DB2 Universal Database for OS/390

Release Planning Guide

Version 6

SC26-9013-02

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix H, “Notices” on page 313.

Third Edition, Softcopy Only (April 2000)

This edition applies to Version 6 of DB2 Universal Database Server for OS/390, 5645-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

This softcopy version is based on the printed edition of the book and includes the changes indicated in the printed version by vertical bars. Additional changes made to this softcopy version of the manual since the hardcopy manual was published are indicated by the hash (#) symbol in the left-hand margin. Editorial changes that have no technical significance are not noted.

Contents

Chapter 1. Introduction to this book and the DB2® for OS/390® library . . 1
Who should read this book . . . 1
How this book is organized . . . 1
Product terminology and citations . . . 2
How to read the syntax diagrams . . . 2
How to use the DB2 library . . . 3
| How to obtain DB2 information . . . 6
| DB2 on the Web . . . 6
| DB2 publications . . . 6
| DB2 education . . . 7
| How to order the DB2 library . . . 7
| Summary of changes to DB2 UDB for OS/390 Version 6 . . . 7
| Capacity improvements . . . 7
| Performance and availability . . . 7
| Data sharing enhancements . . . 9
| User productivity . . . 9
| Network computing . . . 9
| Object-relational extensions and active data . . . 10
| More function . . . 11
| Features of DB2 for OS/390 . . . 11
| Migration considerations . . . 11

Chapter 2. Capacity improvements . . . 13
16-terabyte tables . . . 13
Terminology changes . . . 13
Page set types and new storage limits . . . 14
Creating EA-enabled table spaces and index spaces . . . 14
Changes to utilities . . . 15
Catalog changes . . . 15
| Many more open data sets . . . 15
Larger secondary quantity value . . . 15
More pieces for nonpartitioning indexes . . . 15
Increased total number of extents . . . 16
Buffer pools in data spaces . . . 16
Total storage in the ssnmDBM1 address space . . . 17
Dynamic statement cache in a data space . . . 17

Chapter 3. Improved performance and availability . . . 19
Rebalancing data in a partitioned table space . . . 19
Why rebalance data? . . . 19
The process: ALTER INDEX then REORG . . . 20
REORG-pending status . . . 21
Recovery guidelines . . . 22
Altering variable length columns . . . 23
Changing the limit key value . . . 24
Immediate index access . . . 24
| Dynamically change checkpoint frequency . . . 24
| Why change the checkpoint frequency? . . . 24
| Using the SET LOG command . . . 25
| Displaying the current LOGLOAD value . . . 25

Faster restart and recovery . . . 25
Postponing backout processing at restart . . . 26
Fast log apply . . . 29
| Faster log read . . . 29
Faster, more parallel utilities . . . 29
Faster backup and recovery . . . 30
Parallel index build . . . 32
Inline statistics . . . 33
Faster discard and unload during REORG . . . 33
Decreased elapsed and processor time for online REORG . . . 34
Avoid delete and redefine of data sets . . . 34
Query performance and optimization enhancements . . . 35
Query parallelism enhancements . . . 35
Improvements to join processing . . . 36
Other query optimization enhancements . . . 41
Data sharing enhancements . . . 43
Continuous availability with group buffer pool duplexing . . . 43
Faster checkpointing of group buffer pools . . . 50
Reduced P-lock overhead . . . 51
More caching options . . . 51
More performance and availability improvements . . . 53
Direct row access . . . 53
Declared temporary tables . . . 57
Increased flexibility with 8-KB and 16-KB page sizes . . . 58
Preserving a prior access path . . . 59
More buffer pool tuning options . . . 61
Control of space map copy maintenance . . . 63
Reduced DBD logging for CREATE, ALTER, DROP . . . 63
Improved performance for DROP . . . 63
Larger log buffer sizes . . . 63
| Authorization caching for stored procedures and user-defined functions . . . 64
More flexibility when altering space allocations . . . 64
Deferred allocation of data sets . . . 64
More command concurrency . . . 64
| Increased concurrency for RRSAF and IMS transactions . . . 64

Chapter 4. User productivity . . . 67
Built-in function extensions . . . 67
New ROWID data type . . . 67
Characteristics of the ROWID data type . . . 68
Defining a ROWID column . . . 68
Using a ROWID column as the partitioning key . . . 69
Casting to a ROWID . . . 70
Inserting into a ROWID column . . . 70
Declaring host variables for ROWID columns . . . 71
DB2 REXX Language Support . . . 72
More flexibility and control . . . 72
Predictive governing . . . 72
Statement cost estimation . . . 79
Set default buffer pools . . . 83
More information available for monitoring DB2 . . . 83
IRLM enhancements . . . 85
Display IRLM coexistence information . . . 85
Option to prevent disconnecting IRLM on DB2 shutdown . . . 86

More control over IRLM storage . . . 87
Support for automatic restart manager . . . 89
Improved serviceability . . . 89
More user productivity enhancements . . . 91
DSNTEP2 available in object form . . . 91
| Customized DB2I defaults can be migrated . . . 91
Numeric data type extensions for identity columns . . . 91
Savepoints to undo selected changes . . . 93
More tables allowed in SQL statements . . . 94
SQL extensions . . . 94
More character conversions . . . 95
Utility usability and functionality enhancements . . . 95
Enhanced database commands . . . 97
Support for multi-volume DASD archive log data sets . . . 98
| Remote site recovery copy flexibility . . . 98
| Better retention of installation values across migrations . . . 98
| Better diagnostic information for commands executed through IFI . . . 99

Chapter 5. Improved network computing . . . 101
Java enablement . . . 101
Better performance for Java applications with SQLJ . . . 101
JDBC application support . . . 102
DRDA support for three-part names . . . 103
Benefits of DRDA access . . . 103
Benefits of using three-part names for DRDA access . . . 103
Restrictions on DRDA access programs that use three-part names . . . 104
Preparing applications with three-part names to use DRDA access . . . 105
Moving from DB2 private protocol access to DRDA access . . . 106
Choosing a default database protocol . . . 107
Stored procedure enhancements . . . 107
Creating and modifying stored procedure definitions . . . 107
Changes to stored procedure security . . . 109
Changes to stored procedure invocation . . . 110
Using SQL procedures . . . 113
Improved data transfer with OPTIMIZE FOR n ROWS . . . 115
DB2 ODBC enhancements . . . 118
| Faster ODBC catalog queries . . . 119
Better performance for dynamic SQL applications . . . 119
Improvements for dynamically prepared SQL statements . . . 119
DB2 database connection pooling . . . 120
Using type 2 inactive threads . . . 120
Determining if a thread can become inactive . . . 121
Enabling threads to become inactive . . . 121

Chapter 6. Object-relational extensions and active data . . . 123
Working with large objects (LOBs) . . . 124
Introduction to defining LOBs . . . 124
Declaring LOB host variables and LOB locators . . . 127
LOB materialization . . . 132
Using LOB locators to save storage . . . 132
Deferring evaluation of a LOB expression to improve performance . . . 133
Indicator variables and LOB locators . . . 135
| Valid assignments for LOB locators . . . 136
LOB system processing . . . 136

Managing buffer pools for LOBs . . . 136
Locking LOBs . . . 136
Recovering table spaces that contain LOBs . . . 141
Creating and using user-defined functions . . . 143
Overview of user-defined function definition, implementation, and invocation 143
Defining a user-defined function . . . 146
Implementing an external user-defined function . . . 150
Preparing a user-defined function for execution . . . 188
Invoking a user-defined function . . . 192
Syntax for user-defined function invocation . . . 193
Ensuring that DB2 executes the intended user-defined function . . . 193
Casting of user-defined function arguments . . . 199
What happens when a user-defined function abnormally terminates . . . 200
Other considerations for user-defined function invocation . . . 200
Creating and using distinct types . . . 202
Introduction to distinct types . . . 202
Creating a distinct type . . . 202
Using distinct types in applications . . . 203
Combining distinct types with user-defined functions and LOBs . . . 208
Using triggers for active data . . . 211
Example of creating and using a trigger . . . 212
Parts of a trigger . . . 213
Invoking stored procedures and user-defined functions from triggers . . . 218
Trigger cascading . . . 219
Ordering of multiple triggers . . . 220
Interactions among triggers and referential constraints . . . 221
Creating triggers to obtain consistent results . . . 223
DB2 Extenders™ . . . 225

Chapter 7. Features of DB2 UDB Server for OS/390 . . . 227
Control Center for DB2 UDB . . . 227
DB2 Stored Procedures Builder . . . 228
DB2 Installer . . . 228
DB2 Visual Explain . . . 228
DB2 Estimator . . . 229
Net.Data for OS/390 . . . 229
Query Management Facility . . . 229
DataPropagator Relational . . . 230
DB2 Performance Monitor . . . 230
DB2 Buffer Pool Tool . . . 231
DB2 Administration Tool . . . 231

Chapter 8. Planning for migration and fallback . . . 233
Migration considerations . . . 233
Type 2 indexes are required . . . 233
Data set password protection is removed . . . 233
Shared read-only data is removed . . . 233
Remove views on two catalog tables . . . 234
Private protocol function not enhanced . . . 234
More than 32 K databases are supported . . . 234
Log buffer size increased . . . 234
Consider enlarging BSDS . . . 234
Increase maximum number of data sets open . . . 234
Customized DB2I defaults can be migrated . . . 234

| DB2 online help reader not used . . . 235
Stored procedures . . . 235
| ALTER TABLE changes . . . 236
Utility enhancements . . . 236
Work file database size calculations . . . 236
Changes to Subsystem parameters . . . 236
Release incompatibilities . . . 237
Adjust application programs . . . 238
Examine all new and changed values for DB2I panels . . . 241
Changes to the RLST . . . 241
SYSIBM.SYSPROCEDURES no longer used . . . 241
An 'X' plan in the PLAN_TABLE . . . 242
Limit backouts with system restarts . . . 242
Changes to IFCID fields . . . 242
| DISPLAY BUFFERPOOL changes . . . 242
| Index changes . . . 242
ALTER INDEX syntax . . . 243
RECOVER INDEX becomes REBUILD INDEX . . . 243
Work space formulas changed for utilities . . . 243
| Support for up to 150000 connections . . . 244
| Change to parameter in IRLMPROC startup procedure . . . 244
Release coexistence . . . 244
Coexistence in a distributed data environment . . . 244
Coexistence in a data sharing environment . . . 244
Preparing for fallback . . . 247
Frozen objects . . . 247
Other fallback considerations . . . 249
| Installation changes . . . 250
Version 6 panels . . . 250
| SMP/E changes . . . 251
Changes to installation jobs . . . 251
| Changes to sample jobs . . . 252

Appendix A. Changes to commands . . . 253
New commands . . . 253
Changed commands . . . 254

Appendix B. Changes to utilities . . . 261
New utilities . . . 261
Changed utilities . . . 261
Other utility changes . . . 268

Appendix C. Changes to SQL . . . 269
New SQL statements . . . 269
Changed SQL statements . . . 270
New built-in functions . . . 279
Changed built-in functions . . . 282
Other SQL language changes . . . 282

Appendix D. Catalog changes . . . 285
New catalog tables . . . 285
Changed catalog tables . . . 286
New indexes . . . 288
Revised indexes . . . 290

Appendix E. EXPLAIN table changes 293
Format of the Version 6 PLAN_TABLE 293
Descriptions of new and changed columns 294

Appendix F. New and changed IFCIDs 295
New IFCIDs 295
Changed IFCIDs 295

| **Appendix G. Prerequisites of Version 6 of DB2 for OS/390** 301
DB2 for OS/390 Version 6 prerequisites 301
Hardware requirements 301
Program requirements and optional programs 303
Virtual storage requirements 307
Prerequisites of features of DB2 for OS/390 Version 6 308
DB2 Installer requirements 308
Visual Explain requirements 308
DB2 Estimator requirements 309
Net.Data requirements 309
QMF requirements 310
DB2 Performance Monitor requirements 310
Workstation-Based Analysis and Tuning 311

Appendix H. Notices 313
Programming interface information 315
Trademarks 316

Glossary 317

Bibliography 323

Index 329

Chapter 1. Introduction to this book and the DB2® for OS/390® library

DB2 Release Guide is intended to help you plan for Version 6 of the licensed program DB2 for OS/390. This introduction describes more about the book, its intended users, and information about how to use the DB2 library for Version 6.

Who should read this book

This book is intended for all users of DB2 including application programmers, database administrators, and system programmers. It assumes that the user is familiar with Version 5.

How this book is organized

Each chapter in this book contains a grouping of functional enhancements that are new for DB2 Version 6. Chapters 2 through 6 introduce and describe how to use each enhancement:

- "Chapter 2. Capacity improvements" on page 13
- "Chapter 3. Improved performance and availability" on page 19
- "Chapter 4. User productivity" on page 67
- "Chapter 5. Improved network computing" on page 101
- "Chapter 6. Object-relational extensions and active data" on page 123

Information about the optional features that come with DB2 for OS/390 is included in "Chapter 7. Features of DB2 UDB Server for OS/390" on page 227.

Information about migration and fallback is included in "Chapter 8. Planning for migration and fallback" on page 233.

The appendixes contain the following information:

- Appendix A, "Changes to commands" on page 253
- Appendix B, "Changes to utilities" on page 261
- Appendix C, "Changes to SQL" on page 269
- Appendix D, "Catalog changes" on page 285
- Appendix E, "EXPLAIN table changes" on page 293
- Appendix F, "New and changed IFCIDs" on page 295
- Appendix G, "Prerequisites of Version 6 of DB2 for OS/390" on page 301
- Appendix H, "Notices" on page 313

Some of the functions described in this book are available with Version 5 as functional APARs. When this is the case, the APAR number is identified in the description of those functions.

Product terminology and citations

In this book, DB2 Universal Database Server for OS/390 is referred to as "DB2 for OS/390." In cases where the context makes the meaning clear, DB2 for OS/390 is referred to as "DB2." When this book refers to other books in this library, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM DATABASE 2 Universal Database Server for OS/390 SQL Reference*.)

References in this book to "DB2 UDB" relate to the DB2 Universal Database™ product that is available on the AIX®, OS/2®, and Windows NT™ operating systems. When this book refers to books about the DB2 UDB product, the citation includes the complete title and order number.

The following terms are used as indicated:

DB2® Represents either the DB2 licensed program or a particular DB2 subsystem.

C and C language
Represent the C programming language.

CICS®
Represents CICS/ESA® and CICS Transaction Server for OS/390 Release 1.

IMS™ Represents IMS/ESA®.

MVS Represents the MVS element of OS/390.

How to read the syntax diagrams

The following rules apply to the syntax diagrams used in this book:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►— symbol indicates the beginning of a statement.

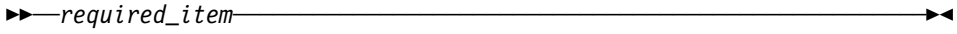
  The —► symbol indicates that the statement syntax is continued on the next line.

  The ►— symbol indicates that a statement is continued from the previous line.

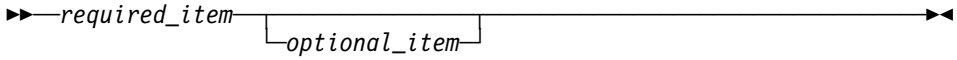
  The —►◄ symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ►— symbol and end with the —► symbol.

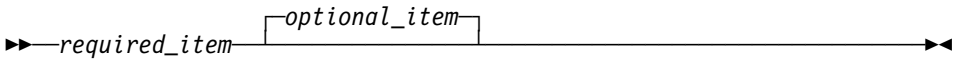
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

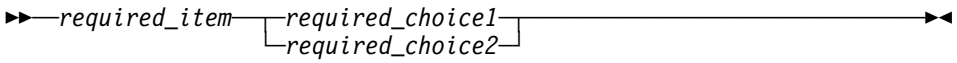


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

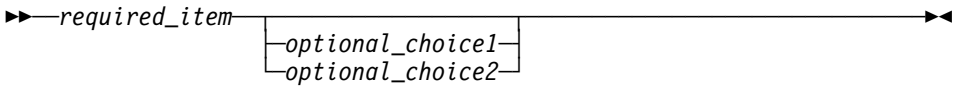


- If you can choose from two or more items, they appear vertically, in a stack.

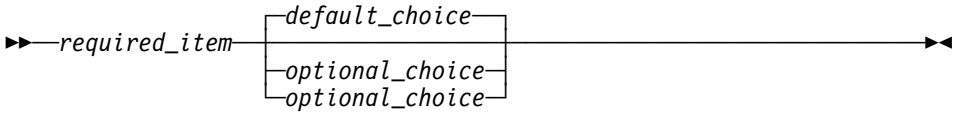
  If you *must* choose one of the items, one item of the stack appears on the main path.



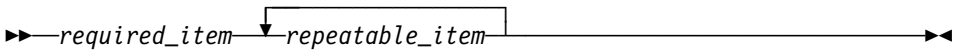
  If choosing one of the items is optional, the entire stack appears below the main path.



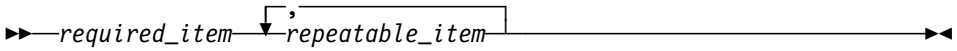
  If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



  If the repeat arrow contains a comma, you must separate repeated items with a comma.



  A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Keywords appear in uppercase (for example, `FROM`). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

How to use the DB2 library

Titles of books in the library begin with DB2 Universal Database for OS/390 Version 6. However, references from one book in the library to another are shortened and do not include the product name, version, and release. Instead, they point directly to the section that holds the information. For a complete list of books in the library, and the sections in each book, see the bibliography at the back of this book.

New Book in Version 6: *DB2 ODBC Guide and Reference* is new in Version 6. It describes how to write applications that use DB2 ODBC to access DB2 servers, and how to write applications that use Open Database Connectivity (ODBC) to access DB2 servers.

Throughout the library, the DB2 for OS/390 licensed program and a particular DB2 for MVS/ESA subsystem are each referred to as "DB2." In each case, the context makes the meaning clear.

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task called *end use*. Other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. The tasks associated with DB2 are grouped into the following major categories (but supplemental information relating to all of the below tasks for new releases of DB2 can be found in this book):

Installation: If you are involved with DB2 only to install the system, *DB2 Installation Guide* might be all you need.

If you will be using data sharing then you also need *DB2 Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

End use: End users issue SQL statements to retrieve data. They can also insert, update, or delete data, with SQL statements. They might need an introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. This information is found in *DB2 Application Programming and SQL Guide* and *DB2 SQL Reference*.

End users can also issue SQL statements through the Query Management Facility (QMF™) or some other program, and the library for that program might provide all the instruction or reference material they need. For a list of the titles in the QMF library, see the bibliography at the end of this book.

Application Programming: Some users access DB2 without knowing it, using programs that contain SQL statements. DB2 application programmers write those programs. Because they write SQL statements, they need *DB2 Application Programming and SQL Guide*, *DB2 SQL Reference*, and *DB2 ODBC Guide and Reference* just as end users do.

Application programmers also need instructions on many other topics:

- How to transfer data between DB2 and a host program—written in COBOL, C, or FORTRAN, for example
- How to prepare to compile a program that embeds SQL statements
- How to process data from two systems simultaneously, say DB2 and IMS or DB2 and CICS®
- How to write distributed applications across platforms
- How to write applications that use DB2 ODBC to access DB2 servers
- How to write applications that use Open Database Connectivity (ODBC) to access DB2 servers
- How to write applications in the Java™ programming language to access DB2 servers

The material needed for writing a host program containing SQL is in *DB2 Application Programming and SQL Guide* and in *DB2 Application Programming Guide and Reference for Java™*. The material needed for writing applications that use DB2 ODBC or ODBC to access DB2 servers is in *DB2 ODBC Guide and Reference*. For handling errors, see *DB2 Messages and Codes* .

Information about writing applications across platforms can be found in *Distributed Relational Database Architecture™: Application Programming Guide*.

System and Database Administration: *Administration* covers almost everything else. *DB2 Administration Guide* divides those tasks among the following sections:

- Section 2 (Volume 1) of *DB2 Administration Guide* discusses the decisions that must be made when designing a database and tells how to bring the design into being by creating DB2 objects, loading data, and adjusting to changes.
- Section 3 (Volume 1) of *DB2 Administration Guide* describes ways of controlling access to the DB2 system and to data within DB2, to audit aspects of DB2 usage, and to answer other security and auditing concerns.
- Section 4 (Volume 1) of *DB2 Administration Guide* describes the steps in normal day-to-day operation and discusses the steps one should take to prepare for recovery in the event of some failure.
- Section 5 (Volume 2) of *DB2 Administration Guide* explains how to monitor the performance of the DB2 system and its parts. It also lists things that can be done to make some parts run faster.

In addition, the appendixes in *DB2 Administration Guide* contain valuable information on DB2 sample tables, National Language Support (NLS), writing exit routines, interpreting DB2 trace output, and character conversion for distributed data.

If you are involved with DB2 only to design the database, or plan operational procedures, you need *DB2 Administration Guide*. If you also want to carry out your own plans by creating DB2 objects, granting privileges, running utility jobs, and so on, then you also need:

- *DB2 SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges
- *DB2 Utility Guide and Reference*, which explains how to run utilities
- *DB2 Command Reference*, which explains how to run commands

If you will be using data sharing, then you need *DB2 Data Sharing: Planning and Administration*, which describes how to plan for and implement data sharing.

Additional information about system and database administration can be found in *DB2 Messages and Codes*, which lists messages and codes issued by DB2, with explanations and suggested responses.

Diagnosis: Diagnosticians detect and describe errors in the DB2 program. They might also recommend or apply a remedy. The documentation for this task is in *DB2 Diagnosis Guide and Reference* and *DB2 Messages and Codes*.

How to obtain DB2 information

DB2 on the Web

Stay current with the latest information about DB2. View the DB2 home page on the World Wide Web. News items keep you informed about the latest enhancements to the product. Product announcements, press releases, fact sheets, and technical articles help you plan your database management strategy.

You can view and search DB2 publications on the Web, or you can download and print many of the most current DB2 books. Follow links to other Web sites with more information about DB2 family and OS/390 solutions. Access DB2 on the Web at the following address:

http://www.ibm.com/software/db2os390

DB2 publications

The DB2 publications for DB2 Universal Database Server for OS/390 are available in both hardcopy and softcopy format.

BookManager® format

Using online books on CD-ROM, you can read, search across books, print portions of the text, and make notes in these BookManager books. With the appropriate BookManager READ product or IBM Library Readers, you can view these books in the OS/390, VM, OS/2, DOS, AIX, and Windows™ environments. You can also view many of the DB2 BookManager books on the Web.

PDF format

Many of the DB2 books are available in Portable Document Format (PDF) for viewing or printing from CD-ROM or the Web. Download the PDF books to your intranet for distribution throughout your enterprise.

CD-ROMs

Books for Version 6 of DB2 Universal Database Server for OS/390 are available on CD-ROMs:

- *DB2 UDB for OS/390 Version 6 Licensed Online Book*, LK3T-3519, containing *DB2 UDB for OS/390 Version 6 Diagnosis Guide and Reference* in BookManager format, for ordering with the product.
- *DB2 UDB Server for OS/390 Version 6 Online and PDF Library*, SK3T-3518, a collection of books for the DB2 server in BookManager and PDF formats.

Periodically, the books will be refreshed on subsequent editions of these CD-ROMs.

The books for Version 6 of DB2 UDB Server for OS/390 are also available on the following collection kits that contain online books for many IBM products:

- *Online Library Omnibus Edition OS/390 Collection*, SK2T-6700, in English
- *IBM Online Library MVS Collection Kit*, SK88-8002, in Japanese, for viewing on DOS and Windows operating systems.

DB2 education

IBM Education and Training offers a wide variety of classroom courses to help you quickly and efficiently gain DB2 expertise. Classes are scheduled in cities all over the world. You can find class information, by country, at the IBM Learning Services Web site:

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

For more information, including the current local schedule, please contact your IBM representative.

Classes can also be taught at your location, at a time that suits your needs. Courses can even be customized to meet your exact requirements. The *All-in-One Education and Training Catalog* describes the DB2 curriculum in the United States. You can inquire about or enroll in these courses by calling 1-800-IBM-TEACH (1-800-426-8322).

How to order the DB2 library

You can order DB2 publications and CD-ROMs through your IBM representative or the IBM branch office serving your locality. If you are located within the United States or Canada, you can place your order by calling one of the toll-free numbers :

- In the U.S., call 1-800-879-2755.
- In Canada, call 1-800-565-1234.

To order additional copies of licensed publications, specify the SOFTWARE option. To order additional publications or CD-ROMs, specify the PUBLICATIONS and SLSS option. Be prepared to give your customer number, the product number, and the feature code(s) or order numbers you want.

Summary of changes to DB2 UDB for OS/390 Version 6

DB2 UDB for OS/390 Version 6 delivers an enhanced relational database server solution for OS/390. This release focuses on greater capacity, performance improvements for utilities and queries, easier database management, more powerful network computing, and DB2 family compatibility with rich new object-oriented capability, triggers, and more built-in functions.

Capacity improvements

16-terabyte tables provide a significant increase to table capacity for partitioned and LOB table spaces and indexes, and for nonpartitioning indexes.

Buffer pools in data spaces provide virtual storage constraint relief for the ssnmDBM1 address space, and data spaces increase the maximum amount of virtual buffer pool space allowed.

Performance and availability

Improved partition rebalancing lets you redistribute partitioned data with minimal impact to data availability. One REORG of a range of partitions both reorganizes and rebalances the partitions.

You can **change checkpoint frequency dynamically** using the new SET LOG command and initiate checkpoints any time while your subsystem remains available.

Utilities that are faster, more parallel, easier to use:

- **Faster backup and recovery** enables COPY and RECOVER to process a list of objects in parallel, and recover indexes and table spaces at the same time from image copies and the log.
- **Parallel index build** reduces the elapsed time of LOAD and REORG jobs of table spaces, or partitions of table spaces, that have more than one index; the elapsed time of REBUILD INDEX jobs is also reduced.
- Tests show **decreased elapsed and processor time for online REORG**.
- **Inline statistics** embeds statistics collection into utility jobs, making table spaces available sooner.
- You can **determine when to run REORG** by specifying threshold limits for relevant statistics from the DB2 catalog.

Query performance enhancements include:

- **Query parallelism extensions** for complex queries, such as outer joins and queries that use nonpartitioned tables
- **Improved workload balancing in a Parallel Sysplex®** that reduces elapsed time for a single query that is split across active DB2 members
- **Improved data transfer** that lets you request multiple DRDA query blocks when performing high-volume operations
- The ability to use an **index to access predicates with noncorrelated IN subqueries**
- **Faster query processing** of queries that include join operations

More performance and availability enhancements include:

- **Faster restart and recovery** with the ability to postpone backout work during restart, and a faster log apply process
- **Increased flexibility with 8-KB and 16-KB page sizes** for balancing different workload requirements more efficiently, and for controlling traffic to the coupling facility for some workloads
- **Direct-row access** using the new ROWID data type to re-access a row directly without using the index or scanning the table
- **Ability to retain prior access path** when you rebind a statement. You almost always get the same or a better access path. For the exceptional cases, Version 6 of DB2 for OS/390 lets you retain the access path from a prior BIND by using rows in an Explain table as input to optimization.
- An **increased log output buffer size** (from 1000 4-KB to 100000 4-KB buffers) that improves log read and write performance

Data sharing enhancements

More caching options use the coupling facility to improve performance in a data sharing environment for some applications by writing changed pages directly to DASD.

Control of space map copy maintenance with a new option avoids tracking of page changes, thereby optimizing performance of data sharing applications.

User productivity

Predictive governing capabilities enhance the resource limit facility to help evaluate resource consumption for queries that run against large volumes of data.

Statement cost estimation of processing resource that is needed for an SQL statement helps you to determine error and warning thresholds for governing, and to decide which statements need tuning.

A **default buffer pool** for user data and indexes isolates user data from the DB2 catalog and directory, and separating user data from system data helps you make better tuning decisions.

More information available for monitoring DB2 includes data set I/O activity in traces, both for batch reporting and online monitors.

Better integration of DB2 and Workload Manager delay reporting enables DB2 to notify Workload Manager about the current state of a work request.

More tables are allowed in SQL statements SELECT, UPDATE, INSERT, and DELETE, and in views. DB2 increases the limit from 15 to 225 tables. The number of tables and views in a subselect is not changed.

Improved DB2 UDB family compatibility includes SQL extensions, such as:

- A VALUES clause of INSERT that supports any expression
- A new VALUES INTO statement

Easier recovery management lets you achieve a single point of recovery and recover data at a remote site more easily.

Enhanced database commands extend support for pattern-matching characters (*) and let you filter display output.

You can easily **process dynamic SQL in batch mode** with the new object form of DSNTEP2 shipped with DB2 for OS/390.

Network computing

SQLJ, the newest Java implementation for the OS/390 environment, supports SQL embedded in the Java programming language. With SQLJ, your Java programs benefit from the superior performance, manageability, and authorization available to static SQL, and they are easy to write.

DRDA® support for three-part names offers more functionality to applications using three-part names for remote access and improves the performance of client/server applications.

Stored procedure enhancements include the ability to create and modify stored procedure definitions, make nested calls for stored procedures and user-defined functions, and imbed CALL statements in application programs or dynamically invoke CALL statements from IBM's ODBC and CLI drivers.

DB2 ODBC extensions include new and modified APIs and new data types to support the object-relational extensions.

ODBC access to DB2 for OS/390 catalog data improves the performance of your ODBC catalog queries by redirecting them to shadow copies of DB2 catalog tables.

Better performance for ODBC applications reduces the number of network messages that are exchanged when an application executes dynamic SQL.

Improvements for dynamically prepared SQL statements include a new special register that you use to implicitly qualify names of distinct types, user-defined functions, and stored procedures.

DDF connection pooling uses a new type of inactive thread that improves performance for large volumes of inbound DDF connections.

Object-relational extensions and active data

The object extensions of DB2 offer the benefits of object-oriented technology while increasing the strength of your relational database with an enriched set of data types and functions. Complementing these extensions is a powerful mechanism, triggers, that brings application logic into the database that governs the following new structures:

- **Large objects (LOBs)** are well suited to represent large, complex structures in DB2 tables. Now you can make effective use of multimedia by storing objects such as complex documents, videos, images, and voice. Some key elements of LOB support include:
  - LOB data types for storing byte strings up to 2 GB in size
  - LOB locators for easily manipulating LOB values in manageable pieces
  - Auxiliary tables (that reside in LOB table spaces) for storing LOB values
- **Distinct types** (which are sometimes called user-defined data types), like built-in data types, describe the data that is stored in columns of tables where the instances (or objects) of these data types are stored. They ensure that only those functions and operators that are explicitly defined on a distinct type can be applied to its instances.
- **User-defined functions**, like built-in functions or operators, support manipulation of distinct type instances (and built-in data types) in SQL queries.
- **New and extended built-in functions** improve the power of the SQL language with about 100 new built-in functions, extensions to existing functions, and sample user-defined functions.

Triggers automatically execute a set of SQL statements whenever a specified event occurs. These statements validate and edit database changes, read and modify the database, and invoke functions that perform operations inside and outside the database.

You can use the **DB2 Extenders** feature of DB2 for OS/390 to store and manipulate image, audio, video, and text objects. The extenders automatically capture and maintain object information and provide a rich body of APIs.

More function

Some function and capability is available to both Version 6 and Version 5 users. Learn how to obtain these functions now, prior to migrating to Version 6, by visiting the following Web site:

http://www.software.ibm.com/data/db2/os390/v5apar.html

Features of DB2 for OS/390

DB2 for OS/390 Version 6 offers a number of tools, which are optional features of the server, that are shipped to you automatically when you order DB2 Universal Database for OS/390:

- DB2 Management Tools Package, which includes the following elements:
  - DB2 UDB Control Center
  - DB2 Stored Procedures Builder
  - DB2 Installer
  - DB2 Visual Explain
  - DB2 Estimator
- Net.Data® for OS/390

You can install and use these features in a “Try and Buy” program for up to 90 days without paying license charges:

- Query Management Facility
- DB2 DataPropagator™
- DB2 Performance Monitor
- DB2 Buffer Pool Tool
- DB2 Administration Tool

Migration considerations

Migration to Version 6 eliminates all type 1 indexes, shared read-only data, data set passwords, use of host variables without the colon, and RECOVER INDEX usage. You can migrate to Version 6 only from a Version 5 subsystem.

Chapter 2. Capacity improvements

DB2 continues to enhance its ability to serve your needs for very large databases while recognizing that processor memory capacity is limited. The following enhancements are described in this chapter:

- "16-terabyte tables"
- "Many more open data sets" on page 15
- "Larger secondary quantity value" on page 15
- "More pieces for nonpartitioning indexes" on page 15
- "Increased total number of extents" on page 16
- "Buffer pools in data spaces" on page 16
- "Dynamic statement cache in a data space" on page 17

16-terabyte tables

Previously: The maximum amount of data for a single table space was 1 terabyte.

Now: In Version 6, one table space can hold 16 terabytes of data, either compressed or uncompressed. The maximum number of partitions remains at 254, but each partition can be up to 64 GB. This feature is available for partitioned table spaces, partitioning indexes, nonpartitioning indexes, and LOB table spaces.

Requirements: The following requirements for this enhancement are:

- DFSMS Version 1.5 (available in OS/390 Release 7, or subsequent releases)

  DFSMS Version 1.5 supports *extended addressability* for the VSAM linear data sets. This lets individual data sets grow to up to 64 GB in size.

- The data sets must be managed by SMS

  For DB2-managed data sets, use '*' in the VOLUMES clause of the CREATE STOGROUP statement to indicate that you want all data sets in this DB2 storage group to be managed by SMS.

  For both user-managed and DB2–managed data sets, then, you use SMS automatic class selection routines to assign data sets to a data class. See "Creating EA-enabled table spaces and index spaces" on page 14 for more information.

Terminology changes

Because DFSMS's extended addressability function is necessary to create data sets larger than 4 GB, the term for page sets that are enabled for extended addressability is *EA enabled*. An EA-enabled table space or index space is one in which you specify that the individual partitions (or pieces, for LOB table spaces) can be greater than 4 GB. You do this with the new DSSIZE option of CREATE TABLESPACE. DSSIZE stands for "data set size."

For example, the following specification creates an EA-enabled table space:

```
DSSIZE 8G NUMPARTS 8
```

Another terminology change for Version 6 is with regard to the term LARGE. In Version 5, LARGE referred to table spaces that were created with the LARGE option. These table spaces have 5-byte RIDs. In Version 6, any table space that is

created with the new DSSIZE option has 5-byte RIDs, no matter how large the table space becomes. In cases when this book discusses the RID size of a table space, the context clarifies how that table space was created (either with the LARGE or DSSIZE option).

Recommendation: To prepare for future enhancements, use the DSSIZE option instead of LARGE.

Page set types and new storage limits

Table 1 summarizes the data set size limits for the different types of page sets.

Table 1. Data set or partition storage limits. Changes for Version 6 are indicated with italics

Page set type	Data set units	Not EA-enabled	EA-enabled
Partitioned page set	Partitions	254 of 4 GB	***254 of 64 GB***
Nonpartitioning index	Pieces	***254 of 4 GB***	***254 of 64 GB***
LOB table space	Pieces	***254 of 4 GB***	***254 of 64 GB***

Creating EA-enabled table spaces and index spaces

To use EA-enabled page sets, you must:

1. Use SMS to manage the data sets associated with the EA-enabled page sets.
2. Associate the data sets with a *data class* (an SMS construct) that specifies the extended format and extended addressability options.

