select a maintenance activity to configure. In this example, we chose **backup**, **reorg**, and **runstats**. You can specify filters, such as tables for **runstats** and **reorg**, and **backup** options, such as backup location, online or offline, and criteria for backup. For example, we define the maximum time between backups and maximum log space used between backups by clicking the **Customize** button on the Backup Criteria window. In this example, we use the default values.

Finally, the summary window (see Figure 12-4) shows you all the options you have selected.

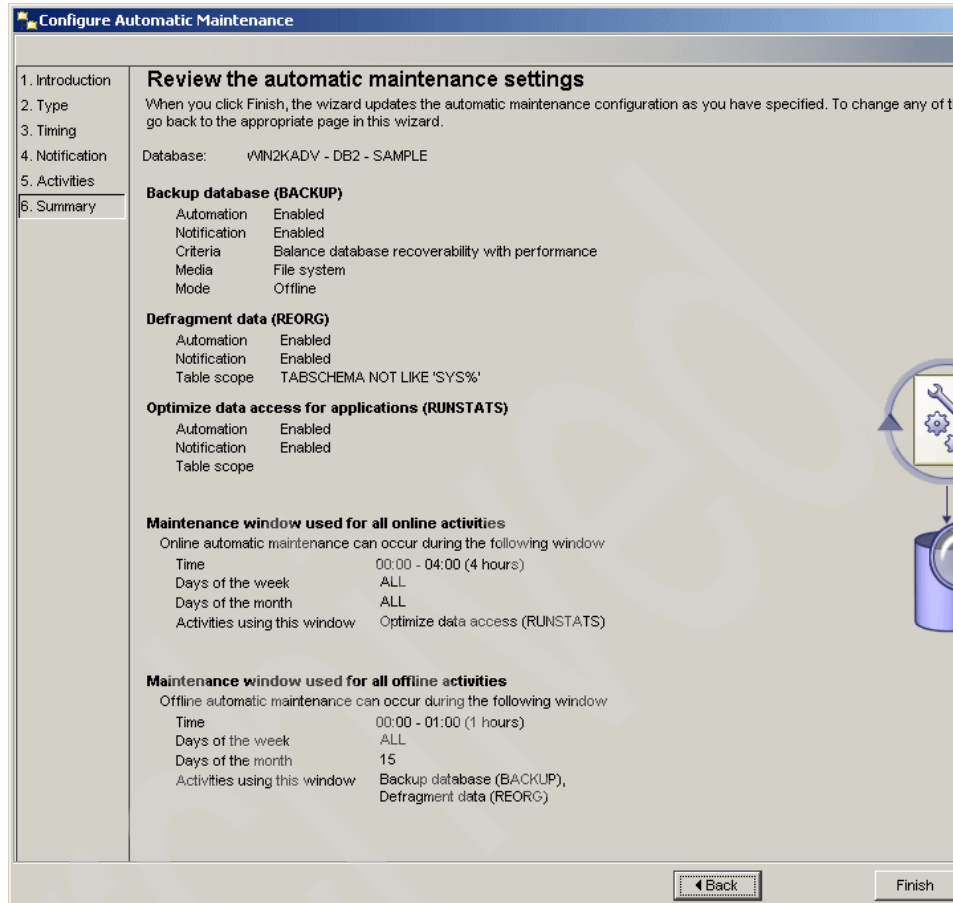


Figure 12-4 Configure automatic maintenance summary

Click **Finish** to complete the Configure Maintenance wizard and to turn on maintenance.

Automation maintenance windows

The automatic maintenance features previously described consume resources on your system and can affect the performance of your database when they are run. Automatic reorganization and offline database backup also restrict access to the tables and database when these utilities are run. It is, therefore, necessary to provide appropriate periods of time when these maintenance activities can be scheduled for execution.

Offline database backups and table and index reorganizations are run in the offline maintenance time period. These features run to completion, even if they

go beyond the time period specified. The internal scheduling mechanism learns over time and estimates job completion times. If the offline time period is too small for a particular database backup or reorganization activity, the scheduler does not start the job the next time and relies on the health monitor to provide notification of the need to increase the offline maintenance time period.

Automatic statistics collection and profiling, as well as online database backups, are run in the online maintenance time period. To minimize the impact on the system, they are throttled by the adaptive utility throttling mechanism. The internal scheduling mechanism uses the online maintenance time period to start the online jobs. These features run to completion even if they go beyond the time period specified.

Creating new databases with maintenance enabled

You can enable various automatic maintenance features when creating a new database using Control Center or First Steps. The automatic maintenance features can:

- ▶ Create a new database on the disk or directory of your choice
- ▶ Assign disk space for the data
- ▶ Configure the new database for performance
- ▶ Turn on automatic maintenance
- ▶ Configure notification by e-mail or pager if the database needs attention

These features can also be manually configured and enabled/disabled as previously described.

12.4 Other performance tuning advice

The performance of a DB2 UDB database can be influenced by many factors including the type of workload, the type and quantity of hardware resources, application design, database design, and instance and database configuration. Default database configuration settings should be customized for your specific environment before ever being deployed in a production environment. This section focuses on a number of DB2 UDB performance tuning tips that can be used when determining an initial configuration.

12.4.1 Table spaces

Recall from the section 11.3, “Managing database storage” on page 272 that three table spaces are created by default at database creation time:

- ▶ 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.
- ▶ USERSPACE1 - User table space for storing table data and indexes

In SMS table spaces, reading and writing data from/to tables is buffered by the operating system, and space is allocated according to operating system conventions. When a table is initially created, only one page is allocated to it on disk. When records are inserted into a table, DB2 UDB extends the table file by one page at a time by default.

When the workload involves performing many insert operations, extending files by only one page at a time can be a very expensive operation. To minimize the overhead of table space extension, multi-page file allocation can be enabled. 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) versus one page at a time.

To check whether multi-page file allocation is enabled, examine the database configuration and search for the string `Multi-page`.

Example 12-2 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 12-2 `Multi-page` is 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 must be disconnected. After `db2empfa` is run against the target database, check the multi-page file allocation parameter for the status again (see Example 12-3).

Example 12-3 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
...	

Improved insert performance can also be achieved using *Database Managed Space (DMS)* table space because containers are pre-allocated and the management of I/O operations is shifted to the database engine. In DB2 UDB, DMS containers can be added, dropped, or modified in size. Data is automatically rebalanced across containers when any of the previously mentioned operations occur, unless specified otherwise.

For optimal performance, large data volumes and indexes should be placed on DMS table spaces, which, if possible, reside on separate devices. System catalogs and system temporary table spaces should generally use SMS table spaces. The system catalogs contain large objects, which are not cached in DMS table spaces, but are cached by the operating system when using SMS table spaces. In an OLTP environment, there is typically no need for creating large temporary objects or staging areas for data, so an SMS table space is a good starting point for system temporary table spaces.

12.4.2 Physical placement of database objects

When creating a new database, an important design decision involves determining the type of hardware resources required. The ideal situation is to have the fastest disks possible and at least five to ten disks per processor (for an intensive high I/O transaction processing workload). However, hardware is often chosen based on other considerations, thus in order to achieve optimal performance, the placement of database objects needs to be carefully planned.

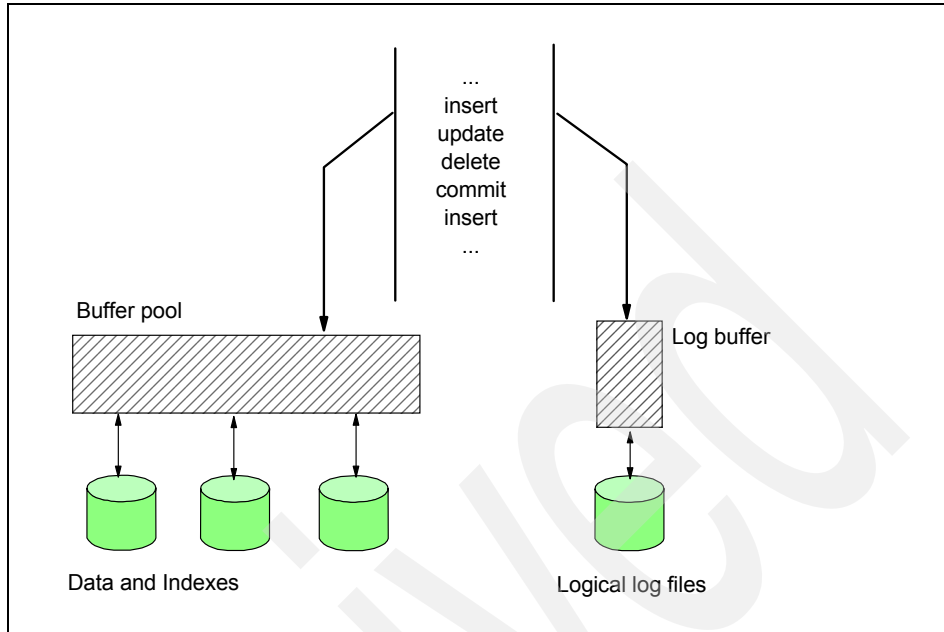


Figure 12-5 Explaining logical log

As shown in Figure 12-5, all data modifications are not only written to table space containers, but are also logged to ensure recoverability. Because every INSERT, UPDATE, or DELETE operation is recorded in the transaction log, the flushing speed of the logical log buffer can be crucial for database performance. To understand the importance of logical log placement, you should keep in mind that the time to write data to disk depends on the physical data distribution on disk. Many random read and write operations mean more disk head movements are required. Flushing (writing) the logical log buffer to disk is a sequential operation by nature, and should not be interfered with by other read or write requests. Locating the logical log files on separate physical devices ensures uninterrupted sequential writes by other processes or queries.

To change the physical location of logical log files, the NEWLOGPATH database parameter can be modified, as shown in Example 12-4. The log files are relocated to the new path after the next database activation. It may take some time to create the log files if they do not already exist.

Example 12-4 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 optimal value of these two parameters depends on the type of workload and available resources. As a start, the following guidelines can be used in determining the values for these parameters:

- ▶ 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 12-5 shows how to set the initial values of the parameters, assuming a two processor machine with six disks are available to DB2 UDB.

Example 12-5 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 might 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 in this case. If you want to allow parallel I/O operations on a single container, the DB2_PARALLEL_IO registry variable should be set before starting the DB2 UDB engine.

By executing the following command, I/O parallelism will be enabled within a single container for all table spaces:

```
db2set DB2_PARALLEL_IO="*"
```

The following example show how parallel I/O can be enabled for only for two table spaces: DATASP1 and INDEXSP1:

```
db2set DB2_PARALLEL_IO="DATASP1,INDEXSP1"
```

To check the current value for the parameter, the following command can be issued:

```
db2set DB2_PARALLEL_IO
```


12.4.3 Buffer pools

Generally, the more active pages of data that can be retained in memory, the better the query performance. Therefore, large buffer pools generally result in improved performance, since more data can be kept in memory and the database is less likely to fetch the data from disk. However, a buffer pool that is too large could cause memory paging at the OS level, so a guideline is to allocate the largest buffer pool possible without causing paging.

The default size of a buffer pool is very small: only 250 pages (~ 1 MB) for Windows and 1000 pages (~ 4 MB) for Linux and UNIX platforms. The overall buffer pool size has a great effect on DB2 UDB performance since it can significantly reduce I/O, which is typically the most expensive operation. We highly recommend increasing the default buffer pool sizes. However, the total buffer pool size should not be set too high, because over-allocating memory to them can also cause problems. To calculate the maximum buffer pool size, all other DB2 UDB memory related parameters such as the database heap, agent memory, lock storage, as well as operating system and other application requirements should be taken into account.

Initially, the total size of buffer pools should be set to approximately 10% to 20% of available memory. You can always monitor the system later and correct it if necessary. DB2 UDB allows dynamic modification buffer pool sizes, meaning the changes take effect immediately, without shutting down the database. The ALTER BUFFERPOOL statement with the IMMEDIATE option causes the changes to 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 for buffer pools, this area can be divided into multiple, independent buffer pools to improve utilization. For example, suppose that a database has many frequently used small tables, which would normally reside in the buffer pool in their entirety, and thus would be accessible very fast. Now suppose a query runs against a very large table, using the same buffer pool as the smaller tables. When this query runs, the pages from the small, frequently used tables are replaced, making it necessary to re-read them when they are needed again. If the smaller tables had used their own buffer pool, the situation described above where a large query would cause their pages to be replaced could be avoided.

You can create additional buffer pools for caching user data and leave the IBMDEFAULTBP buffer pool for exclusive use by system catalogs. Creating an extra buffer pool for system temporary data also can be valuable for system performance, especially in an OLTP environment where temporary objects are

relatively small. In a data warehousing environment, temporary table spaces are used more heavily, so the buffer pools used by them should also be larger.

Example 12-6 shows how to create buffer pools. It assumes that an additional table space called DATASPACE is already created for storing data and indexes and that there is enough memory in the system. This example can be used as a starting point for buffer pool configuration for a system with a minimum of 2 GB RAM.

Example 12-6 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

In addition to creating larger buffer pools, other parameters must also be adjusted accordingly. The CHNGPGS_THRESH parameter specifies the percentage of changed pages at which the asynchronous page cleaners are started. Asynchronous page cleaners write changed pages from the buffer pool to disk. The default value for the parameter is 60%. When the threshold is reached, some users may experience a slower response time. Having larger buffer pools means

more modified pages in memory and more work to be done by page cleaners, as shown in Figure 12-6. To guarantee a more consistent response time as well as a shorter recovery phase, we recommend using a smaller value for this parameter (50 or 40). The value can be changed by issuing the following command:

```
db2 update db cfg for redbook using CHNGPGS_THRESH 40
```

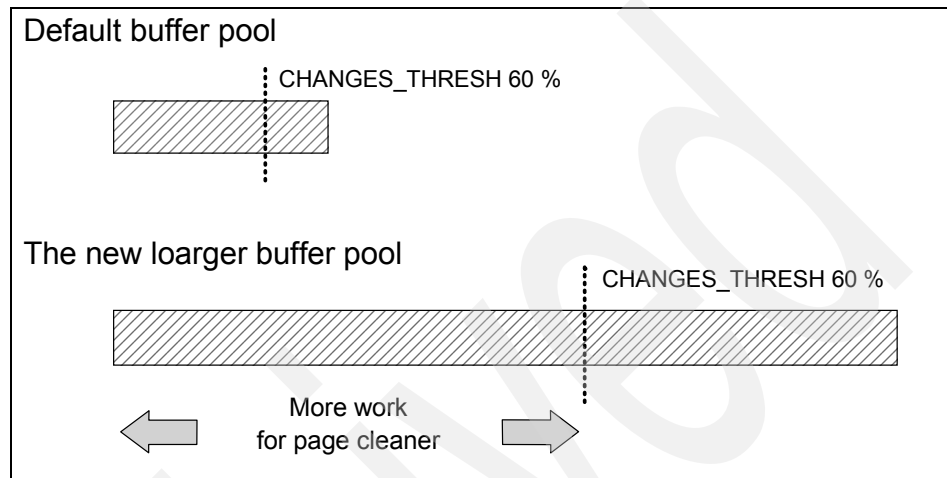


Figure 12-6 Visualizing *CHNGPGS_THRESH* parameter

With 32-bit versions of DB2 UDB, there is a limit on the total size of the buffer pools that can be defined regardless of the available real memory. No such restrictions apply to 64-bit versions of DB2 UDB, where real memory limits that cause operating system paging are most likely a problem with large buffer pools. An advantage of a single buffer pool is that it exploits the efficient page management algorithm of DB2 UDB, which maintains a high buffer hit ratio by keeping the most active and important pages, such as index pages, in memory while migrating less frequently used pages to disk. Having one buffer pool also requires no tuning after its size is chosen. The main disadvantage of a single buffer pool is that it will not be able to distinguish between high priority access data with potentially low access rates when there are lower priority table spaces with higher activity rates using the same buffer pool.

The advantages and disadvantages of multiple buffer pools are the opposite of those for a single buffer pool. Multiple buffer pools offer greater flexibility for prioritizing the I/O performance of different table spaces, but require constant monitoring and tuning to keep them performing optimally.

If you decide to use more than one buffer pool, there are two recommended methods:

- ▶ Define at least three buffer pools: a smaller one for randomly accessed tables, a larger one for the rest of the tables, and a larger one for all the indexes. This prevents the tables that are randomly accessed from thrashing the main buffer pool and also separates data pages from index pages.
- ▶ Define at least two buffer pools: one for all of the static tables, and a another for the data and index pages. This configuration allows the static tables to be retained in memory, and the most active data and index pages are also retained.

In summary:

- ▶ Larger buffer pools are better.
- ▶ Set CHNGPGS_THRESH lower if victim/dirty page steals occur.
- ▶ Set NUM_IOCLEANERS to one or two more than the number of physical drives.

For further reference, consult the DB2 UDB documentation *Administration Guide: Performance*, SC09-4821. There is also an article on the IBM developerWorks Web site called “DB2 Basics: Table Spaces and Buffer Pools,” available at:

<http://www.ibm.com/developerworks/db2/library/techarticle/0212wieser/0212wieser.html>

12.4.4 Large transactions

By default, databases are created with relatively small space for transactional logs. Only three log files with each 250 pages are created on Windows while only 1000 pages are created on Linux and UNIX.

A single transaction must be able fit into the available log space. 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 that modify a large number of rows, adequate log space is needed.

The total *log space* available for transactions can be calculated by multiplying the size of one log file (database parameter LOGFILSIZ) by the number of log files (database parameter LOGPRIMARY) .

From a performance perspective, it is better to have a larger log file size because of the high cost of switching from one log to another when a log file fills up. When log archiving is enabled, 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.

In Example 12-7, 400 MB of total log space is allocated.

Example 12-7 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 manages 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 amount of memory 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. Using the default installation parameter values, a maximum of 5688 (Windows) or 11377 (Linux and UNIX) locks can be allocated as shown in Figure 12-7.

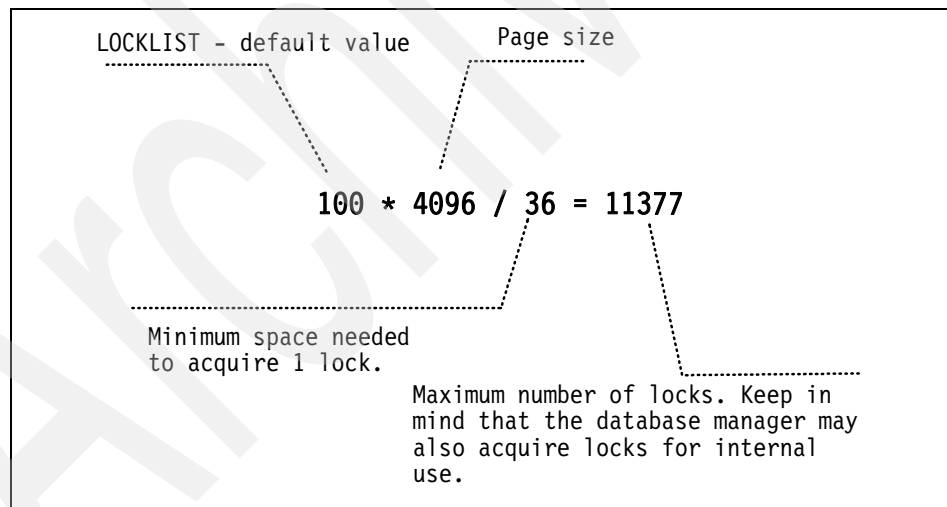


Figure 12-7 Maximum number of locks available for default settings on Linux

When the maximum number of locks has been reached, the database manager replaces existing row level locks with table locks (*lock escalation*). This operation reduces the amount of lock space needed since a transaction will only hold a single 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 before obtaining a lock.

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. In Linux and UNIX, the default value for this parameter is 10, while on Windows, it is 22. Thus, if one application requests more than 10% of the total locks space (LOCKLIST), a lock escalation occurs for that application. For example, inserting 1137 rows of data into a table on Linux within one transaction, using the default parameter values, results in lock escalation because the transaction requests 1138 locks (one per each inserted row plus one internal lock). This requires at least $1138 \times 36 = 40968$ bytes of storage - more than 10% of global lock memory defined by the LOCKLIST parameter default value.

Initial values for LOCKLIST and MAXLOCKS should be based on the maximum number of applications and average number of locks requested by a transaction (for OLTP systems, assume a minimum of 512 locks for every application). When setting MAXLOCKS, you should take into account lock-intensive batch processes that run during daytime hours. To check the current usage of locks, a lock snapshot can be taken, as illustrated in Example 12-8.

Example 12-8 Taking a database lock snapshot

```
db2 get snapshot for locks on sample
```

The snapshot collects the required locking information at the time the command is issued. Figure 12-8 shows the lock snapshot output. In this example, during the time the snapshot was taken, there were two applications connected to the database SAMPLE, and a total of 1151 locks were acquired.

Issuing a subsequent **GET SNAPSHOT** command may produce a different result because other applications connected to the database may have already released the locks they held at the time of the previous snapshot.



Figure 12-8 Explaining lock snapshot information

To check for lock escalations, examine the *db2diag.log* file. A lock escalation message looks similar to Example 12-9.

Example 12-9 Lock escalation message in db2diag.log file

```

2005-02-21-19.05.05.888741 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 of the logical log buffer is eight pages (32 KB), which is often too small for an OLTP workload or long running batch processes. In most cases, log records are written to disk when a transaction issues a **COMMIT** statement, 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. With a larger buffer

size, log records are written to disk less frequently and more log records are written each time. As a starting point, set LOGBUFSZ to 128 (or 256) 4 KB pages. The log buffer area uses memory controlled by the DBHEAP database parameter, so consider increasing this parameter value.

A snapshot for applications can be used to check the log space occupied by transactions. Before taking an application snapshot, the *Unit Of Work* monitor must be switched on, as shown in Example 12-10.

Example 12-10 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

At the time the above snapshot was issued, only three transactions were running in the system. The first transaction used 478 bytes of log space, the second 21324 bytes, and the third 110865 bytes. An application snapshot only provides the counter values at the moment the command was executed. To obtain more valuable information about the use of log space by transactions, take multiple snapshots over time and compare the resulting outputs.

Example 12-11 shows how to obtain information about log I/O activity.

Example 12-11 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 taking a database snapshot, the monitors may need to be reset. The values gathered by a snapshot are accumulated from the time the monitor was last reset or the database was activated.

For convenience, the snapshot output can be re-directed to a file, and analyzed using the Linux **grep/egrep** tool. In Example 12-11, 630 pages were written at the time the snapshot was taken, which is about $630 / (23 + 2 + 1) = 25$ pages per transaction. It is not possible to tell what the average size of a transaction is looking strictly at the value *Log pages written*, because the basic DB2 UDB read or write unit is one page (4 KB). Issuing only one small **INSERT** forces 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 re-written to disk more than once, until it becomes full. This behavior guarantees that log files are contiguous.

When setting the value for the log buffer, also consider the ratio between *log pages read* and *log pages written*. The ideal situation is to have zero log pages read with a large number of log pages written. When there are too many log pages read, it usually suggests that a bigger **LOGBUFSZ** parameter value can improve performance.

12.4.5 Process tuning

DB2 UDB is a database engine that uses multiple agents (processes) and there are numerous configuration parameters that control the quantity of the processes running.

The database manager parameter **MAXAGENTS** limits the maximum number of processes that are available to perform requests on behalf of all the applications connected to all of the databases on the instance. At the database level, the parameter **MAXAPPLS** limits the number of applications that can be connected to the database at one time. Setting **MAXAPPLS** to **automatic** has the effect of allowing any number of connected applications. In this case, DB2 UDB dynamically allocates the resources it needs to support new applications.

If you do not want to set this parameter to **automatic**, the value of this parameter must be equal to or greater than the sum of the connected applications, plus the number of these same applications that might be concurrently in the process of completing a two-phase commit or rollback. Then, add to this sum the anticipated number of in-doubt transactions that might exist at any one time.

The value of **MAXAGENTS** should be at least the sum of the values for **MAXAPPLS** in each database allowing concurrent access.

12.5 Data access strategies

Fast access to data is a critical requirement for any application system. The topic of data access can be discussed from two perspectives. First, access to data typically represents the longest steps in a solution scenario. Therefore, minimizing the time to access data can result in significant decreases in response time for the users. One approach, out of many, is to buffer as much data as possible in memory.

Second, the volumes of data collected and stored by an enterprise today are growing at an exponential rate. Therefore, there is typically much more data on disk than can be buffered in memory for fast access. It is very important, therefore, for the database engine to provide techniques that enable the data to be accessed quickly and efficiently.

In this section, we describe the types of access methods that DB2 UDB offers to retrieve data efficiently. The topic is discussed from a logical (software) view, rather than a physical data storage and access view. Therefore, issues regarding how to organize physical disks is not included. However, you should note that this is also a very important consideration for enabling faster data access and improving performance.

12.5.1 Indexing

Indexes provide an efficient way to access data. Instead of scanning the entire table to locate particular data elements, an index contains a reference to the data page where the desired data can be found. That is, the index points directly to the appropriate data page without the requirement to access each page to determine whether the desired data is located there. DB2 UDB implements its indexing using B+ tree data structures.

The general syntax for creating indexes is shown in Example 12-12.

Example 12-12 Basic CREATE INDEX syntax in DB2 UDB

```
CREATE [UNIQUE] INDEX <name> ON <table>(<columns>)
```

An index name must be unique for a database and the identifier (the name of the index) must not be longer than 128 characters. Special characters are not allowed in the index name.

DB2 UDB allows index columns up to a width of 1024 characters. Keep in mind that it is not always ideal to index very large columns. The larger the indexed column, the less performance since the size of the index tree also grows larger, thus requiring more time to process the additional data.

Index placement

DB2 UDB allows indexes to be created in a different table space than the regular table data. The index table space must be specified at *table creation time*, as shown in Example 12-13. All indexes for a given table are placed in the specified table space.

Example 12-13 Separating index and data pages

```
CREATE TABLE t1 (  
    col1 INTEGER,  
    col2 CHAR (10)  
)  
IN tbs1  
INDEX IN idxtbs1;  
  
CREATE INDEX idx1 ON t1 (col1);
```

Database Managed Spaces (DMS) table spaces must be used in order to separate application data and indexes. Both the table space for the data and the table space for the indexes must be defined as DMS table spaces.

Clustered index

For performance reasons, you might to decide to create a clustered index. The syntax to create a clustered index is shown in Example 12-14.

Example 12-14 Creating a clustered index

```
CREATE [UNIQUE] INDEX <name> ON <table> (<columns>)  
CLUSTER
```

DB2 UDB clustered indexes attempt to insert new rows physically close to the rows for which the key values in the index are in the same range. Therefore, the PCTFREE option becomes important, because it influences the future quality of a clustered index when new rows are inserted (see the following section entitled PCTFREE).

The quality of a clustered index is represented in the system catalog table SYSCAT.INDEXES. The value of CLUSTERRATIO determines the quality of the clustered index. The higher the value, the better the clustering. This value is maintained dynamically by DB2 UDB meaning the **runstats** command does not update the quality of this information.

If the quality of a clustered index becomes poor, you may have to reorganize the index.

PCTFREE

The DB2 UDB parameter PCTFREE specifies how much space should be reserved within index leaf pages for upcoming INSERT statements. The default value is set to 10. This means that 10% of the space in an index page is to be reserved for future modifications. Keep in mind that DB2 UDB has different page sizes. The syntax for this parameter is shown in Example 12-15.

Example 12-15 Index with PCTFREE

```
CREATE [UNIQUE] INDEX <name> ON <table> (<columns>)  
    PCTFREE [0..99]
```

Space requirements for indexes

Indexes might improve response times, but there is a trade-off for that improvement. Indexes are database objects that consume disk space. When converting an existing database environment, there is a great opportunity to plan the sizing of database objects, such as indexes, correctly. Because you already have the data, you do *not* have to estimate how much data *might* occur. Better calculations lead to more effective data storage and thus better performance.

A sample equation to calculate an index size can be given as:

$$2 * r * (i + 9)$$

Where,

r = number of rows in table

i = key size (bytes)

When the index is being created, it consumes temporary disk space. You should estimate how much temporary space is required. You can estimate this using the following formula:

$$3.2 * r * (i + 9)$$

Where,

r = number of rows in tables

i = key size (bytes)

For these equations, unique indexes are assumed. If the calculation is made with indexes that allow duplicates, you must add 5 bytes per row. In addition, if the index allows NULL values, add one byte per row for the NULL indicator.

Be aware that these equations only provide estimates. Therefore, you should validate the estimates before using them.

If you create an index using Control Center's Create Index wizard, you can obtain an estimate of the index size. Click the **Estimate Size** button in the wizard. A new window opens. In this window, enter the number of rows of the table and click **Refresh**, as shown in Figure 12-9.

Estimate Size - REDBOOKS

GB071538XP - DB2 - REDBOOK - STEFAN.REDBOOKS

Table Specifications

Name REDBOOKS

Schema STEFAN

Tablespace USERSPACE1

Number of rows 47 Average row length 85 bytes

New total number of rows 201501 New average row length 85 bytes

Objects

Type	Name	Schema	Tablespace	Current size	Estimated size	Minimum size	Maximum size
Table	REDBOOKS	STEFAN	USERSPACE1	0.01	20.61	12.37	
Index	PK_BOOK_ID	STEFAN	USERSPACE1	0	7.83	6.3	
Index	IDX_REDBOOKS_TITLE	STEFAN	USERSPACE1	0	7.05	6.3	
Index	REDBOOKS	GB071538	USERSPACE1	0	7.83	6.3	

Display size in units of MB

Run statistics... Refresh Close Help

Figure 12-9 Estimating the index size using the Create Index wizard

Deleting index entries

DB2 UDB index entries are not deleted immediately when a DELETE statement is executed. Instead, the deletion of an index entry is processed as follows:

- ▶ During key insertion, keys that are marked deleted and are known to be committed are cleaned up if such a cleanup might avoid the need to perform a page split and prevent the index from increasing in size.
- ▶ During key deletion, when all keys on a page have been marked deleted, an attempt is made to find another index page where all the keys are marked deleted and all those deletions have committed. If such a page is found, it is deleted from the index tree.
- ▶ If there is an X lock on the table when a key is deleted, the key is physically deleted instead of just being marked deleted. During this physical deletion, any deleted keys on the same page are also removed if they are marked deleted and known to be committed.

12.5.2 DB2 UDB index expansions

In the DB2 UDB documentation *SQL Reference - Volume 2*, SC09-4845, you can find the complete CREATE INDEX command syntax. The following sections provide examples of the additional options and parameters that can be specified as part of this command.

Include option

DB2 UDB allows key-only reads. These are reads from the database where all the columns in the SELECT list are available through an index. Key-only reads are very efficient because there is no need to access data pages.

DB2 UDB allows you to include additional columns (see the general syntax in Example 12-16) into a unique index. These index entries that are placed at the *leaf level* only.

Example 12-16 Index with INCLUDE expansion

```
CREATE UNIQUE INDEX idx_name ON tab_name (col_list)
    INCLUDE (col_list)
```

Note: Include indexes are allowed for unique indexes only.

The creation of an INCLUDE index is shown in Example 12-17. First, a table with two columns is created. col1 is unique and used for query qualification, but col2 is added to the index for a key-only read. Rows are then inserted into the table.

Example 12-17 Creation of an ordinary composite index

```
CREATE TABLE t1 (
    col1 INTEGER,
    col2 CHAR (3));

INSERT INTO t1 (col1, col2) VALUES (10, 'IKE');
INSERT INTO t1 (col1, col2) VALUES (11, 'D00');
...
INSERT INTO t1 (col1, col2) VALUES (19, 'TOR');

CREATE UNIQUE INDEX t1_idx1 ON t1 (col1, col2);
```

Figure 12-10 shows a visual representation of the index created in Example 12-17.

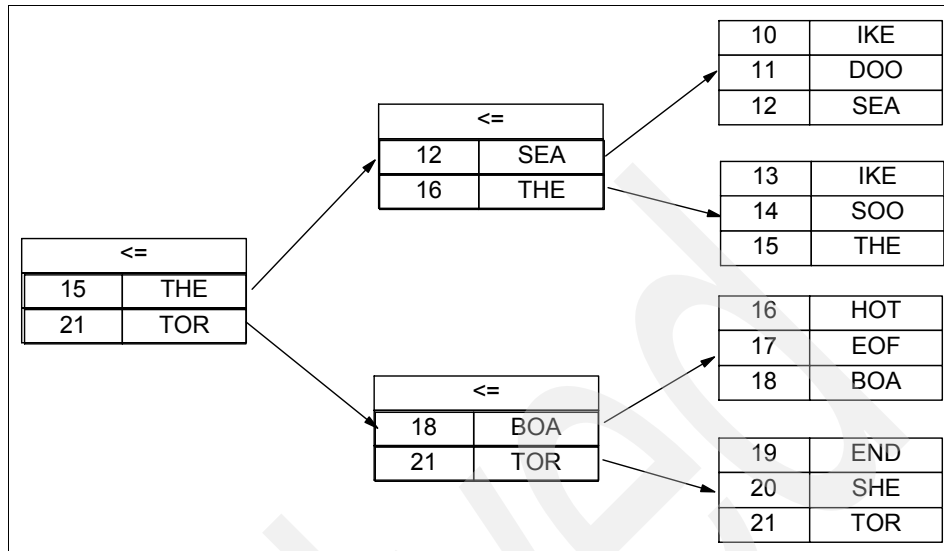


Figure 12-10 Schematic view of a composite index

Examine the INCLUDE index in Example 12-18. The column list of the CREATE INDEX statement does not include the column col2 any longer, but it appears at the end of the statement in the INCLUDE clause. Figure 12-11 on page 370 illustrates the logical structure of the INCLUDE index.

Example 12-18 Index with the INCLUDE option