   To make this association between data sets and the data class, use an automatic class selection (ACS) routine to assign the DB2 data sets to the relevant SMS data class. The ACS routine does the assignment based on the data set name. No performance penalty occurs for having non-EA-enabled DB2 page sets assigned to this data class, too, if you would rather not have two separate data classes for DB2.

   For user-managed data sets, you can use ACS routines or specify the appropriate data class on the DEFINE CLUSTER command when you create the data set.
3. Create the partitioned or LOB table space with a DSSIZE of 8 GB or greater. (The partitioning index for the partitioned table space takes on the EA-enabled attribute from its associated table space.) See *DB2 SQL Reference* for more information about the correct syntax.

   After a page set is created, you cannot use the ALTER TABLESPACE statement to change the DSSIZE. You must drop and re-create the table space.

   Also, you cannot change the data sets of the page set to turn off the extended addressability or extended format attributes. If someone modifies the data class to turn off the extended addressability or extended format attributes, DB2 issues an error message the next time it opens the page set.

Changes to utilities

The following utilities have a new DSSIZE option:

- DSN1COMP
- DSN1COPY
- DSN1PRNT

REPAIR is also enhanced to be more usable and to handle the new data sets.

See Appendix B, “Changes to utilities” on page 261 for more information on the syntax of these utilities.

Catalog changes

To accommodate the very large table sizes, several new floating-point statistics columns are added to the following catalog tables:

- SYSTABLESPACE
- SYSTABLEPART
- SYSTABSTATS
- SYSINDEXTATS

See Appendix D, “Catalog changes” on page 285 for more information about those changes.

Many more open data sets

Previously: The maximum number of DB2 open data sets was 10000. For some installations, this was a limiting factor.

Now: With OS/390 Version 2 Release 6 and with the enhancement included in Version 5 in APAR PQ18543, you can set your DB2 maximum open data set limit (DSMAX subsystem parameter) to a value up to 32767. (The practical limit might be less than 32727, depending on available below-the-line storage.)

Larger secondary quantity value

The maximum value for secondary quantity (SECQTYI) is raised from 131068 KB to 4 GB (4194304 KB).

More pieces for nonpartitioning indexes

Previously: The number of pieces for a nonpartitioning index was limited to 128.

Now: In Version 6, you can have up to 254 pieces, enhancing your ability to reduce I/O contention on the nonpartitioning index. If the nonpartitioning index is defined on an EA-enabled table space, each of the pieces can be up to 64 GB.

Increased total number of extents

With DFSMS Version 1.4, the total number of extents for data sets is increased from 123 to 255. However, the number of extents for a data set in each volume is still limited to 123.

Buffer pools in data spaces

An option to consider for some of your buffer pools is to have DB2 put them in *data spaces*. Like hiperspaces, data spaces are data-only spaces; that is, no program code can run in those areas. With data spaces, though, the system uses the same resources to back data space virtual storage as it uses to back address space virtual storage: a combination of central storage and expanded storage frames (if available), and auxiliary storage slots. The system can move low-use pages of data space storage to auxiliary storage and bring them in again. The paging activity for a data space includes I/O between auxiliary-storage paging devices and central storage.

Figure 1 shows DB2 using a data space for a virtual buffer pool.

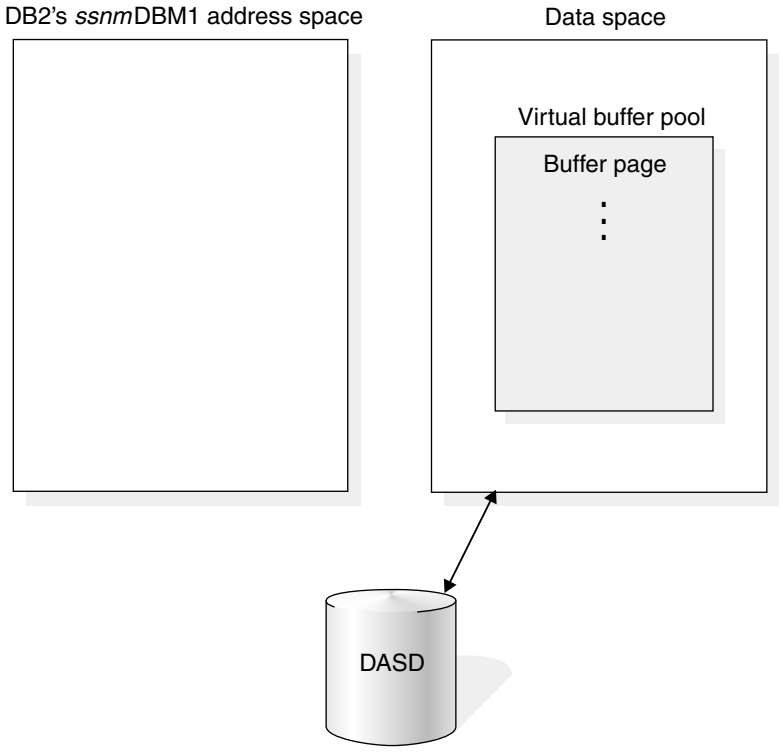


Figure 1. Using a data space for DB2 virtual buffer pools

The main differences between buffer pools in data spaces and in hiperspace are:

- DB2 can put changed pages in data spaces. (Pages in hiperpools must be unchanged.)
- DB2 can do I/O in and out of a data space but not a hiperpool.
- Less *ssnm*DBM1 storage is used for a data space virtual pool than is used for a primary space virtual pool with its associated hiperpool.

- More potential storage is available. The limits for data spaces are much higher than that for hiperpools. In addition, when OS/390 and processors are available that have support for more than 2 GB of real memory per OS/390 image, buffer pools in data spaces will let DB2 use real memory more efficiently.

Each data space can accommodate almost 2 GB of buffers and any single buffer pool can span multiple data spaces. (However, no more than one buffer pool can be in a data space.) The sum of all data space buffers cannot exceed 8 million. This translates to the maximum sizes that are described in Table 2:

Table 2. Maximum amount of storage available for data space buffers

If all buffers are this size...	The total amount of data space storage is...
4 KB	32 GB
8 KB	64 GB
16 KB	128 GB
32 KB	256 GB

Total storage in the ssnmDBM1 address space

Each buffer in a data space requires about 128 bytes of storage in DB2's *ssnm*DBM1 address space. DB2 does not allow more than 1.6 GB of storage in *ssnm*DBM1 address space for virtual pool buffers and data space buffer control storage. Message DSNB508I is issued if the amount of space exceeds 1.6 GB.

Advantages of data spaces

Until processors are available that contain more than 2 GB of real memory, the main reason to choose data spaces is to relieve storage constraints in DB2's *ssnm*DBM1 address space and to provide greater opportunities for caching very large table spaces or indexes. If you are currently using hiperpools for read-intensive workloads and have not reached any DB2 virtual storage limits, there is no immediate benefit to moving to data spaces until large real memory becomes available.

Dynamic statement cache in a data space

If you are constrained on storage in *ssnm*DBM1, and your installation uses dynamic statement caching, move some EDM storage into a data space by specifying a non-zero value for EDMPOOL DATA SPACE SIZE on installation panel DSNTIPC.

Chapter 3. Improved performance and availability

This chapter describes the changes that improve performance and availability in Version 6. Those changes are described under the following categories:

- "Rebalancing data in a partitioned table space"
- "Altering variable length columns" on page 23
- "Dynamically change checkpoint frequency" on page 24
- "Faster restart and recovery" on page 25
- "Faster, more parallel utilities" on page 29
- "Query performance and optimization enhancements" on page 35
- "Data sharing enhancements" on page 43
- "More performance and availability improvements" on page 53

Rebalancing data in a partitioned table space

Previously: Any time you wanted to change the partitioning index to shift data from one partition to another required that you go through the cumbersome process of unloading the entire table space, dropping the table space, re-creating the table space with new index definitions, views, aliases, and authorizations, and then reloading the table space.

Not only is this process cumbersome, but it causes the data to be unavailable for that entire time, and during the time you spend rebinding plans and packages and regranting authorizations.

Now: In DB2 Version 6, you can rebalance data in a partitioned table space with higher availability and with greater ease.

In this section:

- "Why rebalance data?"
- "The process: ALTER INDEX then REORG" on page 20
- "REORG-pending status" on page 21
- "Recovery guidelines" on page 22

Why rebalance data?

When data in a partitioned table space becomes skewed, performance can be negatively affected because of contention for I/O and other resources. Or, maybe you have a situation where some partitions are approaching their maximum size while other partitions have excess space. Now, with a simple ALTER INDEX statement and a REORG job, you can shift data among the partitions, reorganize the affected partitions, and the data is balanced as you specify.

Strategic planning: Work this enhancement into your strategy for table spaces that you want to grow over time. Define dummy partitions that have key values that fall out of the range of any data for which you populate the partition, such as a year in the far future. When needed, you can activate these dummy partitions by altering the out-of-range partitioning key values to a value that causes DB2 to populate the partitions.

Roll changes through: Another possible scenario is to change the limit key values of all or most of the partitions. The old way to do this was to drop and re-create the table space. In Version 6, you can roll the changes through the partitions one or more at a time, making relatively small parts of the data unavailable at any given time.

The process: ALTER INDEX then REORG

Assume data is skewed as that shown in Figure 2.

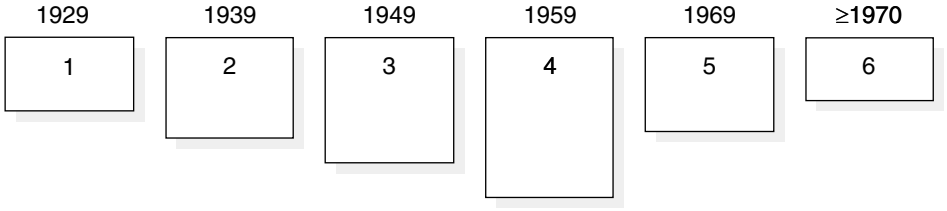


Figure 2. Skewed data, partition 4 is too large

To rebalance data:

1. Issue the ALTER INDEX statement, specifying the changes to the limit key such as:

   ```
   ALTER INDEX birthx PART 1 VALUES ('1935'),
                      PART 2 VALUES ('1943'),
                      PART 3 VALUES ('1953');
   ```

   An additional enhancement to ALTER INDEX is to let you make changes to more than one partition at a time, so you can change the limit key values of some partitions and change other attributes of other partitions. See *DB2 SQL Reference* for more information about the syntax.

   DB2 puts the table space partitions you specified into a new status called *REORG pending*. It also places the next higher partition in REORG pending. Thus, after the above ALTER, partitions 1 through 4 are in REORG pending as shown in Figure 3.

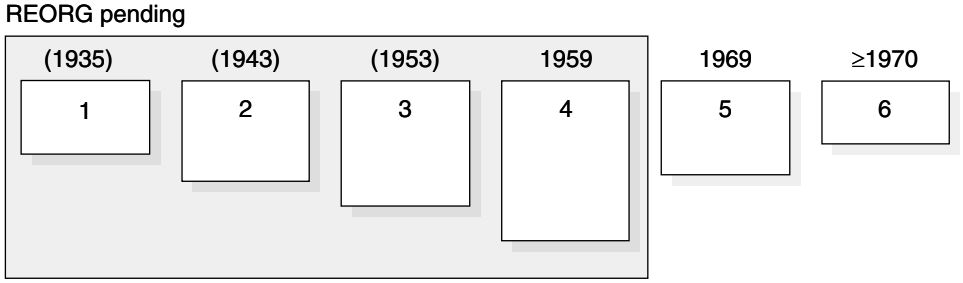


Figure 3. Data partitions in REORG pending status

See “REORG-pending status” on page 21 for more information about the REORG pending status.

DB2 also invalidates the plans and packages that reference the table space. The first application access immediately following the ALTER INDEX statement causes those plans and packages to be rebound. (However, if you have not yet run REORG to remove the REORG pending status, any access to partitions that are in REORG pending status receive a -904 SQLCODE.)

2. Reorganize the entire table space, or just the partitions that are affected by the ALTER. Here is an example of specifying a range of partitions in REORG. Notice also that the STATISTICS option is specified, to gather statistics on those partitions. See *DB2 Utility Guide and Reference* for more information about the REORG utility syntax.

```
REORG TABLESPACE PROD.EMP
  PART 1:4
  STATISTICS
  SORTDEVT SYSDA
   SHRLEVEL NONE
   COPYDDN SYSCOPY
   SORTDATA
```

   A note about the last partition: When a table space is defined with the LARGE or DSSIZE option, the high partitioning key is enforced. Therefore, if an ALTER INDEX causes the last partition to be placed in REORG-pending status, you must specify a discard data set and a punch data set for the REORG job. The discard data set is used in case the ALTER causes existing values to be invalid. The punch data set has generated LOAD statements that let you reload the discarded data, if you want.

   Catalog record: When REORG is complete, the REORG-pending status is turned off, and DB2 writes a SYSCOPY record of a new subtype, A, to the SYSIBM.SYSCOPY catalog table. This record includes the lowest (LOWDSNUM) and highest (HIGHDSNUM) partition in the range that was specified for the REORG job.

3. After the reorganization is complete, data is balanced, as shown in Figure 4.

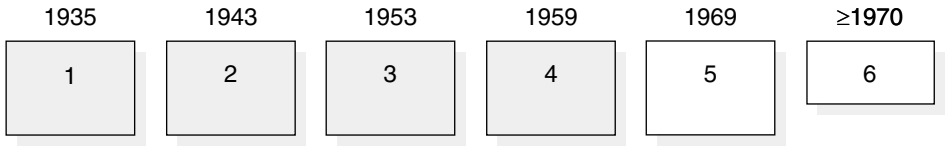


Figure 4. Data is balanced again

4. Rebind plans and packages that are affected by the changed statistics, or which were invalidated by the ALTER INDEX.

REORG-pending status

REORG-pending status (REORP) is a restrictive status that is placed on the data partitions (not the index) that are affected by the ALTER INDEX statement (that is, the specified partition and the next higher partition). When data partitions are in REORG-pending status, you cannot SELECT, INSERT, DELETE, or UPDATE data. This restriction applies to any type of access to the data in those partitions, including access through a partitioning or nonpartitioning index.

You can, however, drop the entire table space while any of its partitions are in REORG-pending status.

The DISPLAY DATABASE command displays REORP status, as shown in *DB2 Command Reference*.

REORG-pending and the last partition

If you change the partitioning key for the last partition, and if the table space is not defined with LARGE or DSSIZE, DB2 does not place that last partition in REORG-pending status.

If the table space is defined with LARGE or DSSIZE, DB2 puts the last partition in REORG pending only if you lower the limit key of the last partition for an ascending index), or if you raise the limit key for a descending index.

Removing REORG-pending status

To remove REORG pending status:

- Run REORG PART *m*:*n* with SHRLEVEL(NONE).
- Run REORG TABLESPACE SHRLEVEL(NONE).
- Run LOAD REPLACE on the table space.
- Drop the table space.

The START DATABASE command with ACCESS(FORCE) does *not* remove the REORG-pending status.

Recovery guidelines

The point at which you reset REORG-pending status is an important point in time for recovery processing:

- Image copies that were taken before this point are not usable for recovery to the current point in time.

  Recommendation: Specify a COPYDDN when you run REORG after altering the partitioning keys. This provides you with an image copy from which you can recover.

- Log records cannot be applied across this point.
- Point-in-time recoveries prior to this point cause REORG pending to be turned back on. By turning REORG-pending status on, you are forced to run REORG to rebalance the partitions and rebuild the indexes.

Point-in-time recovery scenarios

This section describes some scenarios for point-in-time recovery.

Case 1: You try to do a point-in-time recovery of a single partition after you ran a REORG to turn off the REORG-pending status.

In this case, DB2 puts all partitions that were specified in that REORG into REORG-pending status (even those that were not originally in REORG-pending status). For example:

```
Time 1:  You specify ALTER INDEX PART 3 VALUES...
Time 2:  DB2 puts parts 3 and 4 in REORG-pending status.
Time 3:  You run REORG TABLESPACE PART (2:6) SHRLEVEL NONE.
Time 4:  DB2 writes a SYSCOPY STYPE A record for partitions 2 through 6.
Time 5:  You try to recover partition 5 to any time before time 4.
Time 6:  DB2 issues message DSNU556I and return code 8 telling you
             that you must run RECOVER on partitions 1 through 6
             in the same RECOVER job.
Time 7:  You recover partitions 1 through 6 to before time 4.
         DB2 puts partitions 1 through 6 in REORG-pending status.
```

REPORT RECOVERY tells you which image copies will turn REORG pending status back on.

Case 2: You try to recover a partition while it is in REORG-pending status to any time before now.

For example:

```
Time 1:  You specify ALTER INDEX PART 3 VALUES...
Time 2:  DB2 puts parts 3 and 4 in REORG-pending status.
Time 3:  You recover partition 3 to any time before now.
         DB2 puts partition 3 in REORG pending. (Partition 4
         is still in REORG-pending status, too.)
```

Case 3: You use RECOVER LOGONLY to a previous point in time.

You can use RECOVER LOGONLY after data has been redistributed among partitions using REORG. If you are doing a point-in-time recovery, you must keep the offline copies in sync with the SYSIBM.SYSCOPY records; in other words, do not delete any SYSCOPY STYPE A records that might be needed during the recovery. If you delete these records, DB2 does not know that it is to set REORG-pending state on the relevant partitions.

Altering variable length columns

Now, you can increase the length of variable length character columns easily. The ability to alter variable length columns is particularly useful during migration to a new release when the need to store more data requires defining larger variable length character columns on existing tables.

Previously, altering variable length columns required several steps:

- Creating a new table with the larger columns
- Moving data from the existing table to the new table
- Dropping the existing table
- Renaming the new table
- Recreating authorization and dependent objects

This enhancement eliminates these steps. Now you can change the definition of an existing VARCHAR column using the new ALTER COLUMN clause of the ALTER TABLE statement. See *DB2 SQL Reference* for more information.

A VARCHAR column cannot be altered if it is used in a referential integrity relationship, a view, a temporary table, or a table that is defined with DATA

CAPTURE CHANGES. It also cannot be altered if it belongs to a table defined with a user exit (edit procedure, validation routine, field procedure, stored procedure, or user-defined function).

Changing the limit key value

Changing the length of a column of a partitioning index can affect partition boundaries and the limit key of a partition. Now, the length of the limit key can change. If the length of a column that is part of the limit key is altered, and the changed column length affects the partition boundary, the limit key can change. Changing the length of the limit key ensures that the partition boundary does not change.

Changing the length of a column that is not part of the LIMITKEY value does not affect the partition boundary. In this case, the length of the limit key does not require a change.

Immediate index access

When the length of a column of an index key is increased, the index on the altered table remains available, giving you immediate index access.

The maximum number of distinct alters that increase the index key column is less than or equal to sixteen. If the maximum number of alters is exceeded, an error is returned (SQLCODE -148) and you must run a REORG INDEX, REORG TABLESPACE, or REBUILD INDEX. An alter is considered distinct if it is in a different commit scope than the previous alter. For example, two distinct alters are performed if you alter, commit, and then alter again. One distinct alter is performed if you alter, alter again, and then commit. In this case, the second alter replaces the first alter because it occurred in the same commit scope.

If the alter fails because the maximum number of alters has been exceeded and the index is partitioned, then the utilities must be run on all the partitions before you can run a new alter successfully.

Dynamically change checkpoint frequency

Previously: To change the system checkpoint frequency in DB2 required that you change the LOGLOAD subsystem parameter and then start and stop DB2. This limited your ability to have different checkpoint frequencies for different times of the day. Also, there was no easy way to cause a checkpoint to happen when you wanted it to happen.

Now: A new SET LOG command lets you change the LOGLOAD subsystem parameter dynamically. You can initiate a checkpoint by specifying 0 for the LOGLOAD value.

Why change the checkpoint frequency?

DB2 bases its system checkpoint frequency on how many log records are written. At installation time, you tell DB2 that you want it to checkpoint every 'n' number of log records that are written. Restart time is directly affected by how many log records are written after the latest system checkpoint. The more log records, the longer the restart time.

During prime shift, you might have a lower logging rate but fast restart is critical. Therefore, you might choose a lower LOGLOAD value (which means more frequent checkpoints) during prime shift. If you run batch jobs during the off-shift, the logging rate might increase significantly. If you keep the same LOGLOAD value, your system checkpoints more frequently. This frequent checkpointing might not be necessary for these jobs because restart time is not as critical during the off-shift. By altering the LOGLOAD value to a higher value, checkpoints happen less frequently.

Preparing for disaster recovery: Another useful purpose for the SET LOG command is to initiate a system checkpoint. You might want to do this to create recovery logs that have a checkpoint taken at the time the system is quiesced. A typical scenario is:

1. Quiesce all DB2 activity.
2. Issue SET LOG with a LOGLOAD value of 0 to cause a system checkpoint to occur.
3. Issue ARCHIVE LOG and then ship the logs to the disaster recovery site.

Using the SET LOG command

To change the LOGLOAD subsystem parameter, enter a SET LOG command. For example:

```
-DB1G SET LOG LOGLOAD(150000)
```

When you stop and restart DB2, the *ssnm*PARM value for LOGLOAD is used.

To initiate a system checkpoint without changing your normal LOGLOAD value, enter the following command:

```
-DB1G SET LOG LOGLOAD(0)
```

For more information about using the SET LOG command, see *DB2 Command Reference*.

Displaying the current LOGLOAD value

A new command, DISPLAY LOG, lets you display the current value for LOGLOAD and information about offload service tasks. For more information about DISPLAY LOG, see *DB2 Command Reference*.

Faster restart and recovery

DB2 Version 6 makes more improvements in restart and recovery to make data available quicker:

- “Postponing backout processing at restart” on page 26
- “Fast log apply” on page 29
- “Faster log read” on page 29

Postponing backout processing at restart

Previously: Restart times were sometimes inconsistent or very long, especially when a restart entailed backing out a long-running unit of recovery, such as with an errant batch job that does not issue interim commits. The DB2 subsystem was not able to process new work until all restart work was completed, including the sometimes lengthy backout process.

In a data sharing environment with spare capacity, you could reroute work to another member of the group. However, if there was not enough capacity on the other system, or if you were not able to switch workloads because your configuration is such that there is a strong one-to-one relationship between a DB2 member and the workload that runs on that member, the same problem existed as for a non-data-sharing system.

Now: You can tell DB2 to postpone some of the backout work traditionally performed during system restart. By postponing long-running backout work, new work can start more quickly.

In this section:

- "Enabling postponed backout processing"
- "How much processing is done at restart?"
- "Restart-pending and advisory restart-pending statuses" on page 27
- "Resolving postponed units of recovery" on page 27

Enabling postponed backout processing

Specify YES or AUTO on the LIMIT BACKOUT field of installation panel DSNTIPN. By specifying YES or AUTO, DB2 limits backout processing as determined, in part, by the value you specify for another field, BACKOUT DURATION.

The difference between YES and AUTO is that, with YES, you must use the new RECOVER POSTPONED command to perform the rest of the backout work.

Recommendation: Choose AUTO. This option makes data available more quickly.

How much processing is done at restart?

The amount of backout work that is performed as part of restart when LIMIT BACKOUT=YES or LIMIT BACKOUT=AUTO is determined by:

- The frequency of checkpoints.
- The value you specify for BACKOUT DURATION. This value is your indication how many log records are to be read during restart's backward log scan. The BACKOUT DURATION field is a multiplier of the value you specify for the number of log records per checkpoint value (the CHECKPOINT FREQ field on DSNTIPN).

  Use this combination of values to ensure that short-duration URs are handled during restart rather than being postponed.
- The characteristics of the inflight and inabort activity happening when the system failed. In particular, backward processing of the log proceeds until both of the following events occur:
  - The oldest inflight or inabort UR with activity against the catalog or directory is backed out.

– The requested number of log records are processed (as specified by BACKOUT DURATION * CHECKPOINT FREQ).

Restart-pending and advisory restart-pending statuses

Between the completion of restart and automatic backout processing (or a RECOVER POSTPONED command), you can submit new work, and DB2 can process it. However, the objects for which backout work exists are not available to applications.

Non-data-sharing: The particular table spaces (or table space partitions) and index spaces (or physical index space partitions) with backout work pending are put in a new restrictive *restart-pending* (RESTP) status at the end of DB2 restart. An object in RESTP status is unavailable to all but RECOVER POSTPONED and RECOVER INDOUBT processing, or to the automatic backout processing that DB2 performs. The restart-pending status remains until backout processing is complete (or until DB2 is cold started or conditionally restarted).

Data sharing: In data sharing, no restrictive status is set. Access to data with backout work pending is blocked by transaction locks that persist through restart. The following retained locks persist through restart when postponed backout processing is active:

- Retained transaction locks that are held on page sets or partitions for which backout work has not been completed.
- Retained transaction locks that are held on tables, pages, rows, or LOBs of those table spaces or partitions.

The retained transaction locks on any particular page set or partition are freed when all URs using that page set or partition have completed their backout processing. Until that happens, the page set or partition is placed in *advisory restart-pending* (AREST) status.

Utilities are not restricted by the AREST status, but any write claims that are held by postponed-abort URs on the objects in AREST status prevent draining utilities from accessing that page set.

Resolving postponed units of recovery

If you specify LIMIT BACKOUT=AUTO, DB2 resolves all postponed-abort URs.

If you specify LIMIT BACKOUT = YES, you must use the RECOVER POSTPONED command to resolve postponed units of recovery. The RECOVER POSTPONED command completes postponed backout processing for *all* units of recovery; you cannot specify a single unit of work for resolution. The format of the command is:

```
- RECOVER POSTPONED
```

Output from the RECOVER POSTPONED command consists of informational messages. Figure 5 on page 28 shows that backout processing was performed against two table space partitions and two index partitions:

```
DSNV435I - RESOLUTION OF POSTPONED ABORT URS HAS BEEN SCHEDULED
DSN9022I - DSNVRP 'RECOVER POSTPONED' NORMAL COMPLETION
DSNI024I - DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .I PART 00000004.
DSNI024I - DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .PT PART 00000004.
DSNI024I - DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .I PART 00000002.
DSNI024I - DSNIARPL BACKOUT PROCESSING HAS COMPLETED
      FOR PAGESET DSNDB04 .PT PART 00000002.
```

Figure 5. Example output from RECOVER POSTPONED command

Identifying that postponed URs exist: You can identify that pending-backout work exists in several ways:

- Watch for DSNR007I or DSNI023I messages. DSNR007I is issued at the end of restart's backward phase and indicates which URs (if any) will persist through restart with backout work pending. DSNI023I is issued at the end of restart and identifies the page sets and partitions that have backout work pending on behalf of those URs.
- Issue a DISPLAY THREAD command with TYPE (POSTPONED):

  ```
  DSNV401I ! DISPLAY THREAD REPORT FOLLOWS -
  DSNV431I ! POSTPONED ABORT THREADS -
   COORDINATOR               STATUS      RESET URID     AUTHID
   coordinator-name          ABORT-P       urid         authid
   DISPLAY POSTPONED ABORT REPORT COMPLETE
   DSN9022I ! DSNVDT '-DISPLAY THREAD' NORMAL COMPLETION
  ```

- Issue a DISPLAY DATABASE command with the RESTRICT option. The status of any object with postponed UR backout processing is indicated as RESTP (restart pending) in a non-data-sharing environment. In a data sharing environment, the status is shown as AREST (advisory restart pending).
- Issue a DISPLAY GROUP command. The following statuses indicate that work is pending for that member:

  A I This active member has indoubt URs, URs for which backout work is postponed, or both.

  Q I This quiesced member has indoubt URs, URs for which backout work is postponed, or both.

Cold starts and conditional restarts: A cold start of DB2, or a conditional restart of DB2 with BACKOUT=NO, ends all postponed abort URs. Those URs are not resolved. Page sets are removed from restart pending (or advisory restart-pending) status.

If in a data sharing environment, if same page set or partition is updated by two or more DB2s, and each DB2 postponed the backout processing of that page set or partition, a cold or conditional restart of one DB2 does not remove advisory restart-pending status. AREST status is reset when all URs (from all DB2s) that have interest on that page set or partition are resolved.

If a conditional restart truncates the log after the beginning of a postponed-abort UR, that UR is ended without being resolved, and the restrictive restart-pending status is removed.

Fast log apply

Previously: The log apply process was I/O-bound. As each log record was read, the data page affected was read from a buffer or from DASD and changed as required. Then the next log record was read and the process repeated. Also, because many different log records can apply to the same data page, any page might be read many times, unnecessarily increasing the number of I/O operations.

Now: DB2 speeds up the log-apply phase of recovery and restart by sorting log records so that changes that are to be applied to the same page or same set of pages are together. Then, using several log-apply tasks, DB2 can apply those changes in parallel.

Enabling fast log apply: Provide storage for this process on the LOG APPLY STORAGE field of panel DSNTIPL.

Faster log read

Previously: When reading the active log, DB2 read from only one log read buffer, which resulted in wait time when waiting for the log read buffer to refill.

Now: DB2 uses a dual log-read buffer scheme to reduce (and often eliminate) the wait time when reading sequentially through the active logs. While DB2 is processing the log data in a primary log read buffer, another log read buffer is filled asynchronously. When the log-read requester has exhausted the current read buffer, the next buffer is available without having to wait for the I/O request.

This method of reading the log is used whenever reading sequentially through the active logs in a forward direction. The following processes can benefit from this enhancement:

- The current status rebuild and forward log recovery phase of restart
- RECOVER
- REORG SHRLEVEL CHANGE
- START DATABASE for error recovery
- Log read functions that use IFI READS for IFCID 306 (such as DB2 DataPropagator)

Faster, more parallel utilities

In addition to the availability enhancement described in “Rebalancing data in a partitioned table space” on page 19, DB2 adds many performance enhancements to decrease the elapsed time for the utilities, making data available for your applications sooner. Utility enhancements described in this section are:

- “Faster backup and recovery” on page 30
- “Parallel index build” on page 32
- “Inline statistics” on page 33
- “Faster discard and unload during REORG” on page 33
- “Avoid delete and redefine of data sets” on page 34

Other utility enhancements focus on improved usability and increased functionality. See “Utility usability and functionality enhancements” on page 95.

Important

All performance results are based on tests that were done in a control environment with an early level of code. Results you might see will vary based on your actual system configuration and other factors.

Faster backup and recovery

With Version 6, DB2 is making significant gains in the area of backup and recovery performance, thereby making your data available sooner. This section describes the following enhancements:

- "Backup and recover indexes using image copies"
- "Copying a list of objects" on page 31
- "Copying and recovering in parallel" on page 31

The fast log apply function described in "Fast log apply" on page 29 also improves the elapsed time of recovery processing as does "Parallel index build" on page 32.

Backup and recover indexes using image copies

Previously: Because you could not make image copies of indexes, you could recover indexes only by rebuilding the indexes from existing data. This process could be lengthy, especially if index recovery had to wait until the data was recovered, making those indexes unavailable until the rebuild was complete.

Now: You can take a full image copy or a concurrent copy of an index, just as you have always done for table spaces. To recover those indexes, you use the RECOVER utility, which restores the image copy and applies log records.

Enabling indexes for image copies: To enable an index to be copied using the COPY utility, you must create it (or alter it) with COPY YES (the default is COPY NO). Specifying COPY YES gives you two options: copying and recovering the index, or rebuilding the index, as you did before this enhancement.

Catalog record: The COPY attribute of the index is stored in the COPY field of SYSIBM.SYSINDEXES.

Recording recovery information in SYSIBM.SYSLGRNX and SYSIBM.SYSCOPY: With COPY YES, DB2 updates SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX for the index to record recovery information. If you intend to copy a lot of your indexes, you might want to increase the size of those table spaces.

Pending statuses for index spaces: DB2 can place an index that is defined with COPY YES in a new advisory status called *informational COPY pending* (ICOPY) status whenever the index cannot be recovered from the log. Because this is an advisory status not a restrictive status, DB2 allows full read/write access to the affected index. To reset the ICOPY status, take a full image copy of the affected index space.

To find out if an index is in this ICOPY status, issue the DISPLAY DATABASE command with the ADVISORY option.

Indexes defined with COPY YES might also be placed in check pending status (CHKP) if the index might be out of synchronization with the associated table

space. CHKP status on an index is not reflected in the catalog. To find out if this status exists, issue DISPLAY DATABASE with the RESTRICT option.

Managing index recovery: REPORT is enhanced to let you request recovery information for indexes. You can ask for indexes alone, or you can ask for indexes for tables in the table space you specify on REPORT. See the REPORT utility in *DB2 Utility Guide and Reference* for more information.

The MODIFY utility is enhanced to remove recovery information for indexes defined with COPY YES whenever you remove such information from their associated table spaces.

Downlevel detection for indexes: Indexes defined with COPY YES are subject to DB2's downlevel detection process. REPAIR is enhanced to let you repair the level ID for indexes. See *DB2 Utility Guide and Reference* for information about the REPAIR utility syntax.

Copying a list of objects

Previously: Unless you used the CONCURRENT keyword, you could copy only one table space at a time.

Now: You can copy a list of table spaces *and* indexes (if those indexes are defined with COPY YES) in a single COPY utility. There are two main advantages for copying more than one object at a time:

- If you specify a list of objects and also specify SHRLEVEL REFERENCE, that creates a single recovery point for that list of objects. In other words, you can use that recovery point for any point-in-time recoveries for those objects.
- You can specify the PARALLEL option, which lets DB2 copy a number of objects in parallel. See “Copying and recovering in parallel” for more information.

Copying and recovering in parallel

Previously: Although you could specify a list of table spaces to recover, the objects were restored serially. And you could never copy multiple objects before in a single COPY job.

Now: When you specify a list of table spaces and index spaces, you can also copy and recover some or all of those objects in parallel using the new PARALLEL option of COPY and RECOVER. You can tell DB2 how many objects to process in parallel, or you can let DB2 choose an optimal number for you. This new functionality provides a performance advantage, because the RECOVER utility processes the logs for all of the table spaces and index spaces in a single pass.

Restrictions: You cannot make CONCURRENT copies in parallel, and RECOVER cannot restore CONCURRENT copies in parallel.

Full parallel processing is available only for copies made to disk or for restoring those copies from disk.

Parallel index build

Previously: When DB2 rebuilt indexes, it scanned the table space or table space partition, extracted and sorted the index keys for one or more indexes, and built indexes serially. Building the indexes serially meant longer elapsed times for utilities in which indexes were rebuilt.

Now: DB2 can build indexes in parallel, reducing the elapsed time of utilities that rebuild multiple indexes, such as REBUILD INDEX (formerly known as RECOVER INDEX), LOAD, and REORG TABLESPACE.

Figure 6 shows a REORG TABLESPACE flow with parallel index build. To enable the process of rebuilding indexes in parallel, you must specify the SORTKEYS option. DB2 starts multiple subtasks to sort index keys and build indexes in parallel. If you specified STATISTICS, additional subtasks collect the sorted keys and update the catalog table in parallel, eliminating the need for a second scan of the index by a separate RUNSTATS job.

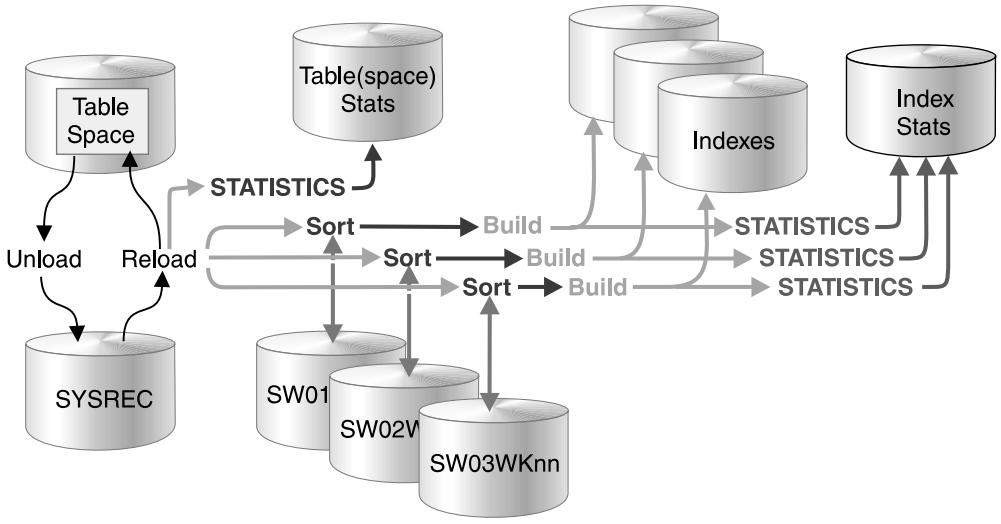


Figure 6. Building indexes using parallel index build

For examples of parallel index build processing for the REBUILD INDEX utility, see *DB2 Utility Guide and Reference*.

Faster index creation and rebuild

Parallel index build sorts index keys, and builds or rebuilds indexes in parallel when you have defined more than one index.

This functional enhancement takes advantage of multiprocessing and multiple processors.

Performance test results for LOAD, REORG, and REBUILD INDEX utilities

Preliminary tests of several utilities demonstrate substantial reductions in elapsed times.

Tests of LOAD and REORG utilities: Two sets of tests measured the Version 6 LOAD and REORG utilities. The first set of tests included the following new capabilities:

- Building indexes in parallel
- Collecting statistics inline

The second set of tests excluded these new capabilities. All tests involved a table space with 10 partitions and 6 indexes. When the LOAD and REORG utilities ran while building indexes in parallel and while collecting statistics inline, the results were impressive:

- Up to a 6 times improvement in elapsed time for LOAD
- Up to a 2.5 times improvement in elapsed time for REORG

Tests of REBUILD utility: A similar comparison of the Version 6 REBUILD utility involved a variety of database configurations that DB2 installations commonly use. Tests included the rebuilding of:

- The partitioning index only
- A single nonpartitioning index only
- The partitioning index and five nonpartitioning indexes

Using the capability to build the indexes in parallel while collecting statistics inline, tests of the REBUILD utility demonstrated excellent reductions in elapsed time:

- Up to 5 times faster for a partitioning index
- Over 3 times faster for a nonpartitioning index
- Almost 8 times faster for six indexes

Inline statistics

Previously: If you wanted the catalog to reflect the new statistics after a LOAD, REBUILD INDEX, or REORG, you had to invoke RUNSTATS separately after those utility jobs were complete.

Now: Now you can invoke RUNSTATS using the STATISTICS keyword from within LOAD, REBUILD INDEX, and REORG. By avoiding a separate invocation of RUNSTATS, the total elapsed time is reduced.

STATISTICS keyword

Specify the STATISTICS keyword on a LOAD, REBUILD INDEX or REORG job to collect inline statistics. A number of RUNSTATS options are now available for inline statistics collection, and are used in conjunction with the STATISTICS keyword.

Benefits of inline statistics

Collecting statistics inline with the LOAD, REORG, and REBUILD INDEX utilities updates catalog and directory tables with a single scan of the data. Accurate statistics can improve access path selection.

For an diagram of inline statistics collection for a REORG TABLESPACE job, see *DB2 Utility Guide and Reference*.

Faster discard and unload during REORG

Previously: You could discard records using an SQL DELETE operation, followed by a REORG. You could unload data using the DSNTIAUL sample application.

Now: With the new options of the REORG utility, you can:

- Select rows to be discarded during a REORG, and optionally write the discarded records to a file

- Perform faster external UNLOAD than the DSNTIAUL sample program provides

Discarding rows during REORG TABLESPACE

You can now efficiently delete a large percentage of rows in a table space, or archive older data in a table space. Specify the DISCARD FROM TABLE ... WHEN option to take advantage of this new function. By specifying the selection criteria, you determine which rows to delete. Rows are only discarded when they match the WHEN criteria. You can also specify the DISCARDDN keyword to designate a discard data set, which stores the discarded records.

This enhancement is available in Version 5 with APAR PQ19897.

If REORG TABLESPACE discards rows which contain primary keys, a subsequent CHECK DATA job with the DELETE YES and LOG NO options does not log the deletes for the referential constraints.

Unloading data during REORG

You can use the new UNLOAD EXTERNAL option to unload data in a format that is acceptable to the LOAD utility of any DB2 subsystem. You can also specify selection conditions to determine which records are unloaded.

This enhancement is available in Version 5 with APAR PQ19897.

Decreased elapsed and processor time for online REORG

Processor and elapsed time for online reorganizations (REORG with SHRLEVEL CHANGE) is greatly reduced. The more disorganized the data, the greater the benefits. Tests in a controlled environment show that some online reorganizations take half the elapsed time they did in Version 5.

You can maximize availability during an online REORG with SHRLEVEL CHANGE by using the new TIMEOUT option. This enhancement is available in Version 5 with APAR PQ18941.

You can minimize the potential for deadlocks during an online REORG by using the new DRAIN ALL option. This enhancement is available in Version 5 with APAR PQ20032.

Avoid delete and redefine of data sets

Previously: For DB2-managed data sets, DB2 did a delete and redefine of the data sets for LOAD REPLACE, REBUILD INDEX, REORG TABLESPACE, RECOVER TABLESPACE, or RECOVER INDEX. Deleting and redefining the data sets takes longer than just doing a logical reset.

In addition to increased elapsed times, deleting and redefining data sets also makes it difficult to manage the placement of data sets that are managed using storage management subsystem (SMS). When data sets are deleted, they might be moved to a different volume when they are defined. You might have to rework your SMS definitions to get those data sets where you want them.

Now: You can specify a new option, REUSE, to tell DB2 to logically reset data sets for DB2-managed data sets rather than to do a delete and redefine. This improves the performance of those utilities and makes it easier to manage using SMS. This enhancement is available in Version 5 with APAR PQ19077.

Do not specify REUSE when the reason for a recovery or rebuild is a media failure, or if you want to reclaim unused extents for your data sets.

Query performance and optimization enhancements

DB2 continues to improve its ability to execute queries quickly.

In this section:

- "Query parallelism enhancements"
- "Improvements to join processing" on page 36
- "Other query optimization enhancements" on page 41

Query parallelism enhancements

Query parallelism is even more effective in Version 6.

Parallelism for nonpartitioned tables

If a nonpartitioned table is accessed through an index or by a table space scan, DB2 can use query parallelism.

This enhancement applies to single-table access and multi-table joins.

Improved workload balancing in a Parallel Sysplex

Previously: For Sysplex query parallelism, DB2 split the work into equal size work ranges based on its knowledge of how many DB2 subsystems were available at bind time. DB2 also made no adjustments based on mixed processor speeds in the data sharing group. For Sysplexes with mixed processor speeds, this is not always the best way to divide the work for maximum efficiency.

Now: DB2 considers processing speeds when developing its work ranges. If, at run time, either the number of members in the group has changed, or if buffer pool resources have changed, DB2 reformulates the parallel plan to take advantage of this new environment. DB2 can even adjust the degree of parallelism upward if more resources are available at run time.

More opportunities for parallelism

The following queries can now benefit from parallelism:

- Outer joins
- IN list index access of an inner table of a parallel group. For example:

```
SELECT Count(*)
FROM T1, T2
WHERE T1.C1 = T2.C1 and
      T1.C2 > 5     and
      T2.C2  IN ( 6 , 7 , 9) ;
```

  In this example, the access path is a nested loop join of T1 to T2. T2 is using IN list index access and can use parallelism.

Improvements to join processing

Processing joins faster continues to be a priority in DB2, with a special focus on outer joins. Some of the outer join enhancements are available in Version 5 with APAR PQ18710.

Faster outer joins

The following optimization enhancements can improve outer join performance:

- An outer join is no longer a criteria for materializing a view or nested table expression.

  The following example demonstrates such a statement:

```
SELECT * FROM V1 X LEFT JOIN              <- Outer SELECT operation
  (SELECT * FROM T2) Y                    <- Inner SELECT operation
  ON X.C1=Y.C1
  LEFT JOIN T3 Z ON X.C1=Z.C1;
```

- DB2 evaluates predicates more aggressively.

  In Version 6, DB2 more aggressively applies predicates, as shown in the example here, in which the predicate is evaluated before the join:

```
SELECT * FROM T1 X LEFT JOIN T2 Y ON X.C1 = Y.C1
  LEFT JOIN T3 Z ON X.C1 = Z.C1
  WHERE X.C2 = 10;
```

  Because this query does not use rows in which X.C2<>10, it is more efficient to screen X to discard X.C2<>10 before the join operation. Because X has a value in every row in the join result, DB2 can move X.C2 = 10 from the join operation to before the join operation.

  Another example: In the following example, DB2 Version 6 can evaluate the predicate after the left join of the department and employee tables; in Version 5, the predicate could not be evaluated until after all the joins:

```
SELECT ...
  FROM DEPT LEFT JOIN EMP ON ...
   LEFT JOIN SALES ..
  WHERE DEPT.LOC <> EMP.LOC OR EMP.DEPTNO IS NULL;
```

- Optimize transitive closure across outer join operations

  In Version 5, DB2 used the transitive closure rule to derive new predicates called *transitive closure predicates*. Only Boolean term predicates that were evaluated before a join operation and had the following formats were considered for the transitive closure rule:

  - COL = COL
  - COL *op value*
  - COL (NOT) BETWEEN *value1* AND *value2*

  op is one of the following operators: =, <=, >, >=, <, or <=. *value* is a constant, host variable, or special register.

  Predicates in the ON clause of an outer join operation were not considered for transitive closure.

  In Version 6, DB2 considers predicates in the ON clause of an outer join operation for transitive closure. Those predicates must have one of the following formats:

- COL = COL
- COL *op value*
- COL (NOT) BETWEEN *value1* AND *value2*

op is one of the following operators: =, <=, >, >=, <, or <=.

When DB2 applies the transitive closure rule, DB2 generates predicates of the following format:

- COL *op value*
- COL (NOT) BETWEEN *value1* AND *value2*

op is one of the following operators: =, <=, >, >=, <, or <=.

- Join operator transformation

  This enhancement transforms some outer join operations into simpler join operations that execute more efficiently.

  The types of predicates that enable join operation transformation have the following characteristics:

  - The predicate is a Boolean term predicate
  - The predicate becomes false if one table supplies a null value for all of its columns

  DB2 does not use join transformation for predicates that use CASE expressions or user-defined functions.

  For example, DB2 can transform the full outer join operation in the first SELECT statement that follows into a left outer join, as shown in the second SELECT statement:

  More complicated SELECT statement

```
SELECT * FROM T1 X FULL JOIN T2 Y
  ON X.C1 = Y.C1
   WHERE X.C2 > 12;
```

  Simpler SELECT statement

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  ON X.C1 = Y.C1
    WHERE X.C2 > 12;
```

- Avoiding work files when possible.

  For a FROM clause that contains a series of outer join or inner join operators, DB2 uses fewer work files.

- DB2 is more aggressive in choosing a variety of access paths for statements that contain multiple left, right, and inner joins. This gives DB2 more choice in choosing the best access path for the join.

Faster join processing when joining columns of different lengths

Previously: In cases like the following two examples, DB2 processed the predicates as stage 2 because the join columns, although of compatible data types, are not the same length. This means that an available index cannot be used.

```
SELECT  *
FROM  T1, T2
WHERE T1.CHAR10 = T2.CHAR5  ....

SELECT  *
FROM  T1, T2
WHERE T1.VACH10 = T2.VACH15....
```

Now: For CHAR and VARCHAR data types, DB2 can process equal and Boolean term join predicates as stage 1, even when the columns are of different lengths.

Example: In this example, the following columns are relevant:

A.ACCESSNAME (CHAR 18)

B.NAME (VARCHAR 18)

An index exists on A.ACCESSNAME. The following statement can now take advantage of stage 1 processing:

```
SELECT A.ACCESSNAME, B.NAME
  FROM PLAN_TABLE A, SYSIBM.SYSINDEXES B
  WHERE A.ACCESSCREATOR = B.CREATOR
  AND   A.ACCESSNAME    = B.NAME  <---Now indexable
```

Enhanced Cartesian join

Previously: DB2's implementation of Cartesian join was limited to a composite table of 5 dimensions.

Now: The composite table size for a Cartesian join is increased to 6 dimensions.

Star schema (star join)

A star schema or star join is a logical database design that is included in decision support applications. A star schema is composed of a fact table and a number of dimension tables that are connected to it. A dimension table contains several values that are given an ID, which is used in the fact table instead of all the values.

You can think of the fact table, which is much larger than the dimension tables, as being in the center surrounded by dimension tables; the result resembles a star formation. The following diagram illustrates the star formation:

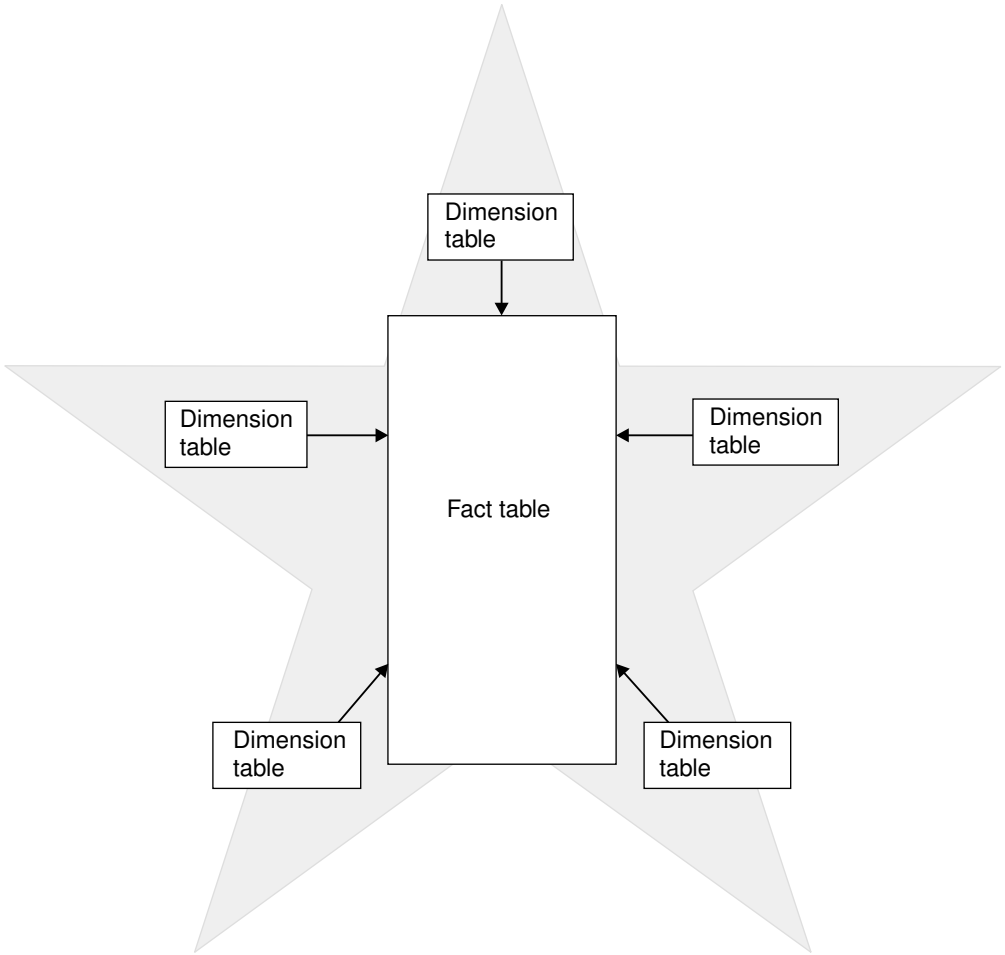


Figure 7. Star schema with a fact table and dimension tables

Example: For an example of a star schema, consider the following scenario. A star schema is composed of a fact table for sales, with dimension tables connected to it for time, products, and geographic locations. The time table has an ID for each month, its quarter, and the year. The product table has an ID for each product item and its class and its inventory. The geographic location table has an ID for each city and its country.

In this scenario, the sales table contains three columns with IDs from the dimension tables for time, product, and location instead of three columns for time, three columns for products, and two columns for location. Thus, the size of the fact table is greatly reduced. In addition, if you needed to change an item, you would do it once in a dimension table instead of several times for each instance of the item in the fact table.

You can create even more complex star schemas by breaking a dimension table into a fact table with its own dimension tables. The fact table would be connected to the main fact table.

When it is used: To access the data in a star schema, you write SELECT statements that include join operations between the fact table and the dimension tables; no join operations exist between dimension tables. When the query meets the following conditions, that query is a star schema:

- The query references at least two dimensions.
- All join predicates are between the fact table and the dimension tables, or within tables of the same dimension.
- All join predicates between the fact table and dimension tables are equi-join predicates.
- All join predicates between the fact table and dimension tables are Boolean term predicates.
- No correlated subqueries cross dimensions.
- No single fact table column is joined to columns of different dimension tables in join predicates. For example, fact table column `F1` cannot be joined to column `D1` of dimension table T1 and also joined to column `D2` of dimension table T2.
- After DB2 simplifies join operations, no outer join operations exist.
- The data type and length of both sides of a join predicate are the same.
- The cardinality of the fact table is at least 25 times the cardinality of the largest dimension that meets both of the following conditions:
  - The dimension is a base table.
  - The dimension is joined directly to the fact table.

Example: query with three dimension tables: Suppose you have a store in San Jose and want information about sales of audio equipment from that store in 2000. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You could write the following query to join the tables:

```
SELECT *
FROM SALES S, TIME T, PRODUCT P, LOCATION L
WHERE S.TIME = T.ID         AND
          S.PRODUCT = P.ID            AND
          S.LOCATION = L.ID     AND
          T.YEAR = 2000               AND
          P.CLASS = 'SAN JOSE';
```

You would use the following index:

```
CREATE INDEX  XSALES_TPL ON SALES (TIME, PRODUCT, LOCATION);
```

Your EXPLAIN output looks like the following table;

QUERYNO	QUERY-BLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN
1	1	0	TIME	S	
1	1	1	PRODUCT	S	
1	1	1	LOCATION	S	
1	1	1	SALES	S	

Figure 8. Plan table output for a star join example with TIME, PRODUCT, and LOCATION

Other query optimization enhancements

Other query optimization enhancements include:

- DB2 can use index screening for access methods with RID processing, such as list prefetch and multi-index access for single-table access and joins. This enhancement avoids processing of unqualified RIDs. Performance is improved because of reduced access to data pages and fewer RIDs to sort. This enhancement is available in Version 5 with APAR PQ15670.
- DB2 can use an index to access predicates with noncorrelated IN subqueries. For example, in the following statement, DB2 was not able to use TAB1's index on PROG to access TAB1 in Version 5.

  ```
  UPDATE TAB1
  SET SDATE = ?, STIME = ?
   WHERE PROG IN
            (SELECT MASTER
             FROM TAB2
             WHERE INCLUDE = ?)
  ```

  In Version 6, DB2 can use TAB1's index on PROG for matching index access for TAB1.

  Another example: Assume that a clustering, nonunique index exists on PRODUCT_NBR. In Version 5, that index could not be used for matching index access for the predicate PRODUCT_NBR IN( ). In Version 6, DB2 can use the matching index.

```
SELECT ....
FROM PRODUCT
WHERE   PRODUCT_NBR IN     <-----Indexable in Version 6
       (SELECT S.PRODUCT_NBR
        FROM SEARCH_TOKENS S
        WHERE S.SEARCH_TOKEN LIKE 'SHOE%')
```

- Improved processing for dynamic SQL statements that refer to temporary tables.

  Previously DB2 did not keep statistics about temporary tables. In some cases, DB2's use of default statistics for the number of rows and for the number of pages could lead to less optimal access paths, especially when temporary tables are joined with other tables.

  Now, DB2 keeps in its memory the number of rows and number of pages for instantiated temporary tables. DB2 can use this information during the optimization of dynamically prepared SQL statements that refer to that table.

  To find out what values DB2 used: IFCID 0022 is enhanced to tell you what values DB2 uses for the number of pages and rows it used to optimize the statement.

  Restrictions: This enhancement is not used for static SQL statements, because the temporary table is not instantiated at the time the access path is chosen. However, if you have an idea of what good values are for CARDF and NPAGES, you can populate SYSYSIBM.SYSTABLES with that information for the temporary table.

  This enhancement also does not apply for dynamic SQL statements that are destined for the dynamic statement cache.

- Improved precision for cluster ratios in the catalog.

  When the access path is chosen for a range predicate on a host variable, DB2 uses the cluster ratio as part of its calculations. However, because CLUSTERRATIO is always a whole percentage value, a table of over 100000 rows can show a CLUSTERRATIO of 99% when 1 row changes or when 1000 rows change. This lack of precision can cause DB2 to choose a less-than-optimal access path.

  Now, DB2 has floating-point columns in SYSINDEXES and SYSINDEXSTATS to maintain the cluster ratio. The new columns are called CLUSTERRATIOF. Run RUNSTATS to populate the new columns.

- More efficient processing of BETWEEN predicates

  DB2 chooses an inefficient access path for a predicate of the form COL BETWEEN *value1* AND *value2*, where *value1* and *value2* are of the same type and length and evaluate to the same value.

  In Version 6, DB2 transforms the BETWEEN predicate to a more efficient equal predicate when the following conditions are true:

  - The predicate is of the form COL BETWEEN *value1* AND *value2*
  - *value1* and *value2* are of the same type and length and evaluate to the same value
  - The predicate is stage 1
  - The bind option REOPT(VARS) is in effect if either *value1* or *value2* is not a constant

- The RUNSTATS utility collects statistics for uniform as well as non-uniform distributions. The extra statistics might improve the access path for queries. This enhancement is available in Version 5 with APAR PQ21014.

Data sharing enhancements

DB2 continues to improve the performance and availability of its Parallel Sysplex data sharing.

This section describes the following topics:

- "Continuous availability with group buffer pool duplexing"
- "Faster checkpointing of group buffer pools" on page 50
- "Reduced P-lock overhead" on page 51
- "More caching options" on page 51
- "More flexible option to wait for retained locks" on page 52

In addition to the topics described here, see the following sections for information about other improvements that affect the performance and availability of data sharing systems:

- "IRLM enhancements" on page 85
- "Increased flexibility with 8-KB and 16-KB page sizes" on page 58
- "Faster restart and recovery" on page 25

Continuous availability with group buffer pool duplexing

Previously: Your only options for recovery in the event of a group buffer pool failure were:

- Group buffer pool *recovery*, by which DB2 recovers data from its logs in case the group buffer pool structure is damaged or if all members lose connectivity to the group buffer pool.
- Group buffer pool *rebuild*, by which DB2 copies pages from group buffer pool to a new allocation of the structure in the alternative coupling facility (or to DASD, if DB2 cannot get enough storage in the alternate coupling facility). DB2 uses this approach when some but not all members lose connectivity to the structure.

Now: With more than one coupling facility, you can duplex the group buffer pools. With duplexing, a secondary group buffer pool is always on standby in another coupling facility, ready to take over if the primary group buffer pool structure fails or if a connectivity failure occurs. If you have three coupling facilities, you can even maintain duplexing while performing maintenance on one of the coupling facilities.

This enhancement is available for Version 5 with APAR PQ17797.

In this section:

- "How duplexing works" on page 44
- "Starting and stopping duplexing for a group buffer pool" on page 45
- "Performance aspects of duplexing" on page 46
- "Monitoring duplexed group buffer pools" on page 47
- "Summary of failure scenarios for duplexed group buffer pools" on page 49
- "Requirements for duplexing" on page 50

How duplexing works

With a duplexed group buffer pool, you have two allocations of the same group buffer pool that use one logical connection. Each structure allocation must be in a different coupling facility.

Recommendation: Configure at least one of the coupling facilities to be non-volatile. If power is lost to both coupling facilities and both coupling facilities are volatile, the group buffer pool must be recovered from the logs.

One allocation is called the *primary structure*. The primary structure is used for cross-invalidation and page registration, and it is the structure from which changed data is cast out to DASD. From an MVS perspective, duplexing is really an extended rebuild, so OS/390 documentation and commands sometimes call the primary structure the *old* structure.

The other allocation of the structure is called the *secondary structure* (referred to by MVS as the *new* structure). As shown in Figure 9, changed data that is written to the primary structure is also written to the secondary structure.

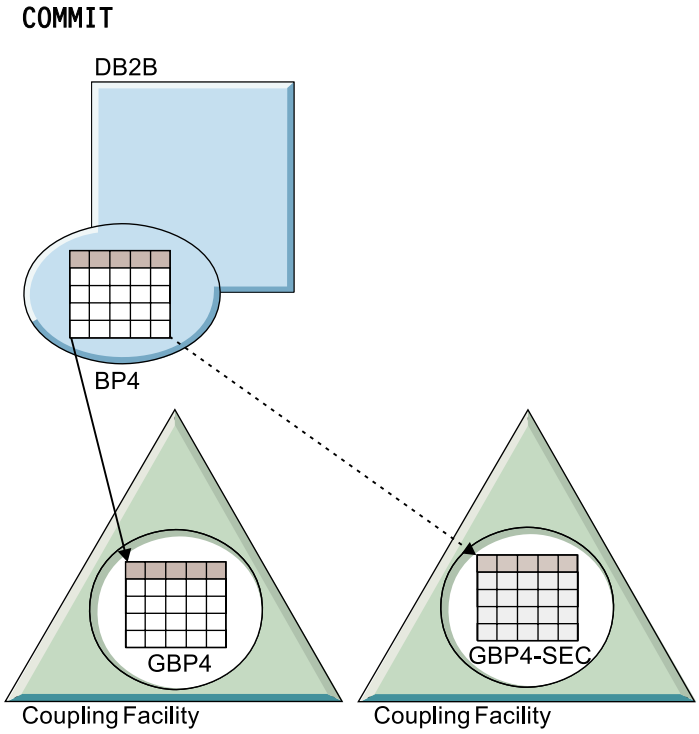


Figure 9. Writing changed data to a primary and secondary group buffer pool

Writing to a duplexed group buffer pool: When a group buffer pool is duplexed, the following events occur:

1. For some fixed number of pages that must be written, DB2 does the following activities for each page:
   a. Write the page to the secondary structure asynchronously.
   b. Write the page to the primary structure synchronously.
2. After all pages have been written to the primary structure, DB2 checks to that the writes to the secondary structure have completed. If any have not completed, DB2 forces the completion of those writes.

Casting out from a duplexed group buffer pool: DB2 casts out data to DASD only from the primary structure. After a set of pages has been cast out, the same set of pages is deleted from the secondary structure. See the DELETE NAME LIST counter in the DISPLAY GROUP BUFFERPOOL MDETAIL report for how many times this event occurs. DB2 ensures that any new pages that are written to the group buffer pool during castout processing are not deleted from the secondary structure.

Characteristics of the secondary structure: The following characteristics of the secondary structure are different than those of the primary structure:

- DB2 does not read data from the secondary structure.
- DB2 does not use the secondary structure for cross-invalidation of pages.
- DB2 does not cast out data to DASD from the secondary structure.

Starting and stopping duplexing for a group buffer pool

This section describes how you can start and stop duplexing for a particular group buffer pool.

Starting duplexing: To start duplexing, at least one DB2 member must be actively connected to the group buffer pool. When duplexing starts, activity to the group buffer pools is quiesced until duplexing is established. This period is generally a few seconds.

Recommendation: Start duplexing during a period of low activity in the system.

Two ways to start duplexing for a group buffer pool are:

- Activate a new CFRM policy with DUPLEX(ENABLED) for the structure. If the group buffer pool is currently allocated, MVS can automatically initiate the process to establish duplexing as soon as you activate the policy. If the group buffer pool is not currently allocated, the duplexing process can be initiated when the group buffer pool is allocated.

  For MVS to automatically initiate duplexing, all CFRM policy parameters other than DUPLEX must be the same as they were before. For example, if you change both the DUPLEX value and the SIZE value, neither value takes effect until you manually rebuild the group buffer pool.
- Activate a new CFRM policy with DUPLEX(ALLOWED) for the structure. If the group buffer pool is currently allocated, use the following command to start the duplexing rebuild:

  ```
  SETXCF START,REBUILD,DUPLEX,STRNAME=strname
  ```

  If the group buffer pool is not currently allocated, wait until it is allocated before starting the duplexing rebuild.

While duplexing is being established, and for the entire time duplexing is in effect, a display of the structure shows the structure as being in DUPLEXING REBUILD. The rebuild phase is called DUPLEX ESTABLISHED, which means that duplexing is truly active for the structure. See Figure 10 on page 48 for an example.

Stopping duplexing: To stop duplexing, you must first decide which instance of the group buffer pool is to remain as the surviving simplexed group buffer pool. If you have a choice, use the primary structure as the surviving group buffer pool. The primary group buffer pool has intact page registration information, while the

secondary GBP has none of this information. So if you switch to the secondary GBP, all GBP-dependent buffers revert to an invalid state and DB2 must refresh the group buffer pool or DASD on the next reference.

To switch temporarily to simplexed mode, use the following method:

1. Optional: If DUPLEX(ENABLED) is specified for the active CFRM policy, activate a new policy specifying DUPLEX(ALLOWED). For the new DUPLEX value to take effect, all other CFRM policy parameters must be the same as before.

   This first step is necessary only if you have at least three coupling facilities, and you do not do not want to automatically reestablish duplexing after stopping it.

2. Use the SETXCF STOP,REBUILD command, specifying KEEP=OLD to revert to using the primary structure as the simplexed structure, or KEEP=NEW to switch to the secondary instance. For example, the following command reverts to using the primary instance as the simplexed group buffer pool:

   ```
   SETXCF STOP,RB,DUPLEX,STRNAME=strname,KEEP=OLD
   ```

If you do not plan on reestablishing duplexing for the group buffer pool in the near future, activate a new CFRM policy specifying DUPLEX(DISABLED) for the structure.

Performance aspects of duplexing

The process of establishing duplexing can be somewhat disruptive because access to the group buffer pool is quiesced while the secondary structure is allocated and changed pages are copied from the primary structure to the secondary structure (or cast out to DASD). Transactions that need access to the group buffer pool during this process are suspended until the process is complete. Because of this disruption, it is best to establish duplexing at a time of low activity on the system. By specifying DUPLEX(ALLOWED) on the CFRM policy, you have more control over when to establish duplexing than you do with DUPLEX(ENABLED). How long the process takes depends upon how many pages are copied to the secondary group buffer pool.

In general, it takes a bit more processor and elapsed time to do duplexed group buffer pool writes and castout processing than it does to do simplexed group buffer pool writes and castout processing. Workloads that are more update-intensive will probably experience a slight increase in host CPU usage when duplexing is activated. In most cases, the majority of the CPU increase will occur in the DB2 address space. Duplexing can cause a slight increase in the transaction elapsed time. Read performance is unaffected by duplexing.

You will also see an increase in the coupling facility CPU usage in the CF that contains the Secondary structure. You can estimate about how much the CF CPU usage will increase when you establish duplexing as follows:

1. Determine the amount of CF CPU that the simplexed Primary structure consumes.
2. Divide the result in half to determine how much CF CPU that the duplexed Secondary structure will consume.

Duplexing should have little or no impact on the CF CPU usage for the CF containing the Primary structure.

The statistics and accounting trace classes contain information about group buffer pool duplexing.

Monitoring duplexed group buffer pools

This section describes the following ways to monitor duplexed group buffer pools:

- "Using MVS DISPLAY XCF command"
- "Using DB2 DISPLAY GROUPBUFFERPOOL command" on page 48
- "Using DB2's trace facility" on page 49

Using MVS DISPLAY XCF command: The following command displays information about GBP1 in group DSNDB0G:

```
D XCF,STR,STRNAME=DSNDB0G_GBP1
```

This particular group buffer pool is duplexed, so you see information about both allocations of the structure (the old structure is the primary structure, and the new structure is the secondary one). Output that is produced is similar to the output shown in Figure 10 on page 48.

```
D XCF,STR,STRNAME=DSNCAT_GBP0
IXC360I  11.13.38  DISPLAY XCF
STRNAME: DSNCAT_GBP0
 STATUS: REASON SPECIFIED WITH REBUILD START:
           OPERATOR INITIATED
         DUPLEXING REBUILD
         REBUILD PHASE: DUPLEX ESTABLISHED  POLICY SIZE    : 32768 K
  POLICY INITSIZE: 5000 K
  REBUILD PERCENT: N/A
  DUPLEX         : ALLOWED
  PREFERENCE LIST: LF01     CACHE01
  EXCLUSION LIST IS EMPTY

DUPLEXING REBUILD NEW STRUCTURE
 ------------------------------
  ALLOCATION TIME: 04/12/1999 11:13:31
  CFNAME         : CACHE01
  COUPLING FACILITY: SIMDEV.IBM.EN.ND0200000000
                     PARTITION: 0   CPCID: 00
  ACTUAL SIZE    : 5120 K
  STORAGE INCREMENT SIZE: 256 K
  VERSION        : B2162049 D1E56F02
  DISPOSITION    : DELETE
  ACCESS TIME    : 0
  MAX CONNECTIONS: 32
  # CONNECTIONS  : 2

 DUPLEXING REBUILD OLD STRUCTURE
 ------------------------------
  ALLOCATION TIME: 04/12/1999 11:12:51
  CFNAME         : LF01
  COUPLING FACILITY: SIMDEV.IBM.EN.ND0100000000
                     PARTITION: 0   CPCID: 00
  ACTUAL SIZE    : 5120 K
  STORAGE INCREMENT SIZE: 256 K
  VERSION        : B2162023 45B3CB06
  ACCESS TIME    : 0
  MAX CONNECTIONS: 32
  # CONNECTIONS  : 2

 CONNECTION NAME  ID VERSION  SYSNAME  JOBNAME  ASID STATE
 ---------------- -- -------- -------- -------- ---- ----------------
 DB2_V61A         02 00020001 UTEC277  V61ADBM1 002F ACTIVE NEW,OLD
 DB2_V61B         01 00010001 UTEC277  V61BDBM1 0033 ACTIVE NEW,OLD
```

Figure 10. MVS command D XCF showing group buffer pool information

For more information about the D XCF command, see *OS/390 MVS System Commands*.

Using DB2 DISPLAY GROUPBUFFERPOOL command

Product-sensitive Programming Interface

The DISPLAY GROUPBUFFERPOOL command of DB2 displays the duplexing status of the group buffer pool and provides some statistics on the secondary instance of that group buffer pool. Figure 11 on page 49 shows what the output

might look like, assuming that the group buffer pool is duplexed. Only the messages relevant to duplexing display.