```

CREATE UNIQUE INDEX t1_idx ON t1 (col1)
  INCLUDE col2
  
```

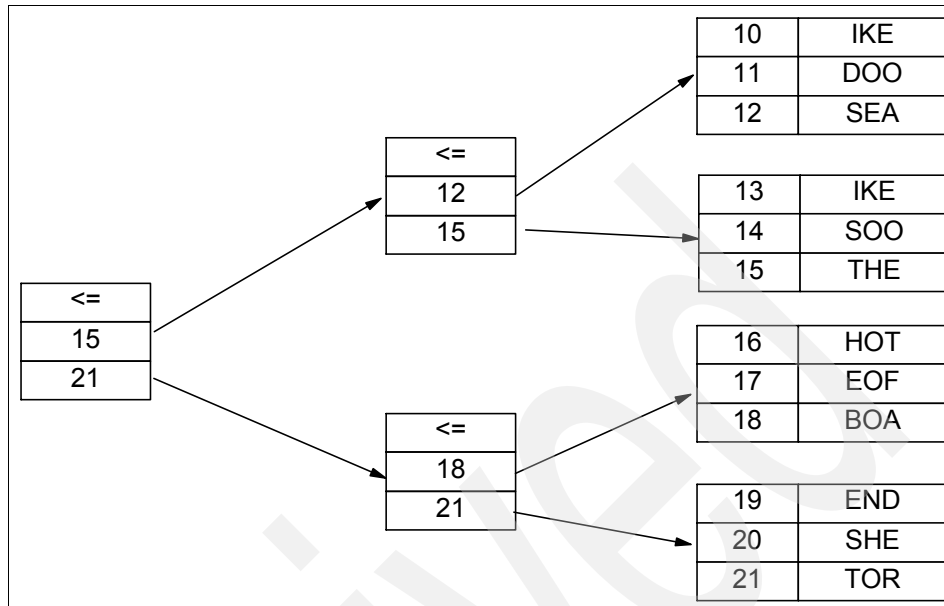


Figure 12-11 Schematic view of an INCLUDE index

Notice that the included column is stored at leaf level only, and it is still possible to use this index for queries by the unique col1 column. If a query selects both columns col1 and col2, it is possible to get a key-only select without reading data pages. Using this technique can increase performance and save index disk capacity.

MINPCTUSED parameter

An index tree is as dynamic as the data changes in the corresponding table. Index leaf pages are split into two new leaf pages if the original leaf page becomes full. If a leaf page becomes empty because rows have been deleted from the corresponding table, DB2 UDB attempts to combine two leaf pages together into one.

The MINPCTUSED parameter defines the percentage at which a leaf page is declared empty and thus subject for compression. The default value for this parameter is 0% which means that a index leaf page must be completely empty before a reverse split occurs. You can set the MINPCTUSED parameter up to 99%, however, we recommend that you do not use values higher than 50%. Example 12-19 shows how to set the value of this parameter.

Example 12-19 DB2 UDB index with the MINPCTUSED parameter

```
CREATE INDEX <name> ON <table> (<columns>)
```


ALLOW REVERSE SCANS option

Indexes are a good instrument to speed-up queries that perform sorts. DB2 UDB index leaf pages are implemented as a doubly-linked list. This allows access to the data in both forward and backward directions.

When you create an index in DB2 UDB, you must specify what sort sequence the index should support. If you do not specify the sorting sequence, the index follows ascending sorting. When `ALLOW REVERSE SCANS` is specified in the `CREATE INDEX` statement (see Example 12-20), the index also can be used for inverse sorting. The default is `DISALLOW REVERSE SCANS`. We recommend using the `ALLOW REVERSE SCANS` option since it adds negligible overhead.

Example 12-20 Index with the ALLOW REVERSE SCANS option

```
CREATE INDEX <name> ON <table> (<columns>)
ALLOW REVERSE SCANS
```

COLLECT

DB2 UDB allows the collection of statistical information of an index during creation time, as demonstrated in Example 12-21. Therefore, there is no need to execute a **runstats** command after the index has been created.

There are three options when specifying the `COLLECT` parameter:

- ▶ **STATISTICS**: Collects basic statistic information about the index.
- ▶ **DETAILED STATISTICS**: Creates a distribution curve of the index data.
- ▶ **SAMPLED DETAILED STATISTICS**: Creates a sample distribution curve of the index data.

Example 12-21 Index with the COLLECT STATISTICS option

```
CREATE INDEX <name> ON <table> (<columns>)
COLLECT [ [SAMPLED] DETAILED ] STATISTICS;
```

Indexes and constraints

When you create primary key or foreign key constraints on columns, a corresponding index is necessary for these columns. If an index does not exist, the DB2 UDB will create one for you. These kinds of indexes are called *implicit indexes*.

If you want to ensure that these implicit indexes are stored in a different table space than the data, you must first create the index manually before the actual constraint is created. Example 12-22 demonstrates this procedure.

Example 12-22 DB2 UDB index used by a constraint

```
CREATE TABLE orders (  
    order_num INTEGER NOT NULL,  
    order_date DATE,  
    customer_num INTEGER NOT NULL  
);  
CREATE UNIQUE INDEX pkidx ON orders (order_num);  
CREATE INDEX fkidx ON orders (customer_num);  
ALTER TABLE orders  
    ADD CONSTRAINT pkorders  
        PRIMARY KEY (order_num);  
ALTER TABLE orders  
    ADD CONSTRAINT fkorders  
        FOREIGN (customer_num) REFERENCES customer;
```

12.5.3 Index reorganization

DB2 UDB Version 8.1 introduced online index reorganization. This feature enables you to perform the reorganization of an index even if users are connected to the database. It is not only possible to run the reorganization continuously online, you can also interrupt it and then continue the process as needed.

During an online index reorganization, the entire index object (that is, all indexes on the table) is rebuilt. A “shadow copy” of the index object is made, leaving the original indexes and the table available for read and write access. Any concurrent transactions that update the table are logged. After the logged table changes have been forward-fitted and the new index (the shadow copy) is ready, the new index is made available. While the new index is being made available, all access to the table is prohibited. The time required to swap files is quite small, typically on the order of seconds. The default behavior of the **reorg indexes** command is **ALLOW NO ACCESS**, which places an exclusive lock on the table during the reorganization process, but you can also specify **ALLOW READ ACCESS** or **ALLOW WRITE ACCESS** to permit other transactions to read from or update the table.

12.6 Advanced access methods

DB2 UDB provides special objects that enable data to be fetched efficiently. The methods introduced in this section are particularly appropriate for data warehousing environments, but they can also be useful in OLTP environments.

Whether these methods enhance your environment depends on the nature of the queries and the availability of resources.

12.6.1 Materialized query tables

Materialized query tables (MQTs) provide a precalculated result of a query. At first glance, an MQT might appear similar to a view. Like views, MQTs are associated with a SELECT statement, whose result is passed to the caller. Contrary to a view, however, an MQT stores the result of the SELECT statement physically on disk when created; it is not resolved at SELECT time. If an MQT is specified by a client or chosen by the optimizer, the data is not retrieved from the underlying tables. This can accelerate query processing, especially if the query that makes up the MQT contains complex and resource-intensive statements. Because the result set that is returned by the MQT is already calculated, no run-time calculations are necessary.

An MQT is represented by a real table, but the optimizer handles MQTs in a special way. Whether an MQT is chosen by the optimizer or not depends on a few requirements. You should always look at the optimizer explanation plan to figure out the type query execution plan was created (see 12.7, “Optimizer” on page 381).

Creating materialized query tables

MQTs are created with the general syntax shown in Example 12-23.

Example 12-23 Syntax for creating an MQT

```
CREATE TABLE <name> AS (  
    SELECT <definition of MQT>  
)  
DATA INITIALLY DEFERRED  
REFRESH <DEFERRED / IMMEDIATE>  
<ENABLE / DISABLE> QUERY OPTIMIZATION  
MAINTAINED BY <SYSTEM / USER>
```

The following list provides a brief description of the options used in Example 12-23:

- ▶ SELECT definition clause: The SELECT statement makes up the definition of MQT. There are also some SQL statements and features that cannot be used with an MQT, such as:
 - References to temporary or typed tables
 - References to the system catalog
 - DATALINK data type
 - Large objects

- Functions that have `EXTERNAL ACTION` or are written in SQL
- ▶ **DATA INITIALLY DEFERRED:** When an MQT is created, data is not loaded into it by default. The data is loaded into the MQT with the inclusion of the `REFRESH TABLE` statement.
- ▶ **REFRESH:** The data stored in the MQT depends on the table or tables that are referenced by the MQT. Most likely, the data of the base tables are subject to change. The question is when should the changes in the base tables be reflected in the MQT. The `REFRESH` option offers the following choices:
 - **DEFERRED:** The data in the MQT is not updated automatically. The update occurs with the execution of a `REFRESH TABLE` statement.
 - **IMMEDIATE:** As soon the data in the base tables changes, the MQT data is automatically updated.
- ▶ **QUERY OPTIMIZATION:** This option defines how the optimizer considers the MQT for query plan calculations. It has the following options:
 - **ENABLE:** This is the default. The optimizer is allowed to use the MQT for query optimization.
 - **DISABLE:** The optimizer is not allowed to use the MQT for query optimization, but the MQT can be used directly by queries.
- ▶ **MAINTAINED BY:** This option identifies the type of modifications allowed on an MQT.
 - **SYSTEM:** This is the default. The MQT data can only be modified by the `REFRESH TABLE` statement.
 - **USER:** A client is allowed to modify the data of the MQT using DML statements. A `REFRESH TABLE` statement is not permitted against a user-maintained MQT.

Prerequisites for MQT

Whether the optimizer chooses an MQT or refers to the base tables depends on a few prerequisites that must be fulfilled:

- ▶ **Costs:** The optimizer performs an estimation of which is the best way to fetch the data. Even if there is a MQT that may seem to be the be the fastest way to access the data, the optimizer follows its own rules and might elect not to use the MQT. Do not be surprised, however, if costs are less by accessing data through the base tables. However, always look at the query plan.
- ▶ **QUERY OPTIMIZATION:** Make sure that an MQT is created with the query optimization enabled.
- ▶ **REFRESH AGE:** For `REFRESH DEFERRED` MQTs, the `CURRENT REFRESH AGE` special register must be set to `ANY`, as shown in Example 12-24. This setting informs the optimizer that the age of the data in the MQT is not a concern and to

always consider it. REFRESH IMMEDIATE MQTs are always kept current and are candidates for optimization regardless of the setting of the CURENT REFRESH AGE register

Example 12-24 Changing the MQT refresh age

```
SET CURRENT REFRESH AGE ANY
```

- ▶ **Dynamic SQL:** The optimizer will choose an MQT instead of using the base tables only for dynamic SQL statements. Static SQL statements will never be candidates for re-routing to an MQT.
- ▶ **Optimization class:** DB2 UDB supports different levels of optimization aggressiveness. To let the optimizer consider an MQT for a query, the database configuration parameter DFT_QUERYOPT must be set to 2, 5, 7, or 9. You can also adjust the query optimizer class on a connection basis, as shown in Example 12-25.

Example 12-25 Changing the query optimization level for a single session

```
SET CURRENT QUERY OPTIMIZATION [2, 5, 7, 9]
```

MQT routing

Figure 12-12 shows the decision tree used by the optimizer for choosing to use an MQT. The first prerequisite for using an MQT (other than those previously mentioned) is that the optimizer is able to rewrite the query. Query rewriting means that the optimizer has recognized other possible query paths to choose. Even if the optimizer has rewritten the query, it still might not use the MQT if the costs for the query with the MQT is higher than with the base tables.

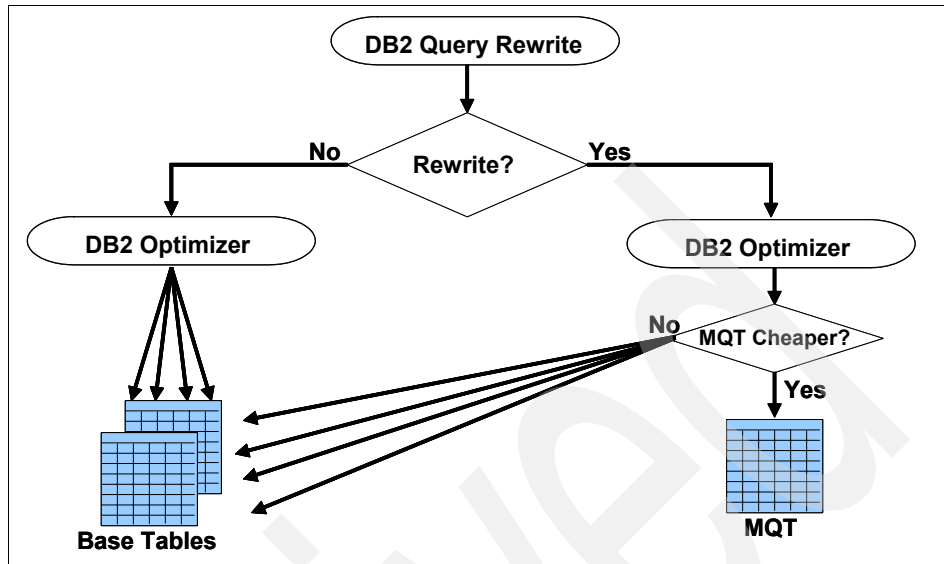


Figure 12-12 MQT routing

Example 12-26 shows the use of an MQT and how it is chosen by the optimizer. You should compare the *Optimized Statement* section and the costs of the query plans (“The db2exfmt utility” on page 387 describes the **db2exfmt** utility). Although the first query must run against the base table (no MQT had been created), the second query (with a FROM clause that specifies the data is to be read from the base table) is rerouted to the MQT. The costs estimated by the optimizer are lower when fetching data from the MQT than those of the base table. The two costs we refer to are in a large, bold (in red if viewed online) font to make it easy to compare the costs.

Example 12-26 Demonstration of MQT and query plan comparison

```

db2 "CREATE TABLE basetable (
    col1 INTEGER
)"

# table is populated with 2,046,118 rows

db2 "EXPLAIN PLAN FOR
    SELECT col1, SUM (twice (col1)), COUNT (*)
    FROM basetable
    WHERE twice (col1) BETWEEN 1000 AND 1000000
    GROUP BY col1
    HAVING SUM (twice (col1)) > 10000"

db2exfmt -d ifmx2db2 -1
  
```

Optimized Statement:

```
-----  
SELECT Q3.$C0 AS "COL1", Q3.$C1, Q3.$C2  
FROM  
  (SELECT Q2.$C0, SUM("DB2INST2"."TWICE"(Q2.$C0)), COUNT(*)  
   FROM  
     (SELECT Q1.COL1  
      FROM DB2INST2.BASETABLE AS Q1  
      WHERE ("DB2INST2"."TWICE"(Q1.COL1) <= 1000000) AND (1000 <=  
        "DB2INST2"."TWICE"(Q1.COL1))) AS Q2  
   GROUP BY Q2.$C0) AS Q3  
WHERE (10000 < Q3.$C1)
```

Access Plan:

Total Cost: 18997.2

```
db2 "CREATE TABLE mqt AS (  
  SELECT col1, SUM (twice (col1)), COUNT (*)  
  FROM basetable  
  WHERE twice (col1) BETWEEN 1000 AND 1000000  
  GROUP BY col1  
  HAVING SUM (twice (col1)) > 10000"  
)  
DATA INITIALLY DEFERRED  
REFRESH DEFERRED  
ENABLE QUERY OPTIMIZATION  
MAINTAINED BY SYSTEM;"
```

runstats omitted

db2 "REFRESH TABLE mqt"

```
db2 "EXPLAIN PLAN FOR  
  SELECT col1, SUM (twice_col1), COUNT (*)  
  FROM basetable  
  WHERE twice_col1 BETWEEN 1000 AND 1000000  
  GROUP BY col1  
  HAVING SUM (twice_col1) > 10000"
```

db2exfmt -d ifmx2db2 -1

Optimized Statement:

```
-----  
SELECT Q1.COL1 AS "COL1", Q1.TWICE_COL1, Q1.THE_COUNT  
FROM DB2INST2.MQT AS Q1
```

Access Plan:

MQT refreshing

When accelerating queries by using MQTs, there are trade-offs. The first is the space required for the MQT, but another issue is the timeliness of the data. Every time a row is inserted into one of the base tables referenced by the MQT, the data in the MQT cannot be refreshed, so it becomes old. Therefore, whether or not the use an MQT is acceptable depends on your business requirements.

There are advanced methods to keep an MQT's data up to date, such as using incremental refreshes. To avoid a full refresh of an MQT, the refresh table command can be issued, as shown in Example 12-27.

Example 12-27 REFRESH TABLE command

```
REFRESH TABLE <tablename> INCREMENTAL;
```

DB2 UDB also tries to refresh only the differences between base tables and the MQT. Refer to the DB2 UDB documentation *SQL Reference - Volume 2*, SC09-4845, for further information.

When to use MQTs

The design of good materialized query tables requires adequate up-front planning and analysis. The designer needs to be familiar with the query workload to identify patterns of table access, aggregation, and summarization.

When deciding whether or not to create a materialized query table, consider the following questions:

- ▶ Will the MQT significantly increase performance?
- ▶ Will many queries benefit? Will the most frequent or most critical or most expensive and long running queries benefit?
- ▶ Will the MQT offer resource savings: communication, I/O, and CPU?
- ▶ Is the loss of disk space that will contain the MQT and its indexes a worthwhile trade for the performance that will be gained?
- ▶ What is the cost of updating or refreshing the MQT?
- ▶ What are the patterns for accessing groups of tables for aggregation and for grouping requirements?
- ▶ How current does the data in the MQT need to be? Does it need to be up-to-the-minute?
- ▶ For MQTs that are maintained in real time, will automatic updates be too slow?

- In a partitioned environment, will co-location of data provide any benefit?
- What will be the logging requirement when large MQTs are refreshed?
- Should the MQT be system or user maintained?

Keep in mind that the DB2 Design Advisor can take the guesswork out of deciding what MQTs to create. It can recommend MQTs, as well as other database objects based on a given workload. See 12.2.1, “Design Advisor” on page 341 for more information.

12.6.2 Multidimensional clustering (MDC) tables

A *multidimensional clustering* (MDC) table is a technique used to group similar data together on disk. Example 12-28 shows the general syntax to create an MDC.

Example 12-28 Syntax to create an MDC

```
CREATE TABLE <name> (
    <column_list>
)
ORGANIZED BY (<column_list>);
```

The table data is physically separated into blocks differentiated by the columns defined in the ORGANIZED BY clause, called *dimensions*. The data blocks (dimensions) are indexed using a *block index*. Every time a query uses one of these dimensions in its WHERE clause (filter), the data can be retrieved using a block index.

An MDC table separates data depending on a value of a column. An MDC table does not spread the data over several table spaces. The data is organized in blocks within *one* table space.

If you create an MDC table, pay attention to the extent sizes that are defined for a table space. The size of an extent equals the size of an MDC block. There should be enough space within one extent/block to accept all rows that should belong to that block. If a block becomes full, additional blocks are allocated on disk. Ideally, the goal is to have only one block per data group, but having small number of blocks per group is acceptable. You should be careful about the number of dimensions specified. As guideline, the more dimensions, the smaller the blocks are.

Example 12-29 demonstrates how to create a two-dimensional MDC table. The data of the sales table is separated by the dimensions (columns) region and year. Figure 12-13 shows a visualization of that MDC.

Special use of an MDC table is shown in the second query of Figure 12-13. The query uses both dimensions of the MDC in its filter. In this case, ANDing of the block indexes is used to retrieve the desired data group. It is also possible to retrieve the desired data group by using block index ORing.

Example 12-29 Two-dimensional MDC table

```
CREATE TABLE sales (
  sales_id INTEGER,
  sales_rep CHAR (3),
  region CHAR (1), -- E, N, S or W
  year INTEGER -- 2002, 2003, 2004, ...
)
ORGANIZED BY (year, region);
```

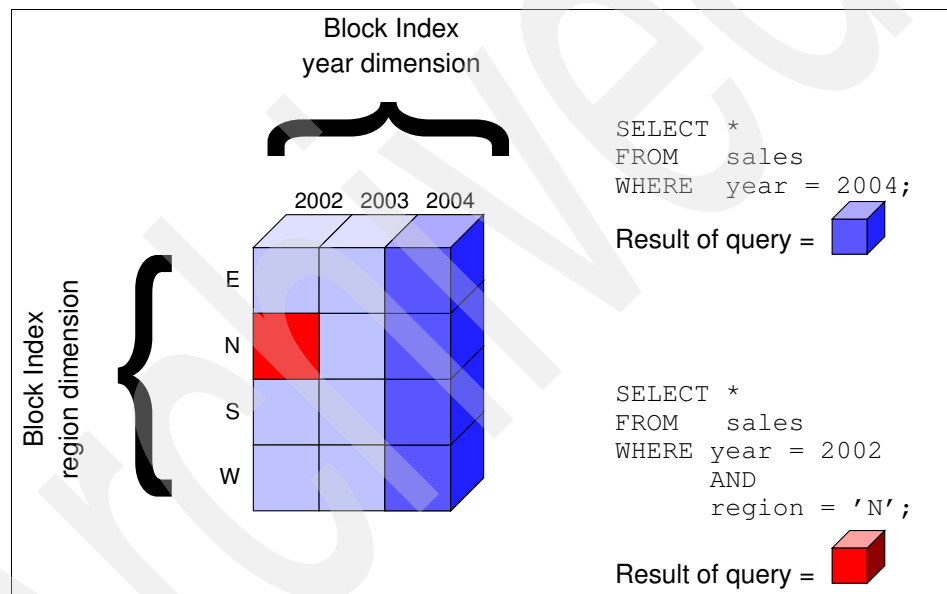


Figure 12-13 Visualization of a two-dimensional MDC table

In Example 12-30, the syntax of a sample MDC table is shown. This table separates data into different blocks, by year, quarter, and region. The advantage of creating data blocks is that if the query retrieves data using a filter on the dimensioned columns, a whole block of data can be read from disk. Contrary to a *clustered index* (see “Clustered index” on page 365), the need for disk-spread does not occur if the blocks are big enough.

Example 12-30 Syntax of a multidimensional clustering table

```
CREATE TABLE sales (
  order_num INTEGER,
```

```

customer_num INTEGER,
year SMALLINT, -- 2001, 2002, ...
quarter CHAR (2), -- Q1, Q2, Q3, Q4
region CHAR (1) -- N, S, W, E
ORGANIZE BY DIMENSION (year, quarter, region))

```

Figure 12-14 shows a visualization of the table created in Example 12-30.

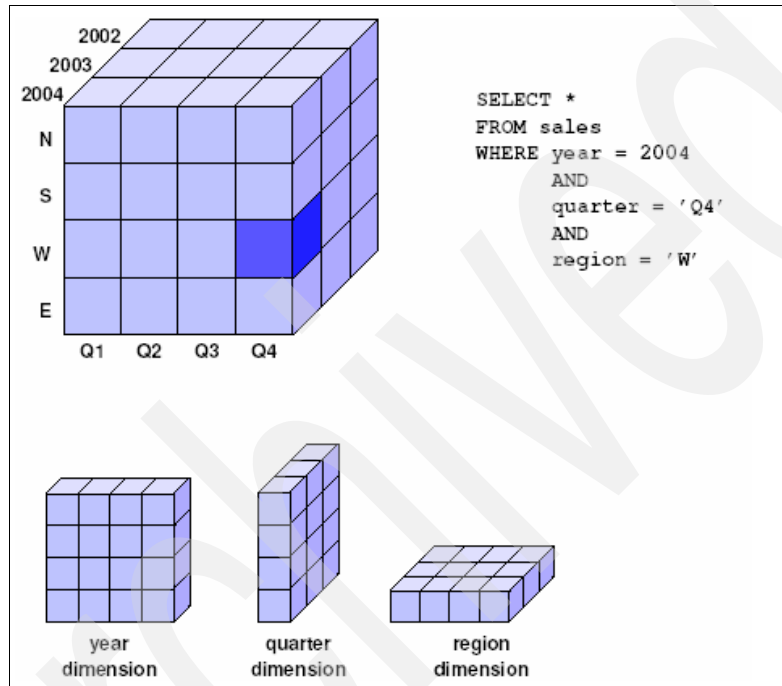


Figure 12-14 Block access using an MDC table

An MDC table is administrated by the database engine. It uses a block index to access the data. However, not every row is recorded in the index, so the index is small. Therefore, index lookups performed with block indexes are even faster than regular index lookups. When using MDC tables, regular indexes on the dimension columns or other columns of the table behind them are still allowed and might be necessary.

12.7 Optimizer

The optimizer is the part of a database engine responsible for creating query plans. A query plan describes how to obtain the data from disk, what indexes can

be used, and what join methods can be employed. If the optimizer produces a suboptimal plan, an application might experience poor query performance.

The optimization process is one of the last elements in a chain of events proceeding actual SQL statement execution. Figure 12-15 on page 382 shows what steps are performed when an SQL statement is submitted by a client to the engine. For a detailed discussion of each step, refer to the DB2 UDB documentation *Administration Guide: Performance*, SC09-4821.

The DB2 UDB optimizer is a cost-based optimizer. Cost-based optimizers require accurate and up-to-date statistical information to produce optimal plans. The **runstats** command can be executed to collect these statistics.

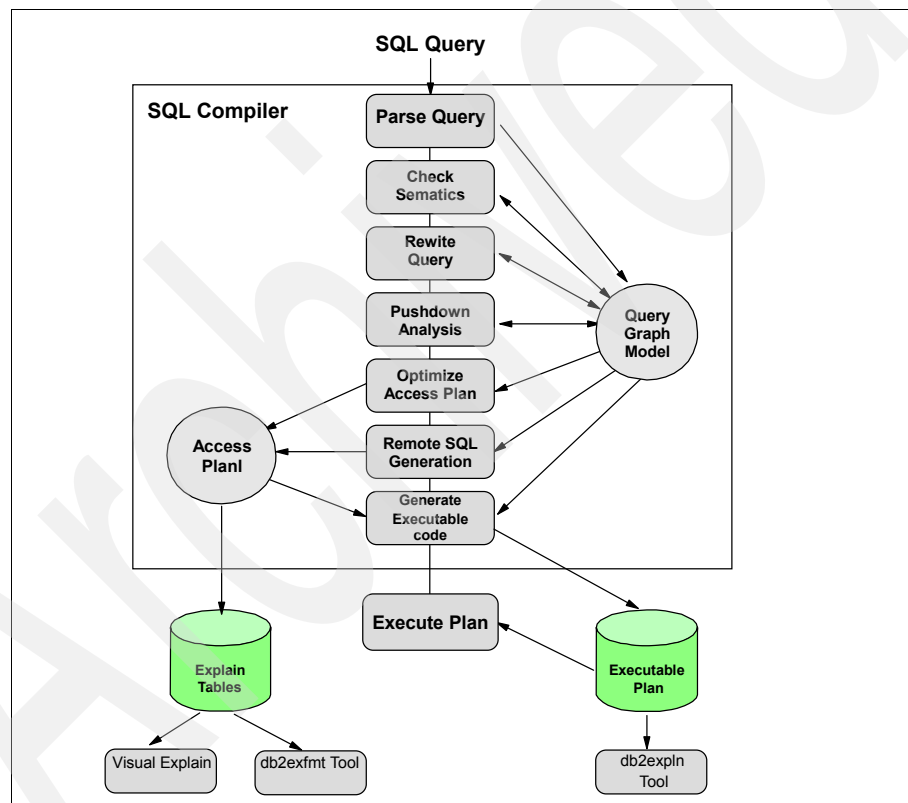


Figure 12-15 Steps performed by the SQL compiler

12.7.1 Optimizer analysis

One of the most important parts of performance tuning is to be able to read and understand what query plans the optimizer has created. DB2 UDB has various tools that enable you to analyze query plans generated by the optimizer.

Preparations

In DB2 UDB, before any kind of optimizer analysis can be performed, a special set of tables (*explain* tables) need to be created in each database. You can find the file that holds the necessary DDL to create the explain tables in C:\Program Files\IBM\SQLLIB\MISC\explain.ddl (see Example 12-31). After the tables have been created, you can proceed with your optimizer analysis.

Example 12-31 Creating the explain tables

```
C:\>db2 -tf "C:\Program Files\IBM\SQLLIB\MISC\explain.ddl"
```

```
***** IMPORTANT *****
```

```
USAGE: db2 -tf EXPLAIN.DDL
```

```
***** IMPORTANT *****
```

```
DB20000I The UPDATE COMMAND OPTIONS command completed successfully.
```

```
DB20000I The SQL command completed successfully.
```

```
[...]
```

```
DB20000I The SQL command completed successfully.
```

Visual Explain

Visual Explain can be launched from Control Center. This tool allows you to analyze an SQL statement. To use Visual Explain, follow these steps:

1. Open Visual Explain. Launch Control Center, right-click the target database, and select **Explain SQL** from the menu, as shown in Figure 12-6.

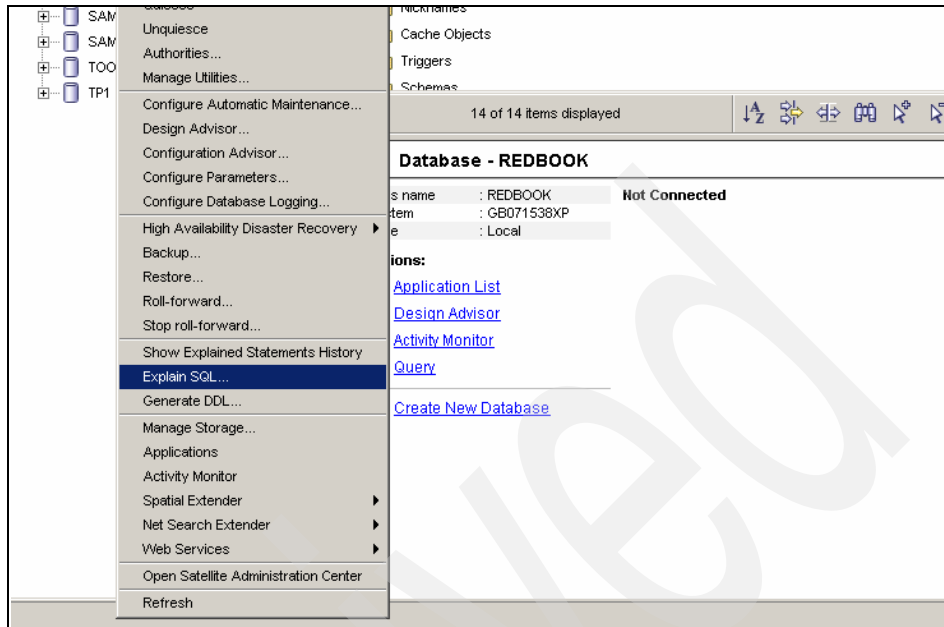


Figure 12-16 Launch Visual Explain from Control Center

2. Ensure a database connection is established and enter an SQL statement, as shown in Figure 12-17.

In the Explain SQL Statement window, there are four adjustable elements:

- Query number: A query identification number.
- Query tag: This tag is a reference that is used within the system catalog.
- Optimization class: Set the optimization class for this query. For more details, see 12.7.2, “Optimizer directives” on page 391.
- Populate all columns in Explain tables: Select the **Populate all columns in the Explain tables** option if you want all the columns of the explain tables to be populated from the dynamic explain; otherwise, only the few columns needed by Visual Explain are populated from the dynamic explain.

In addition, you can open or save a SQL file by clicking the **Get** and **Save** buttons, respectively.

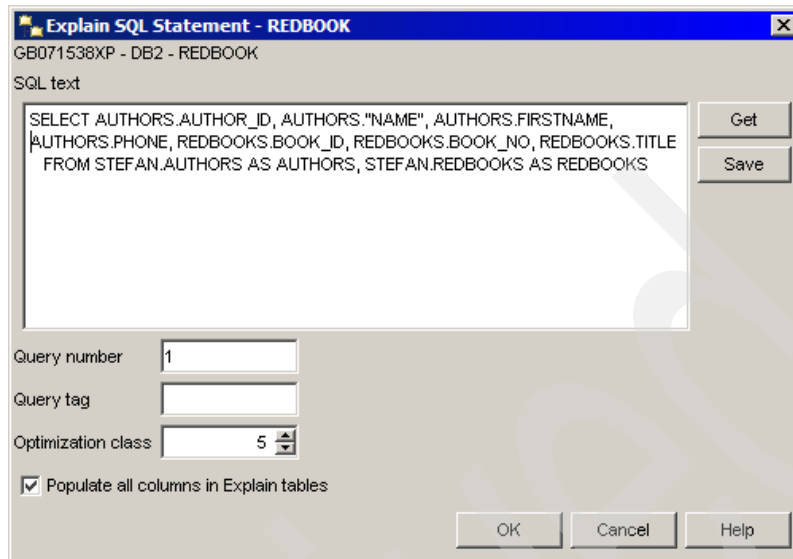


Figure 12-17 Explain SQL statement in Visual Explain

3. After you enter the SQL statement and set the parameters, click **OK** to retrieve the graphical presentation of the query plan, shown in Figure 12-8. The colored boxes represent each step in the data access path that the optimizer has chosen.

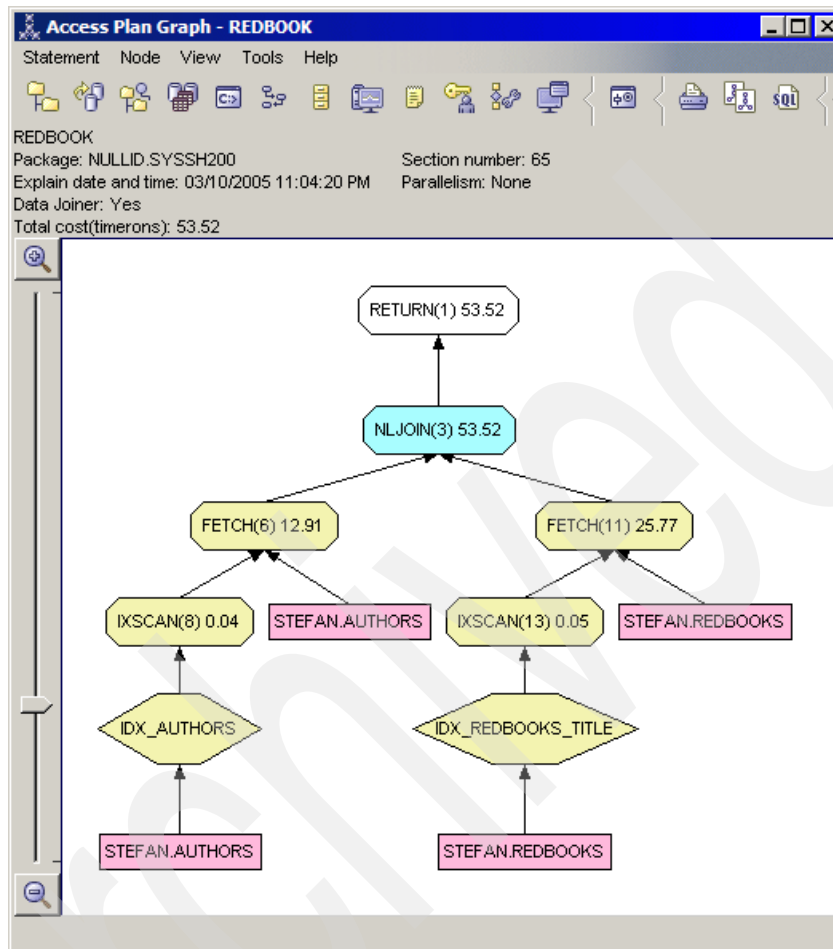


Figure 12-18 Query plan generated by Visual Explain

If you are interested in a particular step of the plan, double-click the corresponding shape in the plan. A detailed explanation of what is done in that step appears in a new window, as shown in Figure 12-19 on page 387.

Cumulative cost	
Total cost	12.91 timerons
CPU cost	156,108 instructions
I/O cost	1 I/Os
First row cost	12.87 timerons
Remote communication cost	0 timerons
Cumulative properties	
Tables	STEFAN.AUTHORS
Columns	STEFAN.AUTHORS.PHONE STEFAN.AUTHORS.FIRSTNAME STEFAN.AUTHORS.NAME STEFAN.AUTHORS.AUTHOR_ID
Order columns	None
Predicates	None
Cardinality	40
Total buffer pool pages used	2
Buffer pool usages	None
Input arguments	
Columns retrieved	STEFAN.AUTHORS.PHONE STEFAN.AUTHORS.AUTHOR_ID
Sargable predicates	None
Residual predicates	None
Block sargable predicates	None
Direct fetch	False
Prefetch	No prefetch
Maximum pages	All
Lock intents	Table: Intent Share Row: Next Key Share

Figure 12-19 Details of an optimizer step Visual Explain

The db2exfmt utility

To analyze query plans from the command line, use the **db2exfmt** utility. This utility provides you with the same information as Visual Explain, but the information is written to a flat file (or standard output).

Before you can begin analyzing an SQL statement with the **db2exfmt** utility, you have to create the explain plan first. This is done by appending the clause **EXPLAIN PLAN FOR** before the actual SQL statement.

The explain file that is generated by the **db2exfmt** utility contains very detailed information about the steps chosen by the optimizer. Most sections of the output are self-explanatory, especially if you are already familiar with terms such as *costs* or *join methods*.

For more information about the **db2exfmt** utility, consult the DB2 UDB documentation *Administration Guide: Performance*, SC09-4821.

Example 12-32 demonstrates the process used to create an explain file with the **db2exfmt** utility.

Example 12-32 Creating an explain plan with the db2exfmt utility

```
$ cat orditem.sql
EXPLAIN PLAN FOR
  SELECT o.order_num, customer_num, order_date
  FROM   orders o, items i
  WHERE  o.order_num = i.order_num
  ORDER BY ship_date DESC;

$ db2 -tsvf orditem.sql
EXPLAIN PLAN FOR SELECT o.order_num, customer_num, order_date FROM   orders o,
items i WHERE  o.order_num = i.order_num ORDER BY ship_date DESC
DB20000I The SQL command completed successfully.