```
DSNB750I -DB1G DISPLAY FOR GROUP BUFFER POOL GBP0 FOLLOWS
.
.
DSNB757I -DB1G MVS CFRM POLICY STATUS FOR DSNDB0G_GBPO      = NORMAL
                MAX SIZE INDICATED IN MVS POLICY            = 61440 KB
                DUPLEX INDICATOR IN POLICY                  = ENABLED
                CURRENT DUPLEXING MODE                      = DUPLEX
                ALLOCATED                                   = YES
DSNB758I -DB1G  ALLOCATED SIZE                              = 61440 KB
                VOLATILITY STATUS                           = NON-VOLATILE
                REBUILD STATUS                              = DUPLEXED
                CFNAME                                      = LF01
                CFLEVEL                                     = 5
.
.
DSNB799I -DB1G SECONDARY GBP ATTRIBUTES
                ALLOCATED SIZE                          = 61440 KB
                VOLATILITY STATUS                       = NON-VOLATILE
                CFNAME                                  = LF01
                CFLEVEL                                 = 5
                NUMBER OF DIRECTORY ENTRIES             = 61394
                NUMBER OF DATA PAGES                    = 11370
```

Figure 11. Partial output from DISPLAY GROUPBUFFERPOOL command

End of Product-sensitive Programming Interface

See *DB2 Command Reference* for more information about the syntax of the command.

Using DB2's trace facility: The “rebuild-for-duplexing” activity and other reasons for rebuilding are reported in IFCIDs 0267 and 0268.

Summary of failure scenarios for duplexed group buffer pools

For duplexed group buffer pools, a failure response is the same for both structure failures and lost connectivity.

Table 3 (Page 1 of 2). Summary of scenarios for both structure failure and lost connectivity for duplexed group buffer pools

Failure occurred for which structure?	DB2 response	Operational response
Primary	Switch to secondary structure in simplex mode DSNB744I DSNB745I If DUPLEX(ENABLED), reduplexing is attempted.	Correct the problem with the failed coupling facility. If you are not using automatic duplexing, use the command SETXCF START,DUPLEX to establish duplexing when the failed coupling facility becomes available.

Table 3 (Page 2 of 2). Summary of scenarios for both structure failure and lost connectivity for duplexed group buffer pools

Failure occurred for which structure?	DB2 response	Operational response
Secondary	Revert to primary structure in simplex mode DSNB743I DSNB745I If DUPLEX(ENABLED), reduplexing is attempted.	Correct the problem with the failed coupling facility. If you are not using automatic duplexing, use the command SETXCF START,DUPLEX to establish duplexing when the failed coupling facility becomes available.
Both (structure failure or 100% lost connectivity)	Damage assessment, GRECP page sets.	None needed if the group buffer pool is defined with AUTOREC(YES) and DB2 successfully recovers the page set. Otherwise, enter START DATABASE commands.

Requirements for duplexing

To enable duplexing requires the following configuration:

- At least two coupling facilities with a CFLEVEL of 5 or higher must exist in the CFRM policy preference list for the group buffer pool. All members of the data sharing group must have physical connectivity to both coupling facilities in which the primary and secondary structures reside.
- All connected DB2 members must be at Version 6 or a subsequent release, or a Version 5 member with PQ17797 applied.
- All connected DB2 members must be running on OS/390 Release 3 or a subsequent release that has APAR OW28460 installed. (The function is included in the base for OS/390 Release 6.)
- The group buffer pool must be defined with GBPCACHE(YES), which is the default.

Faster checkpointing of group buffer pools

Previously: Group buffer pool checkpoints are used, in part, to keep track of the oldest changed page in the group buffer pool at any particular time. This oldest changed page determines how far back in a member's log DB2 must go to recover the data in event of a failure.

DB2 had to interactively invoke the function used to determine the oldest changed page. For a large group buffer pool, this meant scanning perhaps hundreds of thousands of directory entries, which resulted in spikes of coupling facility resource use. This heavy use of coupling facility resources can negatively impact other coupling facility requests.

Now: If the group buffer pool is allocated in a coupling facility that uses CFLEVEL=5 or higher, and if the system is OS/390 Release 3 through Release 5 with MVS APAR OW28460 (or the base level of OS/390 Release 6), DB2 can determine the information it needs by issuing one call to the coupling facility, greatly diminishing its needs for coupling facility resources.

Reduced P-lock overhead

Previously: In DB2 Version 5, page P-lock latch contention could sometimes cause performance and concurrency problems on a busy system.

Now: Page P-lock processing runs faster, improving concurrency for the entire data sharing system.

More caching options

Previously: You had two options for how you wanted pages to be cached in the group buffer pool (CHANGED or ALL).

Now: Two additional options on CREATE and ALTER TABLESPACE give you the most flexibility in satisfying the needs of your data sharing applications. Additionally, you can specify whether you want data cached at the group buffer pool level. The following enhancements are described here:

- "GBPCACHE SYSTEM (for LOBs)"
- "GBPCACHE NONE ("no caching" option)"

GBPCACHE SYSTEM (for LOBs)

You can specify GBPCACHE SYSTEM on the CREATE or ALTER TABLESPACE statement for LOB table spaces only. When you do, the only pages that are written to the group buffer pool are LOB space map pages. All other data pages are written directly to DASD. SYSTEM is the default for LOB table spaces.

Recommendation: For LOB table spaces, choose GBPCACHE SYSTEM to avoid having large LOB values overwhelm the group buffer pool. Also, for LOB table spaces with the LOG NO attribute, GBPCACHE SYSTEM ensures that LOB values are written to DASD by commit time, thereby avoiding possible recovery problems that are caused by missing log data.

GBPCACHE NONE ("no caching" option)

You have the option of not using the group buffer pool to cache data. Instead, the group buffer pool is used solely for the purpose of cross-invalidating buffers. At every commit, any pages that were updated by the transaction and still have not yet been written are synchronously written to DASD during commit processing. This can have a severe impact on performance for most types of transactions.

Reasons to consider not caching: For workloads that consist of heavy sequential updates (INSERT, DELETE, UPDATE) and in which there is little likelihood that a page will be re-referenced, the following benefits are possible when you do not cache data in the group buffer pool:

- Reduced coupling facility costs and faster coupling facility response time
- Reduced processor time on the host system
- Better transaction throughput at a small cost in higher transaction response time

If you choose not to cache in the group buffer pool, consider enabling DASD Fast Write and setting the vertical deferred write threshhold VDWQT=0. By setting this threshold to 0, you let deferred writes happen continuously before the commit, thus avoiding a large surge of write activity at the commit.

One other advantage of not caching in the group buffer pool is that data does not need to be recovered from the log if the coupling facility fails. However, because DB2 still depends on the cross-invalidation information that is stored in the group buffer pool, a coupling facility failure still means some data might not be available. That is why specifying an alternate coupling facility in the CFRM policy is recommended.

If you are looking for a high-availability option, consider duplexing the group buffer pool rather than suffering the performance ramifications of writing directly to DASD at every commit. See “Continuous availability with group buffer pool duplexing” on page 43 for more information about duplexing your group buffer pool.

How to specify "no caching": Two ways to specify that you do not want to cache data in the group buffer pool are:

- Specify GBPCACHE NONE for the page set.
- Assign the page set to a group buffer pool that is specified with GBPCACHE(NO). To enable a new GBPCACHE attribute for a group buffer pool, change the attribute using the ALTER GROUPBUFFERPOOL command, as described in *DB2 Command Reference* . The new option doesn't take effect until the group buffer pool is reallocated, which you can do with the MVS command SETXCF START,REBUILD.

  Although you can specify GBPCACHE(NONE) on the page set level, specifying it at the group buffer pool level is sometimes appropriate because it is easier to change the attribute on the group buffer pool than on the page set. Because the GBPCACHE(NO) attribute takes precedence over the GBPCACHE option on the page set, you can plan for using different types of processing at different times of day.

  For example, assume that you want to run transactions or queries during the day, but you want to do batch updates at night. If you determine that your batch process can benefit by avoiding the group buffer pool for data caching, you could:

  1. Define the table space as GBPCACHE CHANGED and put it into its own buffer pool.
  2. Define the corresponding group buffer pool as GBPCACHE(YES) for day-time processing.
  3. At night, use the ALTER GROUPBUFFERPOOL command to change the group buffer pool to GBPCACHE(NO).
  4. Set the SETXCF START,REBUILD command to enable the new attribute.
  5. In the morning, use the ALTER GROUPBUFFERPOOL command to change the group buffer pool back to GBPCACHE(YES).
  6. Issue the SETXCF START,REBUILD command to enable the new attribute.

More flexible option to wait for retained locks

Previously: The subsystem parameter RETLWAIT (wait for retained locks) lets you specify either YES or NO. If you specified YES, DB2 would wait for the normal connection timeout value if that connection was waiting on a resource that was held by an incompatible retained lock.

| ***Now:*** This parameter is moved to installation panel DSNTIPI and is called
| RETAINED LOCK TIMEOUT. Now you can specify a timeout *multiplier*, just as you
| do with utility timeout multiplier and other similar options.

| For example, if the retained lock multiplier is 2, then the timeout period for a call
| attachment connection that is waiting for a resource that is held by an incompatible
| retained lock is 1 x 2 (1 for the normal CAF timeout period, 2 for the additional time
| specified for retained locks). If the retained lock multiplier is zero (0), then agents
| do not wait for incompatible retained locks, but instead the lock request is
| immediately rejected and the application receives a “resource unavailable”
| SQLCODE.

| ***Migration considerations:*** In Version 5, the allowed values were YES and NO.
| For migration purposes, note that in Version 6, a 0 corresponds to NO and a 1 to
| YES.

More performance and availability improvements

The following performance and availability improvements are delivered in Version 6:

- “Direct row access”
- “Declared temporary tables” on page 57
- “Increased flexibility with 8-KB and 16-KB page sizes” on page 58
- “Preserving a prior access path” on page 59
- “More buffer pool tuning options” on page 61
- “Control of space map copy maintenance” on page 63
- “Reduced DBD logging for CREATE, ALTER, DROP” on page 63
- “More flexibility when altering space allocations” on page 64
- “Larger log buffer sizes” on page 63
- “Authorization caching for stored procedures and user-defined functions” on page 64
- “More flexibility when altering space allocations” on page 64
- “Deferred allocation of data sets” on page 64
- “More command concurrency” on page 64
- “Increased concurrency for RRSAF and IMS transactions” on page 64

Direct row access

If an application selects a row from a table that contains a ROWID column, the row ID value implicitly contains the location of the row. If you use that row ID value in the search condition of subsequent SELECTs, DB2 might be able to navigate directly to the row. This access method is called *direct row access*.

Direct row access is very fast, because DB2 does not need to use the index or a table space scan to find the row. Direct row access can be used on any table that has a ROWID column.

To use direct row access, you first select the column values for a row into host variables. The value that is selected from the ROWID column contains the location of that row. Later, when you perform queries that access that row, you include the row ID value in the search condition. If DB2 determines that it can use direct row access, it uses the row ID value to navigate directly to the row.

Predicates that qualify for direct row access

For a query to qualify for direct row access, the search condition must be a Boolean term, *stage 1* predicate that fits one of these descriptions:

1. A simple Boolean term predicate of the form COL=*noncolumn expression*, where COL has the ROWID data type, and *noncolumn expression* contains a row ID
2. A simple Boolean term predicate of the form COL IN *list*, where COL has the ROWID data type, and the values in *list* are row IDs, and an index is defined on COL
3. A compound Boolean term that combines several simple predicates using the AND operator, and one of the simple predicates fits description 1 or 2

However, just because a query qualifies for direct row access does not mean that that access path is always chosen. If DB2 determines that another access path is better, direct row access is not chosen.

Examples: In the following predicate example, ID is a ROWID column in table T1. A unique index exists on that ID column. The host variables are of the ROWID data type.

```
WHERE ID IN (:hv_rowid1,:hv_rowid2,:hv_rowid3)
```

The following predicate also qualifies for direct row access:

```
WHERE ID = ROWID(X'F0DFD230E3C0D80D81C201AA0A280100000000000203')
```

Reverting to ACCESSTYPE

Although DB2 might plan to use direct row access, circumstances can cause DB2 to not use direct row access at run time. DB2 remembers the location of the row as of the time it is accessed. However, that row can change locations (such as after a REORG) between the first and second time it is accessed, which means that DB2 cannot use direct row access to find the row on the second access attempt. Instead of using direct row access, DB2 uses the access path that is shown in the ACCESSTYPE column of PLAN_TABLE.

If the predicate you are using to do direct row access is not indexable, and if DB2 is unable to use direct row access, DB2 uses a table space scan to find the row. This can have a profound impact on the performance of applications that rely on direct row access. Write your applications to handle the possibility that direct row access might not be used. Some options are to:

- Ensure that your application does not try to remember ROWID columns across reorganizations of the table space.

  When your application commits, it releases its claim on the table space; it is possible that a REORG can run and move the row, which disables direct row access. Plan your commit processing accordingly—use the returned row ID value before committing, or re-select the row ID value after a commit is issued.

  If you are storing ROWID columns from another table, update those values after the table with the ROWID column is reorganized.
- Create an index on the ROWID column, so that DB2 can use the index if direct row access is disabled.

- Supplement the ROWID column predicate with another predicate that enables DB2 to use an existing index on the table. For example, after reading a row, an application might perform the following update:

```
EXEC SQL
UPDATE EMP
SET SALARY = :hv_salary + 1200
 WHERE EMP_ROWID = :hv_emp_rowid
 AND EMPNO = :hv_empno;
```

  If an index exists on EMPNO, DB2 can use index access if direct access fails. The additional predicate ensures DB2 does not revert to a table space scan.

Direct row access and other access methods

Parallelism: Direct row access and parallelism are mutually exclusive. If a query qualifies for both direct row access and parallelism, direct row access is used. If direct row access fails, DB2 does not revert to parallelism; instead it reverts to the backup access type (as designated by column ACCESSTYPE in the PLAN_TABLE). This might result in a table space scan. To avoid a table space scan in case direct row access fails, add an indexed column to the predicate.

RID list processing: Direct row access and RID list processing are mutually exclusive. If a query qualifies for both direct row access and RID list processing, direct row access is used. If direct row access fails, DB2 does not revert to RID list processing; instead it reverts to the backup access type.

Example: Coding with row IDs for direct row access

Figure 12 on page 56 is a portion of a C program that shows you how to obtain the row ID value for a row, and then to use that value to find the row efficiently when you want to modify it.

```
/*************************/
/* Declare host variables */
/*************************/
EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE IS BLOB_LOCATOR hv_picture;
   SQL TYPE IS CLOB_LOCATOR hv_resume;
   SQL TYPE IS ROWID hv_emp_rowid;
   short hv_dept, hv_id;
   char hv_name[30];
   decimal hv_salary[5,2];
EXEC SQL END DECLARE SECTION;

/*********************************************************/
/* Retrieve the picture and resume from the PIC_RES table */
/*********************************************************/
strcpy(hv_name, "Jones");
EXEC SQL SELECT PR.PICTURE, PR.RESUME INTO :hv_picture, :hv_resume
  FROM PIC_RES PR
  WHERE PR.Name = :hv_name;
/*********************************************************/
/* Insert a row into the EMPDATA table that contains the  */
/* picture and resume you obtained from the PIC_RES table */
/*********************************************************/
EXEC SQL INSERT INTO EMPDATA
  VALUES (DEFAULT,9999,'Jones', 35000.00, 99,
  :hv_picture, :hv_resume);

/*********************************************************/
/* Now retrieve some information about that row,          */
/* including the ROWID value.                             */
/*********************************************************/
hv_dept = 99;
EXEC SQL SELECT E.SALARY,  E.EMP_ROWID
  INTO :hv_salary, :hv_emp_rowid
  FROM EMPDATA E
  WHERE E.DEPTNUM = :hv_dept AND E.NAME = :hv_name;
```

Figure 12 (Part 1 of 2). Example of using a row ID value for direct row access

```
/*********************************************************/
/* Update columns SALARY, PICTURE, and RESUME.  Use the   */
/* ROWID value you obtained in the previous statement     */
/* to access the row you want to update.                  */
/* smiley_face and update_resume are                      */
/* user-defined functions that are not shown here.        */
/*********************************************************/
EXEC SQL UPDATE EMPDATA
  SET SALARY = :hv_salary + 1200,
  PICTURE = smiley_face(:hv_picture),
  RESUME = update_resume(:hv_resume)
  WHERE EMP_ROWID = :hv_emp_rowid;
/*********************************************************/
/* Use the ROWID value to obtain the employee ID from the */
/* same record.                                           */
/*********************************************************/
EXEC SQL SELECT E.ID INTO :hv_id
  FROM EMPDATA E
  WHERE E.EMP_ROWID = :hv_emp_rowid;

/*********************************************************/
/* Use the ROWID value to delete the employee record      */
/* from the table.                                        */
/*********************************************************/
EXEC SQL DELETE FROM EMPDATA
  WHERE EMP_ROWID = :hv_emp_rowid;
```

Figure 12 (Part 2 of 2). Example of using a row ID value for direct row access

Declared temporary tables

Declared temporary tables, which are defined with the new DECLARE GLOBAL # TEMPORARY TABLE statement, provide another way to temporarily hold or sort # data besides the existing support for created temporary tables, which are defined # with the CREATE GLOBAL TEMPORARY TABLE statement. Like a created # temporary table, the rows of a declared temporary table persist only as long as the # application process in which it was defined. However, unlike a created temporary # table, the description of a declared temporary table is not stored in the DB2 catalog # tables, is not shareable across application processes, and persists only for the life # of the application process. Thus, multiple application processes can refer to a # declared temporary table with the same name, but each application process can # have its own unique description of the table. For example, application process A # might define a declared temporary table named TEMP1 with 15 columns while # application process B defines a declared temporary table named TEMP1 with 5 # columns.

Some other important differences between the two types of temporary tables # include:

- # A declared temporary table can have indexes and can be modified by the # searched as well as the positioned forms of the UPDATE and DELETE # statements.
- # Some locking, logging, and limited recovery do apply to declared temporary # tables. Share-level locks on the table space and DBD are acquired. A # segmented table lock is acquired when all the rows are deleted from a declared # temporary table or a declared temporary table is dropped. Changes to the rows

of a declared temporary table can be undone (or rolled back) to a savepoint or the last commit.

- A declared temporary table does not require an associated cursor that is declared WITH HOLD to keep its rows across a commit operation.
- DB2 stores declared temporary tables in segmented table spaces in a database that must be defined AS TEMP. Declared temporary tables can be used only if at least one table space has been explicitly created in that database.
- The qualifier for a declared temporary table must be SESSION (the second part of a three-part name and the first part of a two-part name). If you do not explicitly specify SESSION as the qualifier in the DECLARE GLOBAL TEMPORARY TABLE statement, DB2 implicitly uses SESSION. You must specify SESSION when referring to the declared temporary table in any other SQL statement; otherwise, DB2 assumes that the reference is to a base table.

As with created temporary tables, declared temporary tables provide some performance advantages over persistent base tables. For declared temporary tables, some of the locking, DB2 catalog table updates, and DB2 restart forward and backward log recovery that are associated with base tables are avoided. For a more detailed comparison between declared temporary tables, created temporary tables, and base tables, see Section 2 of *DB2 Administration Guide*. For a description and syntax of the new DECLARE GLOBAL TEMPORARY TABLE statement, see *DB2 SQL Reference*.

Increased flexibility with 8-KB and 16-KB page sizes

Previously: Your only options for data page sizes were 4 KB and 32 KB.

Now: You can choose 8-KB and 16-KB pages. These additional options give you the flexibility to manage DASD space more efficiently without sacrificing I/O performance. In a data sharing environment, these options let you manage coupling facility space more efficiently with less overhead for reads and writes.

New buffer pool names

The size of the data page is determined by the buffer pool in which you define the table space. For example, a table space that is defined in a buffer pool with 8-KB buffers has 8-KB page sizes. (Indexes must be defined in a buffer pool with 4-KB buffers.)

You can have up to ten buffer pools, each with 16-KB buffers and 8-KB buffers. Their names are:

- BP8K0, BP8K1,.....,BP8K9 for 8-KB buffers
- BP16K0, BP16K1,.....,BP16K9 for 16-KB buffers

Recommendations for choosing a page size

Initially, use the default of 4-KB page sizes when access to the data is random and only a few rows per page are needed. If row sizes are very small, use the 4-KB page size.

However, there are situations in which larger page sizes are needed or recommended:

- When the size of individual rows is greater than 4 KB, you must use a larger page size. When considering the size of work file table spaces, remember that

some SQL operations, such as joins, can create a result row that does not fit into a 4-KB page. For this reason, define at least one buffer pool with 32-KB buffers. (Work files cannot use 8-KB or 16-KB pages.)

- When you can achieve higher density on DASD, choose a larger page size. For example, only one 2100-byte record can be stored in a 4-KB page, wasting almost half of the space, but storing the record in a 32-KB page can significantly reduce this waste. The disadvantage with this approach is that there is the potential of incurring higher buffer pool storage costs or higher I/O costs. If you access only a small number of rows, you are bringing a larger amount of data from DASD into the buffer pool.

  Using 8-KB or 16-KB page sizes can let you store more data on your DASD with less impact on I/O and buffer pool storage costs. If you use a larger page size and access is random, you might have to go back and increase the size of the buffer pool to achieve the same read-hit ratio you do with the smaller page size.

- When a larger page size can reduce data sharing overhead, choose a larger page size. One way to reduce the cost of data sharing is to reduce the number of times the coupling facility must be accessed. Particularly for sequential processing, larger page sizes can reduce this number. More data can be returned on each access of the coupling facility, and fewer locks must be taken on the larger page size, further reducing coupling facility interactions.

  If data is returned from the coupling facility, each access that returns more data is more costly than those that return smaller amounts of data. However, because the total number of accesses is reduced, coupling facility overhead is reduced.

  For random processing, using an 8-KB or 16-KB page size instead of a 32-KB page size might improve the read hit-ratio to the buffer pool and reduce I/O resource consumption.

Preserving a prior access path

In Version 6, you can have more direct control over the access path that DB2 chooses in those rare situations when the chosen path is not optimal or if you want to temporarily bypass a chosen path.

Only experienced database administrators should attempt to use optimization hints. Always test and analyze the results of any query that uses optimization hints.

Consider giving optimization hints for the following situations:

- You have rebound a plan or package and have encountered a performance problem. If the database or application has changed, or if DB2 has new function that causes it to choose a different access path, it is handy to have the ability to use an old access path if the new one does not perform as well.

  It is always a good idea to save the access paths of your important programs and queries before migrating to a new release of DB2.

- You want to bypass the access path chosen by DB2.

Use the new version PLAN_TABLE

Appendix E, “EXPLAIN table changes” on page 293 shows the SQL statements to create the Version 6 PLAN_TABLE. For best performance, create an ascending index on the PLAN_TABLE with the following columns:

- QUERYNO
- APPLNAME
- PROGNAME
- VERSION
- COLLID
- OPTHINT

The DB2 sample library, in member DSNTESC, contains an appropriate CREATE INDEX statement that you can modify.

Enabling optimization hints

On the subsystem where the application is bound or where dynamic queries are prepared, specify YES in the OPTIMIZATION HINTS field of installation panel DSNTIP4. If you specify NO, DB2 ignores any hints.

Creating the hint

This section describes the procedure you use to create a hint out of an existing access path. You can also populate PLAN_TABLE to force a different access path, but that scenario is not described here.

Assign a query number to the SQL statement for which you want to create a hint. Use the new QUERYNO clause, and use that query number to create a corresponding row in the PLAN_TABLE (or to update an existing row). Use EXPLAIN to put rows into the PLAN_TABLE.

Overview of procedure:

1. Optionally, assign the statement a query number, as described in “Correlating SQL statements with PLAN_TABLE rows.”
2. Run EXPLAIN to put the rows into PLAN_TABLE.
3. Update the rows in PLAN_TABLE, as described in “Update the PLAN_TABLE” on page 61.

Correlating SQL statements with PLAN_TABLE rows: Using the new QUERYNO clause makes correlating a statement with a row in the PLAN_TABLE easier. You do not need to assign a query number to use optimization hints. If you don't assign a query number, DB2 uses the statement number. However, assigning a query number is especially useful in the following cases:

- For dynamic statements

  The query number for dynamic applications is the statement number in the application *where the prepare occurs*. For some applications, such as DSNTEP2, the same statement in the application prepares each dynamic statement, resulting in the same query number for each dynamic statement. Assigning a query number to each statement that uses optimization hints eliminates ambiguity as to which rows in the PLAN_TABLE are associated with each query.

- For static statements

  If you change an application that has static statements, the statement number might change, causing rows in the PLAN_TABLE to be out of sync with the modified application. Statements that use the QUERYNO clause are not dependent on the statement number. You can move those statements around within the application source without affecting the relationship between rows in the PLAN_TABLE and the statements that use those rows in the application.

Here is an example of the QUERYNO clause:

```
SELECT * FROM T1
  WHERE C1 = 10 AND
  C2 BETWEEN 10 AND 20 AND
  C3 NOT LIKE 'A%'
  QUERYNO 100;
```

Update the PLAN_TABLE: Make the PLAN_TABLE rows for that query (QUERYNO=100) into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is OLDPATH:

```
UPDATE PLAN_TABLE
  SET OPTHINT = 'OLDPATH' WHERE
  QUERYNO = 100 AND
   APPLNAME = ' ' AND
   PROGNAME = 'DSNTEP2' AND
   VERSION = '' AND
   COLLID = 'DSNTEP2';
```

Setting the special register

For dynamic statements, you can use a new special register, SET CURRENT OPTIMIZATION HINT to specify to DB2 the name of the hint to use.

```
SET CURRENT OPTIMIZATION HINT = 'OLDPATH';
```

You can put the name of the optimization hint in a host variable. If you don't explicitly set the special register, the value you specify for the bind option OPTHINT is used.

Rebinding

Rebind the plan or package to see if DB2 uses the hint. Use the OPTHINT bind option, described in *DB2 Command Reference*. As a result of the BIND, DB2 issues warning SQLCODEs to indicate whether it used the hints. If EXPLAIN is being done during the REBIND, the PLAN_TABLE rows indicate if the hints are used.

More buffer pool tuning options

You can tune the buffer pool by choosing a page-stealing method and by specifying VDWQT as an absolute number of pages.

Choosing a page-stealing method

When DB2 takes away a page in the buffer pool to make room for a newer page, this is called “stealing” the page from the buffer pool. DB2 usually uses a least-recently-used (LRU) algorithm for managing pages in storage. That is, it takes away older pages so that more recently used pages can remain in the virtual buffer pool.

However, by using the ALTER BUFFERPOOL command, you can also choose to have DB2 use a first-in, first-out (FIFO) algorithm. With this simple algorithm, DB2 does not keep track of how often a page is referenced—the pages that are oldest are moved out, no matter how frequently they are referenced. This approach to page stealing results in a small decrease in the cost of doing a getpage operation, and it can reduce internal DB2 latch contention in environments that require very high concurrency.

Recommendations:

- In most cases, keep the default, LRU.
- Use FIFO for buffer pools that have little or no I/O; that is, the table space or index remains in the buffer pool. Because all the pages are there, there is no need to pay the additional cost of a more complicated page management algorithm, and it can cause even cause a severe performance degradation.
- Keep objects that can benefit from the FIFO algorithm in different buffer pools from those that benefit from the LRU algorithm.

Specifying VDWQT as an absolute number of pages

Vertical deferred write threshold (VDWQT) is similar to the deferred write threshold, but it applies to the amount of updated pages for a data set in the buffer pool. If the percentage or number of updated pages for the data set exceeds the threshold, writes are scheduled for that data set.

You can specify this threshold in one of two ways:

- As a *percentage* of the virtual buffer pool that might be occupied by updated pages from a single page set.

  The default value for this threshold is 10%. You can change the percentage to any value from 0% to 90%.
- As the total *number of buffers* in the virtual buffer pool that might be occupied by updated pages from a single page set.

  You can specify the number of buffers from 0 to 9999. If you want to use the number of buffers as your threshold, you must set the percentage threshold to 0.

Changing the threshold: Change the percent or number of buffers by using the VDWQT keyword on the ALTER BUFFERPOOL command.

Because any buffers that count toward VDWQT also count toward DWQT, setting the VDWQT percentage higher than DWQT has no effect: DWQT is reached first, write operations are scheduled, and VDWQT is never reached. Therefore, the ALTER BUFFERPOOL command does not allow you to set the VDWQT percentage to a value greater than DWQT. You can specify a number of buffers for VDWQT that is higher than DWQT, but again, with no effect.

This threshold is overridden by certain DB2 utilities, which use a constant limit of 64 pages rather than a percentage of the virtual buffer pool size. LOAD, REORG, and RECOVER use a constant limit of 128 pages.

Setting VDWQT to 0: If you set VDQWT to zero for both the percentage and number of buffers, the minimum number of pages is MIN(32,1%) of VP size.

Control of space map copy maintenance

Previously: When an application rapidly updated a single table space, there was possible contention on the space map pages as DB2 tracked changes to pages within the table space. DB2 tracks these changes to help make incremental image copies run faster, but the space map page contention can cause performance problems, especially in a data sharing environment.

Now: With the TRACKMOD NO option on CREATE and ALTER TABLESPACE, DB2 avoids tracking changes and thus avoids contention on the space map. Do not specify this option if you depend on fast incremental image copies for your backup strategy. However, if you rely on other means of making backups, TRACKMOD NO can reduce the overhead that is associated with tracking changes in the space map.

For more information about the syntax, see *DB2 SQL Reference*.

Reduced DBD logging for CREATE, ALTER, DROP

Previously: When an application entered multiple CREATE, ALTER, or DROP statements in a single unit of recovery, DB2 logged the changed database descriptor (DBD) once for each statement. Depending on the size of the DBD, this can be a significant amount of logging activity.

Now: DB2 logs the changed DBD once for each unit of recovery rather than once for each statement. For applications that do a lot of data definition within the same database and within a unit of recovery, this means significantly less logging overhead. The X lock on the DBD is held until the commit is issued.

Improved performance for DROP

In addition to the enhancement in "Reduced DBD logging for CREATE, ALTER, DROP," two enhancements improve the performance of some DROP operations:

- DROP STOGROUP performance is improved because of new indexes on catalog tables SYSINDEXPART and SYSTABLEPART.
- DROP TABLESPACE and DROP INDEX performance is improved for DB2-managed data sets. This same enhancement applies for utilities that logically reset the data set for DB2-managed data sets, including LOAD REPLACE, REORG, RECOVER, and REBUILD INDEX.

Larger log buffer sizes

The default for the log output buffer size is increased in Version 6 to 4 MB. You can modify that size to be up to 400 MB (but be sure our system can handle a very large size). By increasing this buffer size, DB2 can perform better for both log reads and log writes. Log writes improve because DB2 is less likely to need to wait for a buffer. Log reads improve because DB2 searches the buffer first when it must read the log. If the information is in the log buffer, going to DASD is unnecessary.

In Version 6, the default log read buffer size is increased to 60 KB.

Authorization caching for stored procedures and user-defined functions

The routine authorization cache stores a list of authorization IDs that hold the EXECUTE privilege on stored procedures and user-defined functions. You set the cache size using the ROUTINE AUTH CACHE field on installation panel DSNTIPP.

By checking the routine authorization cache for routine authorization IDs, DB2 avoids the performance expense of catalog lookups. This enhancement offers the most benefit for packages that run frequently in different collections.

More flexibility when altering space allocations

Previously: Before you could change the primary or secondary space allocation for a page set, you had to stop the page set.

Now: With the enhancement delivered in Version 5 APAR PQ04053, you can change those attributes without first stopping the page set. If you change the secondary quantity, DB2 uses that new value the next time it extends the data set, but that new value is not reflected in the integrated catalog until you run REORG, LOAD REPLACE, RECOVER, or REBUILD INDEX.

Deferred allocation of data sets

Previously: When you created a tablespace or index space, the underlying VSAM data sets were always allocated immediately.

Now: With the DEFINE NO clause of the CREATE TABLESPACE and CREATE INDEX statements, you can defer the physical allocation of the underlying VSAM data sets until data if first inserted into the table space or index space. DEFINE NO is applicable only for DB2-managed data sets. Deferring the allocation of the data sets allows the CREATE TABLESPACE and CREATE INDEX statements to execute faster and helps optimize the use of DASD resources. These advantages might be particularly beneficial when you use an applicaton package that defines numerous tables that you might not use.

More command concurrency

Previously: The following two commands sometimes could cause concurrency problems:

- DISPLAY DATABASE with the LOCKS option
- MODIFY irlmproc,STATUS

Now: Both commands now allow greater concurrency. The DISPLAY DATABASE enhancement is also available with IRLM APAR PQ15854.

Increased concurrency for RRSAF and IMS transactions

The Recoverable Resource Manager Services attachment facility (RRSAF) relies on an OS/390 component called OS/390 Transaction Management and Recoverable Resource Manager Services (OS/390 RRS). OS/390 RRS provides system-wide services for coordinating two-phase commit operations across MVS products. For RRSAF applications and IMS transactions that run under OS/390 RRS, you can group together a number of DB2 agents into a single global transaction. A global transaction allows multiple DB2 agents to participate in a single global transaction

and thus share the same locks and access the same data. When two agents that are in a global transaction access the same DB2 object within a unit of work, those agents will not deadlock with each other. The following restrictions apply:

- A global transaction is restricted to a single DB2 subsystem running on a CEC, which has no Parallel Sysplex support.
- Because each of the "branches" of a global transaction are sharing locks, uncommitted updates issued by one branch of the transaction are visible to other branches of the transaction.
- Claim/drain processing is not supported across the branches of a global transaction, which means that attempts to issue CREATE, DROP, ALTER, GRANT, or REVOKE may deadlock or timeout if they are requested from different branches of the same global transaction.
- Attempts to update a partitioning key may deadlock or timeout because of the same restrictions on claim/drain processing.
- LOCK TABLE may deadlock or timeout across the branches of a global transaction.

Chapter 4. User productivity

This chapter describes user productivity enhancements that improve DB2 family consistency, expand your access choices, and make administrative, operational, and programming tasks easier.

- "Built-in function extensions"
- "New ROWID data type"
- "DB2 REXX Language Support" on page 72
- "More flexibility and control" on page 72
- "IRLM enhancements" on page 85
- "More user productivity enhancements" on page 91

Built-in function extensions

In addition to the user-defined functions introduced in Version 6, DB2 provides many new built-in functions and extends existing functions to improve compatibility with the same functions across the DB2 family of products.

The new built-in functions include column functions, scalar functions, a number of new functions for date and time, and functions for manipulating strings. Many of the built-in functions support the object-relational extensions; for example, LOB values can be used in the string built-in functions.

See Appendix C, "Changes to SQL" on page 269 for a detailed description of each function.

New ROWID data type

DB2 introduces a new data type called ROWID. With the ROWID data type, a unique value for each row in a table is generated by DB2.

Because row IDs are always generated by DB2, you cannot generate your own values for a ROWID column. You can specify whether the value of a ROWID column can be supplied at insertion time, but this ability is only provided to facilitate data propagation. It is not intended to allow you to generate your own row IDs.

Purposes: A value in a ROWID column is used to uniquely and permanently identify rows in a DB2 table. Other uses for ROWID columns include:

- For direct access to a row, as described in "Direct row access" on page 53.
- To locate LOB values in a row, as described in "Introduction to defining LOBs" on page 124.
- As a partitioning key value for a partitioned table space, as described in "Using a ROWID column as the partitioning key" on page 69.

Characteristics of the ROWID data type

The characteristics of the ROWID data type are:

- It is a new data type with its own rules of operations. ROWIDs are only compatible with ROWIDs.
- A ROWID can never be null.
- Its value is generated by DB2. The unique identity of a row is maintained across table space reorganizations. (This is in contrast with record IDs (RIDs), that can change whenever a table is reorganized.)
- It can be used in a SELECT list or as a search condition. For example:

```
SELECT EMP_ROWID INTO :hv_emp_rowid
                 FROM EMP
                 WHERE ...