$ db2exfmt -d ifmx2db2 -l -o orditem.expl
DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool
Connecting to the Database.
Connect to Database Successful.
Output is in orditem.expl.
Executing Connect Reset -- Connect Reset was Successful.
```

The db2expln utility

The purpose of the **db2expln** utility is primarily to extract query plans from packages.

Since the query plans of packages are created at compile time, there is the potential risk of reduced performance if the query plan becomes outdated. The **db2expln** utility is able to analyze the query plan of a package. It also reports the *isolation level* or *optimization class* of a package.

Example 12-23 demonstrates a small ESQL/C application that contains static SQL. The static SQL is place in a the bind file and optimized at compile time.

Example 12-33 ESQL/C sample application with a single, static SQL statement

```
#include <stdio.h>
#include <string.h>
#include <sql.h>

int main ()
```

```

{
    EXEC SQL BEGIN DECLARE SECTION;
    sqlint32 rowcount;
    char dbname[9];
    char uid[9];
    char pwd[9];
    EXEC SQL END DECLARE SECTION;

    struct sqlca sqlca;

    rowcount = 0;
    strcpy (dbname, "sample");
    strcpy (uid, "db2admin");
    strcpy (pwd, "db2udb");

    printf ("Conneting %s as user %s.\n", dbname, uid);
    EXEC SQL CONNECT TO :dbname USER :uid USING :pwd;

    EXEC SQL
    SELECT COUNT (*) INTO :rowcount FROM syscat.tables;

    printf ("syscat.table contains %d rows.\n", rowcount);

    EXEC SQL DISCONNECT ALL;
    return 0;
}

```

After the application has been precompiled, compiled, and bound to the database, you can run the **db2expln** utility, as shown in Example 12-34.

Example 12-34 Query plan of package “connect”

```
C:\>db2expln -d sample -schema tedwas -package connect -t
```

```

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL Explain Tool

```

```
***** PACKAGE *****
```

```
Package Name = "TEDWAS"."CONNECT" Version = ""
```

```

Prep Date = 2005/03/14
Prep Time = 16:28:35

```

```
Bind Timestamp = 2005-03-14-16.28.35.474000
```

```

Isolation Level      = Cursor Stability
Blocking              = Block Unambiguous Cursors

```

Query Optimization Class = 5

Partition Parallel = No

Intra-Partition Parallel = No

SQL Path = "SYSIBM", "SYSFUN", "SYSPROC", "TEDWAS"

----- SECTION -----
Section = 1

SQL Statement:

```
SELECT COUNT (*)INTO :H00001
FROM syscat.tables
```

Section Code Page = 1252

Estimated Cost = 0.036384

Estimated Cardinality = 1.000000

Access Table Name = SYSIBM.SYSTABLES ID = 0,2

| Index Scan: Name = SYSIBM.IBM137 ID = 6

| | Regular Index (Not Clustered)

| | Index Columns:

| | | 1: TID (Ascending)

| | | 2: FID (Ascending)

| #Columns = 0

| #Key Columns = 0

| | Start Key: Beginning of Index

| | Stop Key: End of Index

| Index-Only Access

| Index Prefetch: None

| Lock Intents

| | Table: Intent Share

| | Row : Next Key Share

| Sargable Index Predicate(s)

| | Predicate Aggregation

| | | Column Function(s)

| Aggregation Completion

| | Column Function(s)

| Return Data to Application

| | #Columns = 1

End of section

| | Predicate Aggregation

| | | Column Function(s)

```
Aggregation Completion
| Column Function(s)
Return Data to Application
| #Columns = 1
```

End of section

12.7.2 Optimizer directives

The database configuration parameter `DFT_QUERYOPT` defines how aggressively the optimizer behaves. This parameter sets the default optimization level for all queries against that database. Example 12-35 demonstrates how this parameter can be changed.

Example 12-35 Setting the query optimization level for the whole database

```
UPDATE DATABASE CFG FOR <dbname> USING DFT_QUERYOPT = [0, 1, 2, 3, 5, 7, 9]
```

Some queries might benefit from a higher level of optimization. The optimization level can also be set on a session basis with the `SET CURRENT QUERY OPTIMIZATION` command, as shown in Example 12-36.

Example 12-36 Setting the query optimization level for a single session

```
SET CURRENT QUERY OPTIMIZATION = [0, 1, 2, 3, 5, 7, 9]
```

The higher the value of the optimization class, the more aggressive the optimizer becomes.

Note: The `SET CURRENT QUERY OPTIMIZATION` statement only affects dynamic SQL statements. To specify an optimization level for static SQL statements, you must specify it at bind time using the `QUERYOPT` parameter in the `BIND` statement.

The following optimizer classes (levels) can be specified for query compilation:

Optimization class 0

This optimization class has the following characteristics:

- ▶ Minimal optimization.
- ▶ Non-uniform distribution statistics are not considered by the optimizer.
- ▶ Only basic query rewrite rules are applied.
- ▶ Greedy join enumeration occurs.
- ▶ Only nested loop join and index scan access methods are enabled.
- ▶ List prefetch and index ANDing are not used in generated access methods.

- ▶ The star-join strategy is not considered.

This class should only be used in circumstances that require the lowest possible query compilation overhead. Query optimization class 0 is appropriate for an application that consists entirely of very simple dynamic SQL statements that access well-indexed tables.

Optimization class 1

This optimization class has the following characteristics:

- ▶ Non-uniform distribution statistics are not considered by the optimizer.
- ▶ Only a subset of the query rewrite rules are applied.
- ▶ Greedy join enumeration occurs.
- ▶ List prefetch and index ANDing are not used in generated access methods although index ANDing is still used when working with the semi-joins used in star-joins.

Optimization class 1 is similar to class 0 except that merge scan joins and table scans are also available.

Optimization class 2

This class directs the optimizer to use a degree of optimization significantly higher than class 1, while keeping the compilation cost significantly lower than classes 3 and above for complex queries. This optimization class has the following characteristics:

- ▶ All available statistics, including both frequency and quantile non-uniform distribution statistics, are used.
- ▶ All query rewrite rules are applied, including routing queries to materialized query tables, except computationally intensive rules that are applicable only in very rare cases.
- ▶ Greedy join enumeration is used.
- ▶ A wide range of access methods are considered, including list prefetch and materialized query table routing.
- ▶ The star-join strategy is considered, if applicable.

Optimization class 2 is similar to class 5 except that it uses Greedy join enumeration instead of dynamic programming. This class has the most optimization of all classes that use the Greedy join enumeration algorithm, which considers fewer alternatives for complex queries, and therefore consumes less compilation time than classes 3 and above. We recommend class 2 for very complex queries in a decision support or online analytic processing (OLAP) environment. In such environments, specific queries are rarely repeated exactly,

so a query access plan is unlikely to remain in the cache until the next occurrence of the query.

Optimization class 3

This class requests a moderate amount of optimization. This class comes closest to matching the query optimization characteristics of DB2 for MVS/ESA™, OS/390, or z/OS. This optimization class has the following characteristics:

- ▶ Non-uniform distribution statistics, which track frequently occurring values, are used if available.
- ▶ Most query rewrite rules are applied, including subquery-to-join transformations.
- ▶ Dynamic programming join enumeration, as follows:
 - Limited use of composite inner tables
 - Limited use of Cartesian products for star schemas involving lookup tables
- ▶ A wide range of access methods are considered, including list prefetch, index ANDing, and star joins.

This class is suitable for a wide range of applications. This class improves access plans for queries with four or more joins. However, the optimizer might fail to consider a better plan that might be chosen with the default optimization class.

Optimization class 5

This class directs the optimizer to use a significant amount of optimization to generate an access plan. This optimization class has the following characteristics:

- ▶ All available statistics are used, including both frequency and quantile distribution statistics.
- ▶ All of the query rewrite rules are applied, including the routing of queries to materialized query tables, except for those computationally intensive rules that are applicable only in very rare cases.
- ▶ Dynamic programming join enumeration, as follows:
 - Limited use of composite inner tables
 - Limited use of Cartesian products for star schemas involving lookup tables
- ▶ A wide range of access methods are considered, including list prefetch, index ANDing, and materialized query table routing.

When the optimizer detects that the additional resources and processing time are not warranted for complex dynamic SQL queries, optimization is reduced.

The extent or size of the reduction depends on the machine size and the number of predicates.

When the query optimizer reduces the amount of query optimization, it continues to apply all the query rewrite rules that would normally be applied. However, it does use the Greedy join enumeration method and reduces the number of access plan combinations that are considered.

Query optimization class 5 is an excellent choice for a mixed environment consisting of both transactions and complex queries. This optimization class is designed to apply the most valuable query transformations and other query optimization techniques in an efficient manner.

Optimization class 7

This class directs the optimizer to use a significant amount of optimization to generate an access plan. It is the same as query optimization class 5 except that it does not reduce the amount of query optimization for complex dynamic SQL queries.

Optimization class 9

This class directs the optimizer to use all available optimization techniques. These include:

- ▶ All available statistics
- ▶ All query rewrite rules
- ▶ All possibilities for join enumerations, including Cartesian products and unlimited composite inners
- ▶ All access methods

This class can greatly expand the number of possible access plans that are considered by the optimizer. You might use this class to find out whether more comprehensive optimization would generate a better access plan for very complex and very long-running queries that use large tables. Use Explain and performance measurements to verify that a better plan has actually been found.

Figure 12-20 shows the impact on the resulting query plan using different optimization classes for a sample query.

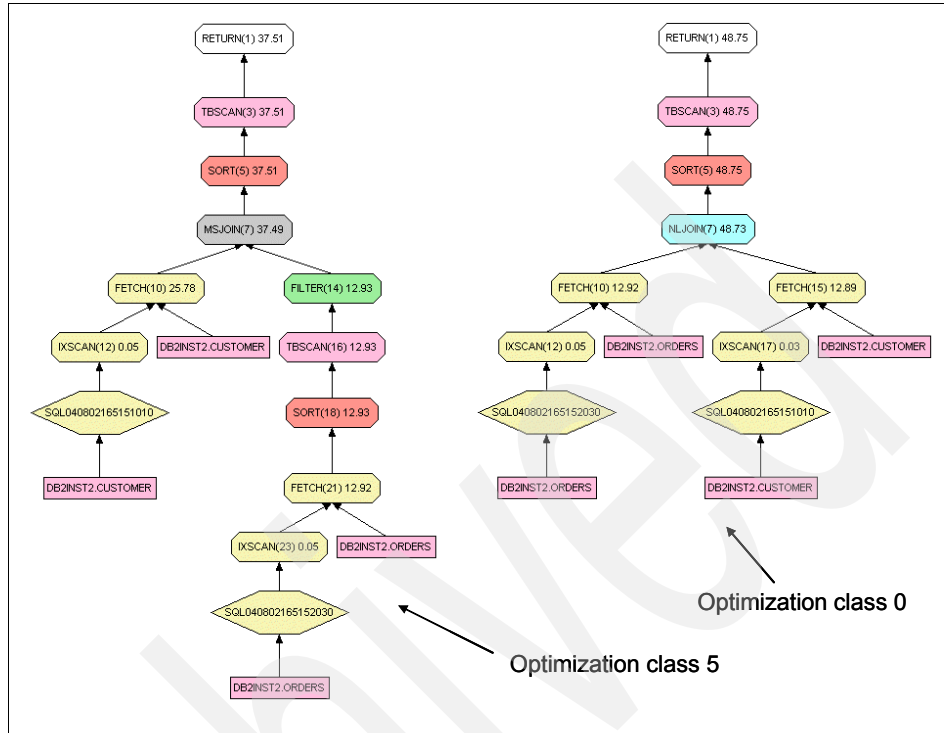


Figure 12-20 Comparison of query plans with optimization classes 0 and 5

Notice how the plan produced with an optimization class of 5 performs better than the plan produced with an optimization class of 0 (37.51 timerons versus 48.75 timerons).

Testing and troubleshooting

This chapter discusses how to verify that data and application functionality were ported completely and correctly.

The different phases of testing include:

- ▶ Test planning
- ▶ Data checking
- ▶ Code and application testing
- ▶ Troubleshooting

All stages of the migration process should be validated by running a series of carefully designed tests. The purpose of the tests is to determine the differences between the expected results in the source environment and the observed results in the migrated application. Any changes detected should be synchronized with the development stages of the project. This chapter describes the test objectives and a generic testing methodology, which can be employed to test migrated applications.

13.1 Planning your testing

The test planning details the activities, dependencies, and effort required to conduct the test of the converted solution.

13.1.1 Principals 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.

13.1.2 Test documentation

The test documentation is a very important part of the project. The *ANSI/IEEE Standard 829-1983 for Software Test Documentation* has a detailed description of software testing procedures. 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?

13.1.3 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 processes entail 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 may cause applications to function improperly.

Functional testing

Functional testing is a set of tests in which new and existing functionality of the system is 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.

13.1.4 Time planning and time exposure

The time planning should be based on realistic and validated estimates. If the estimates for the migration are inaccurate, the entire project plan will slip including testing.

It is always best to couple all test dates directly to their associated 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 13-1 shows the test phases during a typical migration project. The test cases, and all the following tasks, must be completed for all test phases.

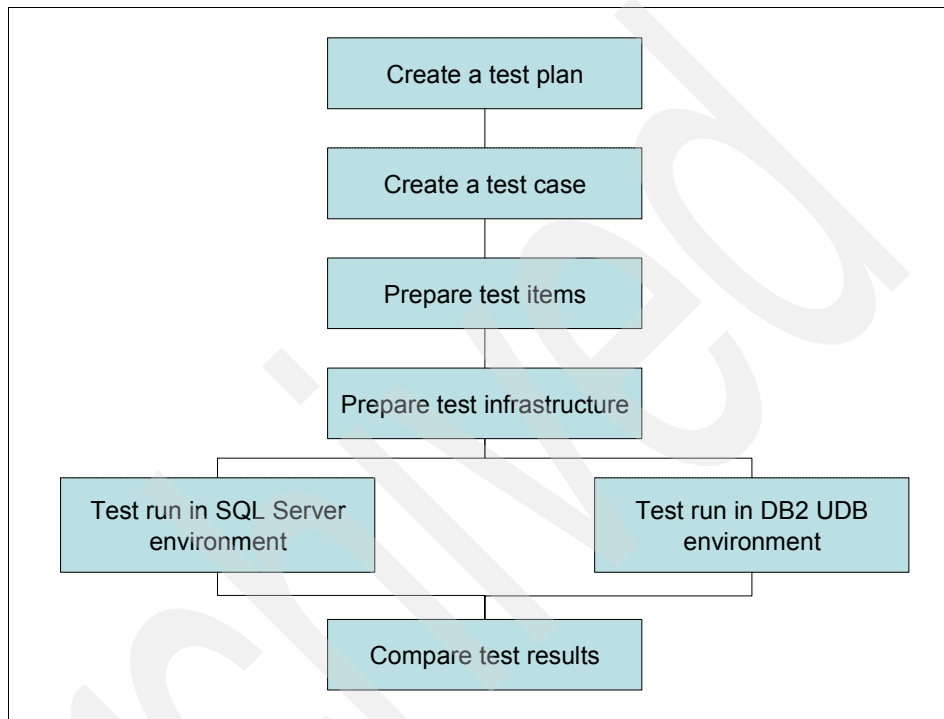


Figure 13-1 Test phases during a migration project

The time exposure of tests depends on the availability of an existing test plan and already prepared test items. The effort also depends on the degree of changes made during the application and database migration.

Note: Testing can occupy between 50% and 70% of the total migration effort.

13.2 Data checking techniques

Data migration must occur before any testing can take place. Tests should detect if all rows were successfully imported into the target database, and verify that data type conversions were successful. The data checking process can 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 database tables and compare them.
- ▶ Create scripts that automatically perform data checks.
- ▶ Train the administrative staff familiar with the application and its data to perform random data checks.

13.2.1 IMPORT/LOAD messages

You should always check the messages generated by the **IMPORT** or **LOAD** commands. Example 13-1 shows an example of the common messages generated by an **IMPORT** command. You should not only read the summary at the end of the listing, but also pay attention to any warning messages.

Example 13-1 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 "ITS0.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 above example, 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, look at the data source file and the table definition (**db2look** command). For Example 13-1, the table definition is presented in Figure 13-2, and the data file in Figure 13-3.

```
CREATE TABLE TABLE01 (  
  C1 SMALLINT,  
  C2 CHAR(3),  
  C3 SMALLINT CHECK( C3 IN (1,2,3)))
```

Figure 13-2 Table definition for Example 13-1

```
1,"abc",1  
2,"abc",4  
32768,"abc",2  
4,"abcd",3
```

Figure 13-3 Data file for Example 13-1

The first row from the input file (Figure 13-3) 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 the SMALLINT data type range (warning SQL3117W) and string abcd from the last row was truncated to abc because it was longer than the associated column definition (warning SQL3125W).

The LOAD utility generates messages in a similar format, but since it is designed for speed, it bypasses the SQL engine, and inserts data directly into table spaces without performing constraint checking. Inserting the same *table01.unl* file (Figure 13-3) into *table01* (Figure 13-2) with the LOAD utility generates messages without SQL3148W, SQL0545N, SQL3185W warnings, as shown in Example 13-2.

Example 13-2 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 in a *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 statement should be used to move a table into a usable state. Example 13-3 shows a sample invocation of this command. All rows that violate constraints are moved to exception table *table01_e*.

Example 13-3 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 such as only checking the integrity of new data, turning integrity checking off, or specifying exception tables with additional diagnostic information.

13.2.2 Data checking

Scripts that perform logical data integrity checks help automate the data verification process and save manual effort.

For small tables (less than 50,000 rows), you can write a script that compares data byte-by-byte. The script can extract sorted rows from SQL Server and DB2 UDB to files in the same ASCII format. The files can be binary compared to determine if they are the same.

For larger tables, comparing all rows byte-by-byte can be very inefficient. The data migration check should be evaluated by comparing aggregate values, such as 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 SQL Server database. Table CK_ROW_COUNT shown in Example 13-4 can be used for this purpose.

Example 13-4 Table for storing number of rows in SQL Server

```
CREATE TABLE CK_ROW_COUNT (
    tab_name VARCHAR(30), -- table name
    row_count INT, -- number of rows
    sys_name VARCHAR(5), -- code to distinguish the system: MSSQL or DB2
    time_ins DATETIME ) -- 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 dbo.CK_ROW_COUNT SELECT 'tablename', COUNT(*), 'MSSQL',
getdate() FROM dbo.tablename
```

The table CK_ROW_COUNTS *and its data* can be manually migrated to the target DB2 UDB database. Example 13-5 shows the DB2 UDB syntax to the table.

Example 13-5 Table for storing number of rows in DB2 UDB

```
CREATE TABLE CK_ROW_COUNT (
    tab_nameE VARCHAR(30),
    row_count INT,
    sys_name VARCHAR(5),
    time_ins TIMESTAMP
)
```

On the DB2 UDB system, you should submit the equivalent INSERT statement:

```
INSERT INTO ck_row_count SELECT 'TAB_NAME', count(*), 'DB2', CURRENT
TIMESTAMP FROM tab_name
```

After performing the steps described above, both CK_ROW_COUNT tables should contain information about the number of rows counted on SQL Server and DB2 UDB respectively. Example 13-6 shows what the table content should look like.

Example 13-6 Example of CK_ROW_COUNTS contents

```
SELECT tab_name, row_count, sys_nameE, time_ins FROM ck_row_count
[...]
TABLE_A 39001 MSSQL 2004-02-13-10.13.39
TABLE_A 39001 DB2 2004-02-13-10.32.13
TABLE_B 60003 MSSQL 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 table is very convenient, because with a single query such as:

```
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)
```

you can get the table names that contain a different number of rows in the source and target database.

13.3 Code and application testing

The most important part of the testing process is to verify that each component of the system functions as it did before migrating. All components of the DB2 UDB system should be verified.

13.3.1 T-SQL to SQL PL object check

All stored procedures, user defined functions, and triggers should be unit tested individually before they are promoted for further testing. This means that after objects are migrated (either manually, with MTK, or a combination of both) they should be checked to ensure they function properly. Basic problems can be discovered by running stored procedures through the Development Center.

Within the Development Center there are options to run selected procedures in debug mode (see Figure 13-4).

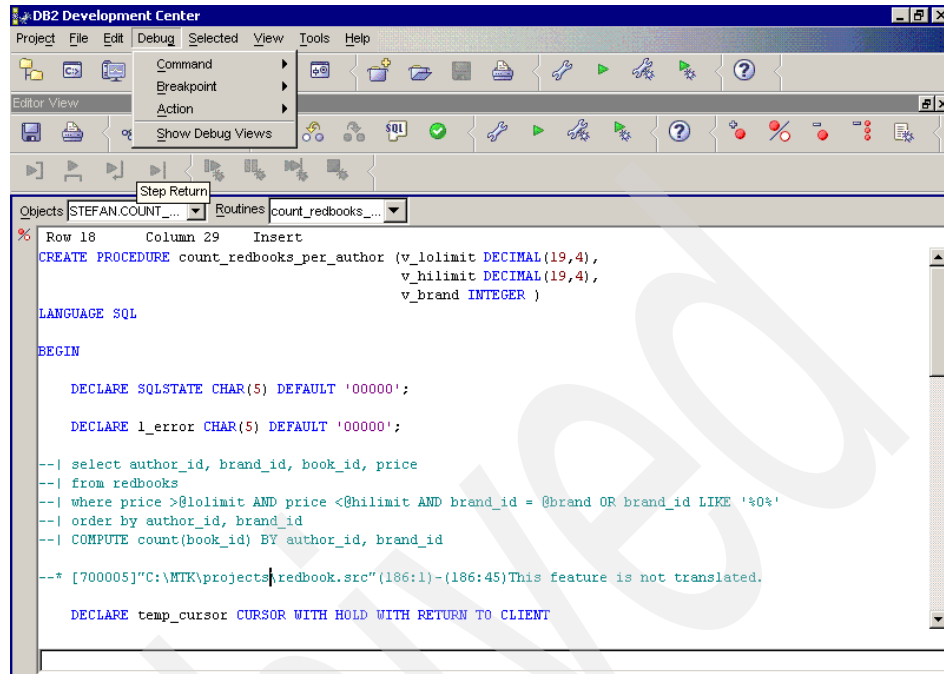


Figure 13-4 Development Center debug window

13.3.2 Application code check

The scope of application testing depends on the migrated application. All queries should be independently tested to ensure that they return the expected results. When the application queries are successfully migrated and tested, the surrounding client programs should be rebuilt, then tested against the target database. Each module of the application should be run and be checked for errors or improper functionality. All supported interfaces should also be checked.

You should document all test conditions and result, 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, creating documentation can become overwhelming. Typically, specialized software is used for these cases.

Application testing is an iterative process of planning, designing the test cases, executing the test cases, and evaluating the results.

Together with various functional testing, the application should also be checked for performance. Since there are many architectural differences between SQL Server and DB2 UDB, some SQL statements might require further optimization.

Observing the performance differences during the testing stages increases the chance to prepare more optimal code in the new environment.

Before going into production, the migrated database should be verified under high volume and loads that emulate the production environment. This can help determine if further application or database tuning is necessary. The stress load can also reveal other hidden problems, such as locking issues, which can typically only be observed in a production environment.

13.3.3 Security testing

Before moving a system into production, security must be checked in detail. Although SQL Server handles security differently than DB2 UDB there are some common factors. DB2 UDB uses the operating system to authenticate users, but SQL Server can also use specific SQL server user IDs for authentication. Privileges for all database objects also should be verified for each user.

13.3.4 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>

13.4 Troubleshooting

The first step of problem determination is to know what information and tools are 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 in diagnostic files, depending on the diagnostic level set in the DB2 UDB

configuration. In this section we discuss the DB2 UDB 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 UDB 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

13.4.1 Interpreting DB2 informational messages

Start your investigation using return codes. 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 of DB2 UDB error messages types:

- ▶ AMI: messages generated by MQ Application Messaging Interface
- ▶ ASN: messages generated by DB2 Replication
- ▶ CCA: messages generated by the Client Configuration Assistant
- ▶ CLI: messages generated by Call Level Interface
- ▶ DBA: messages generated by the Database Administration tools
- ▶ DBI: messages generated by installation and configuration
- ▶ DBT: messages generated by the Database tools
- ▶ DWC: messages generated by the Data Warehouse Center
- ▶ DB2: messages generated by the command line processor
- ▶ DLFM: messages generated by Data Links File Manager
- ▶ DQP: messages generated by Query Patroller
- ▶ GSE: messages generated by the DB2 Spatial Extender
- ▶ ICC: messages generated by the Information Catalog Center
- ▶ MQL: messages generated by MQ Listener
- ▶ SAT: messages generated in a satellite environment
- ▶ SPM: messages generated by the sync point manager
- ▶ SQL: messages generated by the database manager when a warning or error condition has been detected

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 UDB command: **db2 ? error-code**. See Example 13-7.

Example 13-7 Explaining DB2 UDB 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 the DB2 UDB documentation *Messages Reference*, Volumes 1 and 2, GC09-4840-00, and GC09-4841-00.

13.4.2 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 UDB problems, 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 UDB 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:

Windows:

<INSTALL PATH>\<DB2INSTANCE>

Where, <INSTALL PATH> is the directory where DB2 UDB is installed, and <DB2INSTANCE> is the name of DB2 UDB instance.

Linux and UNIX:

\$HOME/sqllib/db2dump

Where, \$HOME is the DB2 instance owner's home directory.

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 UDB 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 13-8 contains an extract of a db2diag.log taken at **DIAGLEVEL 3**.

Example 13-8 Example of db2diag.log file

```
(1)2005-02-15-13.51.33.328000-480 E1009655H459      (2)LEVEL: Warning
(3)PID      : 3356                               (4)TID   : 2472      (5)PROC   : db2syscs.exe
(6)INSTANCE: DB2                               (7)NODE  : 000      (8)DB     : SAMPLE
(9)APPHDL   : 0-277                             (10)APPID: *LOCAL.DB2.050215215129
(11)FUNCTION: DB2 UDB, data management, sqlEscalateLocks, probe:3
(12)MESSAGE : ADM5502W The escalation of "326" locks on table
               "SYSTOOLS.STMG_OBJECT" 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) Type (level) of log entry
- ▶ (3) Process ID of the application or agent
- ▶ (4) Thread ID of the application or agent
 - This is only used on the Windows platform.
- ▶ (5) The Process name of the application or agent
- ▶ (6) The name of the Instance
- ▶ (7) Node or partition number
 - This number is always 0 in a single partition configuration.
- ▶ (8) Name of the database
- ▶ (9) The application handle
- ▶ (10) The application ID
 - This corresponds to the LIST APPLICATIONS command output, each application has a unique application ID.
- ▶ (11) Name of the function, component and probe point in the function
 - This corresponds to a location in the source code of the function that has returned an error or information.
- ▶ (12) Diagnostic information

The above example shows a administration warning about lock escalation (326 row locks were successfully replaced by one table lock) on table SYSTOOLS.STMG_OBJECT.

Notify files

DB2 UDB also records diagnostic information in the administration notification log in the case of a failure. It is located in the same directory as the db2diag.log file. On Windows, all administration notification messages are written to the Event Log. On Linux and UNIX platforms, the administration notification log is a text file called *<instance>.nfy*, where *<instance>* is the name of the instance.

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

Other components such as the Health Monitor, the Capture and Apply programs, and user applications using the db2AdminMsgWrite API function can also write to the notify logs.

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 UDB signal handler, a trap file is generated in the DB2 UDB diagnostic directory. The files are created use the following naming convention:

Windows:

- ▶ DBpppttt.TRP
 - ppp : The process ID (PID)
 - ttt : The thread ID (TID)

Example: DB123654.TRP

Linux and UNIX:

- ▶ tpppppp.nnn
 - pppppp: The process ID (PID)
 - nnn: The node where the trap occurred
 - Example: t123456.000

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 for additional diagnostics, to a

complete DB2 UDB instance shutdown due to a serious internal or external problem.

Dump files

When DB2 UDB determines that internal information needs to be collected, it will often create binary dump files in the diagnostic path. These files are generated in the following format:

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

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

Messages files

Some DB2 UDB utilities like **BIND**, **LOAD**, **EXPORT** and **IMPORT** provide an option to write 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.

DB2 problem determination tutorials

The problem determination tutorials are designed to assist you in developing and supporting a DB2 UDB database environment. You can gain familiarity with the DB2 UDB problem determination techniques and tools available.

Topics that are covered include:

- ▶ Installation
- ▶ Connectivity
- ▶ Problem Determination Tools
- ▶ Database Engine
- ▶ Performance
- ▶ Problem Determination in a Multi-Node Environment
- ▶ Applications
- ▶ DB2 and OS Diagnostics

The tutorials can be downloaded from:

<http://www.ibm.com/software/data/support/pdm/db2tutorials.html>

13.4.3 DB2 support information

Identifying what information is required to solve a problem 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 shown in Example 13-9, 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. We recommend to periodically check if the newest available FixPaks are installed. DB2 UDB maintenance updates are freely available at:

<ftp://ftp.software.ibm.com/ps/products/db2/fixes>

Example 13-9 DB2LEVEL output

```
C:\>db2level
DB21085I  Instance "DB2" uses "32" bits and DB2 code release "SQL08021" with
level identifier "03020106".
Informational tokens are "DB2 v8.1.8.762", "s041221", "WR21348", and FixPak
"8".
Product is installed at "C:\SQLLIB\".
```

db2support utility

The `db2support` utility is designed to automatically collect all DB2 UDB and system diagnostic data. This program generates information about a DB2 UDB 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.

In one simple step, the tool can gather database manager snapshots, configuration files, trap and dump files, and operating system parameters, which should make the problem determination quicker. Below is sample command syntax to run the utility:

```
db2support . -d sample -c
```

The dot represents the current directory where the output file is to be stored. Other parameters are optional. the **-d** and **-c flags** instruct the utility to connect

to the SAMPLE database, and to gather information about database objects such as table spaces, tables, and buffer pools.

Note: To obtain the most complete output from the db2support command we recommend that you run the command as the instance owner.

DB2DIAG utility

DB2 UDB has a new message format in V8.2 and the **db2diag** utility is a tool used for filtering and formatting db2diag.log files. It's a command line tool that has a range of options designed to assist with problem determination using the db2diag.log.

For example, the following command syntax will return *severe* issues reported against the REDBOOK database:

```
db2diag -gi db=REDBOOK -l Severe
```

There are many command options available for **db2diag**. Listed below are some common ones useful for obtaining help and viewing examples.

- ▶ db2diag -help
 - provides a short description of the options
- ▶ db2diag -h brief
 - provides a description of all the options without examples
- ▶ db2diag -h notes
 - provides usage notes and restrictions
- ▶ db2diag -h examples
 - provides a small set of examples to get started
- ▶ db2diag -h tutorial
 - provides examples for all available options
- ▶ db2diag -h all
 - provides the most complete list of options

DB2PD utility

A new utility that can be used to retrieve statistics from a running DB2 UDB instance or database is **db2pd**.

The tool can provide a wide range of information useful for troubleshooting and problem determination, performance improvements, and application development design, including:

- ▶ locks
- ▶ bufferpools
- ▶ tablespaces
- ▶ containers
- ▶ dynamic SQL statements
- ▶ agents
- ▶ applications
- ▶ memory pools and sets
- ▶ transactions
- ▶ logs

Figure 13-10 shows how you can use the tool to return information about the buffer pools.

Example 13-10 DB2PD tool usage