DELETE FROM EMP
 WHERE EMP_ROWID = :hv_emp_rowid
```

- ROWID columns contain bit data and are not subject to character conversion. Their format is internal and transparent to the SQL end user.
- A ROWID column takes up to 19 bytes of physical storage and 17 bytes for each index key on a ROWID column.

Restrictions: ROWID columns have the following restrictions:

- A table cannot have more than one ROWID column.
- A trigger cannot modify a ROWID column, nor can ROWID columns be updated.
- A ROWID column must be defined as NOT NULL.
- A ROWID column cannot have field procedures or check constraints defined on it.
- A table with a ROWID column cannot have an edit procedure defined on it.
- The ROWID data type cannot be used with DB2 private protocol.

Defining a ROWID column

Create a ROWID column for a table as you would any other column, using the ALTER TABLE or CREATE TABLE statement as described in *DB2 SQL Reference*. There are two options for how row IDs should be generated: GENERATED ALWAYS or GENERATED BY DEFAULT. Each of those options is described in this section. These options also have implications for LOAD, as described in “Loading columns defined as ROWID” in *DB2 Utility Guide and Reference*.

GENERATED ALWAYS

The clause GENERATED ALWAYS means that you want DB2 to generate a row ID and that users are not allowed to specify a value. For most cases, GENERATED ALWAYS is the recommended option:

```
CREATE TABLE EMP
  (EMP_ROWID   ROWID NOT NULL GENERATED ALWAYS,
   ID          SMALLINT,
   NAME        CHAR(30),
...
```

GENERATED BY DEFAULT

GENERATED BY DEFAULT means that DB2 accepts valid row IDs as inserted values into a row. If you do not specify a value, DB2 generates the value for you. GENERATED BY DEFAULT is recommended for tables that are populated by propagation from another table.

Required index: If you specify GENERATED BY DEFAULT, you must define a unique, single-column index on the ROWID column. Until this index is created, you cannot add rows using INSERT.

If you create the table using SQLRULES(STD), the index on the ROWID column is created implicitly. The name is "I" concatenated with the first ten characters of the column name followed by seven characters that DB2 chooses.

Using a ROWID column as the partitioning key

The generated value of a ROWID column acts like a random value in the range of X'00000...' through X'FFFFF...'. Because of this semi-random nature, a ROWID column used as a partitioning key distributes the rows of a partitioned table across the partitions in a random way. Therefore, if you would like to partition your table, and it has no other column that would serve as a good partitioning key, you can include a ROWID column in your table and use it as your partitioning key.

Example: Assume you want to create four partitions for an employee table, but because employee numbers are assigned in ascending order, the employee number is not a good partitioning key. You can include a ROWID column in your table definition and use that as the partitioning key as shown here:

```
CREATE TABLESPACE EMPSPACE NUMPARTS 4;

CREATE TABLE EMP
               (Id          SMALLINT,
                Name        CHARACTER(30),
                Salary      DECIMAL(7,2),
                Deptnum     SMALLINT,
                Emp_Rowid   ROWID NOT NULL GENERATED ALWAYS,
                Picture     BLOB(300K),
                Resume      CLOB(100K))
             IN EMPSPACE;

CREATE INDEX EMPARTIX ON EMP(Emp_Rowid)
       CLUSTER (PART 1 VALUES(X'3FFF'),
                PART 2 VALUES(X'7FFF'),
                PART 3 VALUES(X'BFFF'),
                PART 4 VALUES(X'FFFF'));
```

Because DB2 pads the ascending PART values, the CREATE INDEX statement does not need to specify a complete value of the entire length of a row ID. (In a WHERE clause, though, the value compared to a row ID is not padded, so complete values must be specified to confine an SQL statement to a particular partition.)

Casting to a ROWID

ROWID values can only be assigned to ROWID columns or ROWID variables. To assign a character string to ROWID, you must first cast it to the ROWID data type. There are two ways to cast a CHAR or VARCHAR expression, or a HEX literal, to ROWID:

- Using the CAST function:

  ```
  CAST (expression  AS ROWID)
  ```

  ```
  CAST (X'hex_literal' AS ROWID)
  ```

- Using the ROWID function:

  ```
  ROWID (varchar-expression)
  ```

When casting a value to a ROWID, the CAST or ROWID function only returns a valid ROWID value if the argument to the function is a ROWID value that was previously generated by DB2. For example, you can use the ROWID function to convert a ROWID value that was cast to a CHAR value back to a ROWID value. You cast a ROWID value to a VARCHAR as follows:

```
CAST (rowid-expression AS VARCHAR(40))
```

Inserting into a ROWID column

To insert into a ROWID column, use the new DEFAULT clause of the INSERT statement. The DEFAULT clause lets you insert a default value into a column (assuming that default values are allowed for the column). If the column is a ROWID column, DB2 generates a unique value for the column.

Example: Insert a row with a generated row ID and set the employee address to its default value:

```
INSERT INTO T2 (EMPROWID, EMPNAME, EMPADDR)
  VALUES (DEFAULT,:hv_name,DEFAULT);
```

If the ROWID column is defined as GENERATED BY DEFAULT and has the unique single-column index defined on it, you can insert a valid row ID. A valid row ID is one that was previously generated by DB2. Do not attempt to create and insert your own row ID.

Example:

```
INSERT INTO T2 (EMPROWID, EMPNAME, EMPADDR)
  VALUES (:hv_valid_rowid,:hv_name,DEFAULT);
```

The above statement does not work for a ROWID column that is defined as GENERATED ALWAYS.

INSERT with a subselect: The rules for INSERT with a subselect are similar to those for an INSERT with a VALUES clause. You can specify a ROWID value only if the ROWID column is defined as GENERATED BY DEFAULT. The following example is valid only when the ROWID column of T2 is defined as GENERATED BY DEFAULT:

```
INSERT INTO T2
  SELECT * FROM T1;
```

If you have a table with a column defined as ROWID NOT NULL GENERATED ALWAYS, you can propagate all non-ROWID columns to another table with the same definitions in one of the following ways:

- By not specifying the ROWID column in the SELECT list:

```
INSERT INTO T2 (intcol1)
  SELECT intcol1 FROM T1;
```

- By specify the OVERRIDING USER VALUE clause to tell DB2 to ignore any values that you supply for the ROWID column:

```
INSERT INTO T2 (INTCOL2,ROWIDCOL2) OVERRIDING USER VALUE
  SELECT INTCOL1, ROWIDCOL1 FROM T1;
```

Declaring host variables for ROWID columns

Use a new host variable type, SQL TYPE IS ROWID, to store row ID values. You can select a row ID into a host variable and then use it in subsequent SQL statements, such as in a search condition. This gives an application the means of directly referencing a row with no other qualifications needed.

Restriction: You cannot use a host variable of type SQL TYPE IS ROWID in datetime or numeric operations or with the concatenation operator.

Table 4 shows the SQL declaration and equivalent host language declarations for ROWID host variables. ROWID host variables are supported in host structures for C, C++, and PL/I and in group items for COBOL.

Table 4. Host language declarations for ROWID host variables

Host language	You declare this variable	DB2 generates this variable
Assembler	*host-variable* SQL TYPE IS ROWID	`host-variable DS H,CL40`
C	SQL TYPE IS ROWID *host-variable;*	struct { short int length; char data[40]; } *host-variable;*
C++	SQL TYPE IS ROWID *host-variable*;	struct { short int length; char data[40]; } *host-variable;*
COBOL	level-1 *host-variable* <USAGE <IS> > SQLTYPE IS ROWID.	01 *host-variable*. 02 host-variable-LEN PIC S9(4). 02 host-variable-TEXT PIC X(40).
Note: For COBOL, USAGE IS is optional. The declaration for COBOL ROWID host variable can be coded as USAGE IS SQL TYPE IS ROWID, USAGE SQL TYPE IS ROWID, or SQL TYPE IS ROWID.		
FORTRAN	SQLTYPE IS ROWID *host-variable*	CHARACTER 40 *host-variable*
PL/I	*host-variable* SQL TYPE IS ROWID	DCL *host-variable* CHARACTER(40) VARYING

DB2 REXX Language Support

IBM DATABASE 2 Universal Database Server for OS/390 REXX Language Support, is a separately-orderable feature of DB2. DB2 REXX Language Support provides the ability to write SQL application programs in the REXX programming language. See *DB2 Installation Guide* and the DB2 REXX Language Support Program Directory for information on installing DB2 REXX Language Support. See *DB2 Application Programming and SQL Guide* for information on writing and running REXX SQL applications and stored procedures.

More flexibility and control

These changes are described in the sections that follow:

- "Predictive governing"
- "Statement cost estimation" on page 79
- "Set default buffer pools" on page 83
- "More information available for monitoring DB2" on page 83

Predictive governing

DB2's resource limit facility (governor) lets you set warning and error thresholds by which the governor can inform users (via your application programs) that a certain processing limit might be exceeded for a particular dynamic SELECT, INSERT, UPDATE, or DELETE statement. This function is called *predictive governing.*

DB2's predictive governing capability has an advantage over the reactive governor, which stops a currently executing dynamic SQL statement that exceeds the processor limit that is specified for that statement. With a predictive governor, you can stop a statement from starting to execute and thus avoid wasting resources. Resource is spent to roll back a statement when a reactive governor stops a statement, and there is also the resource used to execute the statement. With predictive governing both the costs of executing and rolling the statement back are saved.

Requirement: For complete predictive governing support for incoming client requests, the requester must be at DRDA level 4. DB2 for OS/390 Version 6 and DB2 Connect Version 5.2 provide this support.

In this section:

- "Overview of predictive governing process" on page 73
- "Creating an RLST" on page 74
- "Descriptions of the RLST columns" on page 75
- "Combining reactive and predictive governing" on page 78
- "Writing an application to handle predictive governing" on page 79

Overview of predictive governing process

With predictive governing, at run time, DB2 determines which *cost category* a particular statement belongs to. A cost category is DB2's way of distinguishing statements for which enough information exists to make a good cost estimate from those that do not. Cost category A is a statement for which adequate information exists to make a good estimate. See “Understanding the implications of cost categories” on page 82 for more information about cost categories. You can tell which category a statement falls into by looking at the COST_CATEGORY column of the DSN_STATEMNT_TABLE for that statement.

See Figure 13 for an overview of how predictive governing works.

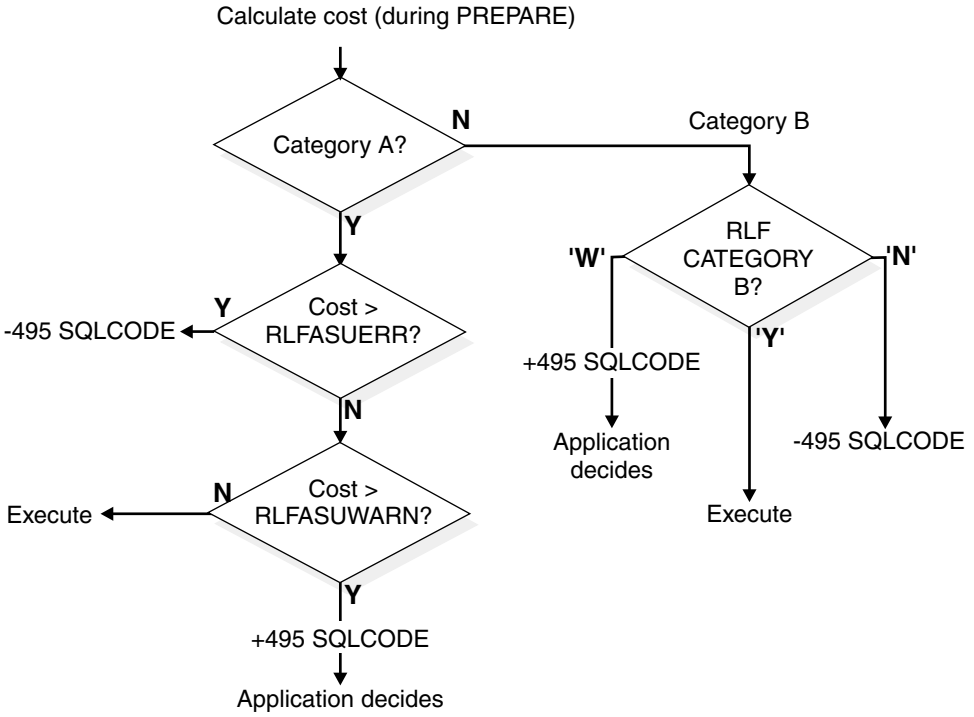


Figure 13. Processing for predictive governing.

At prepare time for a dynamic SELECT, INSERT UPDATE, or DELETE statement, DB2 searches the active RLST to determine if the processor cost estimate exceeds the error or warning threshold that you set in RLFASUWARN and RLFASUERR columns for that statement. DB2 compares the cost estimate for a statement to the thresholds you set, and the following actions occur:

- If the cost estimate is in cost category A and the error threshold is exceeded, DB2 returns a -495 SQLCODE to the application at PREPARE time, and the statement is not prepared or executed.
- If the estimate is in cost category A and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.
- If the estimate is in cost category B, DB2 takes the action you specify in the RLF_CATEGORY_B column; that is, it prepares and executes the statement, it does not prepare or execute the statement, or it returns a warning SQLCODE, which lets the application decide what to do.

Example: Table 5 on page 74 is an RLST with two rows that use predictive governing.

Table 5. Predictive governing example

RLFFUNC	AUTHID	RLFCOLLN	RLFPKG	RLF-ASU-WARN	RLF-ASU-ERR	RLF_CATEGORY_B
7	(blank)	COLL1	C1PKG1	900	1500	Y
7	(blank)	COLL2	C2PKG1	900	1500	W

The rows in the RLST for this example cause DB2 to act as follows for all dynamic INSERT, UPDATE, DELETE, and SELECT statements in the packages listed in this table (C1PKG1 and C2PKG1):

- Statements in cost category A that are predicted to be less than 900 SUs will execute.
- Statements in cost category A that are predicted to be between 900 and 1500 SUs receive a +495 SQLCODE.
- Statements in cost category A that are predicted to be greater than 1500 SUs receive SQLCODE -495, and the statement is not executed.

Cost category B: The two rows in Table 5 differ only in how statements in cost category B are treated. For C1PKG1, the statement will execute. For C2PKG2, the statements receive a +495 SQLCODE, and the user or application must decide whether to execute the statement.

Creating an RLST

You use resource limit specification tables (RLSTs) to give governing information to DB2. Resource limit specification tables can reside in any database, but, because a database has some special attributes while the resource limit facility is active, they should reside in a database of their own.

When you install DB2, installation job DSNTIJSG creates a database, table space, table, and descending index for the resource limit specification. You can tailor those statements.

To create a new resource limit specification table, use the statements in Figure 14 on page 75, also included in installation job DSNTIJSG. You must have sufficient authority to define objects in the DSNRLST database and to specify *authid*, which is the authorization ID specified on field RESOURCE AUTHID of installation panel DSNTIPP.

```
CREATE TABLE authid.DSNRLSTxx
       (AUTHID   CHAR(8)      NOT NULL WITH DEFAULT,
        PLANNAME CHAR(8)      NOT NULL WITH DEFAULT,
        ASUTIME  INTEGER,
        -------3-column format --------
        LUNAME   CHAR(8)      NOT NULL WITH DEFAULT,
        -------4-column format --------
        RLFFUNC  CHAR(1)      NOT NULL WITH DEFAULT,
        RLFBIND  CHAR(1)      NOT NULL WITH DEFAULT,
        RLFCOLLN CHAR(18)     NOT NULL WITH DEFAULT,
        RLFPKG   CHAR(8)      NOT NULL WITH DEFAULT),
        -------8-column format --------
        RLFASUERR  INTEGER,
        RLFASUWARN INTEGER,
        RLF_CATEGORY_B CHAR(1) NOT NULL WITH DEFAULT)
        -------11-column format --------
         IN DSNRLST.DSNRLSxx;
```

Figure 14. Creating a resource limit specification table (RLST).

To create an index for the 11-column format, use the following SQL:

```
CREATE UNIQUE INDEX authid.DSNARLxx
       ON authid.DSNRLSTxx
         (RLFFUNC, AUTHID DESC, PLANNAME DESC,
          RLFCOLLN DESC, RLFPKG DESC, LUNAME DESC)
         CLUSTER CLOSE NO;
```

Table name: The name of the table is *authid*.DSNRLST*xx*, where *xx* is any two-character alphanumeric value, and *authid* is specified when DB2 is installed. Because the two characters *xx* must be entered as part of the START command, they must be alphanumeric—no special or DBCS characters.

Adding your own columns: All future column names defined by IBM appear as RLF*xxxxx*. To avoid future naming conflicts, begin your own column names with characters other than RLF.

Index name: The *xx* in the index name (DSNARL*xx*) must match the *xx* in the table name (DSNRLST*xx*), and the index must be a descending index.

Descriptions of the RLST columns

This section contains a complete description for all RLST columns. In no case will all columns in a particular row be populated; the columns you must populate depend on the function being performed by that row (determined by the value in RLFFUNC) and how narrowly you want to qualify values. For example, you can qualify broadly by leaving the AUTHID column blank, which means that the row applies to all authorization IDs. Or, you can qualify very narrowly by specifying a different row for each authorization ID for which the function applies.

AUTHID

The resource specification limits apply to this primary authorization ID. A blank means that the limit specifications in this row apply to all authorization IDs for the location that is specified in LUNAME.

PLANNAME

The resource specification limits apply to this plan. If you are specifying a function that applies to plans (RLFFUNC=blank or '6'), a blank means that the limit specifications in this row apply to all plans for the location that is specified in LUNAME. When dynamic statements are issued from a remote location to access data at your local location using DB2 private protocol, they are governed by a row in the RLST with RLFFUNC=blank and the LUNAME of the originating location first, PUBLIC second.

Qualify by plan name only if the dynamic statement is issued from a DBRM bound in a plan, not a package; otherwise, DB2 does not find this row. If the RLFFUNC column contains a function for packages ('1,' '2,' or '7'), then this column must be blank; if it is not blank, the row is ignored.

ASUTIME

The number of processor service units allowed for any single dynamic SELECT, INSERT, UPDATE, or DELETE statement. Use this column for reactive governing. A null value in this column means that *no limit* exists. A zero or a negative value means that *no* dynamic SELECT, INSERT, UPDATE, or DELETE statements are permitted.

Service units are independent of processor changes. The processing time for a particular SQL statement varies according to the processor on which it is executed, but the required service units remain constant. The governor samples the processing time in service units.

LUNAME

The LU name of the location where the request originated. A blank value in this column represents the local location, *not* all locations. The value PUBLIC represents all of the DBMS locations in the network; these locations do not need to be DB2 subsystems.

RLFFUNC

Specifies how the row is used:

- Blank means that the row governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by plan name.
- '1' means that the row governs bind operations.
- '2' means that the row governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by package or collection name.
- '3' means that the row disables query I/O parallelism.
- '4' means that the row disables query CP parallelism.
- '5' means that the row disables Sysplex query parallelism.
- '6' means that the row predictively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by plan name.
- '7' means that the row predictively governs dynamic SELECT, INSERT, UPDATE, or DELETE statements by package or collection name.

- All other values are ignored.

RLFBIND
Shows whether bind operations are allowed. An 'N' implies that bind operations are not allowed. Any other value means that bind operations are allowed. This column is used only if RLFFUNC is set to '1'.

RLFCOLLN
Specifies a package collection. A blank value in this column means that the row applies to all package collections from the location that is specified in LUNAME. Qualify by collection name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1', or '6', then RLFCOLLN must be blank.

RLFPKG
Specifies a package name. A blank value in this column means that the row applies to all packages from the that is location specified in LUNAME. Qualify by package name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1', or '6', then RLFPKG must be blank.

RLFASUERR
Used for predictive governing (RLFFUNC= '6' or '7'), and only for statements that are in cost category A. The error threshold number of system resource manager processor service units allowed for a single dynamic SELECT, INSERT, UPDATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the error threshold, an SQLCODE -495 is returned to the application.

Other possible values and their effects are:

null No error threshold

0 (zero) or a negative value
All dynamic SELECT, INSERT, UPDATE, or DELETE statements receive SQLCODE -495.

RLFASUWARN
Used for predictive governing (RELFFUNC= '6' or '7'), and only for statements that are in cost category A. The warning threshold number of processor service units that are allowed for a single dynamic SELECT, INSERT, UPDATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the warning threshold, an SQLCODE +495 is returned to the application.

Other possible values and their effects are:

null	No warning threshold
0 (zero) or a negative value	All dynamic SELECT, INSERT, UPDATE, or DELETE statements receive SQLCODE +495.

Important: Make sure the value for RLFASUWARN is less than that for RLFASUERR. If the warning value is higher, the warning is never reported. The error takes precedence over the warning.

RLF_CATEGORY_B
Used for predictive governing (RLFFUNC= '6' or '7'). Tells the governor the default action to take when the cost estimate for a given statement falls into cost category B, which means that the predicted cost is indeterminate

and probably too low. You can tell if a statement is in cost category B by running EXPLAIN and checking the COST_CATEGORY column of the DSN_STATEMNT_TABLE.

The acceptable values are:

' ' (blank)
By default, execute the SQL statement.

Y Prepare and execute the SQL statement.

N Do not prepare or execute the SQL statement. Return SQLCODE -495 to the application.

W Complete the prepare, return SQLCODE +495, and allow the application logic to decide whether to execute the SQL statement or not.

Combining reactive and predictive governing

A dynamic statement can be governed both before and after the statement is executed. For example, if the processing cost estimate is in cost category B and you decide to run the statement, you can use the RLST to terminate the statement after a certain amount of processor time, the same as it does today. To use both modes of governing, you need two rows in the RLST as shown in Table 6.

Table 6. Combining reactive and predictive governing

RLFFUNC	AUTHID	PLANNAME	ASUTIME	RLFASUWARN	RLFASUERR	RLF_CATEGORY_B
6	USER1	PLANA	0	800	1000	W
(blank)	USER1	PLANA	1100	0	0	(blank)

The rows in the RLST for this example cause DB2 to act as follows for a dynamic SQL statement that runs under PLANA:

Predictive mode:

- If the statement is in COST_CATEGORY A and the cost estimate is greater than 1000 SUs, USER1 receives SQLCODE -495 and the statement is not executed.
- If the statement is in COST_CATEGORY A and the cost estimate is greater than 800 SUs but less than 1000 SUs, USER1 receives SQLCODE +495.
- If the statement is in COST_CATEGORY B, USER1 receives SQLCODE +495.

Reactive mode: In either of the following cases, a statement is limited to 1100 SUs:

- The cost estimate for a statement in COST_CATEGORY A is less than 800 SUs
- The statement is in COST_CATEGORY A or COST_CATEGORY B and the user chooses to execute the statement

If the cost estimate for a COST_CATEGORY A dynamic SQL statement run under PLANA is less than 800 SUs, or if USER1 decides to execute the COST_CATEGORY A or COST_CATEGORY B statement after receiving SQLCODE +495, the statement that runs under PLANA is limited to 1100 SUs.

Writing an application to handle predictive governing

If your installation uses predictive governing, you need to modify your applications to check for the +495 and -495 SQLCODEs that predictive governing can generate after a PREPARE statement executes. The +495 SQLCODE in combination with deferred prepare requires that DB2 do some special processing to ensure that existing applications are not affected by this new warning SQLCODE.

Handling the +495 SQLCODE: If your requester uses deferred prepare, the presence of parameter markers determines when the application receives the +495 SQLCODE. When parameter markers are present, DB2 cannot do PREPARE, OPEN, and FETCH processing in one message. If SQLCODE +495 is returned, no OPEN or FETCH processing occurs until your application requests it.

- If there are parameter markers, the +495 is returned on the OPEN (not the PREPARE).
- If there are no parameter markers, the +495 is returned on the PREPARE.

Normally with deferred prepare, the PREPARE, OPEN, and first FETCH of the data are returned to the requester. For a predictive governor warning of +495, you would ideally like to have the option to choose beforehand whether you want the OPEN and FETCH of the data to occur. For downlevel requesters, you do not have this option. The level of DRDA that fully supports predictive governing is DRDA Level 4.

The products that include DRDA support for predictive governing are DB2 for OS/390 Version 6 and DB2 Connect Version 5.2 with appropriate maintenance. All other requesters are considered downlevel with regards to predictive governing support through DRDA.

For downlevel requesters: If SQLCODE +495 is returned to the requester, OPEN processing continues but the first block of data is not returned with the OPEN. Thus, if your application does not continue with the query, you have already incurred the performance cost of OPEN processing.

For enabled requesters: If your application does not defer the prepare, SQLCODE +495 is returned to the requester and OPEN processing does not occur.

If your application does defer prepare processing, the application receives the +495 at its usual time (OPEN or PREPARE). If you have parameter markers with deferred prepare, you receive the +495 at OPEN time as you normally do. However, an additional message is exchanged.

Recommendation: Do not use deferred prepare for applications that use parameter markers and that are predictively governed at the server side.

Statement cost estimation

You can use EXPLAIN to populate a statement table, *owner*.DSN_STATEMNT_TABLE, at the same time as your PLAN_TABLE is being populated. DB2 provides cost estimates, in service units and in milliseconds, for SELECT, INSERT, UPDATE, and DELETE statements, both static and dynamic. The estimates do not take into account several factors, including cost adjustments that are caused by parallel processing, or the use of triggers or user-defined functions.

Use the information that is provided in the statement table to:

- Help you determine if a statement is not going to perform within range of your service-level agreements and to tune accordingly.

  DB2 puts its cost estimate into one of two *cost categories*: category A or category B. Estimates that go into cost category A are the ones for which DB2 has adequate information to make an estimate. That estimate is not likely to be 100% accurate, but is likely to be more accurate than any estimate that is in cost category B. For information on what goes into cost category B, see What goes into cost category B? on page 82.
- Give a system programmer a basis for entering service-unit values by which to govern dynamic statements.

This section describes the following tasks to obtain and use cost estimate information from EXPLAIN:

1. "Creating a statement table"
2. "Populating and maintaining a statement table" on page 82
3. "Retrieving rows from a statement table" on page 82
4. "Understanding the implications of cost categories" on page 82

Creating a statement table

To collect information about a statement's estimated cost, create a table called DSN_STATEMNT_TABLE to hold the results of EXPLAIN. A copy of the statements that are needed to create the table are in the DB2 sample library, under the member name DSNTESC.

Figure 15 shows the format of a statement table.

```
CREATE TABLE DSN_STATEMNT_TABLE
  ( QUERYNO            INTEGER      NOT NULL WITH DEFAULT,
    APPLNAME           CHAR(8)      NOT NULL WITH DEFAULT,
    PROGNAME           CHAR(8)      NOT NULL WITH DEFAULT,
    COLLID             CHAR(18)     NOT NULL WITH DEFAULT,
    GROUP_MEMBER       CHAR(8)      NOT NULL WITH DEFAULT,
    EXPLAIN_TIME       TIMESTAMP    NOT NULL WITH DEFAULT,
    STMT_TYPE          CHAR(6)      NOT NULL WITH DEFAULT,
    COST_CATEGORY      CHAR(1)      NOT NULL WITH DEFAULT,
    PROCMS             INTEGER      NOT NULL WITH DEFAULT,
    PROCSU             INTEGER      NOT NULL WITH DEFAULT,
    REASON             VARCHAR(254) NOT NULL) WITH DEFAULT;
```

Figure 15. Format of DSN_STATEMNT_TABLE

Table 7 shows the content of each column. The first five columns of the DSN_STATEMNT_TABLE are the same as PLAN_TABLE.

Table 7 (Page 1 of 2). Descriptions of columns in DSN_STATEMNT_TABLE

Column Name	Description
QUERYNO	A number that identifies the statement being explained. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique.
APPLNAME	The name of the application plan for the row, or blank.
PROGNAME	The name of the program or package containing the statement being explained, or blank.

Table 7 (Page 2 of 2). Descriptions of columns in DSN_STATEMNT_TABLE

Column Name	Description
COLLID	The collection ID for the package, or blank.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN, or blank.
EXPLAIN_TIME	The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.
STMT_TYPE	The type of statement being explained. Possible values are: **SELECT** SELECT **INSERT** INSERT **UPDATE** UPDATE **DELETE** DELETE **SELUPD** SELECT with FOR UPDATE OF **DELCUR** DELETE WHERE CURRENT OF CURSOR **UPDCUR** UPDATE WHERE CURRENT OF CURSOR
COST_CATEGORY	Indicates if DB2 was forced to use default values when making its estimates. Possible values: **A** Indicates that DB2 had enough information to make a cost estimate without using default values. **B** Indicates that some condition exists for which DB2 was forced to use default values. See the values in REASON to determine why DB2 was unable to put this estimate in cost category A.
PROCMS	The estimated processor cost, in milliseconds, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 milliseconds, which is equivalent to approximately 24.8 days. If the estimated value exceeds this maximum, the maximum value is reported.
PROCSU	The estimated processor cost, in service units, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 service units. If the estimated value exceeds this maximum, the maximum value is reported.
REASON	A string that indicates the reasons for putting an estimate into cost category B. **HOST VARIABLES** The statement uses host variables, parameter markers, or special registers. **TABLE CARDINALITY** The cardinality statistics are missing for one or more of the tables that are used in the statement. **UDF** The statement uses user-defined functions. **TRIGGERS** Triggers are defined on the target table of an INSERT, UPDATE, or DELETE statement. **REFERENTIAL CONSTRAINTS** Referential constraints of the type CASCADE or SET NULL exist on the target table of a DELETE statement.

Populating and maintaining a statement table

You populate a statement table at the same time as you populate the corresponding plan table.

Just as with the plan table, DB2 just adds rows to the statement table; it does not automatically delete rows. To clear the table of obsolete rows, use DELETE, just as you would for deleting rows from any table. You can also use DROP TABLE to drop a statement table completely.

Retrieving rows from a statement table

To retrieve all rows in a statement table, you can use a query like the following statement, which retrieves all rows about the statement that is represented by query number 13:

```
SELECT * FROM JOE.DSN_STATEMNT_TABLE
   WHERE QUERYNO = 13;
```

The QUERYNO, APPLNAME, PROGNAME, COLLID, and EXPLAIN_TIME columns contain the same values as corresponding columns of PLAN_TABLE for a given plan. You can use these columns to join the plan table and statement table:

```
SELECT A.*, PROCMS, COST_CATEGORY
 FROM JOE.PLAN_TABLE A, JOE.DSN_STATEMNT_TABLE B
   WHERE A.APPLNAME = 'APPL1' AND
   A.APPLNAME = B.APPLNAME AND
   A.PROGNAME = B.PROGNAME AND
   A.COLLID   = B.COLLID AND
   A.BIND_TIME = B.EXPLAIN_TIME
ORDER BY A.QUERYNO, A.QBLOCKNO, A.PLANNO, A.MIXOPSEQ;
```

Understanding the implications of cost categories

Cost categories are DB2's way of differentiating estimates for which adequate information is available from those for which it is not. You probably wouldn't want to spend a lot of time tuning a query based on estimates that are returned in cost category B, because the actual cost could be radically different based on such things as what value is in a host variable, or how many levels of nested triggers and user-defined functions exist.

Similarly, if system administrators use these estimates as input into the resource limit specification table for governing (either predictive or reactive), they probably would want to give much greater latitude for statements in cost category B than for those in cost category A.

Because of the uncertainty involved, category B statements are also good candidates for reactive governing.

What goes into cost category B? DB2 puts a statement's estimate into cost category B when any of the following conditions exist:

- The statement has user-defined functions.
- Triggers are defined for the target table:
  - The statement is INSERT, and insert triggers are defined on the target table.
  - The statement is UPDATE, and update triggers are defined on the target table.

  - The statement is DELETE, and delete triggers are defined on the target table.
- The target table of a DELETE statement has referential constraints that are defined on it as the parent table, and the delete rules are either CASCADE or SET NULL.
- The WHERE clause predicate has one of the following forms:
  - COL op literal, and the literal is a host variable, parameter marker, or special register. The operator can be >, >=, <, <=, LIKE, or NOT LIKE.
  - COL BETWEEN literal AND literal where either literal is a host variable, parameter marker, or special register
  - LIKE with an escape clause that contains a host variable
- The cardinality statistics are missing for one or more tables that are used in the statement.

What goes into cost category A? DB2 puts everything that doesn't fall into category B into category A.

Set default buffer pools

Previously: When you created a table space or index without specifying a buffer pool, the object went into buffer pool 0 (BP0). BP0 is rarely the best buffer pool in which to put user data and indexes, because BP0 is always used by the catalog and directory. Performance monitoring and tuning for user data and indexes is difficult when all of those objects are in the same buffer pool.

Now: On installation panel DSNTIP1, you can specify a default buffer pool for user data and a default buffer pool for user indexes.

On CREATE DATABASE, you can specify the default buffer pool for table spaces in that database by using the BUFFERPOOL clause. Now in Version 6, you can also specify the default buffer pool for indexes by using the INDEXBP clause. If you do not specify anything for these values, the installation-specified defaults are used. See *DB2 SQL Reference* for more information about the syntax.

More information available for monitoring DB2

To help you more effectively monitor DB2 performance, several enhancements are added to DB2; these are described in the following sections:

- "Wait times reported more clearly" on page 84
- "Data set I/O statistics in a trace" on page 84
- "Message for lock escalation" on page 84
- "New IFCID for active log shortage" on page 85
- # "Easier monitoring of data sharing groups" on page 85

Wait times reported more clearly

In Version 6, it is easier to pinpoint where and why wait times occur.

Better reporting of some DB2 wait times in IFC traces: Before Version 6, some wait times were reported all together, determining which activities were contributing to the total wait time was difficult.

Now, the following changes are in the accounting class 3 trace:

- The “wait for I/O” field is separated into two separate fields:
  - Wait for log write I/O
  - Wait for database I/O
- The “wait for service task” field is separated into the following fields:
  - Wait for commit phase 2 or abort
  - Wait for open or close service task (including HSM recall)
  - Wait for SYSLGRNX recording service task
  - Wait for data set extend, delete, or define service task
  - Wait for other service tasks

Better reporting of DB2 delays in SMF 72 records: Before Version 6, all in DB2 time was reported as part of the calling region's other product time in an RMF workload activity report. Determining what portion of the delay was caused by DB2 was difficult.

Now, DB2 reports the following delays in the SMF 72 record::

- I/O suspension
- Acquiring a lock or latch
- Queuing for stored procedures execution

By reporting this information in SMF 72 records, performance monitors, including RMF, can report this time as a subset of the total time from the calling region's point of view.

Data set I/O statistics in a trace

Previously: Important information about DB2's use of data sets was available only through the DISPLAY BUFFERPOOL command. Such information was not available to performance monitors.

Now: You can use a new IFCID, 0199, to help you pinpoint I/O delays for particular data sets. The IFCID is activated with statistics class 8 or monitor class 1 traces. The IFCID can let you answer such questions as “what is the average write I/O delay for this data set over the past 10 minutes?”

If you want to use an online performance monitor to gather this data, it is helpful gather statistics frequently. You can use the DATASET STATS TIME field of installation panel DSNTIPN to indicate how often you want those statistics to be gathered for an online performance monitor. The default value is 5 minutes.

Message for lock escalation

Previously: When lock escalation occurred for a table space or a table of a segmented table space, no external notification was made.

Now: A new message, DSNI03I, is issued when a lock is escalated.