```
C:\>db2pd -db redbook -bufferpools
```

```
Database Partition 0 -- Database REDBOOK -- Active -- Up 0 days 00:14:55
```

```
BufferPools:
```

```
First Active Pool ID      1
Max Bufferpool ID          2
Max Bufferpool ID on Disk  2
Num Bufferpools            6
```

Address	Id	Name	PageSz	PA-NumPgs	BA-NumPgs
BlkSiz					
e ES NumTbsp		PgsLeft	CurrentSz	PostAlter	SuspdnTSCt
0x04018720	1	IBMDEFAULTBP	4096	250	0
N 5		0	250	250	0
0x040189B0	2	MTK32K	32768	30	0
N 3		0	30	30	0
0x0293BAF0	4096	IBMHIDDENBP4K	4096	16	0
N 0		0	16	16	0
0x0293BD80	4097	IBMHIDDENBP8K	8192	16	0
N 0		0	16	16	0
0x04018200	4098	IBMHIDDENBP16K	16384	16	0
N 0		0	16	16	0
0x04018490	4099	IBMHIDDENBP32K	32768	16	0
N 0		0	16	16	0

DB2 Technical Support site

An invaluable place to look for support the *DB2 Technical Support Site for Linux, Windows, and UNIX* located on the Web at:

<http://www.ibm.com/software/data/db2/udb/support>

The site has the most recent copies of the documentation, a knowledge base to search for technical recommendations or DB2 UDB defects, links to product updates, the latest support news, and many useful DB2 UDB related links.

To search for related problems in the DB2 Knowledge Base, use keywords that may include the command that was run, the symptoms, and tokens from the diagnostics messages. The Knowledge Base offers an option to search through DB2 UDB documentation, TechNotes, and DB2 UDB defects (APARs).

TechNotes are a set of recommendations and solutions for specific problems.

Authorized Program Analysis Reports (APARs) are defects in the DB2 UDB code 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 on multiple platforms. Fixes for APARs are provided through DB2 UDB FixPaks.

On the DB2 Support Site there is also the possibility to search for closed, open, and HIPER APARs. A *closed* APAR indicates a resolution for the problem has been verified and included in FixPaks. *Open* APARs represent DB2 UDB defects that are currently being worked on or are 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 is 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>

Conversion scenario

This chapter presents a complete conversion scenario of a SQL Server database called REDBOOK using the IBM DB2 Migration Toolkit (MTK). The REDBOOK database is small, but is designed to make the conversion somewhat complex by including some features that are not directly portable to DB2 UDB. It purposefully exposes you to common conversion issues, then shows you how to use MTK effectively and work around them. It is important to remember that MTK is not a “magic box” that can convert everything in one pass. Refinements and customizations are usually needed.

While no scenario can prepare you for all the issues you may potentially encounter during a conversion, we show you how to manage different *classes* of problems. Therefore, your goal in following along with this scenario should be to *understand* the reasoning of each step in the process and the lessons being taught.

This scenario covers the following activities:

- ▶ Set up the conversion environment
- ▶ Create an MTK project
- ▶ Core database extraction and deployment
- ▶ Other database object conversion
- ▶ Application conversion

14.1 Set up the conversion environment

The environment used in this scenario comprises two IBM xSeries machines. Each machine meets the minimum hardware requirements for each of the software applications being used. The first machine, *Boron*, is designated as the SQL Server source. The second system, *Lead*, is designated as the target DB2 UDB system.

On the source system (*Boron*), the following products are installed and the system is configured as follows:

- ▶ Windows 2000 Server with Service Pack 4.
- ▶ Microsoft SQL Server with Service Pack 3a.
 - REDBOOK database

The REDBOOK database is a small database containing tables, views, stored procedures, triggers, and user defined functions. Although it is small, the objects in it include features that are not directly portable to DB2 UDB, thus making the conversion non-trivial. A script to create the REDBOOK database is available for download. Refer to Appendix G, “Additional material” on page 509 for more information.

- ▶ DB2 UDB Application Development Client Version 8.1 with FixPak 8.
- ▶ MTK Version 1.3.
- ▶ A Data Source Name (DSN) called SQLServer has been created for SQL Server.

When accessing SQL Server databases from MTK, an ODBC DSN must be created. This can be configured from the ODBC Data Source Administrator panel in Windows 2000, available from **Start → Settings → Administrative Tools → Data Sources (ODBC)**.

- ▶ A DB2 UDB database, also called REDBOOK, residing on the target system (*Lead*) has been cataloged. This task can be performed once the REDBOOK database is created on *Lead*.

On the target system (*Lead*), the following products are installed and the system is configured as follows:

- ▶ Windows 2000 Server with Service Pack 4
- ▶ DB2 UDB Enterprise Server Edition Version 8.1 with FixPak 8
- ▶ A DB2 UDB database called REDBOOK has been created using the command:
CREATE DATABASE redbook

14.2 Create an MTK project

Before we can begin the conversion using MTK, a new project must be created in MTK. The following steps describe how to create a new MTK project for this scenario:

- 1. Launch MTK. Go to **Start → Programs → IBM DB2 Migration Toolkit 1.3 → Toolkit**

This launches the *MTK welcome* window (Figure 14-1). Click **Launch the DB2 Migration Toolkit product** to load MTK.

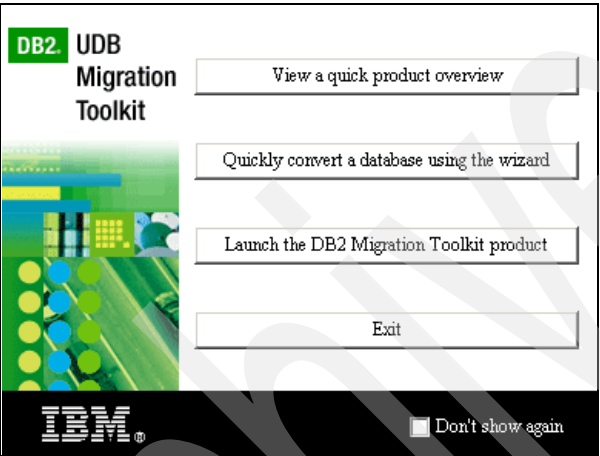


Figure 14-1 MTK - Welcome window

A progress window is displayed while MTK loads. If you select the **Don't show again** checkbox on the welcome window, the next time MTK starts, the toolkit is launched immediately.

- 2. Setup a new project in MTK. In the *Project management* dialog window (Figure 14-2), under the *New Project* tab, enter the project information contained in Table 14-1 and click **OK**. This creates a folder with the given project name under the C:\MTK\projects folder.

Table 14-1 Input to MTK project management dialog

Input Parameter	Value
Project Name	redbook
Project Path	C:\MTK\projects
Project description	This is the conversion scenario for the REDBOOK database.

Input Parameter	Value
Source database	Microsoft SQL Server
DB2 UDB Platform and Version	8.2 for Linux, UNIX and Windows

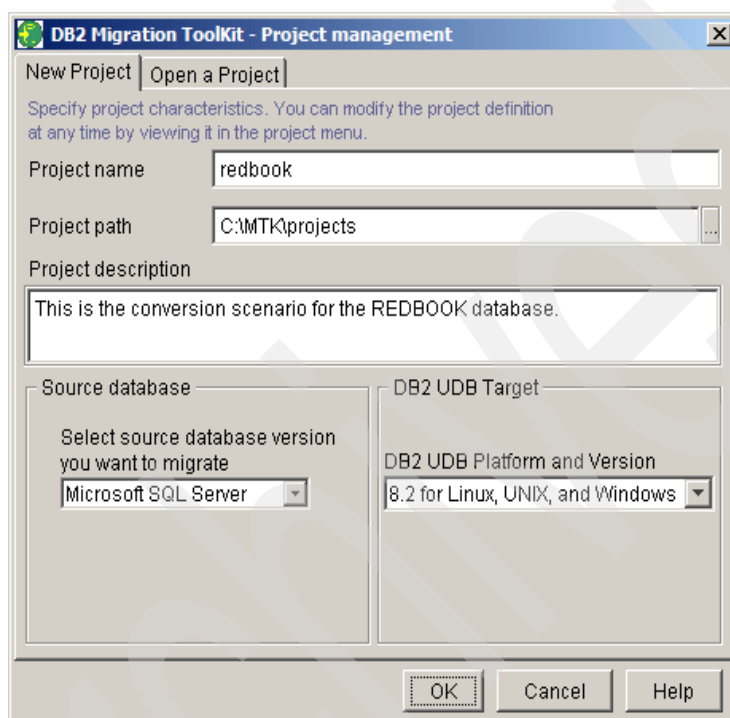


Figure 14-2 MTK - Project Management dialog window

14.3 Core database extraction and deployment

Once the project is created, you are presented with the main MTK user interface window. There are five tabs at the top of the window:

- ▶ Specify Source
- ▶ Convert
- ▶ Refine
- ▶ Generate Data Transfer Scripts
- ▶ Deploy to DB2

Each tab represents a phase of the MTK conversion process. The database conversion process using MTK is summarized in Figure 14-3.

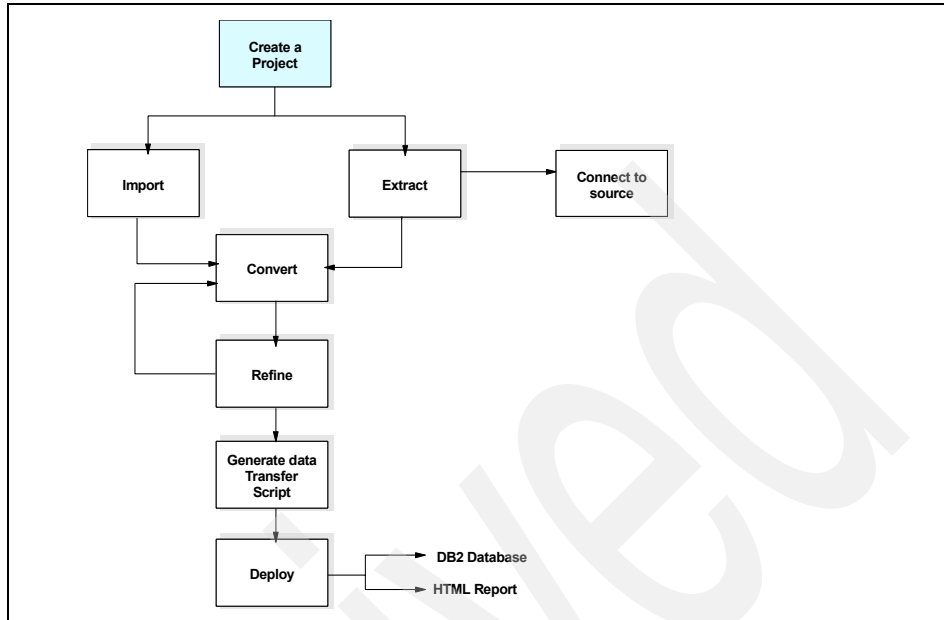


Figure 14-3 Database conversion process using MTK

We now review each step in the process.

14.3.1 Specify Source

On the *Specify Source* tab (Figure 14-4), there are two options:

- ▶ Import
- ▶ Extract

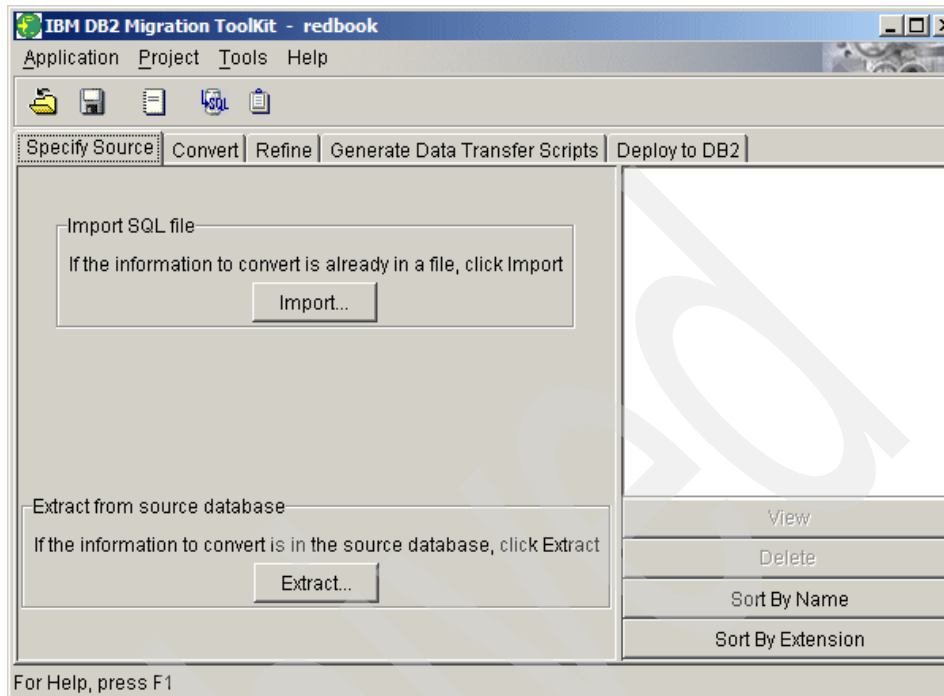


Figure 14-4 MTK - Specify Source tab

MTK can either convert a database from an existing SQL Server script or connect to a SQL Server database to extract the information it needs. By using the **Extract** option, it is possible to:

- ▶ Extract only a subset of objects into their own file (e.g., extract tables into their own file, extract triggers into their own file, etc.).
- ▶ Automatically resolve dependencies (e.g., when extracting a procedure, also extract all the dependent objects, such as tables).
- ▶ Extract each stored procedure and trigger into its own file.

In this scenario, we extract the information directly from the SQL Server REDBOOK database.

Note: We could have also used the REDBOOK database creation script provided as the input conversion source (**Import** option). However, during your conversion, you will likely be extracting the information directly from a SQL Server database.

1. Click **Extract**. In the *Connect to Database* dialog window (Figure 14-5) that appears, provide the connection details for the SQL Server REDBOOK database. In our environment, we created a DSN called SQLServer. We also use the user ID tedwas and the associated password for this user. The values you use in your environment will depend on how you configured your DSN as well what user accounts exist. Fill in the appropriate information and click **OK**.

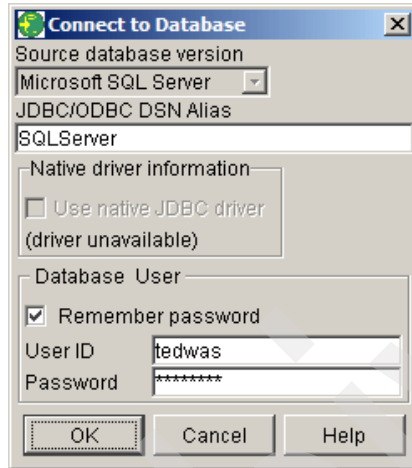


Figure 14-5 MTK - Connect to Database dialog window

2. Upon successful connection, the *Extract* dialog window (Figure 14-6) appears, which shows the tree of the available SQL Server databases. Expand the REDBOOK database option and select the objects you want to convert. In this step, we select all the tables under dbo schema. Enter tables in the *File name* field. The SQL Server table and index definitions are saved in this file. Select the **do not include** option under the *Include other needed objects* section. (Note that it is also possible to extract dependent object definitions in the same file, or a different file). It is also possible to extract GRANT statements and specify whether quoted identifiers should be used (i.e., if the objects being converted have any space characters in their names). We do not use these options in this scenario.

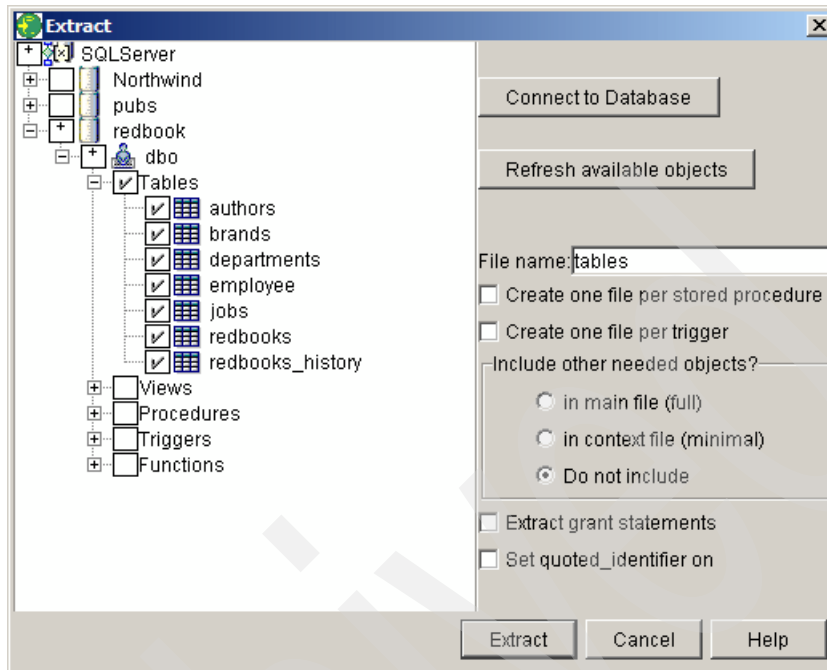


Figure 14-6 MTK - Extract dialog window for tables

Tip: Although it is possible to extract all the object definitions at once, we highly recommend extracting each set of objects (tables, views, procedures, and triggers) separately. This makes the conversion process more structured and manageable.

Click **Extract** to begin the extraction process. MTK begins extracting the table and index object definitions from the SQL Server REDBOOK database. Note that this step may take a while if the source database has a lot of objects.

3. Once the object definitions are extracted, the newly created script(s) appear in the right pane of *Specify Source* tab (Figure 14-7).

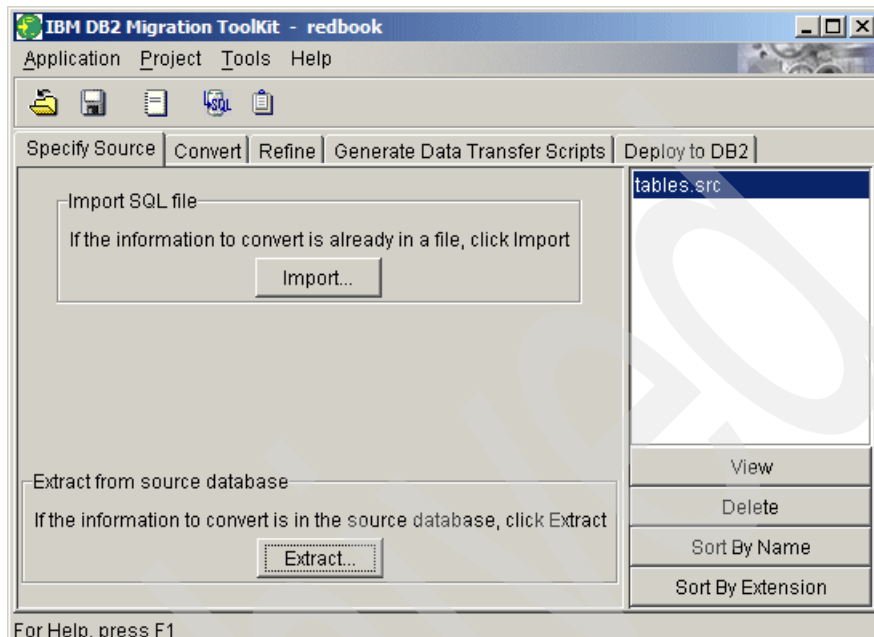


Figure 14-7 MTK - *Specify Source* tab with extracted table script selected

Take a moment to familiarize yourself with the contents of the script by clicking the **View** button.

14.3.2 Convert

Customize how MTK converts the extracted object definitions to DB2 UDB.

1. Select the *Convert* tab (Figure 14-8). On the left window pane, a listing of all source (input) files is provided. In the middle pane, the **Convert** button initiates the conversion process of the *currently selected source file*. The right pane contains the translated (output) files. Click the **View Output File** button in the right pane to view the contents of the converted file. The contents of the file open in a text editor. Ensure you close the file in the text editor before proceeding to the next step.

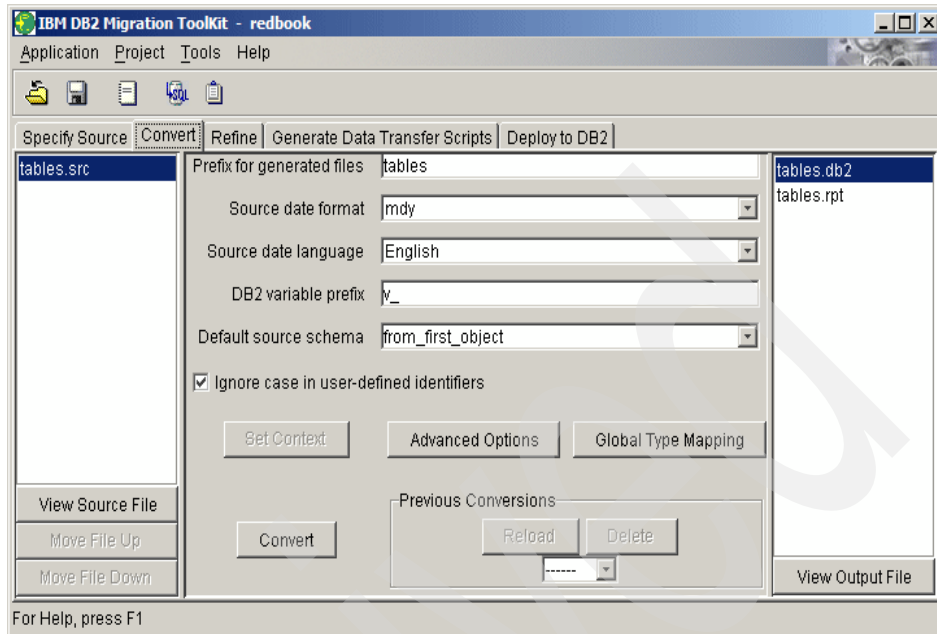



Figure 14-8 MTK - Convert tab after converting tables

2. Click the **Global Type Mapping** button to open the *Global Type Mapping* window (Figure 14-9). In this window, you can specify the data type mapping you want to use between SQL Server data types and DB2 UDB data types. In the *Target type* column, you can modify a target DB2 UDB data type by clicking the  icon next to the type. Alternatively, you can also click in the target cell you want to change. A *Data Type Editor* dialog window is displayed (Figure 14-10) where you can specify the target DB2 UDB data type and its attributes. Click **Apply** to apply your changes. In this example, we change all GRAPHIC and VARGRAPHIC data types to CHAR and VARCHAR, respectively. Clicking the **Restore Previous Mapping** button re-loads the mapping used in the last conversion activity.

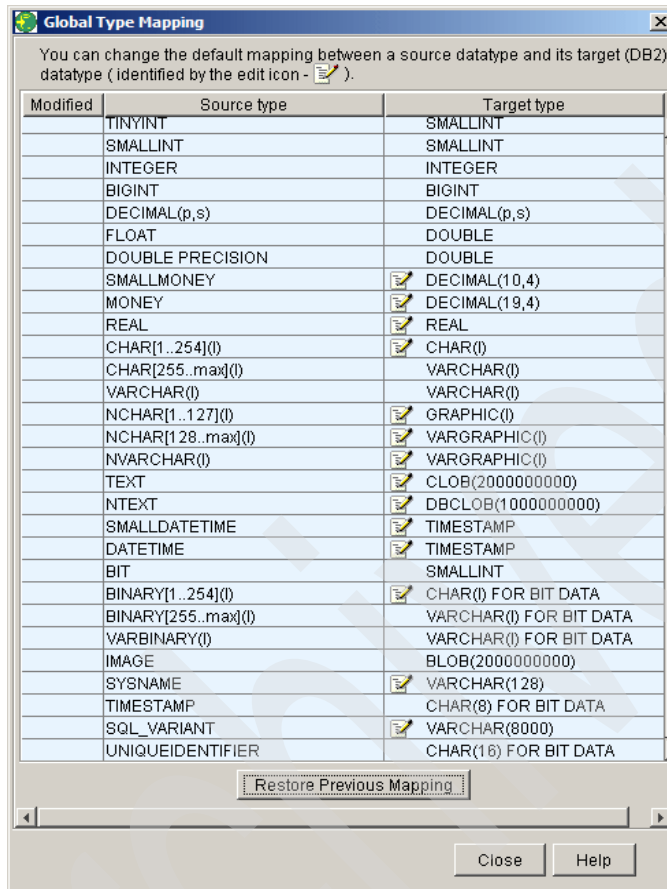


Figure 14-9 MTK - Global Type Mapping window

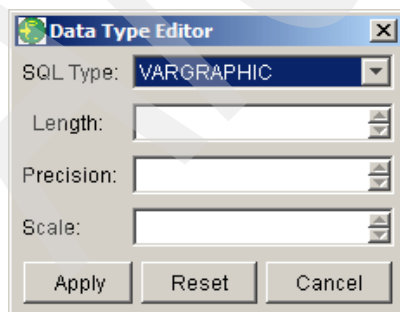


Figure 14-10 MTK - Data Type Editor dialog window

- After we perform the conversion described above, the *Global Type Mapping* window appears like Figure 14-11. Notice how the rows where the mapping is changed appear with a checkmark in the *Modified* column. Click **Close** to close this window and return to *Convert* tab.

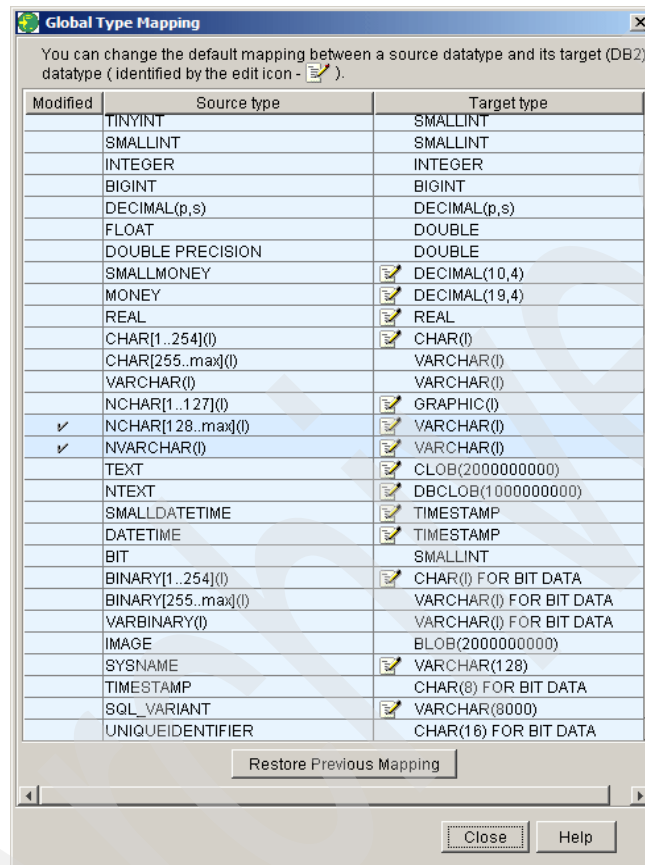


Figure 14-11 MTK - Global Type Mapping changes

- Click the **Advanced Options** button. This opens the *Advanced Options* window (Figure 14-12) where advanced conversion properties can be set, such as how the original source code should be copied into the converted file.

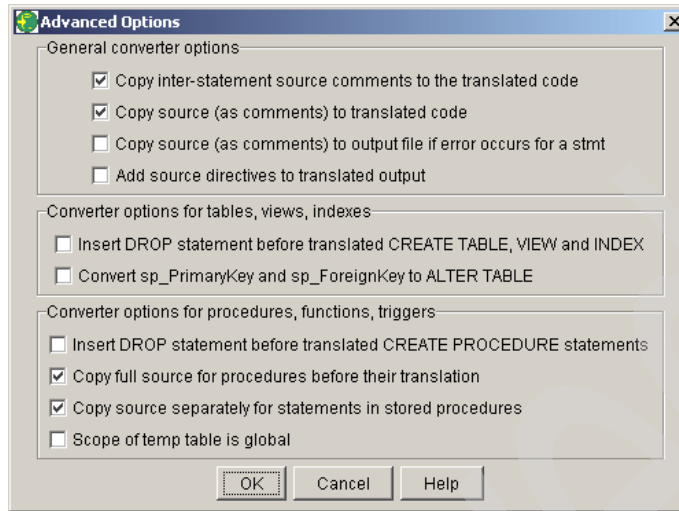


Figure 14-12 MTK - Advance Options window (for conversions)

In this scenario, we use all the default options. Click **OK** to go back to the *Convert* tab.

5. In the *Convert* tab, click **Convert** to begin converting the SQL Server table and index definitions to DB2 UDB. When the conversion completes, MTK automatically advances to the *Refine* tab.

14.3.3 Refine

During the refine stage, you have the opportunity to review any errors, warnings and other information about the conversion.

1. The *Refine* tab (Figure 14-13) has two main panels. The left panel contains a *messages* tree. The right panel contains additional information about the currently selected node in the messages tree. When the root tree node is selected, a summary of the errors encountered is displayed (listed in order of decreasing importance). This summary information is typically used to target the most critical conversion issues first.

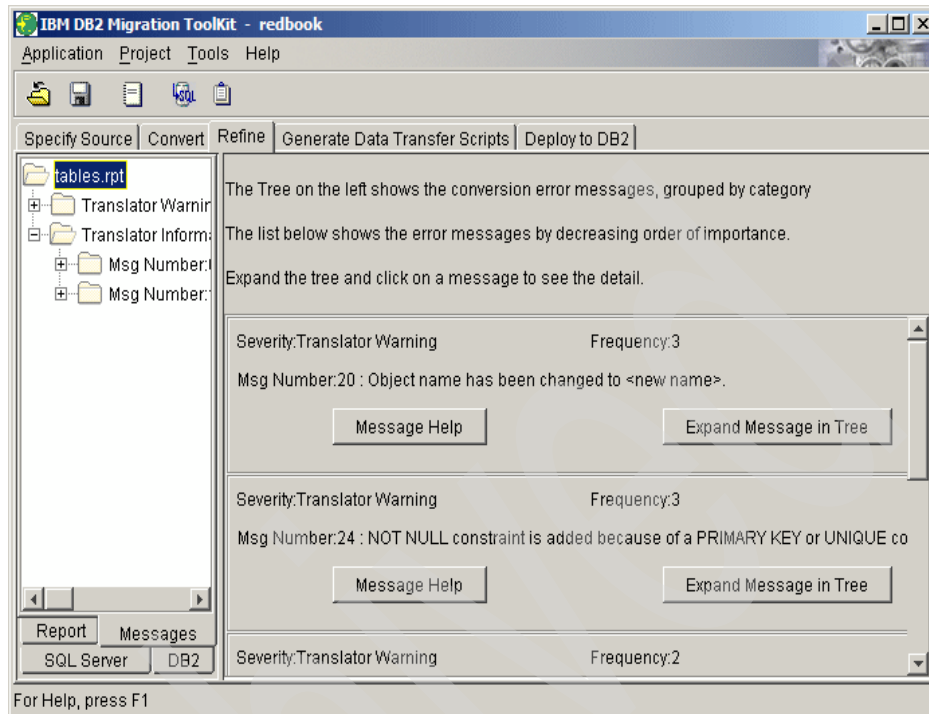


Figure 14-13 MTK - Refine tab

2. Expand the messages tree, as shown in Figure 14-14.



Figure 14-14 MTK - Translator Information message tree (for tables)

This general message displays the version of MTK being used: **041209.0332**. The version number you see may vary from this. No action is required here.

3. Expand the *Translator Warning* message tree as shown in Figure 14-15.

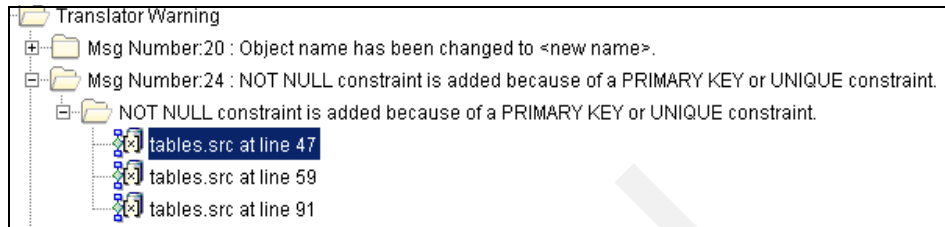


Figure 14-15 MTK - Translator Warning message tree (for tables)

This message is commonly encountered in conversions to DB2 UDB; columns with a UNIQUE constraint must be defined as NOT NULL in DB2 UDB. SQL Server constraints allow nullable columns. MTK has detected this and added a NOT NULL constraint to the converted code.

In many cases, converting a column to NOT NULL is not a serious problem, especially when it is a primary key column. However, you should determine this on your own based on your knowledge of the application and database.

For now, we assume that this conversion is satisfactory and we simply ignore this message. The purpose of this step was to simply introduce this commonly encountered message and help you understand its implications.

4. Expand the *Translator Warning* message tree as shown in Figure 14-16.

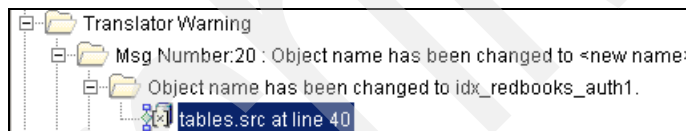


Figure 14-16 MTK - Translator Warning message tree (for tables)

On the right pane of the *Refine* tab, you should see two sub-tabs, as shown in Figure 14-17.

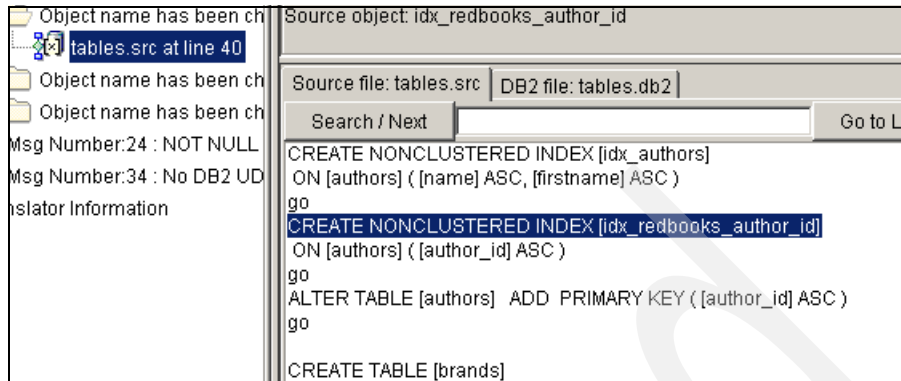



Figure 14-17 MTK - Refine tab including source and target code subtabs, for tables

Click the source SQL Server file tab (Source file: tables.src) to see the problematic code and the target DB2 UDB file tab (DB2 file: tables.db2) to see how MTK converted it.

This is another common warning message encountered in conversions to DB2 UDB. Index names in DB2 UDB are limited to 18 characters, so MTK suggests a new name for the index. The original index name was `idx_redbooks_author_id`. MTK renamed this index to `idx_redbooks_auth1`.

Usually, the renaming of indexes is harmless since applications generally do not need to reference indexes by name. However, you might have a standard naming convention for object names and the name suggested by MTK might be inappropriate. You are able to provide a different name for the converted index by completing the following steps:

- Click the **Go to Source Object** button in the top right pane of the *Refine* tab. This displays a panel where you can see both the original object name and the renamed DB2 UDB object.
- Click the  icon beside the recommended DB2 UDB object name and a dialog window appears where you are able to provide a new name for the object.
- In this example, we rename the new DB2 UDB index `redbook_aid` and click **Apply** to apply the new name. Notice how the `idx_redbooks_author_id` index is now mapped to `redbook_aid` (Figure 14-18).

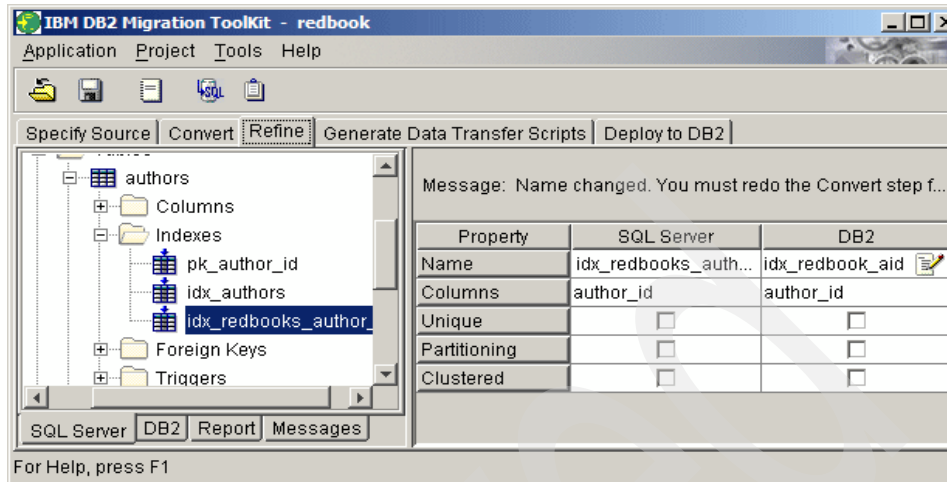


Figure 14-18 MTK - Refine tab (change recommended object name for indexes)

- Return to the previous view of the *Refine* tab by selecting the *Messages* tab in the tab panel underneath the messages tree. You can always return to the current screen (Figure 14-18) by selecting the *SQL Server* tab.
5. Now that you are back at the *Messages* tab, notice how the index name in the DB2 file has not changed. For the change to take effect, we must convert the file again by:
 - a. Selecting the *Convert* tab at the top of the MTK window
 - b. Clicking the **Convert** button

Clicking the **Convert** button causes the original source file to be re-read and applies any changes defined to generate a new tables.db2 file. The previous tables.db2 file is discarded and a new tables.db2 file is generated. If you expand the *Messages* tree once more, the warning message (Msg Number 20) about object name change for this particular index no longer exists.
 6. Follow the procedure described above to rename any other objects in which the MTK-generated name is not preferred.
 7. Ensure all other messages are read and cleared if possible. Expand the messages tree again, as shown in Figure 14-19.

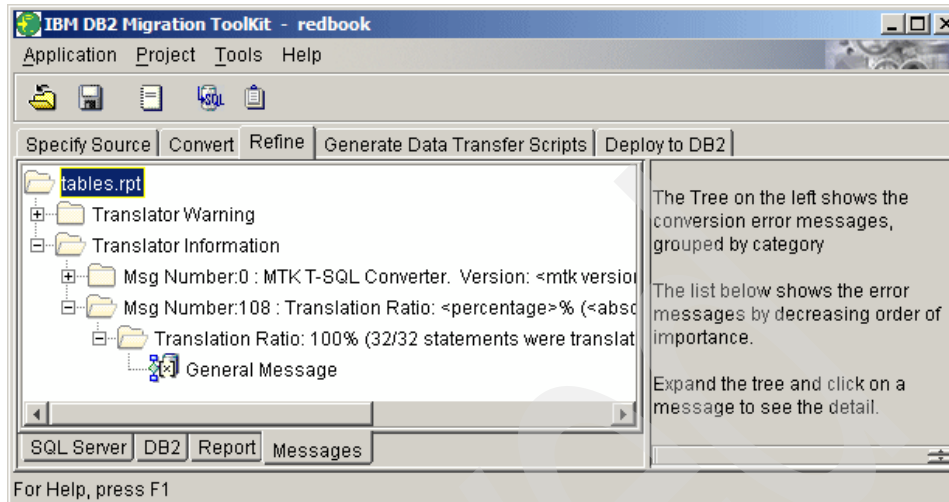


Figure 14-19 MTK - Refine tab (Translation success ratio)

Observe that the script translation ratio is 100%. That is, 100% of the statements in the input script are accounted for and converted without error. Warnings are not critical and are not included in the calculation.

Other important observations

- ▶ All DB2 UDB scripts generated by MTK use the exclamation mark (!) as a statement delimiter. These scripts can be run in CLP or Command Editor.
- ▶ It is important to keep detailed notes during the conversion process, such as, the default assignment of tables and indexes to table spaces. At some point, usually during a performance tuning exercise, you can determine the most appropriate table space(s) and buffer pool(s) to create. Initially, the focus should be on producing a clean DB2 UDB object creation script to form the foundation for all subsequent converted objects.

14.3.4 Generate Data Transfer Scripts

The *Generate Data Transfer Scripts* tab (Figure 14-20) is used to generate:

- ▶ Scripts which can later be used to extract data to flat files from a SQL Server database
- ▶ Scripts to LOAD/IMPORT data from flat files into a DB2 UDB database.

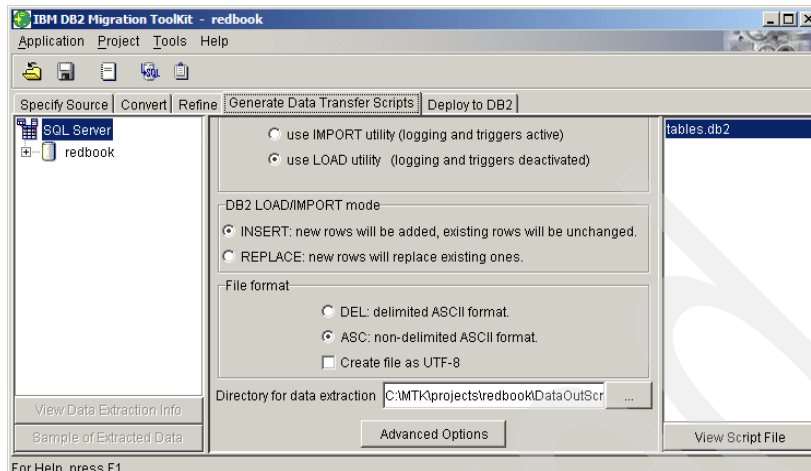


Figure 14-20 MTK - Generate Data Transfer Scripts tab

In your environment, if SQL statements are used to “seed” or populate the database, then you should convert statements using MTK instead.

A common misconception is that data is moved from SQL Server to DB2 UDB in this step. In fact, no data is actually moved in this step - only the scripts to perform the data extraction and deployment are generated.

This step creates scripts that can be used by the current machine or on another machine to perform the deployment. This is very useful when the volume of the data precludes using the machine where MTK is installed.

DB2 UDB Data Loading Options

There are two utilities available to load data from flat files into DB2 UDB tables, IMPORT and LOAD. MTK supports both (Figure 14-21), and they are described briefly in Table 14-2.

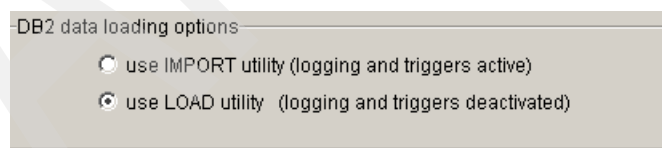


Figure 14-21 MTK - Generate Data Transfer Scripts tab (data loading options)

Table 14-2 A comparison of DB2 UDB's IMPORT and LOAD utilities

Utility	Description
IMPORT	IMPORT reads data from a flat file and generates corresponding INSERT statements. COMMIT statement frequency can be reduced to improve performance. Because INSERT statements are used, constraints are enforced and triggers are activated. IMPORT performance is largely determined by buffer pool size and placement of database logs (logging is performed).
LOAD	LOAD reads data from a flat file and loads data at the data page level. Because SQL statements are not used, triggers are not activated as rows are loaded. Performance of this method is largely determined by the utility heap size. Logging is not performed.

Recommendation: In most cases, you should use the LOAD utility since you need to move the data from SQL Server “as-is”, without activating triggers.

DB2 UDB LOAD/IMPORT Mode

The LOAD and IMPORT utilities can replace existing data in tables, or append new data to the table. These options are shown in Figure 14-22. Select the option that is most appropriate for your application and environment.

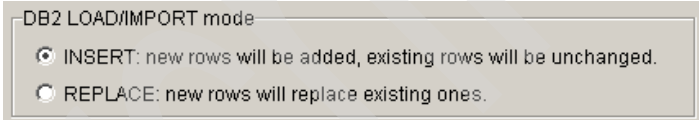


Figure 14-22 MTK - Generate Data Transfer Scripts tab (LOAD/IMPORT mode)

File format

MTK supports both ASCII delimited (DEL) and ASCII positional (ASC) files, as shown in Figure 14-23. In most cases, the default (ASC) works fine. The DEL format requires less disk space for extracted flat files, but you may encounter difficulty selecting a unique column delimiter depending on the data.

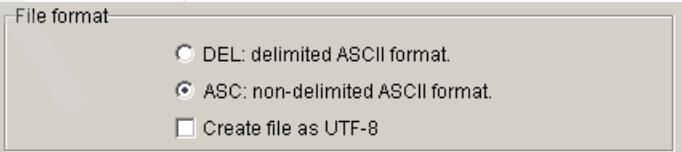


Figure 14-23 MTK - Generate Data Transfer Scripts tab (file format options)

Directory for data extraction

This path specifies where the data extracted from SQL Server is stored. Be sure to choose a device with sufficient disk space.

Advanced options

Depending on the data loading method selected (LOAD or IMPORT), you will see a different menu for advanced options. The options you select may affect the overall performance of the data migration process. For more details about the LOAD and IMPORT commands, consult the DB2 Information Center at:

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

Navigate to:

Reference → Commands → Command Line Processor (CLP) → IMPORT; and
Reference → Commands → Command Line Processor (CLP) → LOAD

Create Scripts button

Click the **Create Scripts** button to initiate the *creation* of data movement scripts, which are used by MTK to perform data extraction and loading in the *Deploy to DB2* tab (the next phase of the conversion).

14.3.5 Deploy to DB2

Once you have generated the data transfer scripts, you are ready to deploy the database structure and data to DB2 UDB. Click the *Deploy to DB2* tab (Figure 14-24).

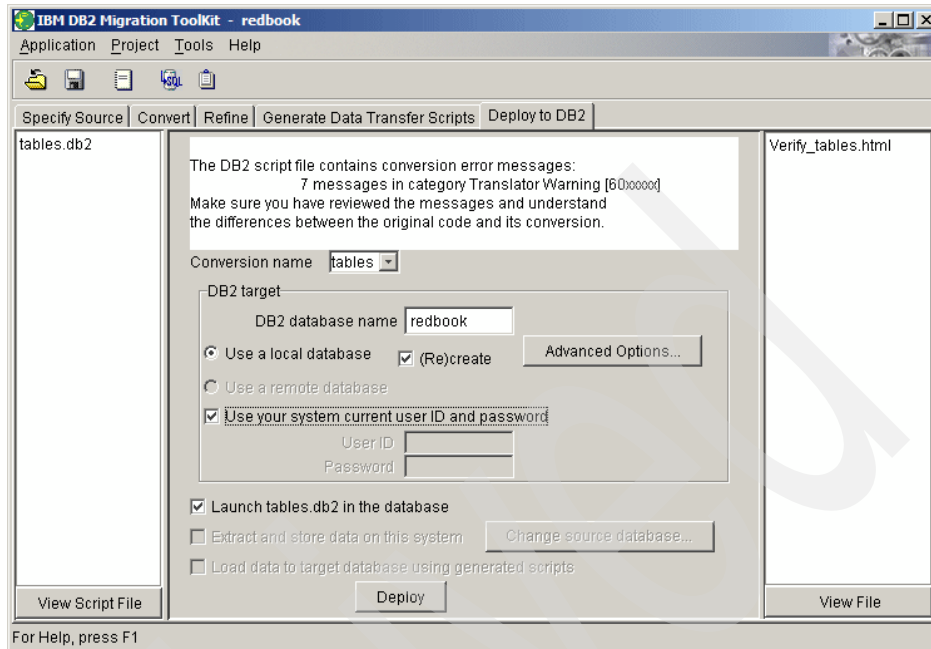


Figure 14-24 MTK - Deploy to DB2 tab (for tables)

Deployment is really a three part process:

- ▶ Create objects in DB2 UDB by executing the converted script
- ▶ Extract data from source database to flat files (using scripts generated previously)
- ▶ Deploy data from flat files to DB2 UDB (using scripts generated previously)

We now describe this process in more detail:

1. Specify the conversion name. In this case, the conversion name is tables (tables.db2) and is the only deployable conversion at this time. After you have completed function, trigger, and stored procedure conversions, this drop-down list will have more conversions to choose from.
2. Select the target DB2 UDB database. The database may be a local or remote database. MTK can deploy the converted objects to an existing database or create a new database for you. In this scenario, we deploy the table and index objects to the previously created database called REDBOOK, by selecting the **Use a local database** option and specifying REDBOOK as the database name.

Note: If you require a Unicode database, the *Advanced Options* window, available by clicking the **Advanced Options** button in the *Deploy to DB2* tab, has an option to specify a UTF-8 codeset.

3. Create the DB2 UDB database objects and load the data. At the bottom of the *Deploy to DB2* tab, three deployment steps (check boxes) are available:

- Launch tables.db2 in the database
- Extract and store data on this system
- Load data to target database using general scripts

We do NOT recommend performing all three steps at the same time. Rather, each step should be performed independently. This gives you the most control and avoids trying to do too much in one step:

- a. Start with the first step (Figure 14-25) and click **Deploy**. This creates the tables and indexes in the DB2 UDB database and gives you a chance to review the verification report (Figure 14-25).

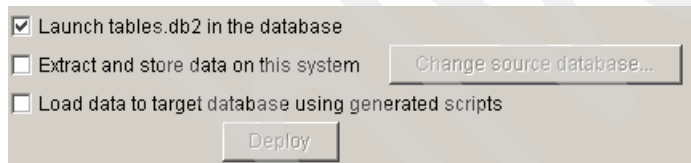


Figure 14-25 MTK - Deploy to DB2 tab (first deployment step)

- b. When you are comfortable with the results of the previous step, select only the second step (Figure 14-26). Click **Deploy** once again to perform data extraction to flat files.

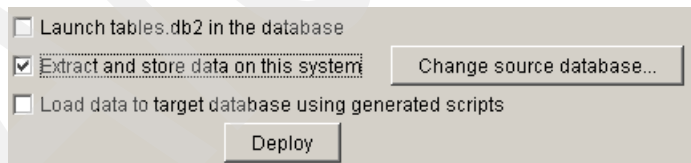


Figure 14-26 MTK - Deploy to DB2 tab (second deployment step)

- c. If the extraction completes successfully, select only the third step (Figure 14-27) and click **Deploy**. This loads the data from the flat files created in the previous step into the DB2 UDB database.

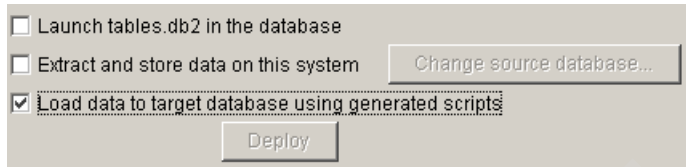


Figure 14-27 MTK - Deploy to DB2 tab (third deployment option)

- d. Later, once any functions, triggers, and stored procedures are converted, you can repeat step a) for each of those scripts. You do not have to perform steps b) or c) again.

When deployment completes (after clicking the **Deploy** button), a Verification Report (Figure 14-28) is displayed to show you the results. Confirm that everything deployed properly. If there are no errors in the verification report, you have successfully converted your tables and indexes to DB2 UDB.

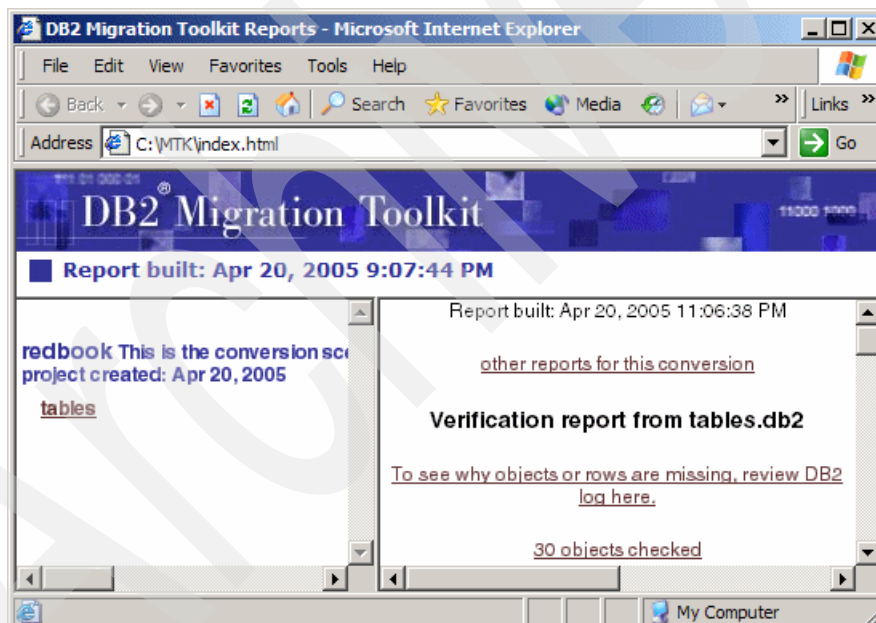


Figure 14-28 MTK - Deploy to DB2 verification report (for tables)

14.4 Other database object conversion

Now that the tables and indexes have been converted, database objects such as views, user defined functions, stored procedures, and triggers can be converted.

In your own conversion, it is critical that you validate your database structure (tables and indexes) before converting other database objects. The structure conversion is the foundation for converting all other database objects.

14.4.1 View conversion

In this section, we describe the process of converting SQL Server Views to DB2 UDB. To perform this conversion using MTK, the following steps can be followed:

1. In the current project, click the *Specify Source* tab at the top of the MTK window.
2. Extract the View definitions from SQL Server into a file called `views`. Follow the same process as we did for tables, described in 14.3.1, “Specify Source” on page 425. Instead of selecting tables to export, select Views, as shown in Figure 14-29.

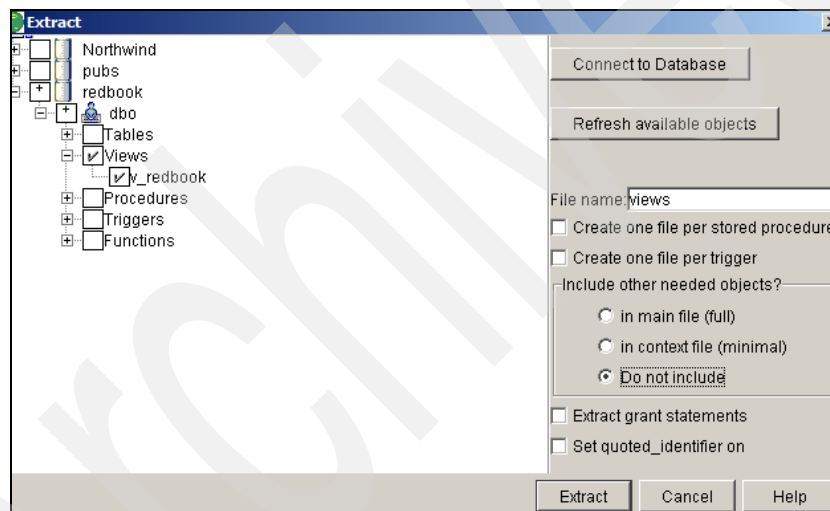



Figure 14-29 MTK - Extract dialog window for Views

3. After completing this step, two files should appear in the MTK project, `tables.src` and `views.src`.
4. Click the *Convert* tab.
To convert these Views, we must first set the context of the conversion because the Views reference some of the tables defined in `tables.src`. Click the **Context** button. Understanding how context affects a conversion is critical to effective use of MTK.
5. In the *Set Context* window, indicate to MTK that your conversion has dependencies on the `tables.src` file by selecting the `tables.src` file and

clicking the  button to move it to the right panel. Click **OK** to close the window.

6. Return to the *Convert* tab.

Notice that a context indicator (**context**) appears next to the `tables.src` file, as shown in Figure 14-30.

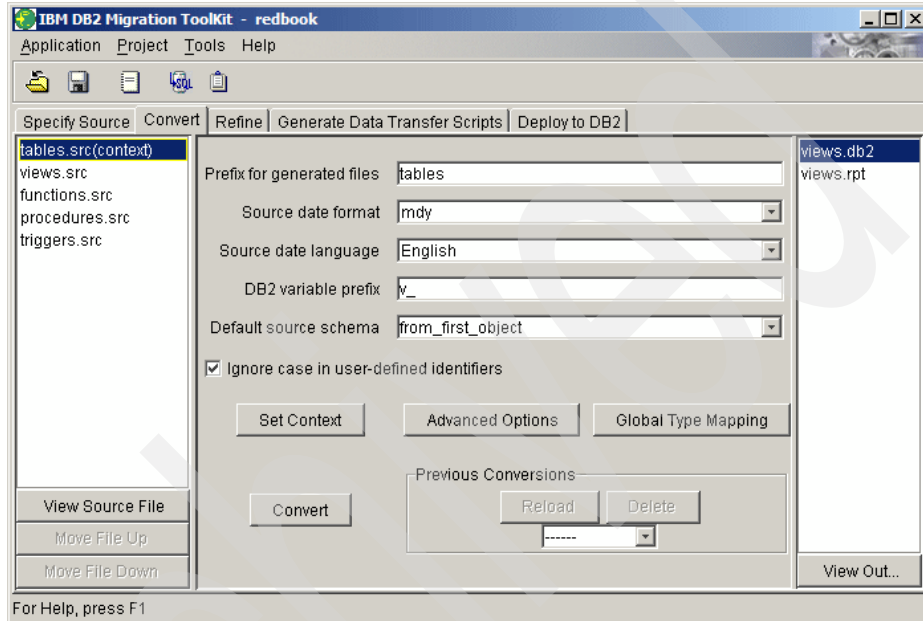


Figure 14-30 MTK - Convert tab after setting the context (for Views)

7. Select the *Refine* tab. Expand the messages tree as shown in Figure 14-31:

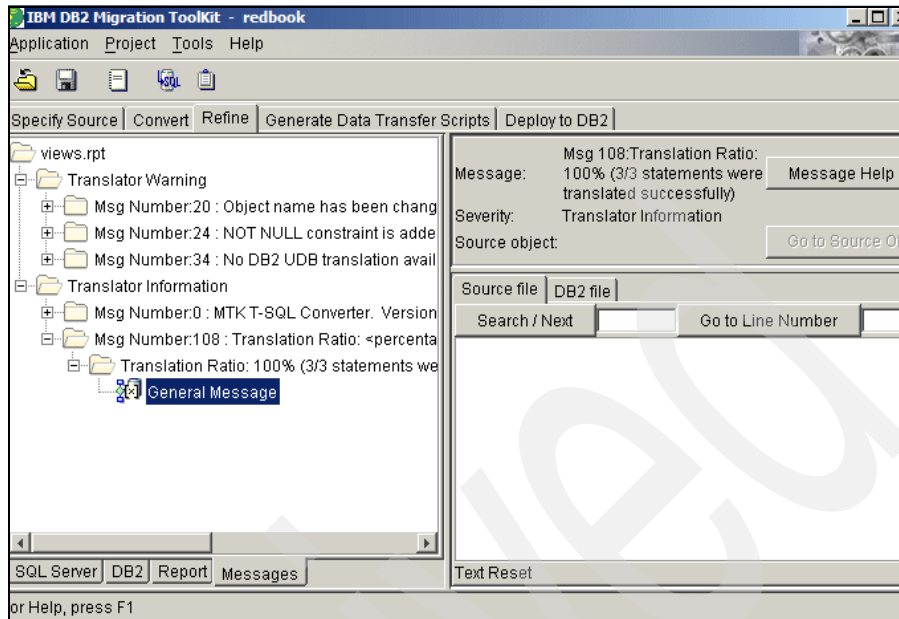


Figure 14-31 MTK - Refine tab after conversion (translation success ratio for Views)

Observe that the script translation ratio is 100%. That is, 100% of the statements in the `views.src` script are accounted for and translated without error. Warnings are not critical and are not included in the calculation. Since there are no Translator Error messages listed in the Messages tree, we can be confident that MTK will convert all the Views to DB2 UDB.

8. Deploy the Views to DB2 UDB.

Click the *Deploy to DB2* tab. From the *Conversion name* pull-down list, select **views**. Ensure that the target database name indicates REDBOOK. Leave the default values for all other settings. Ensure the **(re)create database** option is NOT selected. Click **Deploy**.

After deployment completes, you should see that all views have been created in the DB2 UDB REDBOOK database.

Note: You may need to click the **Refresh** button on your Web browser to see the latest Validation Report.

Other important observations

Examine the right hand panel of the *Convert* tab (Figure 14-32). You might be wondering what happened to `tables.db2` file (the translated output of `tables.src` we generated in 14.3, “Core database extraction and deployment” on page 424).

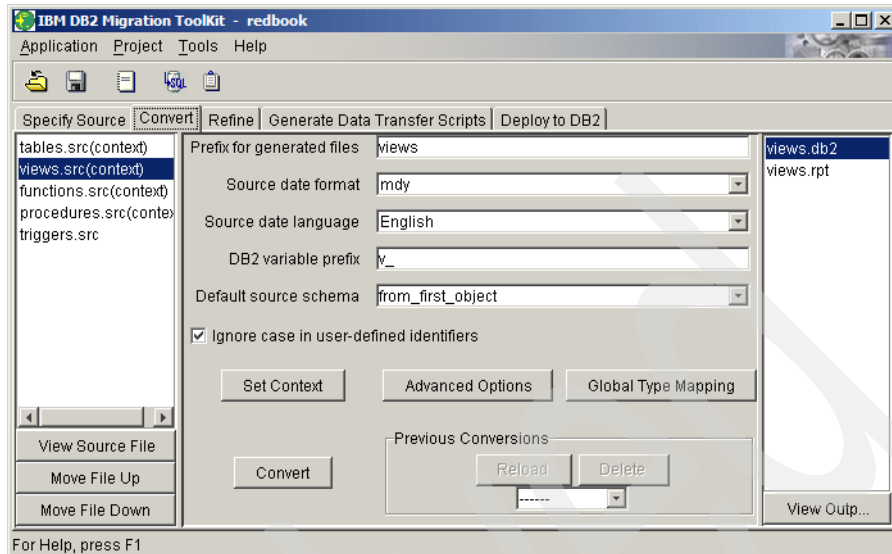


Figure 14-32 MTK - After deploying functions to DB2 UDB

Although it does not appear on the screen, the `tables.db2` file remains in your project folder. It is simply not listed as part of the current conversion activity. If you need to refer to previous conversions, select **tables** from the pull-down list in *Previous Conversions* sub-panel (Figure 14-33) and click **Reload**. You can switch back and forth between conversions without having to re-convert.

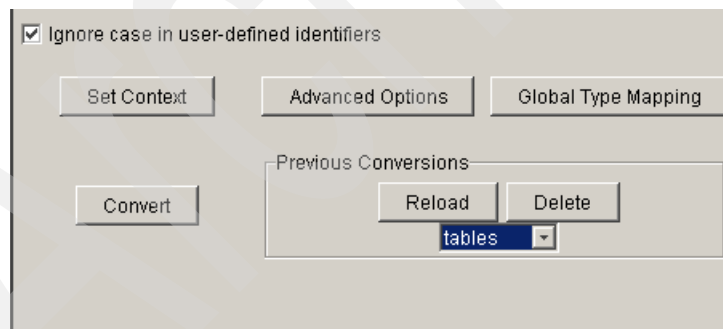


Figure 14-33 MTK - Convert tab (choosing a previous conversion activity)

14.4.2 Function conversion

In this section, we describe the process of converting SQL Server user defined functions to DB2 UDB. To perform this conversion using MTK, the following steps can be performed:

1. In the current project, click the *Specify Source* tab at the top of the MTK window.
2. Extract the function definitions from SQL Server into a file called functions. Follow the same process as we did for tables, described in 14.3.1, “Specify Source” on page 425. Instead of selecting the tables to export, select the functions, as shown in Figure 14-34. After completing this step, three files should appear in your MTK project, tables.src, views.src, and functions.src.

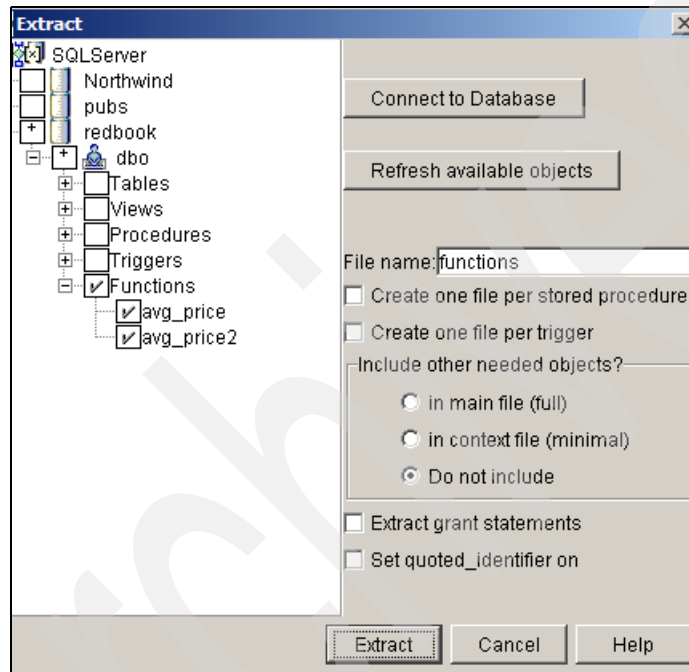



Figure 14-34 MTK - Extract dialog window for user defined functions

3. Click the *Convert* tab.
To convert these functions, we must first set the context of the conversion because the functions may reference some of the tables defined in tables.src and the views defined in views.src. Click the **Context** button.
4. In the *Set Context* window, indicate to MTK that the conversion has dependencies on tables.src and views.src files by selecting each of these files in the left pane, then clicking the  button to move them to the right pane. Click **OK** to close the window.

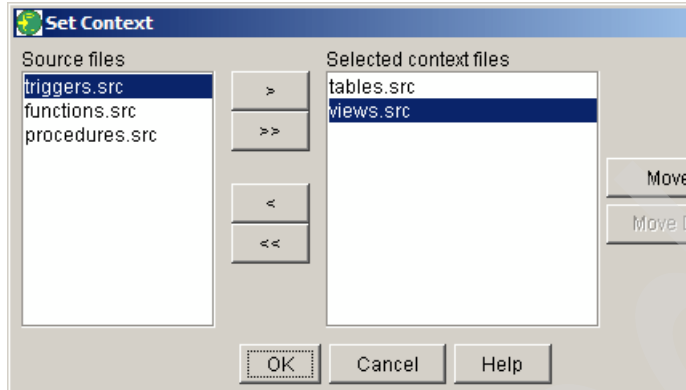


Figure 14-35 MTK - Set Context window (for functions)

5. Return to the *Convert* tab.

Notice that a context indicator (**context**) appears next to the `tables.src` and `views.src` files, as shown in Figure 14-36.

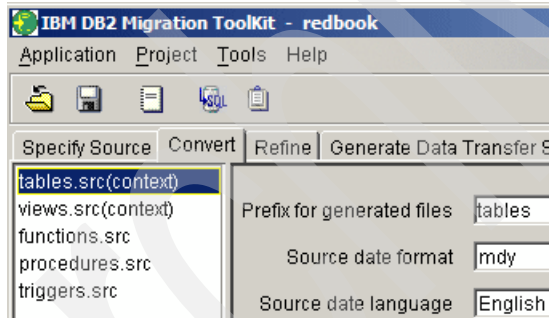


Figure 14-36 MTK - Convert context indicator (for functions)

We can now proceed with converting functions:

- Select the `functions.src` file
- Click **Convert**

This begins converting the functions to DB2 UDB and automatically moves to the *Refine* tab upon completion. We now begin the task of making any necessary refinements to the converted functions, in the same way we refined the conversion of tables and indexes (i.e., working through the *Messages* tree).

6. Expand the messages tree as shown in Figure 14-37:

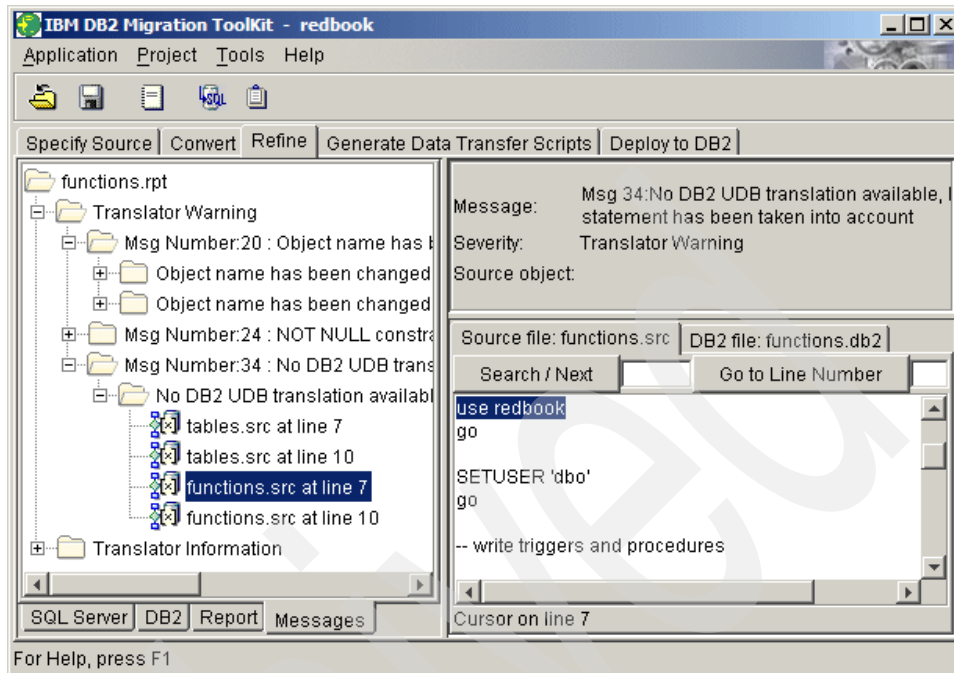


Figure 14-37 MTK - Refining converted functions

This warning message is being reported since the SQL Server `USE redbook` and `SET USER 'dbo'` statements have no equivalents in DB2 UDB. However, MTK has determined that these statements are not required in the conversion and that their behavior is accounted for in the converted code. In the absence of any other significant warning messages, the functions need no further refinements and can now be deployed to DB2 UDB.

7. Deploy the functions to DB2 UDB.

Click the *Deploy to DB2* tab. From the *Conversion name* pull-down list, select **functions**. Ensure that the target database name indicates REDBOOK. Leave the default values for all other settings. Ensure the **(re)create database** option is NOT selected. Click **Deploy**.

After deployment completes, you should see that all functions have been successfully created in the DB2 UDB REDBOOK database.

Note: You may need to click the **Refresh** button on your Web browser to see the latest Validation Report.

Tip: If you prefer a cleaner output file, you can disable the copying of source code by clicking the **Advanced Options** button in the *Convert* tab, de-selecting the option **Copy source (as comments) to translated code**, clicking **OK** to close the Advanced Options window, then clicking **Convert** again to re-convert the input file.

14.4.3 Stored procedure conversion

In this section, we describe the process of converting SQL Server stored procedures to DB2 UDB. To perform this conversion using MTK, the following steps can be performed:

1. In the current project, click the *Specify Source* tab at the top of the window.
2. Extract the stored procedure definitions from SQL Server into a file called *procedures*. Follow the same process as we did for tables, described in 14.3.1, “Specify Source” on page 425. Instead of selecting the tables to export, select the stored procedures. After completing this step, four files should appear in your MTK project, *tables.src*, *views.src*, *functions.src*, and *procedures.src*.
3. Click the *Convert* tab.

The context for the stored procedure conversion includes tables, functions, and views. Ensure the context includes the appropriate source files, as previously described. On the left pane of the *Convert* tab (the source file listing), select the *procedures.src* file and click **Convert**. This begins the process of converting the stored procedures to DB2 UDB. Upon completion, MTK automatically advances to the *Refine* tab.

4. In the *Refine* tab, expand the messages tree, as illustrated in Figure 14-38:

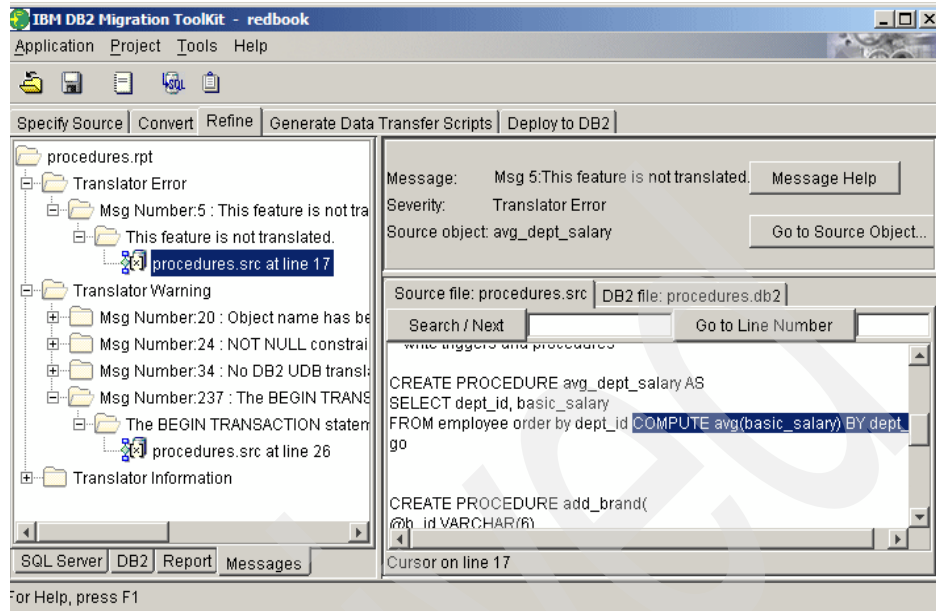


Figure 14-38 MTK - Refine tab translation message error (for stored procedures)

Translator Error messages generally require more attention since something was not directly convertible to DB2 UDB (although an intelligent work-around might exist). In the *Source File* tab (in the bottom right pane of the *Refine* tab), we can see that there is a conversion issue. The procedure `avg_dept_salary()` uses a `SELECT` statement with the `COMPUTE` clause specified. There is not a similar `COMPUTE` clause available in DB2 UDB, but workarounds are possible.

Also, notice that in the *DB2 file: procedures.db2* tab (in the bottom right pane of the *Refine* tab), the converted `avg_dept_salary()` code has not in fact been completely converted. It appears like:

```
CREATE PROCEDURE avg_dept_salary()
LANGUAGE SQL
BEGIN
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE l_error CHAR(5) DEFAULT '00000';

  --| SELECT dept_id, basic_salary
  --| FROM employee order by dept_id COMPUTE avg(basic_salary) BY dept_id
  --* [700005]"C:\MTK\projects\redbook\procedures.src"(17:32)-(17:67)This
  -- feature is not translated.

  DECLARE temp_cursor CURSOR WITH HOLD WITH RETURN TO CLIENT FOR
```

```

SELECT dept_id, basic_salary
FROM employee
ORDER BY dept_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET l_error = '00000';

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING
BEGIN
    SET l_error = SQLSTATE;

    IF SUBSTR(l_error, 1, 1) >= '5' AND SUBSTR(l_error, 1, 1) <= '9'
    THEN
        RESIGNAL;
    END IF;
END;

--| SELECT dept_id, basic_salary
--| FROM employee order by dept_id COMPUTE avg(basic_salary) BY dept_id
--* [700005]"C:\MTK\projects\redbook\procedures.src"(17:32)-(17:67)This
-- feature is not translated.

OPEN temp_cursor;
END!

```

If you were to deploy/run this script (procedures.db2) in its current state, the procedure avg_dept_salary() would create successfully, but without the COMPUTE functionality expected.

Manual intervention is needed here in order to preserve this functionality. Make a note to manually add this functionality back later.

5. Return to the *Convert* tab. In the left pane, select the procedures.src file. Click the **View Source File** button on the bottom left hand corner. Locate the code for avg_dept_salary() and change the SELECT statement from:

```

SELECT dept_id, basic_salary
FROM employee ORDER BY dept_id COMPUTE AVG(basic_salary) BY dept_id
to:

SELECT dept_id, basic_salary
FROM employee ORDER BY dept_id

```

Save and close the text editor (closing the text editor is important to avoid confusion later if you open the file again). We have effectively changed the input file to the MTK. Click the **Convert** button once again. The translator re-reads the input file and generates a new procedures.db2 file. On the *Refine* tab, notice that the conversion of avg_dept_salary() has now completed without errors or warnings. Msg Number: 5 no longer appears (Figure 14-39). Just remember to manually edit the DB2 UDB script later on to compensate

for the removed COMPUTE functionality. Solutions for duplicating SQL Server's COMPUTE clause are discussed in 4.6.1, "SELECT statements" on page 79.

6. Expand the messages tree as shown in Figure 14-39:

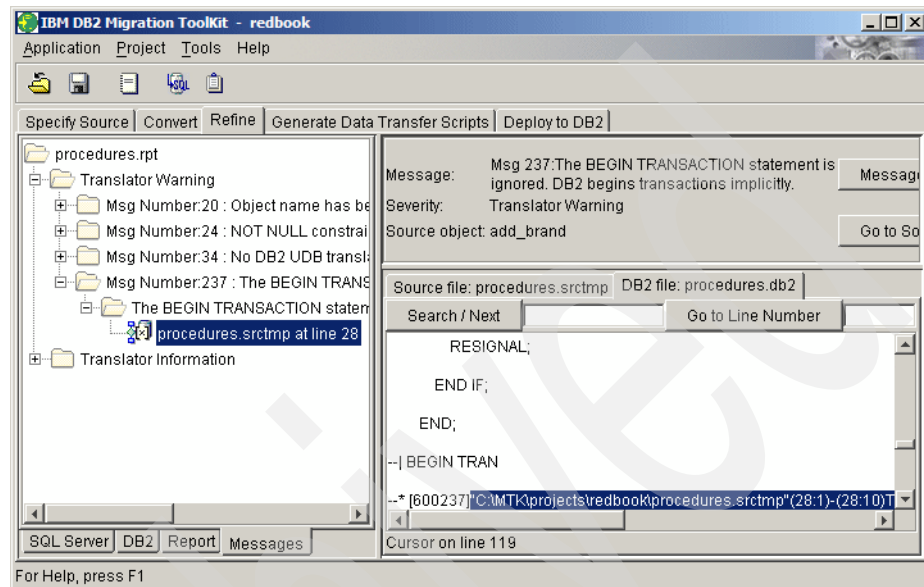


Figure 14-39 MTK - Refine tab after modifying a SQL Server stored procedure

There is no equivalent SQL Server's BEGIN TRANSACTION statement in DB2 UDB. In DB2 UDB, transactions are started implicitly after the first statement after connecting to a database or proceeding a COMMIT or ROLLBACK statement. MTK simply ignores this statement when converting. In the absence of any other significant warning messages, the stored procedures need no further refinements and can now be deployed to DB2 UDB.

7. Deploy the functions to DB2 UDB.
 - a. Select the *Deploy to DB2* tab.
 - b. From the *Conversion name* pull-down list, select **procedures**.
 - c. Ensure that the target database name indicates REDBOOK.
 - d. Click **Deploy**.

After deployment completes, check to make sure that all stored procedures are successfully created in DB2 UDB.

14.4.4 Trigger conversion

In this section, we convert triggers to DB2 UDB. The issues encountered when converting triggers are similar to those encountered with user defined functions.

1. In the current project, click the *Specify Source* tab at the top of the MTK window.
2. Extract the trigger definitions from SQL Server into a file called triggers. Follow the same process as we did for tables, described in 14.3.1, “Specify Source” on page 425. Instead of selecting the tables to export, select the triggers. After completing this step, five files should appear in your MTK project, tables.src, views.src, functions.src, procedures.src, and triggers.src.
3. Click the *Convert* tab.

The context for the trigger conversion includes tables, views, functions, and stored procedures. Ensure the context includes the appropriate source files as previously described. On the left pane of the *Convert* tab (the source file listing), select the triggers.src file and click **Convert**. This begins the process of converting the triggers to DB2 UDB. Upon completion, MTK automatically advances to the *Refine* tab.

4. In the *Refine* tab, expand the messages tree as shown in Figure 14-40:

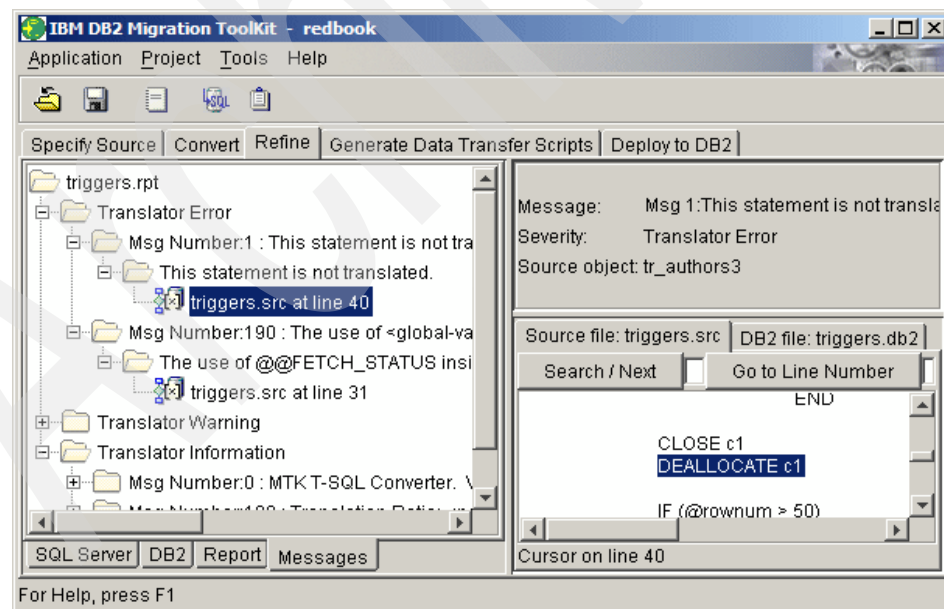


Figure 14-40 MTK - Refine tab (for triggers)

Two Translator Error messages appear (Msg Number 1 and Msg Number 190). Both messages relate to the use of explicit cursor statements (DEALLOCATE and @@FETCH_STATUS) in a trigger. These types of statements are not supported in DB2 UDB triggers. MTK is not able to convert these statements, thus manual intervention is required. For now, we make a note of this and continue investigating the other messages.

5. Expand the messages tree again, as shown in Figure 14-41.

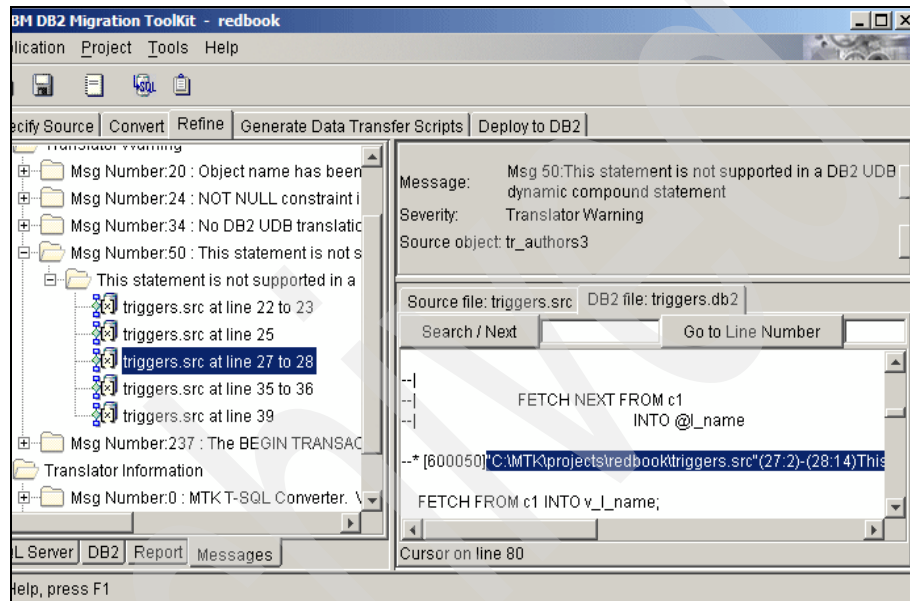


Figure 14-41 MTK - Refine tab (for triggers)

The other Translation Warning messages also involve the various cursor statements in the SQL Server trigger code. Since explicit cursor declarations or statements involving cursors are not supported in DB2 UDB triggers, this trigger should be manually converted to DB2 UDB. Possible options for converting this include:

- Replacing the explicit cursor declaration and statements with implicit cursor operations
- Moving the cursor logic into a stored procedure, then invoking the stored procedure from the trigger
- Performing the cursor logic in the application code, instead of in a trigger. This would involve additional manual conversion effort.

For a more complete discussion about conversion possibilities, refer to 9.4.4, “Overcoming inline SQL PL limitations in triggers” on page 219. For now,

remove this trigger from the source SQL Server trigger file (triggers.src) or leave it in the file as comments. In your conversion notes, make a reminder to manually convert this trigger later.

- Expand the messages tree as shown in Figure 14-42:

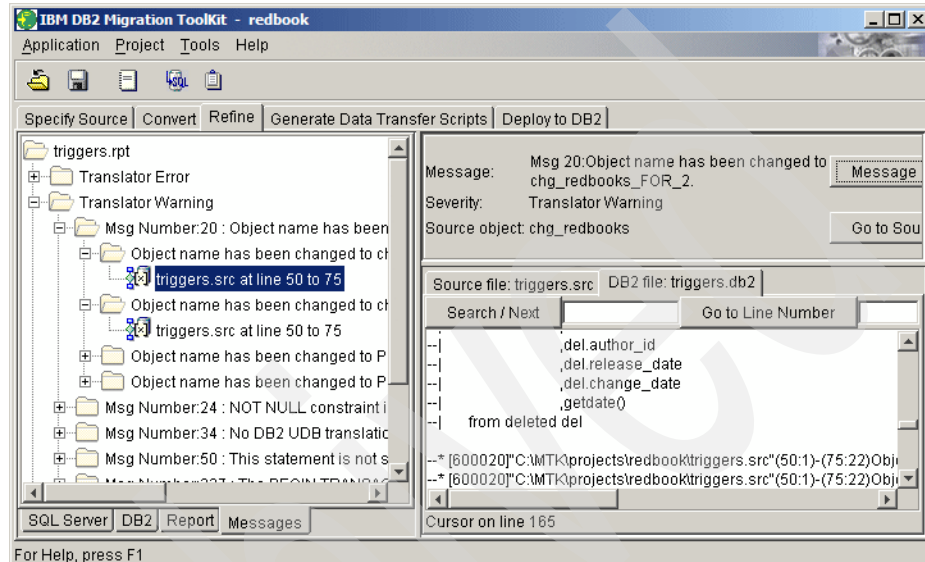


Figure 14-42 MTK - Refine tab (renaming of multi-statement triggers)

DB2 UDB does not support combined UPDATE/DELETE triggers, but multiple triggers can be defined on the same table. Therefore, the single SQL Server UPDATE/DELETE trigger is converted as two separate triggers - one for UPDATE and one for DELETE. MTK simply converts the trigger twice and renames the triggers slightly. The original SQL Server trigger:

```
CREATE TRIGGER chg_redbooks
ON redbooks
FOR UPDATE, DELETE
AS
INSERT INTO redbooks_history(
    book_id,
    book_no,
    title,
    brand_id,
    price,
    notes,
    author_id,
    release_date,
    change_date,
    timestamp)
```



```

SELECT
    del.book_id,
    del.book_no,
    del.title,
    del.brand_id,
    del.price,
    del.notes,
    del.author_id,
    del.release_date,
    del.change_date,
    getdate()
FROM deleted del

```

is converted as:

```

CREATE TRIGGER chg_redbooks_FOR_1
AFTER UPDATE ON redbooks
REFERENCING OLD_TABLE AS DELETED NEW_TABLE AS INSERTED
FOR EACH STATEMENT
MODE DB2SQL
BEGIN ATOMIC
    INSERT INTO redbooks_history (
        book_id,
        book_no,
        title,
        brand_id,
        price,
        notes,
        author_id,
        release_date,
        change_date,
        timestamp)
    SELECT del.book_id,
        del.book_no,
        del.title,
        del.brand_id,
        del.price,
        del.notes,
        del.author_id,
        del.release_date,
        del.change_date,
        CURRENT_TIMESTAMP
    FROM deleted del;
END!

```

```

CREATE TRIGGER chg_redbooks_FOR_2
AFTER DELETE ON redbooks
REFERENCING OLD_TABLE AS DELETED
FOR EACH STATEMENT
MODE DB2SQL

```

```

BEGIN ATOMIC
  INSERT INTO redbooks_history (
    book_id,
    book_no,
    title,
    brand_id,
    price,
    notes,
    author_id,
    release_date,
    change_date,
    timestamp)
  SELECT del.book_id,
    del.book_no,
    del.title,
    del.brand_id,
    del.price,
    del.notes,
    del.author_id,
    del.release_date,
    del.change_date,
    CURRENT_TIMESTAMP
  FROM deleted del;
END!

```

7. Go back to the *Convert* tab. Click the **Convert** button to re-convert the trigger.src file. Observe how the file now converts successfully after the problematic trigger is removed. In the absence of any other significant warning messages, the triggers need no further refinements and can now be deployed to DB2 UDB.
8. Deploy the trigger to DB2 UDB.
 - a. Select the *Deploy to DB2* tab.
 - b. From the *Conversion name* pull-down list, select **triggers**.
 - c. Ensure that the target database name indicates REDB00K. Leave the default values for all other settings.
 - d. Click the **Deploy** button.

After deployment completes, check to make sure that all triggers are successfully created in DB2 UDB. Also remember to manually convert the problematic trigger at a later time and re-add it to the deployment script.

14.5 Application conversion

Once the database conversion is complete, some modifications are required in the application code accessing the DB2 UDB database. We illustrate this process using a small application written in C#.

You can download the application code for this sample application. Refer to Appendix G, “Additional material” on page 509 for download instructions.

14.5.1 Overview of the C# sample application

The sample application enables a user to search for the author of an IBM Redbook. The application has two user interface windows. The first window is the login window where a user logs in to the application. It is shown in Figure 14-43.

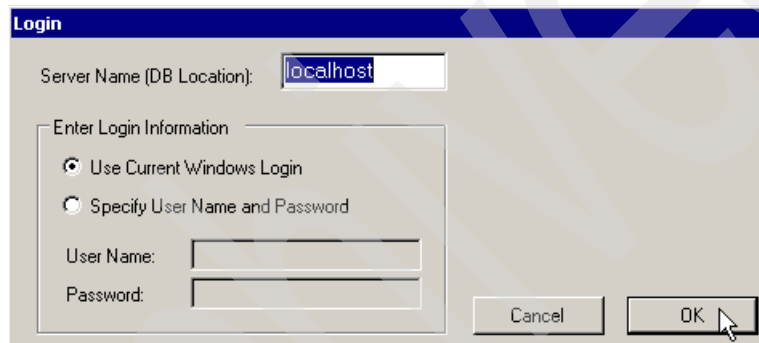


Figure 14-43 Sample C# application - Login window

The other window is the Author search screen where a user can search for IBM Redbooks contained in a database. This window is shown in Figure 14-44.

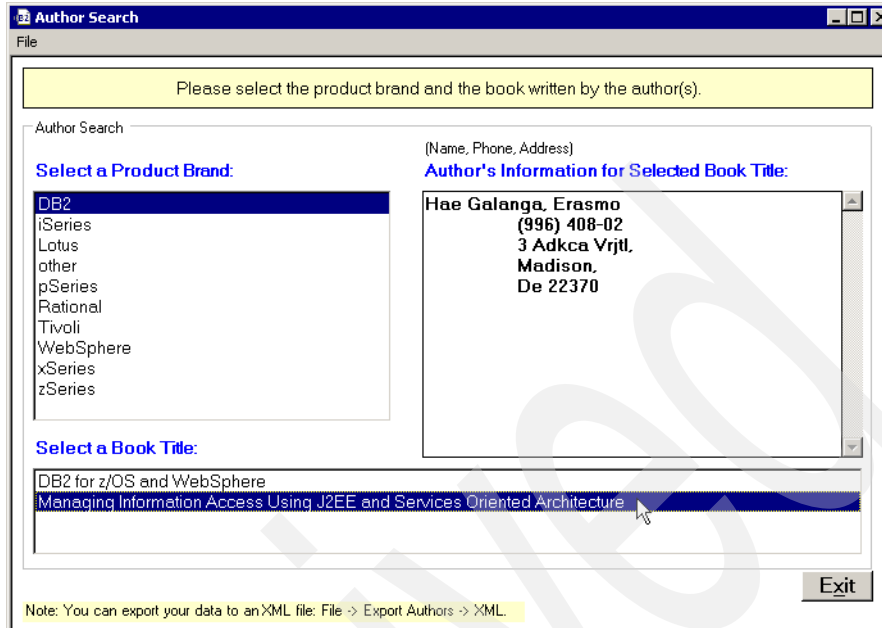


Figure 14-44 Sample C# application - Author Search window

The application uses the Microsoft Sql .NET data provider to interface natively with SQL Server. A change in the interface driver to the IBM DB2 .NET data provider is used in this scenario in order to allow the application to work with a DB2 UDB database with minimal changes.

The application can switch between a SQL Server and DB2 UDB database by setting a boolean variable in the application code:

```
// Switch between SQL Server: false, and DB2 UDB: true
static public Boolean usingDB2UDB = true;
```

The code to interface with SQL Server using the Sql .NET data provider and with DB2 UDB using the DB2.NET data provider is included in the application code for comparison purposes.

14.5.2 Set up Visual Studio .NET for the DB2 UDB environment

To use the DB2 .NET Provider, you must register the IBM DB2 Development Add-In, if you installed Visual Studio .NET after installing DB2 UDB.

Install the add-in by executing the following command from a command window:

```
C:\%DB2HOME%\SQLLIB\BIN\db2vsregister.bat
```

In order for your Visual Studio .NET application to use the add-in, you must add a reference to the data provider library to your project. This can be done using the following steps:

1. **Project** menu → **Add Reference**.
2. In the *.NET* tab, search for **IBM.Data.DB2.dll** in the component name column.
3. Highlight it and click **Select**.
4. Click **OK** to add the reference.

14.5.3 Add a DB2 library reference to .NET classes

In a class file that interfaces with DB2 UDB, insert a reference to the DB2 name space as follows:

```
using System;  
[...]  
using System.IO;  
using IBM.Data.DB2;
```

14.5.4 Replace interface-specific objects

When connecting to SQL Server using the Microsoft data provider, a `SqlConnection` component is used. The corresponding component in the IBM DB2 provider is `DB2Connection`. Similarly, `SqlDataAdapter` is replaced by `DB2DataAdapter`, `SqlCommand` by `DB2Command`, `SqlParameter` by `DB2Parameter`, and so on.

The following summarizes this mapping:

- ▶ `SqlConnection` → `DB2Connection`
- ▶ `SqlCommand` → `DB2Command`
- ▶ `SqlDataAdapter` → `DB2DataAdapter`
- ▶ `SqlDataReader` → `DB2DataReader`

Example 14-1 shows how to replace the `SqlConnection` with `DB2Connection`.

Example 14-1 Changing `SqlConnection` to `DB2Connection` in the application code

SQL Server

```
private System.Data.SqlClient.SqlConnection sqlConnection1;  
[...]  
this.sqlConnection1.Open();  
pubIdCommand.Connection = this.sqlConnection1;  
m_objPubId = pubIdCommand.ExecuteScalar();
```

```
this.sqlConnection1.Close();
```

DB2 UDB

```
private IBM.Data.DB2.DB2Connection db2Connection1;  
[...]  
this.db2Connection1.Open();  
pubIdCommand.Connection = this.db2Connection1;  
m_objPubId = pubIdCommand.ExecuteScalar();  
this.db2Connection1.Close();
```

In this sample program, there are no specific data type enumerations to change. In your environment, you may need to change data type enumerations, for example:

- ▶ SqlDbType.Int → DB2Type.Int32
- ▶ SqlDbType.NVarChar → DB2Type.String
- ▶ SqlDbType.DateTime → DB2Type.DateTime

14.5.5 Database connection statement

The source application code for database connection is shown in Example 14-2.

Example 14-2 Connection statement changes in the application code

SQL Server:

```
m_strConnectionString = "SERVER=" + this.TBServerName.Text  
    + ";UID=" + this.TBUserName.Text + ";PWD=" + this.TBPassword.Text  
    + ";DATABASE=redbook;Trusted_Connection=Yes";
```

DB2 UDB