```
DSNI031I - csect - LOCK ESCALATION HAS OCCURRED FOR
RESOURCE NAME             =  name
LOCK STATE                =  state
PLAN NAME : PACKAGE NAME  =  id1 : id2
STATEMENT NUMBER          =  id3
CORRELATION-ID            =  id4
CONNECTION-ID             =  id5
LUW-ID                    =  id6
THREAD-INFO               =  id7 : id8 : id9 : id10
```

New IFCID for active log shortage

Previously: The console messages indicating that active log data sets are running out of space were not available to online performance monitors.

Now: A new IFCID, 0330 (statistics class 3), is written at the same time the existing DSNJ110E message is issued. This means that IFCID 0330 is written when the last available active log data set is 5% full, and after each additional 5% of the data set space is filled.

Easier monitoring of data sharing groups

DB2 simplifies monitoring of data sharing groups by letting you can collect trace data for an entire data sharing group or one member of a data sharing group using READS or READA calls in an IFI monitor program. DB2 returns the trace data for all members to the return area of the IFI program. For details on how to collect trace data for a data sharing group in an IFI program, see Appendixes (Volume 2) of *DB2 Administration Guide*.

IRLM enhancements

DB2's internal resource lock manager (IRLM) has been significantly improved to provide better serviceability, availability, and usability. The current version of IRLM works with all releases of DB2 beginning with Version 4. All changes here apply to all those versions of DB2:

- "Display IRLM coexistence information"
- "Option to prevent disconnecting IRLM on DB2 shutdown" on page 86
- "More control over IRLM storage" on page 87
- "Support for automatic restart manager" on page 89
- "Improved serviceability" on page 89

Performance enhancements related to IRLM are described in "Reduced P-lock overhead" on page 51 and "More command concurrency" on page 64.

Display IRLM coexistence information

This enhancement is delivered with APAR PQ09947 and makes it easier for you to determine what level of service each of the IRLMs in your data sharing group is at. (This command also works in non-data-sharing environment.)

IRLM function levels

In a data sharing group, you can have two releases of DB2 running at the same time. With IRLM, however, coexistence levels are determined using *function levels*. A function level is an ever-increasing number that each IRLM can use to tell other IRLMs in the group what level of function it supports. The group function level is the minimum of the individual IRLM function levels for all IRLMs that can coexist. Any IRLM that tries to join a data sharing group is prevented from doing so by active members that cannot coexist with the new IRLM's function level.

When the function level for the group changes, that change is serialized by IRLM with lock structure rebuilds. In most cases, however, the lock structure does not actually do a full rebuild. The first phase of rebuild is enough to quiesce the work and cause the function level change to occur. These “partial” rebuilds take place when an IRLM joins or leaves the group and if that activity causes the group function level to change. For example, if the IRLM group is currently at function level n, and the IRLM member that wants to join the group is at n-1, the partial rebuild occurs to lower the group function level. Conversely, if the lowest-level member leaves the group, the partial rebuild might occur if the group can coexist at a higher function level.

Command to display function levels

To display the function levels, enter the following command:

```
MODIFY irlmproc,STATUS,ALLI
```

You get output like that shown here:

```
DXR103I LRLM STATUS IRLMID=007
IRLMS PARTICIPATING IN DATA SHARING GROUP FUNCTION LEVEL=013
IRLM_NAME IRLMID STATUS LEVEL SERVICE MIN_LEVEL MIN_SERVICE
  JRLM      005    UP    015  PQ15290    012      PN90337
  KRLM      006    UP    013  PN92893    006      IRLM2.1
  LRLM      007    UP    014  PN09381    006      IRLM2.1
```

The IRLMs are at group function level 13, which is the lowest level of any of the individual members (KRLM). The MIN_LEVEL field shows the minimum level with which this IRLM can coexist. MIN_SERVICE indicates the service or release that corresponds with that MIN_LEVEL.

Option to prevent disconnecting IRLM on DB2 shutdown

This enhancement is delivered with APAR PQ01040. The new option for the SCOPE parameter, NODISCON, works similar to SCOPE=GLOBAL, except that IRLM remains connected to the data sharing group even when DB2 is stopped (either normally or as a result of a failure). If you want to stop IRLM, you must explicitly stop it.

By leaving IRLM connected when you apply maintenance to DB2, it can make the restart of DB2 quicker. Also, less impact on other systems results when a DB2 fails because MVS is not required to perform certain recovery actions that it normally performs when IRLM comes down.

How to specify the new option

You can specify the NODISCON option in two ways:

- By editing your IRLM start-up procedure and specifying NODISCON for the SCOPE parameter
- By specifying NO for the DISCONNECT IRLM field of installation panel DSNTIPJ

Change to AUTO START field of DSNTIPI

If you specify YES for the AUTO START field of installation panel DSNTIPI and NO for the DISCONNECT IRLM field, DB2 does not automatically stop IRLM when DB2 is stopped. However, if you say YES for DISCONNECT IRLM and YES for AUTO START, DB2 stops IRLM when DB2 terminates.

To summarize:

Table 8. Summary of IRLM actions when DB2 stops, starts, or restarts

DISCONNECT IRLM option	AUTO START option	IRLM action on DB2 stop	IRLM action on DB2 start	IRLM action on DB2 restart
YES	YES	IRLM disconnects from the group and stops itself.	DB2 starts IRLM.	DB2 restarts IRLM.
YES	NO	IRLM disconnects from the group and remains active.	Manual start of IRLM is required.	IRLM rejoins the data sharing group.
NO	NO	IRLM remains active and connected.	Manual start of IRLM is required.	No effect. (IRLM is still active and connected to the data sharing group.)
NO	YES	IRLM remains active and connected.	DB2 starts IRLM.	No effect. (IRLM is still active and connected to the data sharing group.)

More control over IRLM storage

As part of APAR PQ12390, IRLM is better at handling its use of storage by releasing more ECSA storage back to the system when it is no longer needed. Additionally, the value you specify for MAXCSA is now used just for lock control structures, not for the rest of the work space needed by IRLM. This makes it easier to determine what value to specify for MAXCSA.

Other enhancements related to managing IRLM storage are described in this section.

Warning when storage exceeds limits

IRLM processing is critical to the smooth performance of DB2. To alert you when IRLM is having storage problems, IRLM now issues a message when it does not have enough storage:

```
DXR175E irlmnm IRLM IS UNABLE TO OBTAIN STORAGE - storage_type
```

The storage type indicates what type of storage IRLM is short of. For example, MCSA indicates that IRLM has exceeded the MAXCSA value you've specified in the IRLM startup procedure.

This enhancement is delivered in APAR PQ07327.

Dynamically change MAXCSA

This enhancement is delivered in APAR PQ12126 and lets you use a command to dynamically change the value for MAXCSA. (The upper limit for MAXCSA has been raised to 999 MB with APAR PQ13498). Use the following command to change the MAXCSA value:

```
MODIFY irlmproc,SET,CSA=nnn
```

where *nnn* is a value in MB up to 999.

The values you set on a MODIFY,SET command do not persist through a restart of IRLM. If you want to make this change permanent, you must modify the IRLM start-up procedure to use the new value, and then stop and restart IRLM.

Better display of IRLM storage use

This enhancement is delivered in APAR PQ12390 and gives you the ability to better track IRLM storage use, including the current storage allocation and the "high-water" storage allocation since the last time this IRLM was started. The command is:

```
F IR21PROC,STATUS,STOR
```

The display is similar to the following output:

```
DXR1001 IR21 STOR STATS
      A  PC: NO    MAXCSA:    6M
         CSA USE:  B  ACNT:    132K  AHWM:    132K  C  CUR:  4048K  HWM:  4086K
        ABOVE 16M:     72  4033K    BELOW 16M:      6     15K
CLASS   TYPE  SEGS     MEM   TYPE  SEGS     MEM   TYPE  SEGS     MEM
ACCNT   T-1      1     64K   T-2      1     64K   T-3      1      4K
PROC    WRK     11     58K   SRB      3      3K   OTH      2      2K
MISC    VAR     60   4081K   N-V      6     22K   FIX      1     24K
```

The example shows that current allocated storage for IRLM is 4048 KB, and the greatest amount that has been allocated since the last time IRLM was started is 4086 KB. The storage for the locking structures is contained within ECSA, because this IRLM is defined with PC=NO. Use the following information to interpret the display output:

A Displays the current value for the PC and MAXCSA options of the IRLM start-up procedure.

B Shows storage use that is accountable toward the MAXCSA value of the IRLM procedure. In this output, the currently used accountable storage (ACNT) is 132 KB. The high-water mark since the last time IRLM was started (AHWM) is also 132 KB.

C Shows the total current CSA and ECSA usage. In this case, the current usage (CUR) is 4048 KB, and the high water mark (HWM) is 4086 KB. The accountable storage is a subset of this total storage.

The rest of the output contains more detailed information about how storage is used.

Support for automatic restart manager

This enhancement is delivered in APAR PQ06465. With this enhancement, you now have the option of registering IRLM with MVS's automatic restart manager, just as you can do with DB2. This automatic restart only happens when IRLM abnormally shuts down.

Creating the automatic restart policy

To register IRLM in an automatic restart policy, you must know IRLMs *element name*. For a data sharing IRLM, the element name is the IRLM group name, concatenated with the IRLM subsystem name and three-character member ID (such as DXRDB0GDJ1G001).

For a non-data-sharing IRLM, the element name is the IRLM subsystem name, concatenated with the IRLM ID (such as IRLX001).

To specify that IRLM is not to be restarted after a failure, include RESTART_ATTEMPTS(0) in the policy for that IRLM element.

Stopping and deregistering IRLM

To prevent IRLM from automatically restarting when you stop it, use the following command to stop IRLM:

```
MODIFY irlmproc,ABEND,NODUMP
```

However, if your DB2 has YES for the AUTO START option of installation panel DSNTIPI, and if MVS restarts DB2 automatically, DB2 restarts IRLM, too.

Improved serviceability

As a result of improvements to the serviceability of IRLM, problems can be resolved more quickly.

Dynamically control number of trace buffers

This enhancement is delivered in APAR PQ12126 and lets you increase the storage used for IRLM diagnostic traces. By increasing the storage when doing problem diagnosis, you reduce the chances that important trace information will be lost because of wrapped trace entries. To change the storage, use the following command:

```
MODIFY irlmproc,SET,TRACE=nnn
```

where *nnn* is the number of 64 KB trace buffers per trace type. This value is used only when the external CTRACE writer is not activated. The trace buffers are allocated from ECSA. The default is 10.

Valid values are from 10 through 255. If you specify a value outside of this range, IRLM automatically adjusts the value to fall within the range.

IRLM does not immediately acquire the number of trace buffers you set. IRLM allocates buffers as needed, not to exceed the number of buffers you set in this command. If the number of trace buffers you set is less than the number of currently allocated buffers, IRLM brings the number within your specified range by releasing the oldest buffers at the end of the next deadlock or timeout cycle.

Display trace buffer storage using the following command:

```
MODIFY irlmproc,STATUS,TRACE
```

A display similar to the following is produced:

```
DXR179I PR21034 TRACE USAGE
TRACE BUFFER STORAGE IN USE:   256KB
MAXIMUM NUMBER OF TRACE BUFFERS ALLOWED PER TRACE TYPE:  10
TRACE TYPE   ACTIVE   BUFFERS IN USE   CTRACE WRITER
----------   ------   --------------   -------------
   SLM          N            0              N
   XIT          Y            2              N
   XCF          N            0              N
   DBM          N            0              N
   EXP          Y            1              N
   INT          Y            1              N
```

This display shows that the storage currently allocated for IRLM tracing is 256 KB, the maximum number of trace buffers allowed per trace type is set to 10, and the external CTRACE writer is not active.

Prevent unnecessary dumps

This enhancement is delivered in APAR PQ08342 and prevents generation of unnecessary dumps related to delayed child-lock propagation in a data sharing group. Because many of the reasons for this type of delay are outside the scope of IRLM's control (such as member recovery or lost connectivity to the lock structure), a dump is not necessarily the first course of action. IRLM does retry propagating the locks if it detects a delays.

If you suspect a problem with lock propagation in your data sharing group, you can enter a new command:

```
MODIFY irlmproc,DIAG,DELAY
```

This command initiates diagnostic dumps for IRLM subsystems in a data sharing group when responses to XES requests take longer than 45 seconds.

This command is active for only one incident; that is, after an IRLM detects the delay and generates the dump, you must reenter the command to initiate another dump. However, when you enter this command for one member of the data sharing group, *any* member that detects the delay initiates a dump.

IRLM ID in messages

This enhancement is delivered in APAR PQ14255. Some IRLM messages only contain the IRLM name. For Sysplexes in which multiple IRLMs have the same name, the IRLM ID is the only way to tell which IRLM is displaying the message. With this enhancement, the IRLM ID is concatenated to the IRLM name in messages.

For example, message DXR132I now displays as follows:

```
DXR132I JR21003 SUCCESSFULLY JOINED THE DATA SHARING GROUP.  GLOBAL
       INITIALIZATION IS COMPLETE
```

In the message, JR21 is the IRLM name, and 003 is the IRLM ID.

More user productivity enhancements

- "DSNTEP2 available in object form"
- "Customized DB2I defaults can be migrated"
- "Numeric data type extensions for identity columns"
- "Savepoints to undo selected changes" on page 93
- "More tables allowed in SQL statements" on page 94
- "SQL extensions" on page 94
- "More character conversions" on page 95
- "Utility usability and functionality enhancements" on page 95
- "Enhanced database commands" on page 97
- "Support for multi-volume DASD archive log data sets" on page 98
- "Better retention of installation values across migrations" on page 98
- "Better diagnostic information for commands executed through IFI" on page 99

DSNTEP2 available in object form

Previously: The DSNTEP2 sample program, which lets you execute dynamic SQL in batch mode, was useful only if your site had a PL/I compiler.

Now: An object form of DSNTEP2 is shipped with DB2, allowing this useful productivity aid to be used at all installations. Use a new sample program, DSNTEJ1L, to link-edit this object deck with the OS/390 Language Environment® to create an executable load module and to bind a package and plan from the shipped DBRM.

Customized DB2I defaults can be migrated

The DB2I panel variables in the ISPF profile from the previous release can be migrated to the new release. The DSNEMC01 CLIST uses the values specified on installation panel DSNTIPF and stores the results in the ISPF profile member DSNEPROF. Any customized DSNEPROF members can be migrated from Version 5 to Version 6. However, you need to examine any new or changed default panel values to ensure that your customized values are still valid.

Numeric data type extensions for identity columns

DB2 provides new support for defining a table with an *identity column*. When a table has an identity column, DB2 can automatically generate a unique, sequential numeric value for each row that is inserted into the table. Therefore, identity columns are ideal for primary keys. For example, you might want to use identity columns for order numbers, employee numbers, stock numbers, or incident

numbers. Using identity columns to generate unique numbers in a column can reduce concurrency and performance problems that sometimes occur when application programs generate their own unique numbers.

A table can have no more than one identity column and only a column with a SMALLINT, INTEGER, or DECIMAL (with scale zero) can be specified as the table's identity column. You can specify the first value that DB2 should assign to an identity column and the incremental value (the difference) between generated values. For example, you might want the interval between order numbers to be 100, starting with 1000 as the first number (1000, 1100, 1200, and so on). An identity column cannot be defined as nullable, with a default, or with a field procedure. A table with an identity column cannot have an edit procedure.

Defining an identity column: Create an identity column for a table as you would any other column, using the ALTER TABLE or CREATE TABLE statement and specify the AS IDENTITY clause. Similar to a ROWID column, there are two options for how DB2 should generate identity column values: the GENERATED ALWAYS or GENERATED BY DEFAULT clause.

- GENERATED ALWAYS specifies that DB2 always generates a unique number for the column when a row is inserted into the table. You also cannot update the value of an identity column that is defined as GENERATED ALWAYS.
- GENERATED BY DEFAULT, which is recommended when you propagate or load data from another data source, specifies that DB2 generates a value for the column when a row is inserted into the table unless a user value is specified. DB2 can only guarantee uniqueness among the set of values that it generates. If you want to ensure uniqueness for an identity column that is defined as GENERATED BY DEFAULT, define a unique index on the identity column.

The following example shows how to create a table with an identity column and define the identity column such that DB2 will always generate the values for the column. If the AS IDENTITY clause had been specified without the optional START WITH and INCREMENT BY subclauses, DB2 would use the default value of 1 for both the first value to assign to the column and the difference between subsequently assigned values.

```
CREATE TABLE EMPLOYEE
  (EMPNO   INTEGER GENERATED ALWAYS AS IDENTITY START WITH 100 INCREMENT BY 10,
   ID      SMALLINT,
   NAME    CHAR(30),
...
```

Inserting into an identity column Inserting into an identity column is similar to inserting into a ROWID column (see “Inserting into a ROWID column” on page 70). By definition, DB2 always generates a unique, sequential value for an identity column that is defined as GENERATED ALWAYS; therefore, you cannot insert your own value into such a column. To have DB2 successfully insert a generated value, your INSERT statement must do one of the following:

- Not specify a user value for the identity column.
- Use the DEFAULT keyword in the VALUES clause for the identity column
- Include the OVERRIDING USER VALUE clause. When this clause is specified, DB2 ignores any user-specified values and generates a value to insert. If this clause is not specified and a user value is specified, DB2 returns an error.

For example, assume that EMPNO is an identity column that is defined as GENERATED ALWAYS. DB2 generates and inserts a unique, sequential value into the column for either of the following statements:

```
INSERT INTO EMPLOYEE (EMPNO, ID, NAME)
  VALUES (DEFAULT, :hv_id, :hv_name);
```

```
INSERT INTO EMPLOYEE (EMPNO, ID, NAME)
  OVERRIDING USER VALUE
  VALUES (:hv_empno, :hv_id, :hv_name);
```

You can insert your own value into an identity column that is defined as GENERATED BY DEFAULT. DB2 generates the value only if a user value is not specified.

Identity columns versus ROWID columns: Both identity columns and ROWID columns contain values that DB2 generates and guarantees as unique. ROWID columns, which are described in "New ROWID data type" on page 67, are used in large object (LOB) table spaces and can be useful in direct-row access. ROWID columns contain values of the ROWID data type, which returns a 40-byte VARCHAR value that is not regularly ascending or descending. ROWID data values are therefore not well suited to many application uses, such as generating employee numbers or product numbers. For data that is not LOB data and that does not require direct-row access, identity columns are usually a better approach, because identity columns contain existing numeric data types and can be used in a wide variety of uses for which ROWID values would not be suitable.

Savepoints to undo selected changes

Savepoints let you undo selected changes within a transaction. A savepoint represents the state of data at some particular time during a unit of work. An application process can set savepoints within a unit of work, and then as logic dictates, roll back only the changes that were made after a savepoint was set. For example, part of a reservation transaction might involve booking an airline flight and then a hotel room. If a flight gets reserved but a hotel room cannot be reserved, the application process might want to undo the flight reservation without undoing any database changes made in the transaction prior to making the flight reservation. In your programs, you can use the SAVEPOINT statement to set savepoints and the ROLLBACK statement with the TO SAVEPOINT clause to undo changes to a specific savepoint or the last savepoint that was set. Rolling back to a savepoint does not end the unit of work. For example, when the ROLLBACK TO SAVEPOINT statement is executed in the following code, DB2 rolls back work to savepoint B:

```
EXEC SQL SAVEPOINT A;
⋮
EXEC SQL SAVEPOINT B;
⋮
EXEC SQL ROLLBACK TO SAVEPOINT;
```

Savepoints are automatically released at the end of a unit of work. However, if you no longer need a savepoint before the end of a transaction, you can execute the RELEASE SAVEPOINT statement to delete the savepoint.

When savepoints are active, you cannot access remote sites using three-part names or aliases for three-part names. You can, however, use DRDA access with explicit CONNECT statements when savepoints are active. If you set a savepoint before you execute a CONNECT statement, the scope of that savepoint is the local site. If you set a savepoint after you execute the CONNECT statement, the scope of that savepoint is the site to which you are connected.

More tables allowed in SQL statements

DB2 increases the maximum number of tables allowed in a view, and in SELECT, UPDATE, INSERT, and DELETE statements, from 15 to 225. The maximum number of base tables that are allowed in a view, tables in a FROM clause, and subqueries in a statement is 15.

SQL extensions

In addition to the many SQL extensions provided for object support and active data, Version 6 improves DB2 family compatibility and increases the flexibility of SQL with the following changes:

- A VALUES clause of INSERT that supports any expression
- A DEFAULT clause on INSERT that lets you insert default values into columns that allow defaults
- The ability to do a self-referencing subselect on INSERT. For example, you can populate a table by using values from the table itself. The following example populates a table partly from existing values in the table and partly from application values:

```
INSERT INTO TABLEX
  SELECT :hv1, :hv2, COLX, COLY
     FROM TABLEX WHERE .....;
```

- Support for the WHERE and GROUP BY clauses in a SELECT INTO statement
- Support for a subselect in the SET clause of an UPDATE statement. However, the subselect cannot be self-referencing. For example, you can update a table using values from any table except the one being updated.
- A VALUES INTO statement that assigns values to host variables
- An IN predicate that supports any expression
- Support for including SQLDA in a COBOL program on the INCLUDE statement
- A SET assignment statement that you use to assign a value of an expression or the value of null to a host variable or a transition variable; for example:

```
SET :lastmon = CURRENT DATE - 30 DAYS
```

- Extensions to the ON clause, which allow more expressions for LEFT, RIGHT and INNER joins. Here are some examples of statements you can write in Version 6 that you could not write before:

  Example 1: Assume that you have an EMP table with columns for FIRSTNAME, NICKNAME and so on. Assume the SALES table has columns EMPNAME, SALEPRICE, and so on.

  This example shows you how to produce a report of sales in which the first names match the sales record *or* the nickname matches the sales record:

```
SELECT * FROM EMP LEFT JOIN SALES
 ON EMP.FIRSTNAME = SALES.EMPNAME
  OR EMP.NICKNAME = SALES.EMPNAME;
```

Example 2: In addition to the SALES table in Example 1, assume that you have a REGION table with columns REGION_TYPE, REGION_NAME and a PROD table with columns PROD_TYPE and PROD_NAME.

This example shows you how to find the sales data on cola sold in the USA, if any:

```
SELECT * FROM PROD JOIN REGION
  ON 1=1 LEFT JOIN SALES
  ON REGION_TYPE = SALES.REGION
  WHERE REGION_NAME = 'USA' AND PROD_NAME = 'COLA';
```

Example 3: Assume that you have a BONUS table that contains a column called EMPNO.

This example shows you how to produce a report in which only the people whose last names begins with Z get a bonus:

```
SELECT * FROM EMP LEFT JOIN BONUS
 ON EMP.LASTNAME LIKE 'Z%' AND EMP.EMPNO = BONUS.EMPNO;
```

Example 4: In this example, a user-defined function contains the criteria for assigning bonuses:

```
SELECT * FROM EMP LEFT JOIN BONUS
  ON BOSS_OK_UDF(EMP.EMPNO) = 'OKAY'
  AND EMP.EMPNO = BONUS.EMPNO;
```

See Appendix C, “Changes to SQL” on page 269 for more information about these SQL enhancements.

More character conversions

Character data can be represented by different encoding representations, and sometimes character data must be converted from one representation to another. DB2 performs most character conversions based on CCSIDs. If OS/390 Version 2 Release 9 (or a subsequent release) is installed, DB2 can utilize the character conversions provided by Language Environment in addition to those in catalog table SYSIBM.SYSSTRINGS. During character conversion, if SYSIBM.SYSTRINGS does not contain a row that describes the conversion between the source CCSID and the target CCSID, DB2 will try to do the conversion through Language Environment. If the catalog table and Language Environment define the same conversion, DB2 uses the conversion in the catalog table because it looks there first. For a list of the character conversions that Language Environment supports, refer to OS/390 C/C++ Programming Guide.

Utility usability and functionality enhancements

- “Determining when to run REORG” on page 96
- “Easier to quiesce related table spaces” on page 96
- “REPAIR function and usability” on page 96
- “Easier to LOAD test tables and read-only tables” on page 96
- “Improved statistics collection” on page 97

Determining when to run REORG

Previously: You had to perform catalog queries to determine if data needed reorganization.

Now: You can now specify threshold limits in your REORG job, which accesses relevant i8statistics in the catalog. The REORG does not occur unless one of the thresholds you specify is exceeded.

Additionally, you can specify the REPORTONLY option, which prompts DB2 to return information on whether or not a REORG is recommended. When you specify the REPORTONLY option, DB2 does not perform a REORG; if the limits you specified were exceeded, DB2 recommends a REORG.

Easier to quiesce related table spaces

To establish a point of recovery for table spaces that were related by referential integrity, you have to quiesce all table spaces in that set before making the image copies. To quiesce a table space set, you had to do the following steps:

1. Run REPORT TABLESPACESET to find all table spaces in the set
2. Enter those table space names in the QUIESCE SYSIN statement

This procedure was error-prone.

Now, you can use the TABLESPACESET option of QUIESCE to have DB2 quiesce the entire table space set for any particular table space. You no longer have to enter that list of names.

REPAIR function and usability

REPAIR is enhanced in support of the new advisory and restrictive states for LOBs, and to reset level IDs for indexes. (See *DB2 Utility Guide and Reference* for details of the syntax.) REPAIR is also made more usable.

Previously, to indicate which page you wanted to repair, you had to enter a hexadecimal representation of the page and partition number. This process was difficult and error-prone.

Now you can enter a partition as one decimal number and the page within that partition as another decimal number.

Easier to LOAD test tables and read-only tables

The LOAD utility now supports the NOCOPYPEND option, which allows you to avoid making copies for certain tables, such as test tables or read-only tables that can be easily reloaded. Normal completion of a LOAD LOG NO NOCOPYPEND job will be return code 0 if no other errors or warnings exist.

If you want to be able to recover the table space after you specify the NOCOPYPEND option, you should take an inline image copy of the table space. Alternately, you can use the COPY utility after the LOAD job completes.

Improved statistics collection

When you specify the UPDATE SPACE or UPDATE ALL option, the RUNSTATS utility now collects additional space statistics that you can use to estimate the number of data set extents used by objects. These statistics are stored in the SQTY, PQTY, and SECQTYI columns of the SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART catalog tables. Refer to the DSNTESP sample job for sample queries that estimate the number of extents used by objects.

This function is available in Version 5 with APAR PQ25091.

Enhanced database commands

Several improvements to DB2's database commands make identifying the status of databases, table spaces, and index spaces easier. You can also specify the name of a database, table space, or index space using a wildcard character combined with a character string. Additional improvements to the DISPLAY DATABASE command improve your control over the scope of the information displayed.

Improved DSNT736I message content

The STOP DATABASE command now issues a modified DSNT736I message, which includes information about which command completed successfully. This improvement returns specific command syntax in the message, so you can more easily identify which commands completed successfully when you issue multiple STOP DATABASE commands from a console.

Specifying pattern-matching characters within object names

You can now use the pattern-matching character (*) combined with a character string when you specify the name of a database, table space, or index space in the DISPLAY DATABASE, START DATABASE, or STOP DATABASE commands. You can also specify a range of spaces to start, stop, or display with these commands.

Limiting DISPLAY DATABASE output

The DISPLAY DATABASE command accepts the ONLY keyword. When you specify the ONLY keyword without the SPACENAM keyword, DB2 does not display information for the spaces within the database you specified. If you specify the ONLY keyword with the SPACENAM keyword, DB2 displays information for the individual table spaces and indexes. This improvement allows you to narrow the scope of information that is returned when you invoke the DISPLAY DATABASE command.

When you specify the SPACENAM keyword, you can limit the display of table spaces to those that have locks on them, or ones that have write error page range information (WEPR). You can also use the CLAIMERS, USE, or LPL keywords with the ONLY keyword.

When you do not specify the SPACENAM keyword, the ONLY keyword is compatible with the RESTRICT, LIMIT, and AFTER keywords.

Support for multi-volume DASD archive log data sets

Previously: DB2 restricted archive log data sets to a single volume. Because of this restriction, the primary and secondary space quantities must be large to put the entire data set on a single volume. If there is not enough contiguous space available, the log offload fails.

Now: You are no longer restricted to a single volume for archive log data sets. You can now have more, smaller extents when allocating DASD archives, which means a smaller chance of getting "no space available" errors.

Remote site recovery copy flexibility

Previously: In some offsite recovery situations, only the second copy (COPY2) of the archive log is available at the remote site. The BSDS is restored from an archive tape and contains data set information for both COPY1 and COPY2 archives. DB2 always requested the COPY1 archive before the equivalent COPY2 archive. This means that you had to either delete all the unavailable COPY1 archive logs from the BSDS, or you had to reply"N" to the mount requests for the COPY1 archive log volumes, which forced DB2 to request the available COPY2 archive. Either option can lengthen recovery time or be subject to errors.

Now:DB2 has a new subsystem parameter that lets you specify that DB2 is to request COPY2 archive logs for log read requests. In effect, by specifying YES for READ COPY2 ARCHIVE on installation panel DSNTIPO, you are reversing the archive search order. DB2 attempts to satisfy the log read request with the COPY2 archive log and reserves the COPY1 archive logs for error recovery.

Implications for disaster recovery: It is generally recommended that you keep both copies of the archive log at the local site in case the first copy becomes unreadable. However, if you take precautions, such as making another copy of the first archive, you can choose to send the second copy to the recovery site. If a recovery is necessary at the recovery site, make sure you specify YES for the READ COPY2 ARCHIVE field of installation panel DSNTIPO at the recovery site. This option causes DB2 to search for the COPY2 archive first.

Data sharing recommendation: For predictability, make the READ COPY2 ARCHIVE option the same on all members. For any given read request, it is the member that owns the archive log data set that determines which copy is used.

Better retention of installation values across migrations

Previously: Several important subsystem parameter values were not externalized on installation panels. If you changed any of those values in the appropriate macro or DSNTIJUZ job, the new values were not retained when you migrated to a new release.

Now: These options are on installation panels so that you can be assured that the values you choose will be carried over during migration to a new release of DB2. The options that are externalized are shown in Table 25 on page 237.

Better diagnostic information for commands executed through IFI

Previously: When you executed a DB2 command through the Instrumentation Facility Interface (IFI), you received limited information about results from command execution. Those results came only from the DB2 command processor, not from the component that actually executed the command.

Now: The instrumentation facility communication area (IFCA) contains new fields that give you more information about why and when a failure occurred during command execution.

Chapter 5. Improved network computing

In Version 6, DB2 adds a number of enhancements to improve your ability to develop, maintain, and run your e-business and other client/server applications. The following sections contain information about the new functions:

- "Java enablement"
- "DRDA support for three-part names" on page 103
- "Stored procedure enhancements" on page 107
- "Improved data transfer with OPTIMIZE FOR n ROWS" on page 115
- "DB2 ODBC enhancements" on page 118
- "Improvements for dynamically prepared SQL statements" on page 119
- "DB2 database connection pooling" on page 120

Java enablement

Now, you can combine the power of Java with the latest universal database features available in the OS/390 environment. Version 6 introduces IBM's newest Java implementation for the OS/390 environment, *SQLJ*. In Version 5, DB2 for OS/390 delivered JDBC application support, which let you write dynamic SQL applications in Java. With SQLJ support, you can write static SQL applications in Java.

Better performance for Java applications with SQLJ

SQLJ gives you the ability to embed SQL statements in your Java application programs. With SQLJ your Java programs can take advantage of the superior performance, manageability, and authorization that is available to static SQL. SQLJ includes support the following types of SQL statements:

- SELECT, SELECT INTO, or FETCH
- GRANT or REVOKE
- INSERT, searched or positioned UPDATE, searched or positioned DELETE
- COMMIT or ROLLBACK
- CREATE, ALTER, DROP
- CALL, for calls to stored procedures in supported languages
- SET *special register*, SET *host variable*

In addition, SQLJ includes methods for connecting to local DB2 subsystems or remote data sources.

Advantages of using SQLJ include:

- Portable applications across platforms and database management systems
- Strong typing, with compile and bind-time schema checking to ensure that applications are well designed for the database
- The good performance and authorization checking of static SQL

  Static SQL provides control over authorization checking. You cannot manage table privileges for applications using dynamic SQL, because every end user must have table privileges. You can avoid the problem completely by using static SQL. The application uses the table privileges of the package owner.

You prepare and execute SQLJ applications in the OS/390 UNIX environment. To prepare an SQLJ application for execution, you execute these steps:

1. Run the SQLJ translator to produce modified Java source code and serialized profiles.
2. Compile the modified Java source code.
3. Run the SQLJ customizer on each serialized profile to produce a standard DB2 for OS/390 DBRM.
4. Bind the DBRMs for a program into a plan.

Before you can run an SQLJ application, you must specify the DB2 subsystem ID and the plan name in environmental variables. You can then run the program using the command:

```
java
```

JDBC application support

JDBC is a Java application programming interface (API) that Java applications use to access any relational database. DB2 for OS/390's support for JDBC lets you write Java applications that use dynamic SQL to access local DB2 data or remote relational data on a server that supports DRDA.

DB2 JDBC supports the full set of APIs that comprise JDBC. The APIs are defined within 16 classes that support basic SQL functionality for connecting to a database, executing SQL statements, and processing results. These interfaces and classes provide the JDBC capabilities that a Java application uses to access relational data.

DB2 JDBC offers a number of advantages for accessing DB2 data:

- Using the Java language, you can write an application on any platform and execute it on any platform for which the Java Development Kit (JDK) is available.
- JDBC combines the benefit of running your applications in an OS/390 environment with the portability and ease of writing Java applications.
- The ability to develop an application once and execute it anywhere offers the potential benefits of reduced development, maintenance, and systems management costs, and flexibility in supporting diverse hardware and software configurations.
- The JDBC interface offers the ability to change between drivers and access a variety of databases without recoding your Java program.
- JDBC applications do not require precompiles or binds.

A JDBC application performs the following basic steps:

1. Imports the appropriate Java package
2. Loads the appropriate JDBC driver
3. Connects to a database, specifying its location with a URL
4. Passes SQL statements to the database
5. Receives the results
6. Closes the connection

DRDA support for three-part names

DB2 provides two ways to access data at a remote location:

- Through explicit connection to the remote location

  With this technique, you execute a CONNECT statement to connect to the location, and then specify a one- or two-part table name when you execute SQL statements to access data in that table. A one-part name is an unqualified table name. A two-part name is a name of the form *table-owner.table-name*.

- Through implicit connection

  With this technique, you specify a three-part table name when you execute SQL statements to access data in that table. A three-part table name is a name of the form *location.table-owner.table-name*.

In previous releases of DB2 for OS/390, you could use three-part table names only for DB2 private protocol access. With DB2 for OS/390 Version 6, you can use three-part names with DRDA access.

This section discusses the benefits of using DRDA access for your distributed applications, the benefits of coding your DRDA applications using three-part names, and the restrictions on DRDA applications that use three-part names. This section also explains how to prepare applications that use three-part names to use DRDA access and gives you extra information you need to know as you move your existing DB2 private protocol access applications to DRDA access.

Benefits of DRDA access

DRDA access has the following benefits over DB2 private protocol access:

- Access to a broader range of servers

  With DB2 private protocol access, you can access only other DB2 for OS/390 servers. DRDA access lets you access any server that supports DRDA protocols.

- Access to more DB2 function

  Features such as support for TCP/IP, LOBs, and distinct types are available for distributed applications only if you use DRDA access.

- Better performance

  With DB2 private protocol access, DB2 binds statements that execute at the remote server at execution time. With DRDA access, you bind a package at the remote server, so DB2 does less work at execution time.

  In addition, some performance enhancements in recent releases of DB2 are available only for DRDA access. When you switch from DB2 private protocol access to DRDA access, you can take advantage of those enhancements.

Benefits of using three-part names for DRDA access

Using three-part names to access remote data has the following advantages:

- Simpler coding

  Coding a DRDA application to use three-part names is simpler than coding for explicit connections. For example, to access table DSN8610.EMP at locations

SAN_JOSE and LONDON through an explicit connection, you might execute statements like these:

```
EXEC  SQL CONNECT TO SAN_JOSE;
SELECT EMPNO FROM DSN8610.EMP;
EXEC  SQL CONNECT TO LONDON;
SELECT EMPNO FROM DSN8610.EMP;
```

To access the same tables using three-part names, you need to execute only these statements:

```
SELECT EMPNO FROM SAN_JOSE.DSN8610.EMP;
SELECT EMPNO FROM LONDON.DSN8610.EMP;
```

- Coding that is independent of precompiler options

  The SQL statement that you use to reconnect to a remote location depends on precompiler option SQLRULES. However, when you use a three-part name to connect to a location, the coding of your program is independent of the SQLRULES setting because DB2 handles connection management automatically.

- Easy conversion from DB2 private protocol access to DRDA access

  Three-part name support for DRDA access means that you do not need to modify your existing DB2 private protocol access applications to convert them to DRDA access. All you need to do is prepare them, as described in "Preparing applications with three-part names to use DRDA access" on page 105.

- Ability to hop to systems other than DB2 for OS/390 systems

  DB2 lets you connect explicitly to a remote location, and then use a three-part name to connect (hop) to a second remote location. In previous releases of DB2, you could hop to the second location using only DB2 private protocol access. With DRDA access support for three-part names, the second location can be any system that supports DRDA access.

Restrictions on DRDA access programs that use three-part names

DRDA access with three-part names has most of the capabilities of DRDA access with explicit connections. However, there are some restrictions:

- Continuous block fetch is not available.

  Continuous block fetch is available for DB2 private protocol access only. However, you can get the advantages of continuous block fetch for your queries by specifying OPTIMIZE FOR *n* ROWS with a large value of *n*, and by setting installation parameters that cause DB2 to return multiple query blocks on each network transmission. Set those installation parameters through fields EXTRA BLOCKS REQ and EXTRA BLOCKS SRV on installation panel DSNTIP5.

- Recursive hopping is not supported.

  If you perform an explicit connection using DRDA access to access one location and then use DB2 private protocol access to access a second location, you can connect from location A to location B and then connect from location B back to location A. For example:

```
/* Local location is CHICAGO */
EXEC SQL CONNECT TO SAN_JOSE;
EXEC SQL SELECT * FROM CHICAGO.DSN8610.EMP;
```

However, when you switch from DB2 private protocol access to DRDA access for executing statements with three-part names, you cannot connect to another location after you return to the first location (location A).

- CREATE, ALTER, DROP, GRANT, COMMENT ON, LABEL ON, RENAME, and REVOKE statements cannot contain three-part names or aliases for three-part names.

  To execute these statements at a remote location, you must use an explicit connection.

Preparing applications with three-part names to use DRDA access

For applications that use three-part names for remote access, you control the distributed access method when you prepare the applications. To use DRDA access, you must bind an application that contains three-part names:

- Into a package at each location that is specified in a three-part name
- Into a package or plan at the local location with the bind option DBPROTOCOL(DRDA)

DB2 determines the default value for bind option DBPROTOCOL from the value of field DATABASE PROTOCOL on installation panel DSNTIP5.

One method of preparing a program with three-part names to use DRDA access is:

1. Precompile the program.
2. Bind the DBRM into a package at each server that is specified with a three-part name in the program.
3. Bind the DBRM and the packages into a plan using bind option DBPROTOCOL(DRDA) at the local location.

For example, suppose that application program DRDA3PRT contains these SQL statements:

```
SELECT * FROM SAN_JOSE.DSN8610.EMP;
UPDATE LONDON.DSN8610.EMP SET PHONENO='4145'
  WHERE LASTNAME='QUINTANA';
DELETE FROM TOKYO.DSN8610.DEPT WHERE DEPTNO='M01';
```

To prepare program DRDA3PRT to execute using DRDA access, precompile DRDA3PRT, and bind the DBRM into packages at SAN_JOSE, LONDON, and TOKYO. Then bind the three remote packages and the DBRM into a plan using bind option DBPROTOCOL(DRDA).

For an application that uses an explicit connection to one location and a three-part name to access (hop to) a second location, the program preparation process is slightly different. In that case, follow steps like these:

1. Precompile the program.
2. Bind the DBRM into a package at the first location using bind option DBPROTOCOL(DRDA). This causes the connection to the second location to use DRDA access.
3. Bind the DBRM into a package at the second location.
4. Bind the DBRM and the packages into a plan using bind option DBPROTOCOL(DRDA) at the local location.

Moving from DB2 private protocol access to DRDA access

Recommendation: Move from DB2 private protocol access to DRDA access whenever possible. Because DB2 supports DRDA access for three-part names, you can move to DRDA access without modifying your applications. For any application that uses DB2 private protocol access, follow these steps to make the application use DRDA access:

1. Determine which locations the application accesses.

   For static SQL applications, you can do this by searching for all SQL statements that include three-part names and aliases for three-part names. For three-part names, the high-level qualifier is the location name. For potential aliases, query catalog table SYSTABLES to determine whether the object is an alias, and if so, the location name of the table that the alias represents. For example:

   ```
   SELECT NAME, CREATOR, LOCATION, TBCREATOR, TBNAME
     FROM SYSIBM.SYSTABLES
     WHERE NAME='name'
     AND TYPE='A';
   ```

   where *name* is the potential alias.

   For dynamic SQL applications, bind packages at all remote locations that users might access with three-part names.

2. Bind the application into a package at every location that is named in the application. Optionally, bind a package locally.

   For an application that uses explicit CONNECT statements to connect to a second location and then accesses a third location using a three-part name, bind a package at the second location with DBPROTOCOL(DRDA), and bind another package at the third location.

3. Bind all remote packages into a plan with the local package or DBRM. Bind this plan with the option DBPROTOCOL(DRDA).

4. Ensure that aliases resolve correctly.

   For DB2 private protocol access, DB2 resolves aliases at the requesting location. For DRDA access, however, DB2 resolves aliases at the location where the package executes. Therefore, you might need to define aliases for three-part names at remote locations.

   For example, suppose you use DRDA access to run a program that contains this statement:

   ```
   SELECT * FROM MYALIAS;
   ```

   Assume MYALIAS is an alias for LOC2.MYID.MYTABLE. DB2 resolves MYALIAS at the local location to determine that this statement needs to run at LOC2. DB2 does not send the resolved name to LOC2. When the statement executes at LOC2, DB2 resolves MYALIAS using the catalog at LOC2. If the catalog does not contain the alias MYID.MYTABLE for MYALIAS, the SELECT statement does not execute successfully.

   This situation can become more complicated if you use three-part names to access DB2 objects from remote locations. For example, suppose you are connected explicitly to LOC2, and you use DRDA access to execute this statement:

   ```
   SELECT * FROM YRALIAS;
   ```

where YRALIAS is an alias for LOC3.MYID.MYTABLE. When this SELECT statement executes at LOC3, both LOC2 and LOC3 must have an alias YRALIAS that resolves to MYID.MYTABLE at LOC3.

5. If you use the resource limit facility at the remote locations that are specified in three-part names to control the amount of time that distributed dynamic SQL statements run, you must modify the resource limit specification tables at those locations.

   For DB2 private protocol access, you specify plan names to govern SQL statements that originate at a remote location. For DRDA access, you specify package names for this purpose. Therefore, you must add rows to your resource limit specification tables at the remote locations for the packages you bound for DRDA access with three-part names. You should also delete the rows that specify plan names for DB2 private protocol access.

Choosing a default database protocol

The DATABASE PROTOCOL field on installation panel DSNTIP5 determines whether bind operations that do not specify the DBPROTOCOL parameter use DRDA access or DB2 private protocol access. Before you set the default to DRDA, keep the following factors in mind:

- # If you fall back to DB2 Version 5, you cannot use plans and packages that were bound in Version 6 using DBPROTOCOL(DRDA).
- # In a data sharing group that is in coexistence mode, if a data sharing member that is at DB2 Version 6 binds a package or plan using DBPROTOCOL(DRDA), other members that are at earlier DB2 release levels cannot use that package or plan.

Recommendation: Choose a default database protocol of DRDA only after you are sure that your DB2 subsystem is stable at DB2 Version 6. If your installation has data sharing groups that are operating in coexistence mode, wait until all members are at DB2 Version 6 before choosing a default of DRDA.

Stored procedure enhancements

The following sections discuss changes to stored procedures for Version 6:

- "Creating and modifying stored procedure definitions"
- "Changes to stored procedure security" on page 109
- "Changes to stored procedure invocation" on page 110

Creating and modifying stored procedure definitions

DB2 for OS/390 Version 6 introduces the SQL CREATE PROCEDURE, ALTER PROCEDURE, and DROP PROCEDURE statements, which you can use to define stored procedures to DB2. These statements replace the method of inserting, updating, and deleting rows in catalog table SYSIBM.SYSPROCEDURES.

For details on the syntax of the CREATE PROCEDURE, ALTER PROCEDURE, and DROP PROCEDURE statements, see Chapter 6 of *DB2 SQL Reference*.

Example of creating a stored procedure definition: Suppose you have written and prepared a stored procedure that has these characteristics:

- The name is B.

- The stored procedure should be defined in schema RED.
- It takes two parameters:
  - An integer input parameter named V1
  - A character output parameter of length 9 named V2
- It is written in the C language.
- It contains no SQL statements.
- The same input always produces the same output.
- The load module name is SUMMOD.
- The package collection name is SUMCOLL.
- It should run for no more than 900 CPU service units.
- The parameters can have null values.
- It should be deleted from memory when it completes.
- The DBINFO structure is not passed to the stored procedure.
- The Language Environment run-time options it needs are:

  `MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)`
- It is part of the WLM application environment named PAYROLL.
- It runs as a main program.
- It does not access non-DB2 resources, so it does not need a special RACF® environment.
- It can return at most 10 result sets.
- When control returns to the client program, DB2 should not commit updates automatically.

This CREATE PROCEDURE statement defines the stored procedure to DB2:

```
CREATE PROCEDURE RED.B(V1 INTEGER IN, CHAR(9) OUT)
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  EXTERNAL NAME SUMMOD
  COLLID SUMCOLL
  ASUTIME LIMIT 900
  PARAMETER STYLE GENERAL WITH NULLS
  STAY RESIDENT NO
  NO DBINFO
  RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
  WLM ENVIRONMENT PAYROLL
  PROGRAM TYPE MAIN
  SECURITY DB2
  RESULT SETS 10
  COMMIT ON RETURN NO;
```

Example of altering a stored procedure definition: Suppose you need to make the following changes to the preceding stored procedure definition:

- The stored procedure selects data from DB2 tables but does not modify DB2 data.

- The parameters can have null values, and the stored procedure can return a diagnostic string.
- The length of time the stored procedure runs should not be limited.
- If the stored procedure is called by another stored procedure or a user-defined function, the stored procedure uses the WLM environment of the caller.

Execute this ALTER PROCEDURE statement to make the changes:

```
ALTER PROCEDURE RED.B
  READS SQL DATA
  ASUTIME NO LIMIT
  PARAMETER STYLE DB2SQL
  WLM ENVIRONMENT (PAYROLL,*);
```

DB2 no longer uses SYSIBM.SYSPROCEDURES to record stored procedure definitions. When you migrate, DB2 for OS/390 Version 6 automatically creates new definitions of your old stored procedures in the SYSIBM.SYSROUTINES catalog table. However, if you specified values for AUTHID or LUNAME in any old stored procedure definitions, DB2 cannot create new definitions for those stored procedures, and you must manually redefine those stored procedures using the CREATE PROCEDURE statement.

To check for stored procedures with nonblank AUTHID or LUNAME values, execute this query:

```
SELECT * FROM SYSIBM.SYSPROCEDURES
  WHERE AUTHID<>' ' OR LUNAME<>' ';
```

Then use CREATE PROCEDURE to create definitions for all stored procedures that are identified by the SELECT statement. You cannot specify AUTHID or LUNAME using CREATE PROCEDURE. However, AUTHID and LUNAME let you define several versions of a stored procedure, such as a test version and a production version. You can accomplish the same task by specifying a unique schema name for each stored procedure with the same name. For example, for stored procedure INVENTORY, you might define TEST.INVENTORY and PRODTN.INVENTORY.

Changes to stored procedure security

DB2 for OS/390 Version 6 contains improvements to authorization for resources that are internal and external to DB2.

For accessing DB2 resources, DB2 for OS/390 Version 6 introduces the concept of a stored procedure owner. The owner of a stored procedure is the authorization ID under which the CREATE PROCEDURE statement is executed. If the CREATE PROCEDURE statement is embedded in a program, the owner of the stored procedure is the owner of the plan or package that contains the CREATE PROCEDURE statement. If the CREATE PROCEDURE statement is executed dynamically, the owner of the stored procedure is the value of special register CURRENT SQLID when the CREATE PROCEDURE statement is executed.

The owner of a stored procedure can execute the DB2 commands START PROCEDURE, STOP PROCEDURE, and DISPLAY PROCEDURE on that stored procedure. The owner can also grant and revoke the new EXECUTE privilege on the stored procedure. The ability to grant EXECUTE authority on a stored procedure, instead of only on a stored procedure package, lets the owner control

access to stored procedures that contain no SQL statements and therefore have no packages.

For accessing resources that are external to DB2, DB2 can establish a RACF environment for a stored procedure that runs in a WLM-established stored procedures address space. The authority that is used when the stored procedure accesses protected MVS resources depends on the value of SECURITY in the stored procedure definition:

- If the value of SECURITY is DB2, the authorization ID associated with the stored procedures address space is used.
- If the value of SECURITY is USER, the authorization ID under which the CALL statement is executed is used.
- If the value of SECURITY is DEFINER, the authorization ID under which the CREATE PROCEDURE statement was executed is used.

Changes to stored procedure invocation

DB2 for OS/390 Version 6 introduces the following changes to stored procedure invocation:

- Support for passing the new data types as parameters

  Those data types include BLOB, CLOB, DBCLOB, BLOB locators, CLOB locators, DBCLOB locators, DATE, TIME, TIMESTAMP, ROWID, and distinct types

- Passing host structures as parameters in the CALL statement
- Nesting stored procedure and user-defined function invocations

  Stored procedures can contain CALL statements for invoking other stored procedures or invocations of user-defined functions. DB2 supports 16 levels of nesting.

- An additional method for passing parameters

  In previous releases of DB2, you could pass parameters using the SIMPLE (now called GENERAL) convention or the SIMPLE WITH NULLS (now called GENERAL WITH NULLS) convention. DB2 for OS/390 Version 6 adds a new convention called DB2SQL.

  Like GENERAL WITH NULLS, the DB2SQL option lets you supply a null value for any parameter that is passed to the stored procedure. In addition, DB2 passes input and output parameters to the stored procedure that contain this information:

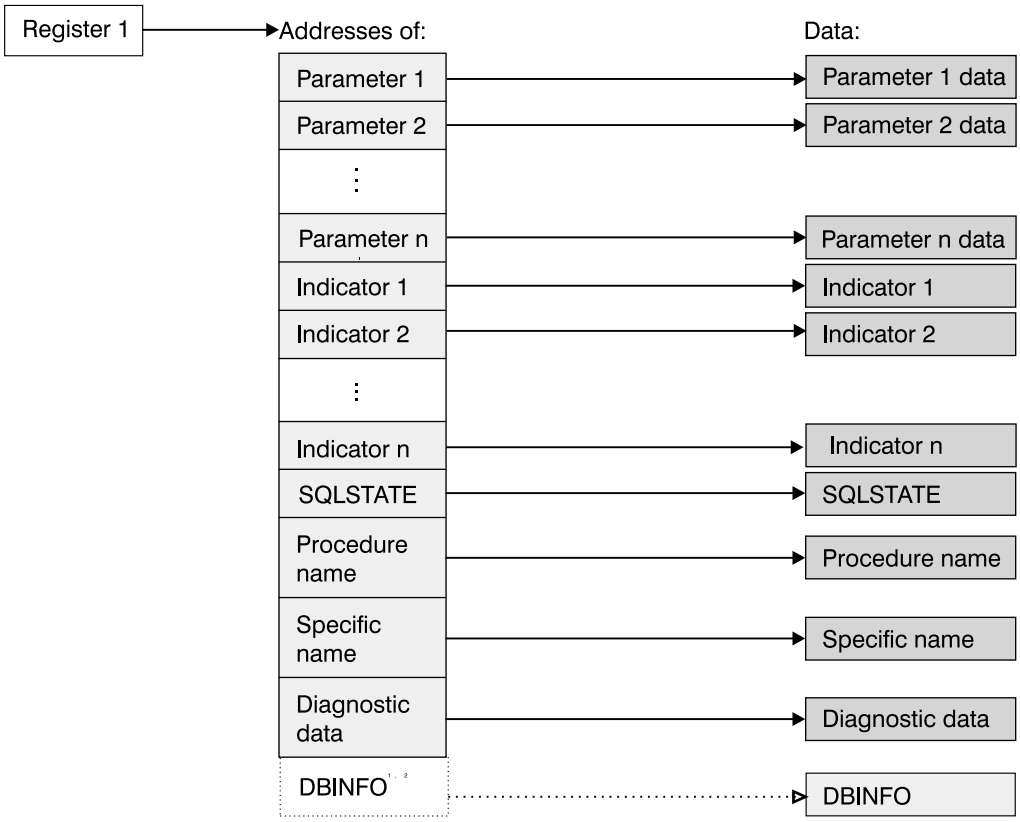
  — The SQLSTATE that is to be returned to DB2.

  — The qualified name of the stored procedure.

  — The specific name of the stored procedure. The specific name is the same as the qualified name.

  — The SQL diagnostic string that is to be returned to DB2.

  Figure 16 on page 111 shows the structure of the parameter list for PARAMETER STYLE DB2SQL.



[1] For PL/I, this value is the address of a pointer to the DBINFO data.
[2] Passed if the DBINFO option is specified in the user-defined function definition

Figure 16. Parameter convention DB2SQL for a stored procedure

Figure 17 on page 112 shows an example of how a stored procedure that is defined with the DB2SQL linkage convention receives parameters.

- Using the schema name to determine which version of a stored procedure to run

  When you execute the CREATE PROCEDURE statement, you implicitly or explicitly specify the name of the schema in which the stored procedure resides. You can create two stored procedures with the same name in two different schemas, and DB2 considers each of those stored procedures to be unique.

  When you execute a CALL statement, you can specify zero, one, or two qualifiers for the stored procedure name. The stored procedure name with its qualifiers uniquely identifies the version of the stored procedure that DB2 should execute. The first qualifier is the name of the location where the stored procedure resides, and the second qualifier is the schema in which the stored procedure resides. If you do not specify a location name, the default location name is the name of the current server. If you specify a location name, a schema name is required. If you do not specify a schema name, DB2 uses the SQL path to determine the default schema name. The SQL path comes from the PATH bind option if you use the CALL *procedure-name* form of the CALL statement, or from the CURRENT PATH special register if you use the CALL *host-variable* form of the CALL statement. See the description of the CALL statement in Chapter 6 of *DB2 SQL Reference* for more information.

```
#pragma runopts(plist(os))
#include <;stdlib.h>
#include <;stdio.h>

main(argc,argv)
  int argc;
  char *argv[];
{
  /*****************************************************/
  /* Assume that the SQL CALL statement includes     */
  /* 3 input/output parameters in the parameter list.*/
  /* Also assume that the DBINFO was not specified   */
  /* in the CREATE PROCEDURE statement.              */
  /* The argv vector will contain these entries:     */
  /*      argv[0]         1    contains load module  */
  /*      argv[1-3]       3    input/output parms    */
  /*      argv[4-6]       3    null indicators       */
  /*      argv[7]         1    SQLSTATE variable     */
  /*      argv[8]         1    qualified proc name   */
  /*      argv[9]         1    specific proc name    */
  /*      argv[10]      + 1    diagnostic string     */
  /*                    ------                       */
  /*                      11    for the argc variable */
  /*****************************************************/
  if argc<>11 {

⋮
  /* We end up here when invoked with wrong number of parms */
  }
  /*****************************************************/
  /* Assume the first parameter is an integer.       */
  /* The code below shows how to copy the integer    */
  /* parameter into the application storage.         */
  /*****************************************************/
  int parm1;
  parm1 = *(int *) argv[1];
  /*****************************************************/
  /* We can access the null indicator for the first  */
  /* parameter on the SQL CALL as follows:           */
  /*****************************************************/
  short int ind1;
  ind1 = *(short int *) argv[4];
  /*****************************************************/
  /* We can use the expression below to assign       */
  /* 'xxxxx' to the SQLSTATE returned to caller on   */
  /* the SQL CALL statement.                         */
  /*****************************************************/
  strcpy(argv[7],"xxxxx/0");
  /*****************************************************/
  /* We obtain the value of the qualified procedure  */
  /* name with this expression.                      */
  /*****************************************************/
  char p_proc[28];
  strcpy(p_proc,argv[8]);
  /*****************************************************/
  /* We obtain the value of the specific procedure   */
  /* name with this expression.                      */
  /*****************************************************/
  char p_spec[19];
  strcpy(p_spec,argv[9]);
```

Figure 17 (Part 1 of 2). An example of DB2SQL linkage in C

```
  /************************************************/
  /* We can use the expression below to assign    */
  /* 'yyyyyyyy' to the diagnostic string returned */
  /* in the SQLCA associated with the CALL statement.*/
  /************************************************/
  strcpy(argv[10],"yyyyyyyy/0");
⋮
}
```

Figure 17 (Part 2 of 2). An example of DB2SQL linkage in C

Using SQL procedures

DB2 introduces SQL procedures, which are stored procedures in which the source program is included in the stored procedure definition. The source program is written using special statements called SQL procedure statements. The SQL procedure statements are:

Assignment statement
Assigns a value to an output parameter or to an SQL variable, which is a variable that is defined and used only within a procedure body. The right side of an assignment statement can include SQL built-in functions.

CALL statement
Calls another stored procedure. This statement is similar to the CALL statement described in Chapter 6 of *DB2 SQL Reference*, except that the parameters must be SQL variables, parameters for the SQL procedure, or constants.

CASE statement
Selects an execution path based on the evaluation of one or more conditions. This statement is similar to the CASE expression, which is described in Chapter 3 of *DB2 SQL Reference*.

GET DIAGNOSTICS statement
Obtains information about the previous SQL statement that was executed.

GOTO statement
Transfers program control to a labelled statement.

IF statement
Selects an execution path based on the evaluation of a condition.

LEAVE statement
Transfers program control out of a loop or a block of code.

LOOP statement
Executes a statement or group of statements multiple times.

REPEAT statement
Executes a statement or group of statements until a search condition is true.

WHILE statement
Repeats the execution of a statement or group of statements while a specified condition is true.

Compound statement
Can contain one or more of any of the other types of statements in this list. In addition, a compound statement can contain SQL variable declarations, condition handlers, or cursor declarations.

The order of statements in a compound statement must be:

1. SQL variable and condition declarations
2. Cursor declarations
3. Handler declarations
4. Procedure body statements (CALL, CASE, IF, LOOP, REPEAT, WHILE, SQL)

SQL statement
A subset of the SQL statements that are described in Chapter 6 of *DB2 SQL Reference*. Certain SQL statements are valid in a compound statement, but not valid if the SQL statement is the only statement in the procedure body. Appendix B of *DB2 SQL Reference* lists the SQL statements that are valid in an SQL procedure.

Figure 18 shows an example of a simple SQL procedure. This SQL procedure demonstates how to use a CASE statement. The procedure receives an employee's ID number and rating as input parameters. The CASE statement modifies the employee's salary and bonus, using a different UPDATE statement for each of the possible ratings.