```
m_strConnectionString = "DATABASE=REDBOOK;" + "Connect Timeout=30;"  
    + "user Id=" + this.TBUserName.Text + ";PWD=" + this.TBPassword.Text;
```

14.5.6 Summary

This section discussed the steps involved in converting a simple application from SQL Server and the Microsoft-specific SqlClient data provider to DB2 UDB and the IBM DB2 .NET data provider.

SQL languages differences must also be identified in any SQL statements in the application code. For this sample application, no changes are required.

Terminology mapping

This appendix provides a comparison between the terminology used in SQL Server and DB2 UDB.

A.1 SQL Server to DB2 UDB terminology comparison

Table A-1 shows the terminology mapping of physical objects between DB2 UDB and MS SQL Server.

Table A-1 Physical objects

SQL Server	DB2 UDB
Filegroup	Tablespace
Database file with FILEGROW>0	SMS tablespace
Database file with SIZE>0, FILEGROW>0	DMS tablespace
n/a	Container (file or RAW device)

Table A-2 shows the terminology mapping of logical objects between DB2 UDB and MS SQL Server.

Table A-2 Logical objects

SQL Server	DB2 UDB
Server	Server
Instance	Instance
Database	Database
Enterprise Manager Registry	Database directory
Enterprise Manager Registry	Node directory
Windows registry	Database Manager configuration file
Windows registry	Database configuration file
System tables (master db or database)	Catalog tables

Table A-3 shows the terminology mapping of database objects between DB2 UDB and MS SQL Server.

Table A-3 Database objects

SQL Server	DB2 UDB
Schema	Schema
Table	Table
Rule and table constraint	Table constraint

SQL Server	DB2 UDB
View	View
Index	Index
Transaction log	Transaction log (also called Recovery log)
Transaction log dump	Archive log
Users, groups and roles	Users and groups (operating system)
Northwind and Pubs	Sample database (called SAMPLE)

Table A-4 shows the terminology mapping of administration and usage between DB2 UDB and MS SQL Server.

Table A-4 Administration and usage

SQL Server	DB2 UDB
Enterprise manager	Control Center
Tables assigned to a filegroup	Tables assigned to a tablespace
Database files assigned to filegroups	Containers assigned to tablespaces
System stored procedures	Administration commands (get db cfg)
n/a	Binding packages
Backup database	Backup database
Backup log	Archive online logs files
Restore database	Restore from backup
Restore log	Roll-forward recovery
Automatic recovery	Crash recovery
Update statistics, Create statistics	Run statistics (runstats command)
BCP (bulk copy program)	Load, Import and Export
SQL Server Query analyzer	Command Editor, Visual Explain
ISQL	Command Line Processor

Data type mapping

This appendix provides a mapping between SQL Server data types and DB2 UDB data types. It also provides a data type mapping between the SQL language data types and the data types available.

B.1 Mapping SQL Server data types to DB2 data types

Table B-1 shows the mapping of the SQL Server data types to DB2 UDB data types. The mapping is one to many and depends on the expected data.

Table B-1 SQL Server data types mapped to DB2 data types

SQL Server data type	DB2 UDB data type	Range of values
CHAR(m)	CHAR(n)	1 <= m <= 8000 1 <= n <= 254
VARCHAR(m)	VARCHAR(n)	1 <= m <= 8000 1 <= n <= 32762
	LONG VARCHAR(n)	if n <= 32700 bytes
TEXT	CLOB(2GB)	if n <= 2 GB
TINYINT	SMALLINT	- 32768 to 32767
SMALLINT	SMALLINT	- 32768 to 32767
INT INTEGER	INT INTEGER	- 231 to (231 - 1)
BIGINT	BIGINT	
DEC(p,s) DECIMAL(p,s)	DEC(p,s) DECIMAL(p,s)	- (1031+1) to (1031 - 1) (p+s <= 31)
NUMERIC(p,s)	NUMp,s) NUMERIC(p,s)	- (1031+1) to (1031 - 1) (p+s <= 31)
FLOAT(p)	FLOAT(p)	
REAL	REAL	
DOUBLE	DOUBLE PRECISION	
BIT	CHAR(1) FOR BIT DATA	0 or 1
BINARY(m)	CHAR(n) FOR BIT DATA	1 <= m <= 8000 1 <= n <= 254
VARBINARY(m)	VARCHAR(n) FOR BIT DATA	1 <= m <= 8000 1 <= n <= 32672
IMAGE	BLOB(n)	if n <= 2 GB
TEXT	CLOB(n)	if n <= 2 GB
NTEXT	DBCLOB(n)	0 <= n <= 2 GB

SQL Server data type	DB2 UDB data type	Range of values
SMALLDATETIME	TIMESTAPMP	Jan 1, 0001 to Dec 31, 9999
DATETIME	TIMESTAPMP	Jan 1, 0001 to Dec 31, 9999
TIMESTAMP	CHAR(8) FOR BIT DATA	
	DATE (MM/DD/YYYY)	year: 0001 to 9999 month: 1 to 12 day: 1 to 31
	TIME (HH24:MI:SS)	hour: 0 to 24 minutes: 0 to 60 seconds: 0 to 60
NCHAR(m)	GRAPHIC(n)	1 <= m <= 4000 1 <= n <= 127
NVARCHAR(m)	VARGRAPHIC(n)	1 <= m <= 4000 1 <= n <= 16336
	LONG VARGRAPHIC(n)	1 <= n <= 16336
SMALLMONEY	NUMERIC(10,4)	
MONEY	NUMERIC(19,4)	
UNIQUEIDENTIFIER	CHAR(13) FOR BIT DATA	

B.2 Supported SQL data types in C/C++

Table B-2 provides a complete list of SQL data types and the corresponding C/C++ data type that should be used. Note that DB2 UDB has multiple definitions for DATE and multiple types for NUMBER.

For more information, refer to DB2 UDB documentation

- ▶ *IBM DB2 UDB SQL Reference, Volume 1, V8, SC09-4844*
- ▶ *IBM DB2 UDB SQL Reference, Volume 2, V8, SC09-4845*
- ▶ *IBM DB2 UDB Application Development Guide: Programming Client Applications V8, SC09-4826*

Table B-2 DB2 UDB SQL data types mapped to C/C++ data types

	SQL data type (sqltype)	C/C++ type	sqlen	Description
integer	SMALLINT (500 or 501)	short	2	<ul style="list-style-type: none"> ▶ 16-bit signed integer ▶ range between (-32,768 and 32,767) ▶ precision of 5 digits
	INTEGER INT (496 or 497)	long	4	<ul style="list-style-type: none"> ▶ 32-bit signed integer ▶ range between (-2,147,483,648 and 2,147,483,647) ▶ precision of 10 digits
	BIGINT (492 or 493)	long	8	<ul style="list-style-type: none"> ▶ 64-bit signed integer
floating point	REAL FLOAT (480 or 481)	float		<ul style="list-style-type: none"> ▶ Single precision floating point ▶ 32-bit approximation of a real number ▶ FLOAT(<i>n</i>) can be synonym for REAL if 0 < <i>n</i> < 25
	DOUBLE (480 or 481) DOUBLE PRECISION	double	8	<ul style="list-style-type: none"> ▶ Double precision floating point ▶ 64-bit approximation of a real number ▶ Range in (0, -1.79769E+308 to -2.225E-307, 2.225E-307 to 1.79769E+308) ▶ FLOAT(<i>n</i>) can be synonym for DOUBLE if 24 < <i>n</i> < 54
decimal	DECIMAL(p,s) DEC(p,s) (484 or 485) NUMERIC(p,s) NUM(p,s)	double / decimal	p/2+1	<ul style="list-style-type: none"> ▶ Packed decimal ▶ If precision /scale not specified, default is (5,0) ▶ Max precision is 31 digits, and max range between (-10E31+1 ... 10E31-1) ▶ Consider using CHAR/DECIMAL functions to manipulate packed decimal fields as CHAR data

	SQL data type (sqltype)	C/C++ type	sqlen	Description
date / time	DATE (384 or 385)	struct { short len; char data[10]; } dt; char dt[11];	10	<ul style="list-style-type: none"> ▶ Null-terminated character form (11 characters) or varchar struct form (10 characters); struct can be divided as desired to obtain the individual fields ▶ Example: 11/02/2000 ▶ Stored internally as a packed string of 4 bytes
	TIME (388 or 389)	char	8	<ul style="list-style-type: none"> ▶ Null-terminated character form (9 characters) or varchar struct form (8 characters); struct can be divided as desired to obtain the individual fields. ▶ Example: 19:21:39 ▶ Stored internally as a packed string of 3 bytes
	TIMESTAMP	char	26	<ul style="list-style-type: none"> ▶ Null-terminated character form(27 characters) or varchar struct form (26 characters); struct can be divided as desired to obtain the individual fields. ▶ Example: 2003-08-04-01.02.03.000000 ▶ Stored internally as a packed string of 10 bytes
character	CHAR (452 or 453)	char	<i>n</i>	<ul style="list-style-type: none"> ▶ Fixed-length character string consisting of <i>n</i> bytes. ▶ Use char[<i>n</i>+1] where 1 ≤ <i>n</i> ≤ 254 ▶ If length not specified, defaults to 1
	VARCHAR (460 or 461)	char	<i>n</i>	<ul style="list-style-type: none"> ▶ Null-terminated variable length character string ▶ Use char[<i>n</i>+1] where 1 ≤ <i>n</i> ≤ 32672

	SQL data type (sqltype)	C/C++ type	sqlen	Description
	VARCHAR (448 or 449)	struct { short len; char data[n] }; }	len	<ul style="list-style-type: none"> ▶ Non null-terminated varying character string with 2-byte string length indicator ▶ Use char[n] in struct form where $1 \leq n \leq 32672$ ▶ Default SQL type
	LONG VARCHAR (456 or 457)	struct { short len; char data[n] }; }	len	<ul style="list-style-type: none"> ▶ Non null-terminated varying character string with 2-byte string length indicator ▶ Use char[n] in struct form where $32673 \leq n \leq 32700$
	CLOB(n) (408 or 409)	clob	<i>n</i>	<ul style="list-style-type: none"> ▶ Non null-terminated varying character string with 4-byte string length indicator ▶ Use char[n] in struct form where $1 \leq n \leq 2147483647$
	CLOB (964 or 965)	clob_locator		▶ Identifies CLOB entities residing on the server
	CLOB (808 or 809)	clob_file		▶ Descriptor for file containing CLOB data
binary	BLOB(n) (404 or 405)	blob	<i>n</i>	<ul style="list-style-type: none"> ▶ Non null-terminated varying binary string with 4-byte string length indicator ▶ Use char[n] in struct form where $1 \leq n \leq 2147483647$
	BLOB (960 or 961)	blob_locator		▶ Identifies BLOB entities on the server
	BLOB (804 or 805)	blob_file		▶ Descriptor for the file containing BLOB data

	SQL data type (sqltype)	C/C++ type	sqlen	Description
double-byte	GRAPHIC(1) GRAPHIC(n) (468 or 469)	sqldbchar	24	<ul style="list-style-type: none"> ▶ sqldbchar is a single double-byte character string ▶ For a fixed-length graphic string of length integer which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed. ▶ Precompiled with CHARTYPE NOCONVERT option
	VARGRAPHIC(n) (464 or 465)	struct { short int; sqldbchar[n] } tag; alternately: sqldbchar[n+1]	$n*2+4$	<ul style="list-style-type: none"> ▶ For a varying-length graphic string of maximum length integer, which may range from 1 to 16336. ▶ Precompiled with WCHARTYPE NOCONVERT option. ▶ Null terminated variable-length
	LONG VARGRAPHIC(n) (472 or 473)	struct { short int; sqldbchar[n] } tag;		<ul style="list-style-type: none"> ▶ For a varying-length graphic string with a maximum length of 16350 and a 2-byte string length indicator $16337 \leq n \leq 16350$ ▶ Precompiled with WCHARTYPE NOCONVERT option
	DBCLOB(n) (412 or 413)	dbclob		<ul style="list-style-type: none"> ▶ For non-null-terminated varying double-byte character large object string of the specified maximum length in double-byte characters. ▶ 4 bytes string length indicator ▶ Use DBCLOBb(n) where $1 \leq n \leq 1073741823$ double-byte characters. ▶ Precompiled with WCHARTYPE NOCONVERT option

	SQL data type (sqltype)	C/C++ type	sqlen	Description
	DBCLOB	dbclob_locat orr		<ul style="list-style-type: none"> ► Identifies DBCLOB entities residing on the server ► Precompiled with WCHARTYPE NOCONVERT option
	DBCLOB	dbclob_file		<ul style="list-style-type: none"> ► Descriptor for file containing DBCLOB data ► Precompiled with WCHARTYPE NOCONVERT option
external data	Datalink(n)		n+54	<ul style="list-style-type: none"> ► The length of a DATALINK column is 200 bytes

B.3 SQL data types for the DB2 .NET Data Provider

Table B-3 shows the mappings between the SQL data types and the .NET Framework data types.

Table B-3 SQL data types mapped to .NET data type

	SQL data type (sqltype)	.NET Framework data type	sqlen	Description
integer	SMALLINT (500 or 501)	Int16	2	16-bit, signed integer
	INTEGER (496 or 497)	Int32	4	32-bit, signed integer
	BIGINT (492 or 493)	Int64	8	64-bit, signed integer
floating point	REAL (480 or 481)	Single		Single precision floating point
	DOUBLE (480 or 481)	Double	4	Single precision floating point
	DOUBLE (480 or 481)	Double	8	Double precision floating point
decimal	DECIMAL(p,s) (484 or 485)	Decimal	n/2	Packed decimal

	SQL data type (sqltype)	.NET Framework data type	sqllen	Description
date / time	DATE (384 or 385)	DateTime	10	10-byte character string
	TIME (388 or 389)	TimeSpan	8	8-byte character string
	TIMESTAMP (392 or 393)	DateTime	26	26-byte character string
character	CHAR (452 or 453)	String	<i>n</i>	Fixed-length character string of length <i>n</i> where <i>n</i> is from 1 to 254
	CHAR FOR BIT DATA	byte[]		Fixed-length character string of length <i>n</i> where <i>n</i> is from 1 to 254
	VARCHAR (448 or 449)	String	<i>n</i>	Variable-length character string, <i>n</i> <= 32672
	VARCHAR FOR BIT DATA	Byte[]		Variable-length character string
	LONG VARCHAR (456 or 457)	String	<i>n</i>	Long variable-length character string, <i>n</i> <= 32672
	CLOB(<i>n</i>) (408 or 409)	String	<i>n</i>	Large object variable-length character string
binary	BLOB(<i>n</i>) (404 or 405)	Byte[]	<i>n</i>	Large object variable-length binary string
double- byte	GRAPHIC(<i>n</i>) (468 or 469)	String	<i>n</i>	Fixed-length double-byte character string
	VARGRAPHIC(<i>n</i>) (464 or 465)	String	<i>n</i> *2+4	Non-null-terminated varying double-byte character string with 2-byte string length indicator

	SQL data type (sqltype)	.NET Framework data type	sqlen	Description
	LONG VARGRAPHIC(n) (472 or 473)	String	<i>n</i>	Non-null-terminated varying double-byte character string with 2-byte string length indicator
	DBCLOB(n) (412 or 413)	String		Large object variable-length double-byte character string

B.4 Supported SQL data types in Java

Table B-4 shows the mapping between SQL data type and Java data type based on the JDBC specification for data type mappings. The JDBC driver converts the data exchanged between the application and the database using the following mapping schema. Use these mappings in your Java applications and your PARAMETER STYLE JAVA stored procedures and user-defined functions.

Table B-4 SQL data types mapped to Java declarations

	SQL data type sqltype	Java type	sqlen	Description
integer	SMALLINT (500 or 501)	short	2	16-bit, signed integer
	INTEGER (496 or 497)	int	4	32-bit, signed integer
	BIGINT (492 or 493)	long	8	64-bit, signed integer
floating point	REAL (480 or 481)	float		Single precision floating point
	DOUBLE (480 or 481)	double	4	Single precision floating point
	DOUBLE (480 or 481)	double	8	Double precision floating point
decimal	DECIMAL(p,s) (484 or 485)	java.math. BigDecimal	n/2	Packed decimal

	SQL data type sqltype	Java type	sqlen	Description
date / time	DATE (384 or 385)	java.sql.Date	10	10-byte character string
	TIME (388 or 389)	java.sql.Time	8	8-byte character string
	TIMESTAMP (392 or 393)	java.sql. Timestamp	26	26-byte character string
character	CHAR (452 or 453)	java.lang. String	<i>n</i>	Fixed-length character string of length <i>n</i> where <i>n</i> is from 1 to 254
	CHAR FOR BIT DATA	byte[]		Fixed-length character string of length <i>n</i> where <i>n</i> is from 1 to 254
	VARCHAR (448 or 449)	java.lang. String	<i>n</i>	Variable-length character string, <i>n</i> <= 32672
	VARCHAR FOR BIT DATA	byte[]		Variable-length character string
	LONG VARCHAR (456 or 457)	java.lang. String	<i>n</i>	Long variable-length character string, <i>n</i> <= 32672
	CLOB(<i>n</i>) (408 or 409)	java.lang. Clob	<i>n</i>	Large object variable-length character string
binary	BLOB(<i>n</i>) (404 or 405)	java.lang.Blob	<i>n</i>	Large object variable-length binary string
double- byte	GRAPHIC(<i>n</i>) (468 or 469)	java.lang. String	<i>n</i>	Fixed-length double-byte character string
	VARGRAPHIC(<i>n</i>) (464 or 465)	java.lang. String	<i>n</i> *2+4	Non-null-terminated varying double-byte character string with 2-byte string length indicator
	LONG VARGRAPHIC(<i>n</i>) (472 or 473)	java.lang. String	<i>n</i>	Non-null-terminated varying double-byte character string with 2-byte string length indicator

	SQL data type sqltype	Java type	sqlen	Description
	DBCLOB(n) (412 or 413)	java.lang. Clob		Large object variable-length double-byte character string

Note: For Java applications connected from a DB2 UDB Version 8.x client to a DB2 UDB Version 7.x server, when the getObject() method is used to retrieve a BIGINT value, a java.math.BigDecimal object is returned.

Function mapping

This appendix provides a mapping of functions from SQL Server to DB2 UDB. The following functions are covered:

- ▶ Mathematical functions
- ▶ Character and string functions
- ▶ Boolean functions
- ▶ Date and time functions
- ▶ Metadata functions
- ▶ Aggregate Functions
- ▶ System functions
- ▶ Security functions
- ▶ Miscellaneous Functions

Differences between SQL Server functions and DB2 UDB functions are highlighted in **bold**.

C.1 Installation of additional built-in functions

MTK provides additional n functions for some SQL Server functions that are not available in DB2. By selecting the *deploy* action in MTK, these functions are automatically installed in DB2 UDB.

If you do not intend to use MTK, you can deploy this functions manually using the following steps:

1. Install MTK

This step is necessary because the function sources are packed with the installation files.

2. Issue the following commands from the DB2 Command Line Processor after connecting to the database where you want the functions to be installed:

```
db2 CALL SQLJ.INSTALL_JAR('C:\MTK\ms7udfs.jar' , 'myproc_jar')
db2 -td! -vf C:\MTK\mtkms7.udf
```

This assumes that MTK is installed in *C:\MTK*. The default schema of the provided functions is *MS7*.

3. Test the installation with the command

```
db2 VALUES ms7.version()
```

The output should looks similar to the following:

```
1
-----
SQL UDF Version: 041209.0332, Java UDF Version: 041201.0314
1 record(s) selected.
```

More information about creating external user-defined functions can be found in the DB2 UDB documentation *Application Development Guide: Programming Server Applications*, SC09-4827.

Note: Manual installation of the additional functions is not necessary if MTK is used to converted the database objects or data.

C.2 Mathematical functions

Table C-1 provides the mapping of mathematical functions from SQL Server to DB2 UDB.

Table C-1 Mathematical function mapping for SQL Server to DB2

SQL Server	DB2 UDB	Notes
%	MOD	Modulus
ABS	ABS	Absolute Value - Returns the absolute, positive value of the given numeric expression
ACOS	ACOS	Arccosine - Returns the angle in radians whose cosine is the given float expression
ASIN	ASIN	Arcsine - Returns the angle in radians whose sine is the given float expression
ATAN	ATAN	Arctangent of n - Returns the angle in radians whose tangent is the given float expression
ATN2	ATAN2	Arctangent of n and m - Returns the angle in radians whose tangent is between the two given float expressions
CEILING	CEILING	Returns the smallest integer greater than or equal to the given numeric expression
COS	COS	Returns the trigonometric cosine of the given angle (in radians)
COT	COT	Returns the trigonometric cotangent of the specified angle (in radians)
DEGREES	DEGREES	Given an angle in radians, returns the corresponding angle in degrees
EXP	EXP	Returns the exponential value of the given float expression
FLOOR	FLOOR	Returns the largest integer less than or equal to the given numeric expression
LOG	LN	Returns the natural logarithm of the given float expression
LOG10	LOG10	Returns the base-10 logarithm of the given float expression
PI	MS7. PI ()^a	Returns the constant value of PI

SQL Server	DB2 UDB	Notes
POWER™	POWER	Returns the value of the given expression to the specified power
RADIANS	RADIANS	Returns radians given a numeric expression in degrees
RAND	RAND	Returns a random float value between 0 and 1
ROUND(x,y)	ROUND(x,y)	Returns a numeric expression, rounded to the specified length or precision
ROUND(x,y,z)	TRUNC(x,y)	Returns x truncated to y places to the right of the decimal point if y is positive, or to the left of the decimal point if y is zero or negative.
SIGN	SIGN	Returns the positive (+1), zero (0), or negative (-1) sign of the given expression
SIN	SIN	Returns the trigonometric sine of the given angle (in radians)
SQUARE	MS7. SQUARE^a	Returns the square of the given expression
SQRT	SQRT	Returns the square root of the given expression
TAN	TAN	Returns the tangent of the input expression

a. This function is provided by the IBM DB2 MTK

C.3 Character and string functions

Table C-2 provides the mapping of mathematical functions from SQL Server to DB2 UDB.

Table C-2 Character and string function mapping for SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
ASCII	ASCII	Returns the ASCII code value of the left most character of a character expression
CHAR	CHAR	Converts an INT ASCII code to a character
CHARINDEX	MS7. CHARINDEX^a	Returns the starting position of a pattern within a string. Wild cards are not allowed.

SQL Server	DB2 UDB	Notes
DIFFERENCE	DIFFERENCE	Returns the difference between the SOUNDEX values of two character expressions as an integer
LEFT	LEFT	Returns the part of a character string starting at a specified number of characters from the left
LEN	LENGTH	Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks
LOWER	LOWER	Returns a character expression after converting uppercase character data to lowercase
LTRIM	LTRIM	Returns a character expression after removing leading blanks
NCHAR	MS7. NCHAR^a	Returns the Unicode character with the given integer code, as defined by the Unicode standard
PATINDEX	MS7. PATINDEX^a	Returns the starting position of the first occurrence of a pattern in a specified expression, or zeros if the pattern is not found, on all valid text and character data types. Wild cards are allowed
QUOTENAME	MS7. QUOTENAME^a	Returns a Unicode string with the delimiter added to make the input string a valid SQL Server delimited identifier
REPLACE	REPLACE	Replaces all occurrences of the second given string expression in the first string expression with a third expression
REPLICATE	REPEAT	Repeats a character expression a specified number of times
REVERSE	MS7. REVERSE^a	Returns the reverse of a character expression
RIGHT	RIGHT	Returns the part of a character string starting from a specified number of characters from the right
RTRIM	RTRIM	Returns a character string after removing all trailing blanks
SOUNDEX	SOUNDEX	Returns a four-character (SOUNDEX) code to evaluate the similarity of two strings

SQL Server	DB2 UDB	Notes
SPACE	SPACE	Returns a string of repeated spaces
STR	MS7. STR ^a	Returns character data converted from numeric data
STUFF	MS7. STUFF ^a	Deletes a specified length of characters and inserts another set of characters at a specified starting point
SUBSTRING	SUBSTR	Returns part of a character, binary, text, or image expression
TEXTPTR	N/A	Returns the text-pointer value that corresponds to a text, ntext, or image column in VARBINARY format
TEXTVALID	N/A	Returns the text-pointer value that corresponds to a text, ntext, or image column in varbinary format
UNICODE	MS7. UNICODE ^a	Returns the integer value, as defined by the Unicode standard, for the first character of the input expression
UPPER	UPPER	Returns a character expression with lowercase character data converted to uppercase

a. This function is provided by the IBM DB2 MTK

C.4 Boolean functions

Table C-3 provides the mapping of boolean functions from SQL Server to DB2 UDB.

Table C-3 Boolean function mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
ALL	ALL	Compares a scalar value with a single-column set of values
AND	AND	Combines two Boolean expressions and returns TRUE when both of the expressions are TRUE
ANY	ANY	Compares a scalar value with a single-column set of values
BETWEEN	BETWEEN	Specifies an inclusive range to test

SQL Server	DB2 UDB	Notes
EXISTS	EXISTS	Specifies a subquery to test for the existence of rows
IN	IN	Determines if a given value matches any value in a subquery or a list
LIKE	LIKE	Determines whether or not a given character string matches a specified pattern
NOT	NOT	Negates a Boolean input
SOME	SOME	Compares a scalar value with a single-column set of values

C.5 Date and time functions

Table C-4 provides the mapping of date and time functions from SQL Server to DB2 UDB.

Table C-4 Date and time function mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
DATEADD	see 4.1, "SQL standard compliance" on page 66	Returns a new date time value based on adding an interval to the specified date
DATEDIFF	MS7. DATEDIFF ^a	Returns the number of date and time boundaries crossed between two specified dates
DATENAME	MS7. DATENAME ^a	Returns a character string representing the specified date part of the specified date
DATEPART	see 4.1, "SQL standard compliance" on page 66	Returns an integer representing the specified date part of the specified date
DAY	DAY	Returns an integer representing the day part of the specified date
GETDATE	see 4.1, "SQL standard compliance" on page 66	Returns the current system date and time in the SQL Server standard internal format for date and time values

SQL Server	DB2 UDB	Notes
MONTH	MONTH	Returns an integer that represents the month part of a specified date
YEAR	YEAR	Returns an integer that represents the year part of a specified date

a. This function is provided by the IBM DB2 MTK

C.6 Metadata functions

Table C-5 lists the metadata functions of SQL Server. DB2 UDB does not provide such functions; however, you can get this data by querying the DB2 catalog tables. The corresponding DB2 UDB catalog table and column is presented in the table.

Metadata about the database itself can be obtained from the database configuration of the database manager configuration. For more information refer to the DB2 UDB documentation *Command Reference*, SC09-4828.

Table C-5 Metadata function mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
COL_LENGTH	SYSCAT. COLUMNS	Returns the defined length of a column
COL_NAME	SYSCAT. COLUMNS	Returns the name of a database column given the corresponding table identification number and column identification number
COLUMNPROPERTY	SYSCAT. COLUMNS	Returns information about a column or procedure parameter
DATABASEPROPERTY	Database (Manager) Configuration	Returns the named database property value for the given database and property name
DB_ID	N/A	Returns the database identification number
DB_NAME	N/A	Returns the database name
FILE_ID	N/A	Returns the file identification number (file ID) for the given logical file name in the current database

SQL Server	DB2 UDB	Notes
FILE_NAME	N/A	Returns the logical file name for the given file identification number (ID)
FILEGROUP_ID	N/A	Returns the filegroup identification number (ID) for the given filegroup name.
FILEGROUP_NAME	N/A	Returns the filegroup name for the given filegroup identification number (ID)
FILEGROUPPROPERTY	N/A	Returns the specified filegroup property value when given a filegroup and property name
FILEPROPERTY	N/A	Returns the specified file name property value when given a file name and property name
FULLTEXTCATALOG PROPERTY	N/A	Returns information about full-text catalog properties
FULLTEXTSERVICE PROPERTY	N/A	Returns information about full-text service-level properties
INDEX_COL	SYSCAT.INDEXES	Returns the indexed column name
INDEXPROPERTY	SYSCAT.INDEXES	Returns the named index property value given a table identification number, index name, and property name
OBJECT_ID	SYSCAT.TABLES SYSCAT.VIEWS	Returns the database object identification number
OBJECT_NAME	SYSCAT.TABLES SYSCAT.VIEWS	Returns the database object name
OBJECTPROPERTY	SYSCAT.TABLES SYSCAT.VIEWS	Returns information about objects in the current database
TYPEPROPERTY	SYSCAT.DATATYPES	Returns information about a data type

C.7 Aggregate functions

Table C-6 provides a mapping of aggregate functions from SQL Server to DB2 UDB.

Table C-6 Aggregate function mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
AVG	AVG	Returns the average of the values in a group
COUNT	COUNT	Returns the number of items in a group
GROUPING (OLAP)	GROUPING	Causes an additional column to be output with a value of 1 when the row is added by either the CUBE or ROLLUP operator, or 0 when the row is not the result of CUBE or ROLLUP
MAX	MAX	Returns the maximum value in the expression
MIN	MIN	Returns the minimum value in the expression
STDEV	$\text{SQRT}(\text{VAR}(x) * \text{COUNT}(x) / (\text{COUNT}(x) - 1))$	Returns the statistical standard deviation of all values in the given expression
STDEVP (population)	STDDEV	Returns the statistical standard deviation for the population for all values in the given expression
SUM	SUM	Returns the sum of all the values, or only the DISTINCT values, in the expression
VAR	$\text{VAR}(x) * \text{COUNT}(x) / (\text{COUNT}(x) - 1)$	Returns the statistical variance of all values in the given expression
VARP (population)	VAR	Returns the statistical variance for the population for all values in the given expression

C.8 System functions

Table C-7 provides a mapping of System functions from SQL Server to DB2 UDB.

Table C-7 System function mapping Sfrom SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
APP_NAME	N/A	Returns the application name for the current session if one has been set by the application
CASE	CASE	Evaluates a list of conditions and returns one of multiple possible result expressions
COALESCE	COALESCE	Returns the first non-null expression among its arguments
CURRENT_TIMESTAMP	CURRENT TIMESTAMP	Returns the current date and time. Equivalent to GETDATE() function
CURRENT_USER	USER	Returns the current user. Equivalent to USER_NAME() function
DATALength	LENGTH	Returns the number of bytes used to represent any expression
FORMATMESSAGE	N/A	Constructs a message from an existing message in sysmessages
GETANSINULL	N/A	Returns the default nullability for the database for this session
HOST_ID	N/A	Returns the workstation identification number
HOST_NAME	N/A	Returns the workstation identification number
IDENT_INCR	N/A	Returns the increment value specified during the creation of an identity column in a table or view that has an identity column
IDENT_SEED	N/A	Returns the seed value specified during the creation of an identity column in a table or a view

SQL Server	DB2 UDB	Notes
IDENTITY	N/A	Used only in a SELECT statement with an INTO table clause to insert an identity column into a new table. NOT the same as IDENTITY Property
ISDATE	N/A	Determines whether an input expression is a valid date
ISNULL	COALESCE	Replaces NULL with the specified replacement value
ISNUMERIC	MS7. ISNUMERIC^a	Determines whether an expression is a valid numeric type
NEWID	UDF	Creates a unique value of type uniqueidentifier
NULLIF	NULLIF	Returns a null value if the two specified expressions are equivalent
PARSENAME	SYSCAT.TABLES SYSCAT.VIEWS	Returns the specified part of an object name - object name, owner name, database name, or server name
PERMISSIONS	SYSCAT.DBAUTH	Returns a value containing a bitmap that indicates the statement, object, or column permissions for the current user
SESSION_USER	CURRENT USER	Allows a system-supplied value for the current session's username to be inserted into a table when no default value is specified
STATS_DATE	N/A	Returns the date that the statistics for the specified index were last updated
SYSTEM_USER	CURRENT USER	Allows a system-supplied value for the current system username to be inserted into a table when no default value is specified
USER_NAME	N/A	Returns a user database username from a given identification number

a. This function is provided by the IBM DB2 MTK

C.9 Security functions

Table C-8 provides a mapping of security functions from SQL Server to DB2 UDB.

Table C-8 Security function mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
IS_MEMBER	N/A	Indicates whether the current user is a member of the specified Microsoft® Windows NT® group or Microsoft SQL Server™ role
IS_SRVROLEMEMBER	N/A	Indicates whether the current user login is a member of the specified server role
SUSER_ID	N/A	Returns the user's login identification number. Equivalent to SUSER_SID
SUSER_NAME	CURRENT USER	Returns the user's login identification name. Equivalent to SUSER_SNAME
SUSER_SID	N/A	Returns the security identification number (SID) for the user's login name
SUSER_SNAME	CURRENT USER	Returns the login identification name from a user's security identification number (SID)
USER_ID	N/A	Returns a user's database identification number
USER	CURRENT SCHEMA	Allows a system-supplied value for the current user's database username to be inserted into a table when no default value is specified

C.10 Miscellaneous functions

This section provides a mapping of miscellaneous functions from SQL Server to DB2 UDB.

Table C-9 Miscellaneous function mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
CAST	CAST	Explicitly converts an expression of one data type to another
CONTAINSTABLE	N/A	Returns a table of zero, one, or more rows for those columns containing character-based data types for precise or “fuzzy” (less precise) matches to single words and phrases, the proximity of words within a certain distance of one another, or weighted matches
CONVERT	CAST	Explicitly converts an expression of one data type to another
CURSOR_STATUS	See 4.6.9, “Cursors” on page 88	A scalar function that allows the caller of a stored procedure to determine whether or not the procedure has returned a cursor and result set for a given parameter
FREETEXTTABLE	N/A	Returns a table of zero, one, or more rows for those columns containing character-based data types for values that match the meaning but not the exact wording of the text in the specified freetext_string
OPENQUERY	N/A	Executes the specified pass-through query on the given linked server, which is an OLE DB data source
OPENROWSET	N/A	Includes all connection information necessary to access remote data from an OLE DB data source

Operator mapping

This appendix provides a mapping of operators from SQL Server to DB2 UDB. The following operator types are covered:

- ▶ Arithmetic operators
- ▶ Assignment operators
- ▶ String concatenation operators
- ▶ Comparison operators
- ▶ Logical operators
- ▶ Bitwise operators

Differences between SQL Server functions and DB2 UDB functions are highlighted in **bold**.

D.1 Arithmetic operators

Table D-1 shows a mapping of arithmetic operators for numeric and date time data types from SQL Server to DB2 UDB.

Table D-1 Arithmetic operator mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
+	+	Addition
-	-	Substraction
*	*	Multiplication
/	/	Division
%	MOD	Modulus

D.2 Assignment operators

Table D-2 shows a mapping of assignment operators for variables from SQL Server to DB2 UDB.

Table D-2 Assignment operator mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
SET	SET	Assigns values to variables or parameter

D.3 String concatenation operators

Table D-3 shows a mapping of string concatenation operators for string data types from SQL Server to DB2 UDB.

Table D-3 String concatenation operator mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
+	CONCAT	Returns the concatenation of two string arguments Note: It may be used as a synonym for CONCAT

D.4 Comparison operators

Table D-4 shows a mapping of comparison operators for numeric, data time and string data types from SQL Server to DB2 UDB.

Table D-4 Comparison operator mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
=	=	Is equal to
>	>	Is greater than
>=	>=	Is greater than or equal to
<	<	Is less than
<=	<=	Is less than or equal to
<>	<>	Is not equal to
!=	!=	Is not equal to
!<	>=	Is not less than
!>	<=	Is not greater than

D.5 Logical operators

Table D-5 shows a mapping of logical operators for results of expressions from SQL Server to DB2 UDB.

Table D-5 Logical operator mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
ALL	ALL	Returns TRUE if all of a set of comparisons are TRUE
AND	AND	Returns TRUE if both Boolean expressions are TRUE
ANY	ANY	Returns TRUE if any one of a set of comparisons are TRUE
BETWEEN	BETWEEN	Returns TRUE if the operand is within a range
EXISTS	EXISTS	Returns TRUE if a subquery contains any rows

SQL Server	DB2 UDB	Notes
IN	IN	Returns TRUE if the operand is equal to one of a list of expressions
LIKE	LIKE	Returns TRUE if the operand matches a pattern
NOT	NOT	Returns Reverses the value of any other Boolean operator
OR	OR	Returns TRUE if either Boolean expression is TRUE
SOME	SOME	Returns TRUE if some of a set of comparisons are TRUE

D.6 Bitwise operators

DB2 UDB does not provide bitwise operators. To perform these operations, we provide you with equivalent functions listed in Table D-6 in the Appendix G, “Additional material” on page 509.

All these functions are implemented using external Java user defined function. Below is the Java code for the `bitand()` function:

```
import java.sql.*;
import java.lang.*;
import COM.ibm.db2.app.*;

class DB2BitOperators extends UDF {
public static int bitand(int v1, int v2) {
    try
    {
        return(v1 & v2);
    }
    catch (Exception e)
    {
        return(0);
    }
} // end bitand
}
```

Compile this Java code to a .class file then generate a JAR file. Copy this JAR file into the `\sql1ib\function\jar` directory. This is the directory where DB2 UDB expects to find the external Java function byte code. The function must then be registered in the database. The following command can be issued from the CLP to register the above `bitand()` function in the database:


```
CREATE FUNCTION rb.BitAnd(v1 INT, v2 INT)
RETURNS INTEGER
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'DB2BitOperators.bitand'
```

Table D-6 lists the bitwise operation functions of SQL Server mapped to the functions provided in Appendix G, “Additional material” on page 509.

Table D-6 Bitwise operators mapping from SQL Server to DB2 UDB

SQL Server	DB2 UDB	Notes
&	rb.bitand()	Bitwise AND
	rb.bitor()	Bitwise OR
^	rb.bitxor()	Bitwise exclusive OR
~	rb.bitnot()	Bitwise NOT

Administrative tasks mapping

In this appendix we compare common administrative tasks in SQL Server and DB2 UDB solutions. SQL Server Enterprise Manager has functionality to perform most of these tasks while the DB2 UDB provides most of this functionality through the Control Center, the Command Line Processor, or the Stored Procedure Builder (SPB).

Table 14-3 SQL Server and DB2 UDB comparable tasks

Task	SQL Server	DB2 UDB
Start a server or instance	sqlservr.exe <parameter>	db2start db2admin start
Stop a server or instance	No example	db2stop db2admin stop
View the server options	sp_configure	GET DBM CFG
View the database options		GET DB CFG FOR <i>dbname</i>

Task	SQL Server	DB2 UDB
Update server options	sp_configure ' <i>option_name</i> ', <i>new_value</i>	UPDATE DBM CFG USING ' <i>config_parameter</i> ', <i>new_value</i>
Update database options		UPDATE DB CFG FOR <i>dbname</i> USING ' <i>config_parameter</i> ', <i>new_value</i>
Display active servers or instances	N/A	db2ilist
Access server or instance	isql -U <i>user</i> -P <i>pswd</i> -S <i>server</i>	ATTACH TO <i>instance_name</i> USER <i>user_name</i> USING <i>pswd</i>
Access database	use <i>dbname</i>	CONNECT TO <i>dbname</i> USER <i>user_name</i> USING <i>pswd</i>
List databases in server or instance	sp_helpdb	LIST DB DIRECTORY
List devices or files used by the databases	sp_helpdevice	LIST TABLESPACES or LIST TABLESPACE CONTAINERS
Find space used or available space	sp_helpdb <i>dbname</i> or sp_spaceused	LIST TABLESPACE CONTAINERS FOR (tsid) SHOW DETAIL or LIST TABLESPACES SHOW DETAIL
List database tables	sp_help	LIST TABLES
List table characteristics	sp_help <i>tablename</i>	DESCRIBE TABLE <i>tablename</i>
List source for stored procedures	sp_helptext <i>procname</i>	Use <i>Get Source</i> function of the DB2 Stored Procedure Builder

Task	SQL Server	DB2 UDB
Administer security	grant revoke sp_helpuser sp_addlogin sp_adduser sp_addalias sp_dropalias sp_dropuser sp_droplogin sp_addgroup sp_helpgroup sp_changegroup sp_password	GRANT REVOKE UPDATE DBM CFG USING SYSADM_GROUP <i>group_name</i> UPDATE DBM CFG USING SYSCTRL_GROUP <i>group_name</i> UPDATE DBM CFG USING SYSMAINT_GROUP <i>group_name</i> All the authentication set up is done by an external security mechanism such as Operating System
Backup database	BACKUP DATABASE <i>db_name</i> TO device	BACKUP DATABASE <i>db_name</i> TO <i>/path/file</i>
Restore database	RESTORE DATABASE <i>db_name</i> FROM device	RESTORE DATABASE <i>db_name</i> FROM <i>/path/file</i>
Export a text file from a table	BCP <i>table_name</i> OUT <i>filename</i> ...	EXPORT TO <i>filename</i> OF type SELECT ...
Load a text file into a table	BCP <i>table_name</i> IN <i>filename</i> ...	LOAD FROM <i>filename</i> OF type INSERT INTO <i>tablename</i> or IMPORT FROM <i>filename</i> OF type INSERT INTO <i>tablename</i>
List connected users	sp_who	LIST APPLICATIONS
Kill connected users	KILL <i>spid_number</i>	FORCE APPLICATION
Generate DDLs	N/A	db2look -e

SQL limits

This appendix shows some relevant limits of SQL Server 2000 and DB2 UDB V8.2. You can find a detailed overview of all DB2 UDB SQL limits in the manual DB2 UDB documentation *SQL Reference Volume 1*, SC09-4844.

F.1 Identifier length limits

Table F-1 lists the identifier limits of SQL Server 2000 and DB2 UDB V8.2.

Table F-1 SQL Server 2000 and DB2 UDB V8.2 identifier limits

Item	SQL Server 2000	DB2 UDB V8.2
table name	128	128
column name	128	30
view name	128	128
index name	128	18
constraint name	128	18
cursor name	128	18
password for data source access	128	32
SQL variable	128	64
user name	128	30

F.2 Database limits

Table F-2 lists database limits of SQL Server 2000 and DB2 UDB V8.2.

Table F-2 SQL Server 2000 and DB2 UDB V8.2 database limits

Item	SQL Server 2000	DB2 UDB V8.2
columns per table	1024	1012
table rows	8036	32677
columns per index	16	16
indexes per table	250	32767
index key [byte]	900	1024
max char() size	8000	254
max varchar() size	8000	32672
longest SQL statement [byte]	16.777.216	2.097.152
columns per SELECT statement	4096	1012
columns per INSERT statement	1024	1012

Item	SQL Server 2000	DB2 UDB V8.2
nested stored procedure levels	32	16

Archived

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246672>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246672.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
ch04_examples.sql	Chapter 4 examples text file

ch09_examples.sql Chapter 9 examples text file

put_line.zip Zipped user-defined function samples to enable file output from pure SQL for AIX, HP, Linux, Sun Solaris and Windows. Includes:

- put_lin_readme.pdf
- put_line_aix.db2v72.tar.Z
- put_line_aix.db2v81.tar.Z
- put_line_hpux.db2v72.tar.Z
- put_line_solaris.db2v72.tar.Z
- put_line_linux.db2v72.tar.Z
- put_lin_w2k.ZIP

bitwise_operators.zip Zipped bitwise operators user-defined function samples for pure SQL for AIX, HP, Linux, Sun Solaris and Windows. Includes:

- bit_operators.db2
- bit_operators.jar
- bit_operators.java

redbook.sql REDB00K database script for SQL Server.
redbook_db2.zip REDB00K database script for DB2 UDB.

ITSOApp.zip .NET sample application, includes

- ITSOApp.csproj: .NET Project File
- /bin/Release/PubsApp.exe: Application executable file
- Bin
- Obj
- bin
- obj
- App.ico
- AssemblyInfo.cs
- Closelcon1.gif
- DB.gif
- DSAuthor.cs
- DSAuthor.xsd
- DSAuthor.xsx
- DSAuthorRB.cs
- DSAuthorRB.xsd
- DSAuthorRB.xsx
- DSBrand.cs
- DSBrand.xsd
- DSBrand.xsx
- DSPubName.cs
- DSPubName.xsd

DSPubName.aspx
DSTitles.cs
DSTitles.xsd
DSType.cs
DSType.xsd
DSType.aspx
ITSOApp.csproj.user
ITSOApp.sln
LoginForm.resx
MainForm.cs
MainForm.resx
PubsApp.sln.old
XML.gif

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 500MB minimum
Operating System: Windows 2000
Processor: Pentium
Memory: 512MB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 515. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Oracle to DB2 UDB Conversion Guide*, SG24-7048
- ▶ *Database Transition: Informix Dynamic Server to DB2 Universal Database*, SG24-6367
- ▶ *MySQL to DB2 UDB Conversion Guide*, SG24-7093

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
- ▶ *DB2 Universal Database Federated Systems 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
- ▶ *DB2 SQL PL: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS*, by Zamil Janmohamed, et. al. Prentice Hall, Second Edition, November 2004, ISBN 0131477005

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 Universal Database
<http://www.ibm.com/software/data/db2/udb/>
<http://www.ibm.com/db2/v8>
- ▶ DB2 developerWork
<http://www.ibm.com/developeworks/db2/>
- ▶ DB2 Universal Database V8 Application Development
<http://www.ibm.com/software/data/db2/udb/ad>

- ▶ DB2 Technical Support
<http://www.ibm.com/software/data/db2/udb/support/index.html>
 - ▶ DB2 Extenders™
<http://www.ibm.com/software/data/db2/extenders/>
 - ▶ IBM Manuals for Data Management Products
<http://www.ibm.com/software/data/technical/B00K/>
 - ▶ DB2 Migrate NOW!
<http://www.ibm.com/db2/migration>
- Microsoft**
- ▶ Microsoft SQL Server home page
<http://www.microsoft.com/sql/>
 - ▶ Microsoft MSDN library
<http://msdn.microsoft.com/sql/>

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

186
186
% 189
.NET 129, 189
.rpt 145
? 199

A

acceptance testing 402
access methods 339, 364, 372, 391, 393
access plan 21–23, 33–35, 377, 393–394
activate database 270, 345
Active Directory 29
Active Directory Helper 17
ActiveX 129, 238–239, 242
Activity Monitor 50, 341
ADO 4, 88, 129, 238–240, 242
agents 16–17, 363, 419
AIX 2, 5, 120, 139, 313
ALTER BUFFERPOOL 295, 355–356
Alter Table Space 283
ALTER TABLESPACE 19, 60, 276, 282, 284, 356
Analyze 4, 54, 127, 132, 256, 261, 383, 387–388
APAR 420
API 14–15, 27, 130, 189, 238, 249, 306, 326, 415
application xxiii–xxv, 1, 4–6, 12, 14, 16–17, 22, 24, 26, 32, 34, 39–40, 46–47, 51, 63–64, 70–71, 78, 87, 119–122, 124–132, 135, 137, 143, 149, 177, 182, 185, 190, 193, 200, 208, 215, 224, 226, 266, 272, 290–291, 339–340, 344–345, 397, 399, 401–402, 422, 461–462, 478, 482, 491, 503
application layer 260
architecture profiling 120
archival 289
archive logging 20–21, 292, 307, 314, 347
ARCHRETRYDELAY 312
assessment 9, 120, 122
ATTACH 265, 502
authentication 26–27, 29, 46, 250, 410, 503
authorities 26–29, 123
Authorized Program Analysis Reports 420
AUTOCONFIGURE 344–345

Automatic database backup 307–308, 346–347
Automatic database statistics 338, 347
automatic maintenance 49, 267–268, 272, 306, 346–347, 349
Automatic reorganization 334, 348–349
automatic table statistics collection 23
automatic statistics profiling 23

B

backup 3, 15–16, 20–21, 23–24, 36–37, 49, 54, 57–58, 63, 127, 129, 134, 143, 263, 268, 289–290, 346–349, 402, 413, 467, 503
BACKUP DATABASE 294, 467, 503
backup image 21, 58, 268, 306, 308, 315, 329, 347
BCP 54, 170, 175, 177, 467, 503
buffer pool tuning 355
buffer pools 13, 15, 70, 231, 263, 270, 279, 294, 345, 355–357, 418
Bulk Copy Program 23, 170, 467
Business Intelligence xxv, 1–2, 4, 45–46, 126
Business intelligence 4, 127
business logic layer 260

C

C 4–5, 39, 53, 120, 126, 130, 139, 153, 176, 189, 281, 388, 412, 423, 482
C++ 189
calculate index sizes 366
CHNGPGS_THRESH 344, 356–358
circular 289
circular logging 20, 291–293
client 21, 26–27, 40, 47, 64, 120, 129–130, 139, 189–190, 234, 242, 252, 320–321, 324, 326, 373–374, 382, 409, 411, 422, 480
clone 329
CLP 16, 52–53, 59, 93, 129, 162, 167, 178, 190, 199, 265, 267, 270, 272
CLR 4–5, 39, 189, 209
clustered index 158, 163–164, 331, 365, 380
COBOL 128, 130, 189
Columns 4, 35, 63, 69–70, 76, 137, 158, 182, 209, 216, 218, 222, 261, 331, 335–336, 340, 364–366, 488, 494, 506

Command

- ATTACH 502
- CONNECT 52, 167, 250, 271, 417, 502
- DESCRIBE TABLE 502
- GET DB CFG 501
- GET DBM CFG 501
- LIST DB DIRECTORY 502
- LIST TABLES 502
- LIST TABLESPACE CONTAINERS 288, 502
- LIST TABLESPACES 286, 502
- PRUNE HISTORY 312
- UPDATE DB CFG 502
- UPDATE DBM CFG 502
- Command Line Processor 52, 93, 129, 178, 190, 199, 271, 344, 411, 467, 482
- common language runtime 4–5, 189
- complexity 2, 9, 122–123, 127, 257
- configuration 4, 11–12, 15, 21, 23–26, 28, 33, 40–41, 46, 48, 50–51, 121, 132–133, 152, 179, 215, 234, 254, 271–274, 340, 342–344, 398–399, 410–411, 466, 488, 511
- Configuration Advisor 3, 49–51, 133, 272, 340, 342
- Configure Client Reroute 319
- CONNECT 6, 26, 28–29, 32, 52–53, 60, 119, 121, 126, 128, 167, 171, 232, 240, 250, 270, 356, 388–389, 463–464, 502
- Connect 270
- Connection Concentrator 3, 16
- constraints 125, 138, 156–157, 371, 398, 405–406
- containers 18–19, 38, 56, 70, 263, 267, 273–274, 276, 282, 352–354, 412, 419, 466–467, 502
- Content management 1–2, 126–127
- Control Center 15–16, 32–36, 49–50, 58–59, 64, 122, 129, 133, 176, 178, 180, 265–268, 342, 346–347, 350, 367, 467, 501
- ControlCenter 122
- conventions 130, 351
- conversion xxiii, 1, 9, 68, 70–71, 75, 119–129, 132, 135, 138, 142, 144–145, 147–149, 185, 192, 216, 223, 226, 228, 341, 401
- Convert 64, 67, 70–71, 73, 121, 123–128, 143–145, 149, 163, 166, 172, 182, 211, 218–219, 223, 227, 429, 432–433, 494
- cost-based optimizer 22, 335
- Create Index Wizard 367
- Cursor Stability 234, 237–238, 345, 389

D

- data checking 397, 403
- data migration testing 401
- data storage 18, 364, 366
- Data Transformation Services 170
- data type support 3
- Database Managed Spaces (DMS) 352
- database population method 315
- datafile 18
- DB2 CLI 4, 88, 246
- DB2 Data Links Manager 3
- DB2 Development Center 4
- DB2 Everyplace 4
- DB2 maintenance utilities 57
- DB2 Net Search Extender 3
- DB2 packaging 5
- DB2 Spatial Extender 3, 411
- DB2 Technical Support site 419
- DB2 version 2, 6, 24, 290, 294, 407, 417
- DB2 Version 8.2 8
- DB2_PARALLEL_IO 354
- db2AdminMsgWrite 415
- db2dasdiag.log 25
- db2dasrm.exe 17
- db2diag.log 24–25, 361, 412–415, 418
- db2empfa 274, 351
- db2fmp.exe 17
- db2ilist 264, 502
- db2jds.exe 17
- db2level 417
- db2licd.exe 17
- db2look 54, 58–59, 61, 405, 503
- db2rcmd.exe 17
- db2sec.exe 17
- db2secv82 29
- db2setup 264
- db2start 266, 330, 501
- DB2STOP 270
- db2support.zip 417
- db2syscs 12, 17, 25, 414
- db2syscs.exe 17
- db2systray.exe 17
- DBHEAP 344, 362
- DBI 130–131, 411
- DDL 1, 54, 58–61, 121, 125, 128, 138, 143–145, 147, 191, 383, 427
- Deletion of index entries 367
- delta backup 21
- DEPLOY 5, 39, 75, 139, 143, 146–147, 165, 174,

255, 482
 Deploy 145, 165, 482
 DESCRIBE TABLE 502
 Design Advisor 49–51, 341–342, 379
 Developer Domain 133
 DGT 186–187
 DIAGLEVEL 24, 413–414
 diagnostic logs 411–412
 directory containers 273
 Disk mirroring 329
 Distributed Transaction Coordinator 17
 DML 67, 374
 DMS 18–19, 57, 70, 152, 154, 273–276, 352, 354, 365, 412, 466
 drop 28, 32–33, 57, 64, 144, 186, 254, 264, 267, 269–270
 DSS 120
 DTS 170, 175
 dump files 412, 416–417

E

EDUs 17
 embedded 2, 34, 66, 84, 128, 130, 238–239, 249
 engine dispatchable units 17
 entity-relation 121
 Entity-Relationship 125
 error log 24–25
 estimation 122, 374
 Event Monitor 133, 233
 executive summary 398
 EXPLAIN 35–36, 48, 54, 236, 337, 376–377, 383, 411, 467
 EXPORT 32, 40, 54, 129, 169–170, 175, 258, 261, 416, 467, 503
 extent 18, 127, 274, 276, 351, 379, 394
 external function 209
 Extracting 58, 63, 74, 122, 128, 138, 143, 401, 428

F

FAILARCHPATH 293–294, 312
 Federated Web Services 3
 Federation 127, 134, 177
 federation 3, 178
 file 12, 18–19, 21, 25–26, 28–29, 52–53, 56–59, 64, 70, 99, 114, 125, 128, 138, 140, 143–145, 152, 174–176, 252, 258, 261, 266, 272, 274–275, 351–352, 354, 358, 404–405, 411–412, 466, 474, 476, 482, 488, 498, 503, 509–511

filegroup 18, 466–467, 489
 FixPak 8, 25, 120, 411, 417, 420, 422
 FORCE APPLICATION 503
 Foreign Key 371
 FORTRAN 130
 functional testing 401, 409
 functions 4, 29, 34–35, 39, 42, 70–71, 73–74, 121, 125–126, 128–130, 138, 182, 190, 208–210, 226, 246–247, 264, 306, 328, 374, 399, 401, 408, 472, 481–484, 498–499

G

Generate Data Transfer 143, 145–146, 172, 438
 GET DB CFG 271, 351, 467, 501
 GET DBM CFG 267, 501
 GET SNAPSHOT 360, 362
 Gigabytes 277
 global temporary table 186, 188
 GUI 19, 31–32, 34, 36–42, 44–47, 52, 58–59, 127, 133–134, 142, 152, 267, 296, 339, 341–342, 402

H

HADR
 See High Availability Disaster Recovery
 See High Availability Disaster Recovery
 Health Center 38–39, 47–48, 307, 334, 338, 347
 Health Center/Monitor 3
 health Monitor 39, 350, 415
 High availability 2, 8, 11, 21, 30, 50, 127, 134, 263, 313–314, 325–328
 High Availability Disaster Recovery 3, 30, 313
 HIPER APAR 420
 HTML xxvii, 6, 9, 53, 61, 69, 76, 78, 129–130, 132–133, 254, 358, 417, 420

I

IBM Software Support Center 420
 IBM support 420
 IBMDEFAULTBP 294, 355–356, 419
 IMMEDIATE 33, 97, 153, 175, 211, 219, 293–295, 324, 355–356, 373, 375, 406, 415
 IMPORT 32, 40, 54, 125–126, 129, 143, 146–147, 169–170, 173, 175, 191, 258, 299–300, 404–405, 416, 467, 498, 503
 Import 54, 144, 146–147, 173, 176, 258, 300, 404, 498
 importing 138, 143–144, 170

- Include option 368–369
- incremental backup 20–21, 305
- index expansions 368
- index leaf pages 331, 366, 370–371
- Index placement 365
- index rebuild 229
- index reorganization 334, 372
- index space estimates 366
- index tree 364, 367, 370
- Indexes 4, 18–19, 21–22, 28, 32, 35, 51, 62, 70, 117, 120–121, 125, 138, 152, 154, 158, 176, 187, 231–232, 254, 273, 275, 284, 340–341, 346, 352, 489, 506
- indexes and constraints 371
- indexing columns 364
- indexing strategies 364
- Information integration 3, 126–127
- installation 9, 12, 17, 29, 41, 48, 99, 139–142, 187, 253, 258, 261–262, 264, 297, 340, 359, 411, 416, 482
- instance 4, 12–17, 24–29, 33, 38, 48, 50, 56, 69, 77–78, 92, 121, 123–124, 187, 190, 212, 214, 249–250, 258, 264–267, 340, 350, 361, 363, 402, 413–415, 466, 501–502
- integration testing 402
- interface 53, 124, 128–132, 142, 226, 285–286, 326, 411, 461–462
- Inter-partition parallelism 24
- inter-query parallelism 24
- Intra-partition parallelism 24
- intra-query parallelism 24
- Isolation Levels 234–235, 345
- Isolation Levels - changing 235
- isql 52, 467, 502

J

- Java 2, 4, 32, 39, 64, 91, 120, 126, 128–131, 138, 189, 191, 209, 238–239, 249, 255, 402, 478–479, 482
- JDBC 2, 4, 17, 88, 130, 138, 144, 162–163, 171, 238–239, 249, 478

K

- Kilobytes 277

L

- language xxiii, 1, 4–5, 65, 68, 72, 75, 120, 122,

- 125–126, 128, 130–131, 189–192, 251
- leaf page 370
- Linux xxv, 1–2, 5, 25, 62–63, 120, 131, 139, 171, 264, 266, 342, 355, 358–359, 413, 415–416, 419, 424
- LIST APPLICATIONS 414, 503
- LIST DB DIRECTORY 502
- LIST TABLES 53, 502
- LIST TABLESPACE CONTAINERS 288, 502
- LIST TABLESPACES 53, 286, 502
- LOAD 4, 23–24, 28, 32, 49, 54, 138, 146–147, 169–170, 173–174, 232, 289, 299, 354, 402, 404–405, 467, 503
- Load 23, 54, 146, 173–174, 330, 402, 404, 410
- load stress testing 402
- Local System account 29
- Lock escalation 229, 233, 359–361, 414
- lock escalation 233, 359–361
- lock snapshot 360–361
- LOCK TABLE 232–233
- Locking 15, 123, 158, 186–187, 226, 229–231, 330, 359–360, 410
- LOCKLIST 359–360
- lock-placement 229
- log mirroring 30, 292, 327
- log space 20, 290, 307, 347–348, 358, 362
- LOGBUFFSZ 362–363
- LOGFILSIZ 290–291, 293–294, 358–359
- logical log 290, 353, 361
- logical log buffer 353, 361
- logins 26
- LOGPRIMARY 290–291, 293–294, 358–359
- LOGRETAIN 294, 311–312
- LSA 29

M

- Managing logs 290
- Manual methods 125
- MAXLOCKS 360
- MDC 4, 233, 331, 341, 379–381
 - dimensions 331, 379
 - extent sizes 379
 - syntax 379–380
 - two-dimensional 379
- Memory Visualizer 3
- messages files 412, 416
- metadata 45–46, 58, 91, 93, 138, 143, 182, 267, 481, 488

- metadata transport 125
- Microsoft SQL Server 1–2, 5, 7, 11, 24, 163, 171, 422, 424, 493
- Migration Toolkit 8, 121, 182, 423
- migration tools xxvi, 182
- mixed-mode 26
- Mobility on demand 4
- model 14, 16–17, 20–21, 25, 44, 122, 125, 129, 253, 291
- modeling tools 121, 125
- monitor 3, 31, 38–39, 44, 50, 61, 132–133, 233, 284–286, 291, 341, 350, 355, 362, 415
- MQT 4, 341, 373–376
 - creation 373
 - Dynamic SQL 375
 - Optimization class 375
 - Query rewriting 375
 - Refresh deferred 374
 - Refresh immediate 374
 - REFRESH TABLE 374, 377
 - refreshing 378
 - when to use 378
- msdtc.exe 17
- MTK 8, 75, 121–122, 126, 128, 131, 137–143, 145, 147–148, 152, 169, 171–172, 226, 408, 422–423, 428, 482, 484, 486, 488
- Multidimensional Cluster
 - See MDC
 - See MDC
- Multidimensional data clustering
 - see MDC
- multi-page 351–352

N

- nesting level 189, 203
- NEWLOGPATH 290, 327, 353
- non-logged operation 21
- notify files 412, 415
- NOTIFYLEVEL 415
- NUM_IO_CLEANERS 354
- NUM_IOSERVERS 354
- NUMARCHRETRY 312

O

- ODBC 4, 8, 40, 88, 128–131, 138, 144, 171, 175, 179, 238–240, 242, 422
- offline backup 21, 291, 307, 347
- OLE DB 4, 88, 129, 175, 238–240, 242, 494

- OLTP 120, 340, 352, 355, 360
- Online load 4
- Online storage management 4
- Open Data Services 16
- optimization class 22–23, 236, 384, 388, 390–391
- optimizer 21–23, 89, 133, 157, 335, 346, 373–375, 381
 - Cartesian products 393
 - classes 23, 391–392
 - composite inner tables 393
 - cost based 22, 382
 - Dynamic Programming 392
 - Greedy join enumeration 392, 394
 - Merge Scan joins 392
 - Non-uniform distribution statistics 391, 393
 - query plans 21, 376, 381, 383
 - star-join strategy 392
 - table scans 392
 - Visual Explain 383, 387

P

- Packages 5, 40, 61, 138, 235, 253, 329, 341, 388, 399, 467
- parallel load 23
- parallelism 5, 23–24, 58, 294, 306, 344, 354
- partitioned databases 3, 16
- partitioning strategies 341
- Performance Expert 62, 133
- performance testing 402, 410
- Performance tuning 11, 35, 119, 132–133, 135, 294, 339–341, 350
- performance tuning 132–134, 339–340, 350, 383
 - buffer pools 355
 - data organization 346
 - database statistics 346
 - instance and database 340
 - multiple buffer pools 357
 - processes 363
- PERL 130–131
- Perl 130–131
- phantom rows 237
- PL/SQL 88, 138
- planning tools 123
- point in time recovery. 20
- Porting 1, 9, 117, 120–124, 126, 128, 132, 135, 151, 294
- preparation 119–120, 124, 140, 258, 297, 317
- Primary Key 156, 163, 165, 187–188, 371–372

privileges 26–29, 91, 160–161, 264, 304, 410
problem determination 25, 410–411, 413, 416–417

Q

query plan analysis 387
Queue replication 43, 328
QUIESCE 266

R

raw device 18, 273, 466
Read Stability 234, 237, 345
recovery 2–3, 8, 20–21, 30, 50, 62, 127, 134, 263, 281, 288–289, 342, 345, 351–352, 467
recovery model 20–21
Redbooks Web site 509, 515
 Contact us xxvii
Referential Integrity 63, 114, 169–170, 174
Refine 134, 143, 145–146, 172, 433
REORG INDEXES 331, 372
REORGCHK 54, 330–332, 348
Repeatable Read 234, 236–238, 345
replication 2–3, 30, 43–45, 127, 134, 170, 177, 180–181, 313, 328, 411
Reports 62, 138, 143, 145, 175, 400, 420
requirements 1, 3, 5, 70, 123, 139, 146, 186, 226, 257, 267, 296, 328, 355, 366, 373, 378, 400, 410, 511
resouce 122
RESTORE DATABASE 311, 323, 467, 503
reverse-engineer 122
Role 26, 161, 493
RUNSTATS 22–23, 54, 134, 263, 308, 330, 332, 336, 346–348, 365, 371, 467
runstats 23, 330, 335–336, 346–348, 377

S

scalar function 209, 494
scripts 13, 36, 117, 119, 121, 129, 138, 143, 145–146, 169, 172, 174, 252, 254, 297, 299–300, 403, 406, 429
Sequences 69, 138
SET INTEGRITY 175, 406
severity indicator 411
showserver 502
SMPO 122
SMS 18–19, 70, 273–275, 351, 412, 466
snapshot 93, 285, 329–330, 360–363

Snapshots 94, 133, 285, 362, 417
sourced function 209
sp_addalias 503
sp_addgroup 503
sp_addlogin 503
sp_adduser 503
sp_changegroup 503
sp_configure 33, 501–502
sp_dropalias 503
sp_droplogin 503
sp_dropuser 503
sp_help 502
sp_helpdb 502
sp_helpdevice 502
sp_helpgroup 503
sp_helptext 502
sp_helpuser 503
sp_password 503
sp_spaceused 502
sp_updatestats 22, 54
sp_who 503
Specify Source 143, 171, 425, 429
split image 315
split mirror function 329
split mirror image 329
SQL xxiii, xxv, 1–2, 4–5, 7–9, 11–16, 20, 22–26, 28–29, 32–37, 39, 41, 45, 47, 51–52, 54, 58, 62–63, 65–68, 119–122, 125–126, 128–131, 133, 137–139, 142–144, 148–149, 151–153, 169–170, 172, 185–188, 226, 263, 266–267, 340–341, 351, 368, 402, 405–406, 408, 411, 422, 424, 465–467, 471, 476, 478, 481–484, 495–498, 501, 505–506
SQL Agent 17
SQL function 209
SQL PL 96, 99–101, 138, 189–190, 192, 199, 250, 408
SQL replication 43, 181, 328
SQL Server xxiii, 7–8, 11–12, 14, 16–18, 20–24, 26–28, 30–33, 35, 40, 42–43, 45, 49, 52, 54, 58, 61, 121–122, 137, 144
SQL standards 79, 125
SQL Translator 143, 145, 148–149, 182, 226
SQL*Plus 138
sqladhl.exe 17
sqlagent.exe 17
SQLCODE 89, 99, 101, 214, 222, 251, 404, 411
sqldiag 24–25
SQLj 2, 88, 130, 239, 249, 482
SQLJ programming interfaces 4

- SQLWays 121, 182
- standby database 312–313, 315–316, 329
- standby instance 314–315
- startserver 501
- statistical information 22, 92, 285, 330–331, 335, 371, 382
- Storage Management 4, 284–285
- Storage Management tool 284
- Storage Manager 284, 291, 307
- storage objects 285
- stored procedure 22, 33, 68, 85, 91, 93, 123, 126, 128, 138, 189, 191–193, 335, 422, 494, 501–502, 507
- structure 14, 44, 91, 103, 109, 114, 119, 121, 123–128, 144, 157–158, 170–171, 186, 251, 257, 273, 329, 335, 359, 369
- Sybase 6, 8–9, 121
- System Managed Space 18–19, 273–274, 287

T

- table function 91, 209–210, 213
- Table reorganization 229, 232, 333
- table space 18–19, 28, 30, 38–39, 49, 55–57, 62, 70, 92, 152, 160, 186–187, 229, 273–275, 350–354
- table space state 286, 289
- TechNotes 420
- temporary table 186–188, 273, 351–352, 354, 356
- TERMINATE 271
- test deliverables 400
- test phases 401, 403
- test strategy 399
- text 2–3, 6, 25, 53, 56–57, 64, 125, 129, 143, 175–176, 183, 244, 254, 402, 415, 464, 470, 485–486, 489, 494, 503
- threads 16–17
- tools xxiii, xxvi, 1–4, 13, 25, 30–34, 36–42, 44–45, 47–48, 52, 61–62, 119–128, 131, 133, 137–138, 143, 162, 167, 169, 175, 182, 226, 254, 257, 267, 296–298, 339, 383, 399–400, 410, 422
- transaction log 16, 19–21, 290, 308, 353, 358, 467
- transaction logging 19–20, 85, 289
- Translator 138, 143, 145, 148–149, 165, 182, 226
- trap files 412–413, 415
- Triggers 32, 96, 117, 121, 126–128, 138, 173, 176, 190, 215–219, 254, 408, 422
- T-SQL 22, 74, 88, 96, 99, 121, 189, 192, 202–203
- Type 2 130, 158, 231–232
- Type 3 130

- Type 4 130
- Types of locks 230
 - Exclusive lock 230
 - Shared lock 230
 - Update (promotable) lock 230

U

- Uncommitted Read 231–232, 234, 236, 238, 345
- UNIX xxv, 1–2, 5, 17, 25, 62–63, 131, 171, 264, 266, 355, 358–360, 413, 415–416, 419, 424
- Unload 54, 63, 138–139, 146, 260
- UPDATE DB CFG 272, 293–294, 312, 353–354, 357, 359, 502
- UPDATE DBM CFG 267, 502–503
- Update Statistics 54, 335–336, 347, 467
- use command 502
- User Mode Scheduler 16
- user-defined functions 126, 128
- USEREXIT 294, 311–312
- userexit 294, 311

V

- variables 77, 80, 99, 124, 130, 211, 250–252, 496
- video 1–2, 6, 129
- Views 4, 32, 64, 78, 91, 95, 121, 138, 148, 159–160, 166, 180, 185, 222–223, 254, 373, 422, 489, 492
- Visual Explain 35, 54, 337–338, 383–386
- volume stress testing 402

W

- Web integration 127
- Web services 3, 129
- WebSphere 4, 6, 39, 47, 64, 116, 169–170, 177, 179, 249, 253, 255–256, 328
- WebSphere MQ 328
- WebSphere Studio 39, 122
- WebSphere Studio Application Developer plug-in 255
- wilcard parameter 189
- Windows authentication 26–27
- wizard 19, 34, 37, 47, 54, 175, 182, 267–269, 272, 341–342, 346–347
- workshop 9, 258
- write-ahead logging 290

X

XML 1, 3–4, 6, 39, 64, 103–106, 129



Redbooks

Microsoft SQL Server to IBM DB2 UDB Conversion Guide

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



Redbooks

Microsoft SQL Server to IBM DB2 UDB Conversion Guide

**Complete guide to
convert from
Microsoft SQL Server
to IBM DB2 UDB**

**Application
enrichment through
advanced DB2 UDB
features**

**Application
conversion with
detailed examples**

DB2 Universal Database (DB2 UDB) has long been known for its technology leadership. This IBM Redbook is an informative guide that describes how to convert the database system and applications from Microsoft SQL Server to DB2 UDB Version 8.2.

This guide presents the best practices in conversion strategy and planning, an architecture overview of the two relational database systems, migration tools, and practical conversion examples. It is intended for technical staff involved in a SQL Server to DB2 UDB conversion project.

We discuss, in detail, the conversion of database structure, data, and database objects including stored procedures, triggers, and user defined functions. We address SQL and application conversion considerations. Examples are used throughout the book to illustrate the conversion and to provide solutions and work-arounds for different scenarios.

In addition, we provide a DB2 tools and wizard usage overview, performance considerations, and testing and troubleshooting techniques. Finally, a migration scenario using the IBM DB2 Migration Toolkit is included to show the complete migration process.

**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