```
CREATE PROCEDURE UPDATESALARY2
 (IN EMPNUMBR CHAR(6),
  IN RATING INT)
 LANGUAGE SQL
 MODIFIES SQL DATA
 CASE RATING
  WHEN 1 THEN
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.10, BONUS = 1000
    WHERE EMPNO = EMPNUMBR;
  WHEN 2 THEN
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.05, BONUS = 500
    WHERE EMPNO = EMPNUMBR;
  ELSE
   UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * 1.03, BONUS = 0
    WHERE EMPNO = EMPNUMBR;
 END CASE
```

Figure 18. Example of an SQL procedure definition

Improved data transfer with OPTIMIZE FOR n ROWS

In previous versions of DB2 for OS/390, client programs could use the clause OPTIMIZE FOR *n* ROWS in SELECT statements to limit the number of data rows that DB2 for OS/390 returned on each DRDA network transmission. DB2 for OS/390 Version 6, enhances the OPTIMIZE FOR *n* ROWS clause to optimize retrieval of a large number of rows.

DB2 for OS/390 Version 6 supports DRDA level 3, which provides support for returning multiple query blocks on each network transmission. To retrieve multiple query blocks on each network transmission, a client program that performs high-volume download operations can specify a large value for *n* in the OPTIMIZE for *n* ROWS clause.

The number of rows that DB2 for OS/390 transmits on each network transmission depends on the following factors:

- If *n* rows of the SQL result set fit within a single DRDA query block, a DB2 server can send *n* rows to any DRDA client. In this case, DB2 sends *n* rows in each network transmission, until the entire query result set is exhausted.
- If *n* rows of the SQL result set exceed a single DRDA query block, the number of rows that are contained in each network transmission depends on the client's DRDA software level and configuration:
  - If the client does not support DRDA level 3, the DB2 server automatically reduces the value of *n* to match the number of rows that fit within a DRDA query block.
  - If the client does support DRDA level 3, the DRDA client can choose to accept multiple DRDA query blocks in a single data transmission. DRDA lets the client establish an upper limit on the number of extra DRDA query blocks in each network transmission.

    The upper limit is the smaller of the following values:

    - The value of EXTRA BLOCKS SRV in install panel DSNTIP5 at the DB2 for OS/390 server

      This is the maximum number of extra DRDA query blocks that the DB2 server returns to a client in a single network transmission.

    - The client's extra query block limit, which DB2 for OS/390 obtains from the DDM MAXBLKEXT parameter received from the client

      When DB2 for OS/390 acts as a DRDA client, DB2 sets the DDM MAXBLKEXT parameter to the value that is specified on the EXTRA BLOCKS REQ field of the DSNTIP5 install panel.

    The number of rows that a DB2 server sends is the smaller of *n* rows and the number of rows that fit within the lesser of these two limitations:

    - The value of EXTRA BLOCKS SRV in install panel DSNTIP5 at the DB2 server

      This is the maximum number of extra DRDA query blocks that the DB2 server returns to a client in a single network transmission.

    - The client's extra query block limit, which is obtained from the DDM MAXBLKEXT parameter received from the client

When DB2 acts as a DRDA client, the DDM MAXBLKEXT parameter is set to the value specified on the EXTRA BLOCKS REQ install option of the DSNTIP5 install panel.

The OPTIMIZE FOR *n* ROWS function is useful in two cases:

- If *n* is less than the number of rows that fit in the DRDA query block, OPTIMIZE FOR *n* ROWS can improve performance by preventing the DB2 server from fetching rows that might never be used by the DRDA client application.
- If *n* is greater than the number of rows that fit in a DRDA query block, OPTIMIZE FOR *n* ROWS lets the DRDA client request multiple blocks of query data on each network transmission. This use of OPTIMIZE FOR *n* ROWS can significantly improve elapsed time for large query download operations.

Specifying a large value for *n* in OPTIMIZE FOR *n* ROWS can increase the number of DRDA query blocks that a DB2 server returns in each network transmission. This function can significantly improve performance for applications that use DRDA access to download large amounts of data. However, this same function can degrade performance if you do not use it properly. The following examples demonstrate the performance problems that can occur when you do not use OPTIMIZE FOR *n* ROWS judiciously.

The example in Figure 19, the DRDA client opens a cursor and fetches rows from the cursor. At some point before all rows in the query result set are returned, the application issues an SQL INSERT. DB2 uses normal DRDA blocking, which has two advantages over the blocking that is used for OPTIMIZE FOR *n* ROWS:

- If the application issues an SQL statement other than FETCH (the example shows an INSERT statement), the DRDA client can transmit the SQL statement immediately, because the DRDA connection is not in use after the SQL OPEN.
- If the SQL application closes the cursor before fetching all the rows in the query result set, the server fetches the number of rows that fits into one query block, which is 100 rows. Basically, the DRDA query block size places an upper limit on the number of rows that are fetched unnecessarily.

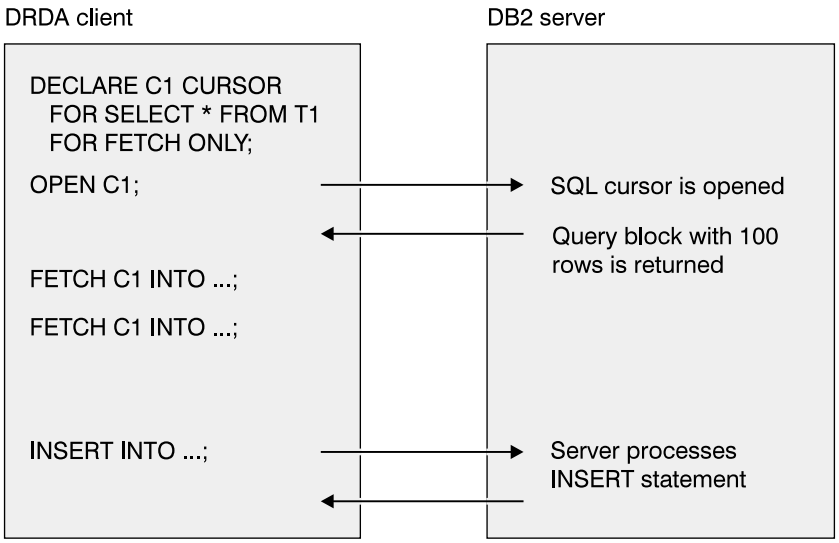


Figure 19. Message flows without OPTIMIZE FOR 1000 ROWS

In Figure 20 on page 117, the DRDA client opens a cursor and fetches rows from the cursor using OPTIMIZE FOR *n* ROWS. Both the DRDA client and the DB2 server are configured to support multiple DRDA query blocks. At some time before the end of the query result set, the application issues an SQL INSERT. Because OPTIMIZE FOR *n* ROWS is being used, the DRDA connection is not available when the SQL INSERT is issued because the connection is still being used to receive the DRDA query blocks for 1000 rows of data. This causes two performance problems:

- Application elapsed time can increase if the DRDA client waits for a large query result set to be transmitted, before the DRDA connection can be used for other SQL statements. Figure 20 shows how an SQL INSERT statement can be delayed because of a large query result set.
- If the application closes the cursor before fetching all the rows in the SQL result set, the server might fetch a large number of rows unnecessarily.

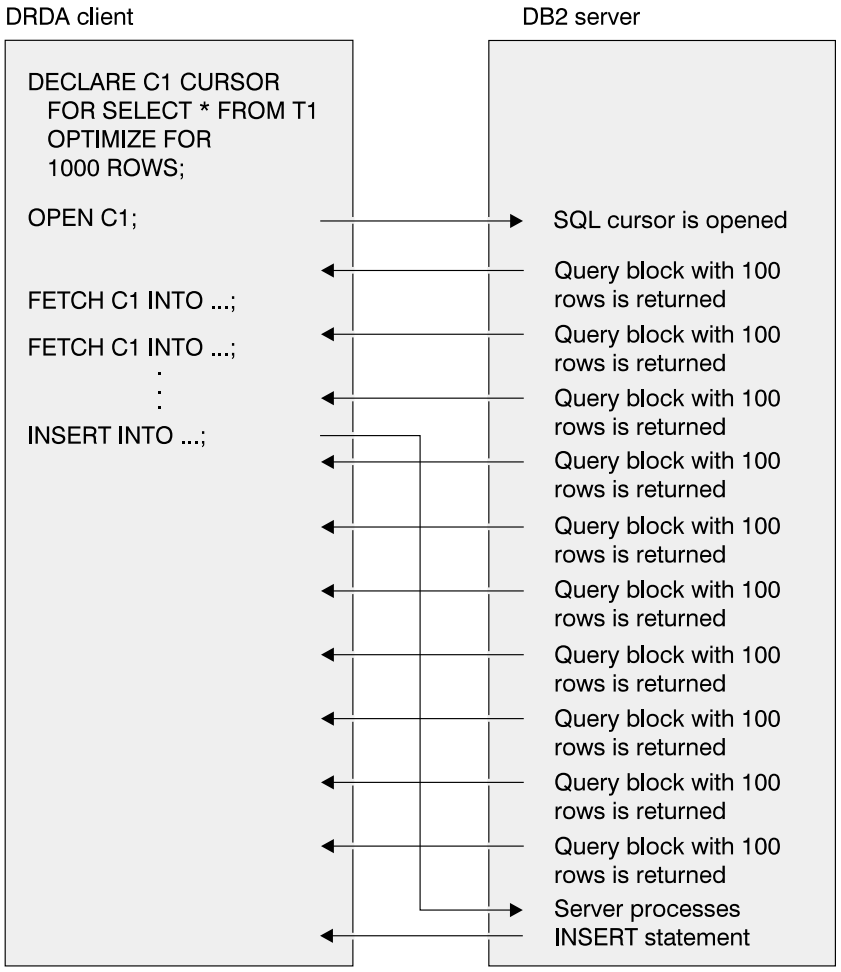


Figure 20. Message flows with OPTIMIZE FOR 1000 ROWS

Recommendation: Use OPTIMIZE FOR *n* ROWS to increase the number of DRDA query blocks only in applications that have all these attributes:

- The application fetches a large number of rows from a read-only query.
- The application rarely closes the SQL cursor before fetching the entire query result set.

- The application does not issue statements other than FETCH to the DB2 server while the SQL cursor is open.
- The application does not execute FETCH statements using multiple cursors that are opened concurrently and defined with OPTIMIZE FOR *n* ROWS.

DB2 ODBC enhancements

In Version 5, DB2 for OS/390 introduced DB2 Call Level Interface (CLI). In Version 6, DB2 CLI is renamed DB2 ODBC (Open Database Connectivity) to align with common industry terminology. This is a change in name only; the support and function remain the same.

Enhancements to DB2 ODBC improve performance, support the object-relational extensions that are introduced in Version 6, and provide usability improvements.

- Performance improvements

  The new DB2 ODBC shadow catalog provides fast catalog access for your ODBC catalog queries. Using the new DB2 ODBC initialization file keyword, CLISCHEMA, you can direct your ODCB catalog queries to the shadow catalog. See “Faster ODBC catalog queries” on page 119 for more information.

  A new API, `SQLDescribeParam()`, describes parameter markers. This API improves performance for ODBC applications, as described in “Better performance for dynamic SQL applications” on page 119.

- Support for object-relational extensions

  Three new APIs support the manipulation of LOBs:

  - `SQLGetLength()` retrieves the length of a LOB value.
  - `SQLGetPosition()` returns the starting position of a string within a LOB value.
  - `SQLGetSubString()` retrieves a portion of a LOB string value.

  A number of the existing APIs now support the object extensions.

  Additional support for the object extensions includes:

  - LOB and row ID data types.
  - LOB locators. For example, ODBC applications can use LOB locators and apply the new APIs to a LOB value using the LOB locator.
  - Support of the new FREE LOCATOR and HOLD LOCATOR SQL statements.
  - Support of distinct types and cast functions.
  - Support of user-defined functions.
  - A new DB2 ODBC initialization file keyword, CURRENTFUNCTIONPATH, for setting the CURRENT PATH special register.

- Usability improvements include more detailed ODBC trace information and changes to trace keywords that make application and diagnostic traces easier to use.

DB2 ODBC's support of multithreading and multiple contexts is available in Version 5 with APAR PQ09901. With these enhancements, you can write multithreaded

applications in an LE POSIX(ON) runtime environment, and restrictions on connection switching are lifted.

For detail about the DB2 ODBC enhancements, see *DB2 ODBC Guide and Reference*.

Faster ODBC catalog queries

Most ODBC end users require access to a small subset of tables in the DB2 system catalog. To enable faster, more efficient ODBC catalog queries, Version 6 introduces the DB2 ODBC shadow catalog. The tables in the shadow catalog are pre-joined and indexed to improve performance and are limited to only the subset of catalog information needed for ODBC operations.

You can easily access and manage the DB2 ODBC shadow catalog.

- The setting you specify in the `CLISCHEMA` keyword in the DB2 ODBC initialization file points your catalog query to a specific set of shadow catalog tables or views.
- DB2 DataPropagator for OS/390 provides an automated process for managing the shadow catalog. The DB2 DataPropagator Capture and Apply process captures the catalog data and synchronizes updates to the shadow copies of the DB2 catalog with the DB2 system catalog.

Better performance for dynamic SQL applications

DB2 adds DRDA support for the new DESCRIBE INPUT statement. DESCRIBE INPUT obtains information about the input parameter markers of a prepared statement. This support improves performance for dynamic SQL applications and many ODBC applications by reducing the number of network messages that need to be exchanged when an application is executing dynamic SQL with input host variables and does not know the correct data type of the input host variables ahead of time. Using DESCRIBE INPUT, you can ask the DBMS to describe what an SQL statement looks like and avoid the expense of catalog lookups for determining input parameter marker data.

See Appendix C, “Changes to SQL” on page 269 for a description of the DESCRIBE INPUT statement.

Improvements for dynamically prepared SQL statements

DB2 uses the SQL path determined by the new CURRENT PATH special register to implicitly qualify names of distinct types, user-defined functions, and stored procedures in dynamically prepared SQL statements. The new register can be very useful for test purposes. You can use it to support both the test and production versions of a single stored procedure within one DB2 subsystem. For more information about multiple versions of stored procedures, see “Changes to stored procedure invocation” on page 110.

For more information about the CURRENT PATH special register, see *DB2 SQL Reference*.

DB2 database connection pooling

DB2 Version 6 adds new support for database connection pooling. In previous releases of DB2, when an application requester established a connection to DB2, a connection to the DB2 database was also established. A connection to the database has a much larger footprint than does a DRDA or a private-protocol connection to a DB2 application server. In Version 6, for DRDA requesters, DB2 maintains a pool of database connections that may be re-used as necessary to process requests from DRDA application requesters. This pool enables DB2 to support up to 150 000 DRDA connections to DB2. The pool of connections to the database consists of a pool of database access threads and the support for this is called type 2 inactive threads.

DB2 supports two types of inactive threads: *type 1* and *type 2*. The differences between the types are that type 2 inactive threads are only available for DRDA connections, use less storage than type 1 inactive threads, and use a pool of database access threads that can be switched among connections as needed. If you have a requirement to support more inbound remote connections than you have database access threads, you should consider using DDF inactive thread support.

The following sections provide information on inactive thread support.

- "Using type 2 inactive threads"
- "Determining if a thread can become inactive" on page 121
- "Enabling threads to become inactive" on page 121

Using type 2 inactive threads

Type 2 inactive threads use a pool of database access threads that can be switched among connections as needed. DB2 always tries to make inactive threads type 2, but in some cases cannot do so. The conditions listed in Table 9 determine if a thread can be a type 2 or a type 1.

Table 9. Requirements for type 1 and type 2 inactive threads

If there is...	Thread can be type 2?	Thread can be type 1?
A hop to another location	Yes	Yes
A connection using DB2 private—protocol access	No	Yes
A package that is bound with RELEASE(COMMIT)	Yes	Yes
A package that is bound with RELEASE(DEALLOCATE)	Yes	No
A held cursor, a held LOB locator, or a package bound with KEEPDYNAMIC(YES)	No	No

When the conditions listed in Table 9 are true, the thread can become inactive when a COMMIT is issued. After a ROLLBACK, a thread can become inactive even if it had open cursors defined WITH HOLD or a held LOB locator because ROLLBACK closes all cursors and LOB locators.

Determining if a thread can become inactive

After a COMMIT or ROLLBACK, DB2 determines if a thread can become inactive and, if so, if that thread can become a type 1 or type 2 inactive thread based on the conditions shown in Table 9 on page 120.

If a thread is eligible to become a type 2 inactive thread, the thread is made inactive and the database access thread is eligible to be used by another connection.

If a thread must become a type 1 inactive thread, DB2 first compares the number of current type 1 inactive threads to the value that is specified for your installation for MAX TYPE 1 INACTIVE on panel DSNTIPR:

1. If the current number of type 1 inactive threads is below the value in MAX TYPE 1 INACTIVE, the thread becomes inactive. It cannot be used by another connection.
2. If the current number of type 1 inactive threads meets or exceeds the value in MAX TYPE 1 INACTIVE, the thread remains active. However, too many active threads (that is, more than MAX REMOTE ACTIVE) can cause the thread and its connection to be terminated.

Enabling threads to become inactive

You must specify INACTIVE on the DDF THREADS field of installation panel DSNTIPR to allow threads to become inactive. To limit the number of type 1 inactive threads that can be created, specify a value in the MAX TYPE 1 INACTIVE field of installation panel DSNTIPR. The default is 0, which means that any thread that does not qualify for being a type 2 inactive thread remains active.

Recommendation: Use type 2 inactive threads if you can. If you can't, set MAX TYPE 1 INACTIVE to the maximum number of concurrent connections that use DB2 private—protocol access.

Chapter 6. Object-relational extensions and active data

With the object extensions of DB2, you can incorporate object-oriented concepts and methodologies into your relational database by extending DB2 with richer sets of data types and functions. With those extensions, you can store instances of object-oriented data types in columns of tables and perform operations on them using functions in SQL statements. In addition, you can control the types of operations that users can perform on those data types.

The object extensions that DB2 provides are:

- Large objects (LOBs)

  The VARCHAR and VARGRAPHIC data types have a storage limit of 32 KB. Although this might be sufficient for small- to medium-size text data, applications often need to store large text documents. They might also need to store a wide variety of additional data types such as audio, video, drawings, mixed text and graphics, and images. DB2 provides three data types to store these data objects as strings of up to 2 GB - 1 in size. The three data types are binary large objects (BLOBs), character large objects (CLOBs), and double-byte character large objects (DBCLOBs).

  For a detailed discussion of LOBs, see “Working with large objects (LOBs)” on page 124.

- Distinct types

  A distinct type is a user-defined data type that shares its internal representation with a built-in data type but is considered to be a separate and incompatible type for semantic purposes. For example, you might want to define a picture type or an audio type which have different semantics but use the built-in data type BLOB for their internal representation.

  For a detailed discussion of distinct types, see “Creating and using distinct types” on page 202.

- User-defined functions

  The built-in functions that are supplied with DB2 are a useful set of functions, but they might not satisfy all of your requirements. You can write user-defined functions to meet the specific needs for your installation. For example, a built-in function might perform a calculation you need, but the function does not accept the distinct types you want to pass to it. You can then define a function based on a built-in function, called a *sourced* user-defined function, that accepts your distinct types. You might need to perform another calculation in your SQL statements for which there is no built-in function. In that situation, you can define and write an *external* user-defined function.

  For a detailed discussion of user-defined functions, see “Creating and using user-defined functions” on page 143.

Triggers help you bring application logic into the database. A trigger defines a set of actions that are to be executed when a specific SQL data change operation occurs on a specified table. The SQL data change operation includes actions initiated by SQL inserts, updates, and deletes, as well as actions of referential integrity constraints. For a detailed discussion of triggers, see “Using triggers for active data” on page 211.

Working with large objects (LOBs)

The term *large object* and the acronym *LOB* refer to DB2 objects that you can use to store large amounts of data. A LOB is a varying-length character string that can contain up to 2 GB - 1 of data.

The three LOB data types are:

- *Binary large object (BLOB)*

  Use a BLOB to store binary data such as pictures, voice, and mixed media.

- *Character large object (CLOB)*

  Use a CLOB to store SBCS or mixed character data, such as documents.

- *Double-byte character large object (DBCLOB)*

  Use a DBCLOB to store data that consists of only DBCS data.

This section presents the following information about LOBs:

- "Introduction to defining LOBs"
- "Declaring LOB host variables and LOB locators" on page 127
- "LOB materialization" on page 132
- "Using LOB locators to save storage" on page 132
- "LOB system processing" on page 136
- "Recovering table spaces that contain LOBs" on page 141

Introduction to defining LOBs

These are the basic steps for defining LOBs and moving the data into DB2:

1. Define a column of the appropriate LOB type and a row identifier (ROWID) column in a DB2 table. Define only one ROWID column, even if multiple LOB columns are in the table.

   The LOB column contains information about the LOB, not the LOB data itself. The table that contains the LOB definition is called the *base table*. DB2 uses the ROWID column to locate your LOB data. You need only one ROWID column in a table that contains one or more LOB columns. You can define the LOB column and the ROWID column in a CREATE TABLE or ALTER TABLE statement. If you are adding a LOB column and a ROWID column to an existing table, you must use two ALTER TABLE statements. Add the ROWID with the first ALTER TABLE statement and the LOB column with the second ALTER TABLE statement.

2. Create a table space and table to hold the LOB data.

   The table space and table are called a LOB table space and an auxiliary table. If your base table is nonpartitioned, you must create one LOB table space and one auxiliary table for each LOB column. If your base table is partitioned, for each LOB column, you must create one LOB table space and one auxiliary table for each partition. For example, if your base table has three partitions, you must create three LOB table spaces and three auxiliary tables for each LOB column. Create these objects using the CREATE LOB TABLESPACE and CREATE AUXILIARY TABLE statements.

3. Create an index on the auxiliary table.

Each auxiliary table must have exactly one index. Use CREATE INDEX for this task.

4. Put the LOB data into DB2.

   If the total length of a LOB column and the base table row is less than 32 KB, you can use the LOAD utility to put the data in DB2. Otherwise, you must use INSERT or UPDATE statements. Even though the data is stored in the auxiliary table, the target of your LOAD utility statement or INSERT statement specifies the name of the base table. Using INSERT can be difficult because your application needs enough storage to hold the entire value that goes into the LOB column.

For example, suppose you want to add a resume for each employee to the employee table. Employee resumes are no more than 1 MB in size. The employee resumes contain single-byte characters, so you can define the resumes to DB2 as CLOBs. You therefore need to add a column of data type CLOB with a length of 1 MB to the employee table. If a ROWID column has not been defined in the table, you need to add the ROWID column before you add the CLOB column. Execute an ALTER TABLE statement to add the ROWID column, and then execute another ALTER TABLE statement to add the CLOB column. You might use statements like this:

```
ALTER TABLE EMP
  ADD ROW_ID ROWID NOT NULL GENERATED ALWAYS;
COMMIT;
ALTER TABLE EMP
  ADD EMP_RESUME CLOB(1M);
COMMIT;
```

Next, you need to define a LOB table space and an auxiliary table to hold the employee resumes. The LOB table space must be in the same database as the base table. You also need to define an index on the auxiliary table. You can use statements like this:

```
CREATE LOB TABLESPACE RESUMETS
  IN DSN8D61A
  LOG NO;
COMMIT;
CREATE AUXILIARY TABLE EMP_RESUME_TAB
  IN DSN8D61A.RESUMETS
  STORES DSN8610.EMP
  COLUMN EMP_RESUME;
CREATE UNIQUE INDEX XEMP_RESUME
  ON EMP_RESUME_TAB;
COMMIT;
```

If the value of bind option SQLRULES is STD, or if special register CURRENT RULES has been set in the program and has the value STD, DB2 creates the LOB table space, auxiliary table, and auxiliary index for you when you execute the ALTER statement to add the LOB column.

Now that your DB2 objects for the LOB data are defined, you can load your employee resumes into DB2. To do this in an SQL application, you can define a host variable to hold the resume, copy the resume data from a file into the host variable, and then execute an UPDATE statement to copy the data into DB2. Although the data goes into the auxiliary table, your UPDATE statement specifies

the name of the base table. The C language declaration of the host variable might be:

```
SQL TYPE is CLOB (5K) resumedata;
```

The UPDATE statement looks like this:

```
UPDATE EMP SET EMP_RESUME=:resumedata
  WHERE EMPNO=:employeenum;
```

In this example, employeenum is a host variable that identifies the employee who is associated with a resume.

After your LOB data is in DB2, you can write SQL applications to manipulate the data. You can use most SQL statements with LOBs. For example, you can use statements like these to extract information about an employee's department from the resume:

```
EXEC SQL BEGIN DECLARE SECTION;
  long deptInfoBeginLoc;
  long deptInfoEndLoc;
  SQL TYPE IS CLOB_LOCATOR resume;
  SQL TYPE IS CLOB_LOCATOR deptBuffer;
EXEC SQL END DECLARE SECTION;
⋮
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT EMPNO, EMP_RESUME FROM EMP;
⋮
EXEC SQL FETCH C1 INTO :employeenum, :resume;
⋮
EXEC SQL SET  :deptInfoBeginLoc =
  POSSTR(:resumedata, 'Department Information');

EXEC SQL SET :deptInfoEndLoc =
  POSSTR(:resumedata, 'Education');

EXEC SQL SET :deptBuffer =
  SUBSTR(:resume, :deptInfoBeginLoc,
  :deptInfoEndLoc - :deptInfoBeginLoc);
```

These statements use host variables of data type large object locator (LOB locator). LOB locators let you manipulate LOB data without moving the LOB data into host variables. By using LOB locators, you need much smaller amounts of memory for your programs. LOB locators are discussed in “Using LOB locators to save storage” on page 132.

Sample LOB applications: Table 10 lists the sample programs that DB2 provides to assist you in writing applications to manipulate LOB data. All programs reside in data set DSN610.SDSNSAMP.

Table 10 (Page 1 of 2). LOB samples shipped with DB2

Member that contains source code	Language	Function
DSN8DLPL		Demonstrates how to create a table with LOB columns, an auxiliary table, and an auxiliary index. Also demonstrates how to load LOB data that is 32KB or less into a LOB table space.

Table 10 (Page 2 of 2). LOB samples shipped with DB2

Member that contains source code	Language	Function
DSN8DLPL	C	Demonstrates the use of LOB locators and UPDATE statements to move binary data into a column of type BLOB.
DSN8DLRV	C	Demonstrates how to use a locator to manipulate data of type CLOB.
DSNTEP2	PL/I	Demonstrates how to allocate an SQLDA for rows that include LOB data and use that SQLDA to describe an input statement and fetch data from LOB columns.

Declaring LOB host variables and LOB locators

When you write applications to manipulate LOB data, you need to declare host variables to hold the LOB data or LOB locator variables to point to the LOB data. See “Using LOB locators to save storage” on page 132 for information on what LOB locators are and when you should use them instead of host variables.

You can declare LOB host variables and LOB locators in assembler, C, C++, COBOL, FORTRAN, and PL/I. For each host variable or locator of SQL type BLOB, CLOB, or DBCLOB that you declare, DB2 generates an equivalent declaration that uses host language data types. When you refer to a LOB host variable or locator in an SQL statement, you must use the variable you specified in the SQL type declaration. When you refer to the host variable in a host language statement, you must use the variable that DB2 generates.

The following examples show you how to declare LOB host variables in each supported language. In each table, the left column contains the declaration that you code in your application program. The right column contains the declaration that DB2 generates.

Important: DB2 supports host variable declarations for LOBs up to 2 GB - 1 bytes in length, however, the sizes of the LOBs you can declare and manipulate depend on the limits of the host language and the amount of storage available to your program.

Declarations of LOB host variables in assembler: Table 11 on page 128 shows assembler language declarations for some typical LOB types.

Table 11. Example of assembler LOB variable declarations

You declare this variable	DB2 generates this variable
blob_var SQL TYPE IS BLOB 1M	blob_var DS 0FL4 blob_var_length DS FL4 blob_var_data DS CL65535[1] ORG blob_var_data+(1048476-65535)
clob_var SQL TYPE IS CLOB 40000K	clob_var DS 0FL4 clob_var_length DS FL4 clob_var_data DS CL65535[1] ORG clob_var_data +(40960000-65535)
dbclob-var SQL TYPE IS DBCLOB 4000K	dbclob_var DS 0FL4 dbclob_var_length DS FL4 dbclob_var_data DS GL65534[2] ORG dbclob_var_data+(8192000-65534)
blob_loc SQL TYPE IS BLOB_LOCATOR	blob_loc DS FL4
clob_loc SQL TYPE IS CLOB_LOCATOR	clob_loc DS FL4
dbclob_var SQL TYPE IS DBCLOB_LOCATOR	dbclob_loc DS FL4

Notes to Table 11:

1. Because assembler language allows character declarations of no more than 65535 bytes, DB2 separates the host language declarations for BLOB and CLOB host variables that are longer than 65535 bytes into two parts.
2. Because assembler language allows graphic declarations of no more than 65534 bytes, DB2 separates the host language declarations for DBCLOB host variables that are longer than 65534 bytes into two parts.

Declarations of LOB host variables in C: Table 12 shows C and C++ language declarations for some typical LOB types.

Table 12. Examples of C language variable declarations

You declare this variable	DB2 generates this variable
SQL TYPE IS BLOB (1M) blob_var;	struct { unsigned long length; char data[1048576]; } blob_var;
SQL TYPE IS CLOB (40000K) clob_var;	struct { unsigned long length; char data[40960000]; } clob_var;
SQL TYPE IS DBCLOB (4000K) dbclob_var;	struct { unsigned long length; wchar_t data[4096000]; } dbclob_var;
SQL TYPE IS BLOB_LOCATOR blob_loc;	unsigned long blob_loc;
SQL TYPE IS CLOB_LOCATOR clob_loc;	unsigned long clob_loc;
SQL TYPE IS DBCLOB_LOCATOR dbclob_loc;	unsigned long dbclob_loc;

Declarations of LOB host variables in COBOL: Table 13 on page 129 shows COBOL declarations for some typical LOB types.

Table 13. Examples of COBOL variable declarations

You declare this variable	DB2 generates this variable
01 BLOB-VAR USAGE IS SQL TYPE IS BLOB(1M).	01 BLOB-VAR. 02 BLOB-VAR-LENGTH PIC 9(9) COMP. 02 BLOB-VAR-DATA. 49 FILLER PIC X(32767).[1] 49 FILLER PIC X(32767). *Repeat 30 times* ⋮ 49 FILLER PIC X(1048576-32*32767).
01 CLOB-VAR USAGE IS SQL TYPE IS CLOB(40000K).	01 CLOB-VAR. 02 CLOB-VAR-LENGTH PIC 9(9) COMP. 02 CLOB-VAR-DATA. 49 FILLER PIC X(32767).[1] 49 FILLER PIC X(32767). *Repeat 1248 times* ⋮ 49 FILLER PIC X(40960000-1250*32767).
01 DBCLOB-VAR USAGE IS SQL TYPE IS DBCLOB(4000K).	01 DBCLOB-VAR. 02 DBCLOB-VAR-LENGTH PIC 9(9) COMP. 02 DBCLOB-VAR-DATA. 49 FILLER PIC G(32767) USAGE DISPLAY-1.[2] 49 FILLER PIC G(32767) USAGE DISPLAY-1. *Repeat 1248 times* ⋮ 49 FILLER PIC X(20480000-1250*32767) USAGE DISPLAY-1.
01 BLOB-LOC USAGE IS SQL TYPE IS BLOB-LOCATOR.	01 BLOB-LOC PIC S9(9) USAGE IS BINARY.
01 CLOB-LOC USAGE IS SQL TYPE IS CLOB-LOCATOR.	01 CLOB-LOC PIC S9(9) USAGE IS BINARY.
01 DBCLOB-LOC USAGE IS SQL TYPE IS DBCLOB-LOCATOR.	01 DBCLOB-LOC PIC S9(9) USAGE IS BINARY.

Notes to Table 13:

1. Because the COBOL language allows character declarations of no more than 32767 bytes, for BLOB or CLOB host variables that are greater than 32767 bytes in length, DB2 creates multiple host language declarations of 32767 or fewer bytes.
2. Because the COBOL language allows graphic declarations of no more than 32767 double-byte characters, for DBCLOB host variables that are greater than 32767 double-byte characters in length, DB2 creates multiple host language declarations of 32767 or fewer double-byte characters.

Declarations of LOB host variables in FORTRAN: Table 14 on page 130 shows FORTRAN declarations for some typical LOB types.

Table 14. Examples of FORTRAN variable declarations

You declare this variable	DB2 generates this variable
SQL TYPE IS BLOB(1M) blob_var	CHARACTER blob_var(1048580) INTEGER*4 blob_var_LENGTH CHARACTER blob_var_DATA EQUIVALENCE(blob_var(1), + blob_var_LENGTH) EQUIVALENCE(blob_var(5), + blob_var_DATA)
SQL TYPE IS CLOB(40000K) clob_var	CHARACTER clob_var(4096004) INTEGER*4 clob_var_length CHARACTER clob_var_data EQUIVALENCE(clob_var(1), + clob_var_length) EQUIVALENCE(clob_var(5), + clob_var_data)
SQL TYPE IS BLOB_LOCATOR blob_loc	INTEGER*4 blob_loc
SQL TYPE IS CLOB_LOCATOR clob_loc	INTEGER*4 clob_loc

Declarations of LOB host variables in PL/I: Table 15 on page 131 shows PL/I declarations for some typical LOB types.

Table 15. Examples of PL/I variable declarations

You declare this variable	DB2 generates this variable
DCL BLOB_VAR SQL TYPE IS BLOB (1M);	DCL 1 BLOB_VAR, 2 BLOB_VAR_LENGTH FIXED BINARY(31), 2 BLOB_VAR_DATA,[1] 3 BLOB_VAR_DATA1(32) CHARACTER(32767), 3 BLOB_VAR_DATA2 CHARACTER(1048576-32*32767);
DCL CLOB_VAR SQL TYPE IS CLOB (40000K);	DCL 1 CLOB_VAR, 2 CLOB_VAR_LENGTH FIXED BINARY(31), 2 CLOB_VAR_DATA,[1] 3 CLOB_VAR_DATA1(1250) CHARACTER(32767), 3 CLOB_VAR_DATA2 CHARACTER(40960000-1250*32767);
DCL DBCLOB_VAR SQL TYPE IS DBCLOB (4000K);	DCL 1 DBCLOB_VAR, 2 DBCLOB_VAR_LENGTH FIXED BINARY(31), 2 DBCLOB_VAR_DATA,[2] 3 DBCLOB_VAR_DATA1(2500) GRAPHIC(16383), 3 DBCLOB_VAR_DATA2 GRAPHIC(40960000-2500*16383);
DCL BLOB_LOC SQL TYPE IS BLOB_LOCATOR;	DCL BLOB_LOC FIXED BINARY(31);
DCL CLOB_LOC SQL TYPE IS CLOB_LOCATOR;	DCL CLOB_LOC FIXED BINARY(31);
DCL DBCLOB_LOC SQL TYPE IS DBCLOB_LOCATOR;	DCL DBCLOB_LOC FIXED BINARY(31);

Notes to Table 15:

1. Because the PL/I language allows character declarations of no more than 32767 bytes, for BLOB or CLOB host variables that are greater than 32767 bytes in length, DB2 creates host language declarations as follows:
   - If the length of the LOB is greater than 32767 bytes and evenly divisible by 32767, DB2 creates an array of 32767-byte strings. The dimension of the array is *length*/32767.
   - If the length of the LOB is greater than 32767 bytes but not evenly divisible by 32767, DB2 creates two declarations: The first is an array of 32767 byte strings, where the dimension of the array, *n*, is *length*/32767. The second is a character string of length *length-n**32767.
2. Because the PL/I language allows graphic declarations of no more than 16383 double-byte characters, for DBCLOB host variables that have more than 16383 characters, DB2 creates host language declarations as follows:
   - If the length of the LOB is greater than 16383 characters and evenly divisible by 16383, DB2 creates an array of 16383-character strings. The dimension of the array is *length*/16383.
   - If the length of the LOB is greater than 16383 characters but not evenly divisible by 16383, DB2 creates two declarations: The first is an array of 16383 byte strings, where the dimension of the array, *m*, is *length*/16383. The second is a character string of length *length-m**16383.

LOB materialization

LOB materialization means that DB2 places a LOB value into contiguous storage in a data space. Because LOB values can be very large, DB2 avoids materializing LOB data until absolutely necessary. However, DB2 must materialize LOBs when your application program:

- Calls a user-defined function with a LOB as an argument
- Moves a LOB into or out of a stored procedure
- Assigns a LOB host variable to a LOB locator host variable
- Converts a LOB from one CCSID to another

Data spaces for LOB materialization: The amount of storage that is used in data spaces for LOB materialization depends on a number of factors including:

- The size of the LOBs
- The number of LOBs that need to be materialized in a statement

DB2 allocates a certain number of data spaces for LOB materialization. If insufficient space is available in a data space for LOB materialization, your application receives SQLCODE -904.

Although you cannot completely avoid LOB materialization, you can minimize it by using LOB locators, rather than LOB host variables in your application programs. See “Using LOB locators to save storage” for information on how to use LOB locators.

Using LOB locators to save storage

To retrieve LOB data from a DB2 table, you can define host variables that are large enough to hold all of the LOB data. This requires your application to allocate large amounts of storage, and requires DB2 to move large amounts of data, which can be inefficient or impractical. Instead, you can use LOB locators. LOB locators let you manipulate LOB data without retrieving the data from the DB2 table. Using LOB locators for LOB data retrieval is a good choice in the following situations:

- When you move only a small part of a LOB to a client program
- When the entire LOB does not fit in the application's memory
- When the program needs a temporary LOB value from a LOB expression but does not need to save the result
- When performance is important

A LOB locator is associated with a LOB value or expression, not with a row in a DB2 table or a physical storage location in a table space. Therefore, after you select a LOB value using a locator, the value in the *locator* normally does not change until the current unit of work ends. However, the value of the LOB can change.

If you want to remove the association between a LOB locator and its value before a unit of work ends, execute the FREE LOCATOR statement. To keep the association between a LOB locator and its value after the unit of work ends, execute the HOLD LOCATOR statement. After you execute a HOLD LOCATOR statement, the locator keeps the association with the corresponding value until you execute a FREE LOCATOR statement or the program ends.

If you execute HOLD LOCATOR or FREE LOCATOR dynamically, you cannot use EXECUTE IMMEDIATE. For more information on the HOLD LOCATOR and FREE LOCATOR statements, see *DB2 SQL Reference*.

Deferring evaluation of a LOB expression to improve performance

DB2 moves no bytes of a LOB value until a program assigns a LOB expression to a target destination. This means that when you use a LOB locator with string functions and operators, you can create an expression that DB2 does not evaluate until the time of assignment. This is called *deferring evaluation* of a LOB expression. Deferring evaluation can improve LOB I/O performance.

The following example is a C language program that defers evaluation of a LOB expression. The program runs on a client and modifies LOB data at a server. The program searches for a particular resume (EMPNO = '000130') in the EMP_RESUME table. It then uses LOB locators to rearrange a copy of the resume (with EMPNO = 'A00130'). In the copy, the Department Information Section appears at the end of the resume. The program then inserts the copy into EMP_RESUME without modifying the original resume.

Because the program uses LOB locators, rather than placing the LOB data into host variables, no LOB data is moved until the INSERT statement executes. In addition, no LOB data moves between the client and the server.

```
EXEC SQL INCLUDE SQLCA;

/*************************/
/* Declare host variables */                                  1
/*************************/
EXEC SQL BEGIN DECLARE SECTION;
   char userid[9];
   char passwd[19];
   long            HV_START_DEPTINFO;
   long            HV_START_EDUC;
   long            HV_RETURN_CODE;
   SQL TYPE IS CLOB_LOCATOR HV_NEW_SECTION_LOCATOR;
   SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR1;
   SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR2;
   SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR3;
EXEC SQL END DECLARE SECTION;
/************************************************/
/* Delete any instance of "A00130" from previous */
/* executions of this sample                     */
/************************************************/
EXEC SQL DELETE FROM EMP_RESUME WHERE EMPNO = 'A00130';

/************************************************/
/* Use a single row select to get the document   */          2
/************************************************/
EXEC SQL SELECT RESUME
           INTO :HV_DOC_LOCATOR1
           FROM EMP_RESUME
          WHERE EMPNO = '000130'
            AND RESUME_FORMAT = 'ascii';
/*****************************************************/
/* Use the POSSTR function to locate the start of    */
/* sections "Department Information" and "Education" */      3
/*****************************************************/
EXEC SQL SET :HV_START_DEPTINFO =
          POSSTR(:HV_DOC_LOCATOR1, 'Department Information');

EXEC SQL SET :HV_START_EDUC =
             POSSTR(:HV_DOC_LOCATOR1, 'Education');

/*****************************************************/
/* Replace Department Information section with nothing */
/*****************************************************/
EXEC SQL SET :HV_DOC_LOCATOR2 =
      SUBSTR(:HV_DOC_LOCATOR1, 1, :HV_START_DEPTINFO -1)
   || SUBSTR (:HV_DOC_LOCATOR1, :HV_START_EDUC);
```

Figure 21 (Part 1 of 2). Example of deferring evaluation of LOB expressions

```
/*******************************************************/
/* Associate a new locator with the Department         */
/* Information section                                 */
/*******************************************************/
EXEC SQL SET :HV_NEW_SECTION_LOCATOR =
    SUBSTR(:HV_DOC_LOCATOR1, :HV_START_DEPTINFO,
   :HV_START_EDUC -:HV_START_DEPTINFO);

/*******************************************************/
/* Append the Department Information to the end        */
/* of the resume                                       */
/*******************************************************/
EXEC SQL SET :HV_DOC_LOCATOR3 =
  :HV_DOC_LOCATOR2 || :HV_NEW_SECTION_LOCATOR;
/*******************************************************/
/* Store the modified resume in the table.  This is    */ 4
/* where the LOB data really moves.                    */
/*******************************************************/
EXEC SQL INSERT INTO EMP_RESUME VALUES ('A00130', 'ascii',
   :HV_DOC_LOCATOR3, DEFAULT);

/*********************/
/* Free the locators */                                  5
/*********************/
EXEC SQL FREE LOCATOR :HV_DOC_LOCATOR1, :HV_DOC_LOCATOR2, :HV_DOC_LOCATOR3;
```

Figure 21 (Part 2 of 2). Example of deferring evaluation of LOB expressions

Notes on Figure 21 on page 134:

1. Declare the LOB locators here.
2. This SELECT statement associates LOB locator HV_DOC_LOCATOR1 with the value of column RESUME for employee number 000130.
3. The next five SQL statements use LOB locators to manipulate the resume data without moving the data.
4. Evaluation of the LOB expressions in the previous statements has been deferred until execution of this INSERT statement.
5. Free all LOB locators to release them from their associated values.

Indicator variables and LOB locators

For host variables other than LOB locators, when you select a null value into a host variable, DB2 assigns a negative value to the associated indicator variable. However, for LOB locators, DB2 uses indicator variables differently. A LOB locator is never null. When you select a LOB column using a LOB locator and the LOB column contains a null value, DB2 assigns a negative value to the associated indicator variable. In a client/server environment, this null information is recorded only at the client.

When you use LOB locators to retrieve data from columns that can contain null values, define indicator variables for the LOB locators, and check the indicator variables after you fetch data into the LOB locators. If an indicator variable is null after a fetch operation, you cannot use the value in the LOB locator.

Valid assignments for LOB locators

Although you usually use LOB locators for assigning data to and retrieving data from LOB columns, you can also use LOB locators to assign data to CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC columns. However, you cannot fetch data from CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC columns into LOB locators.

LOB system processing

The section discusses the following system-related aspects of LOBs:

- "Managing buffer pools for LOBs"
- "Locking LOBs"

Managing buffer pools for LOBs

Put LOB data in buffer pools that are not shared with other data. For both LOG YES and LOG NO LOBs, use a deferred write threshold (DWQT) of 0. LOBs specified with LOG NO have their changed pages written at commit time (*force-at-commit* processing). If you set DWQT to 0, those writes happen continuously in the background rather than in a large surge at commit.

LOBs defined with LOG YES can use deferred write, but by setting DWQT to 0, you can avoid massive writes at DB2 checkpoints.

Locking LOBs

Terminology: A lock that is held on a LOB value in a LOB table space is called a **LOB lock**.

In this section: The following topics are described:

- "Relationship between transaction locks and LOB locks"
- "Hierarchy of LOB locks" on page 138
- "LOB lock modes" on page 138
- "Duration of locks" on page 139
- "When locks on the LOB table space are not taken" on page 140
- "Controlling the number of locks" on page 140
- "The LOCK TABLE statement" on page 140
- "The LOCKSIZE clause for LOB table spaces" on page 141

Relationship between transaction locks and LOB locks

As described in "Introduction to defining LOBs" on page 124, LOB column values are stored in a different table space, a LOB table space, from the values in the base table. An application that reads or updates a row in a table that contains LOB columns obtains its normal transaction locks on the base table. The locks on the base table also control concurrency for the LOB table space. When locks are not acquired on the base table, such as for ISO(UR), DB2 maintains data consistency by using locks on the LOB table space.

DB2 also obtains locks on the LOB table space and the LOB values stored in that LOB table space, but the primary purpose of those locks is:

- To determine whether space from a deleted LOB can be reused by an inserted or updated LOB

  Storage for a deleted LOB is not reused until no more readers (including held locators) are on the LOB and the delete operation has been committed.

- To prevent deallocating space for a LOB that is currently being read

  A LOB can be deleted from one application's point-of-view while a reader from another application is reading the LOB. The reader continues reading the LOB because all readers, including those readers that are using uncommitted read isolation, acquire S-locks on LOBs to prevent the storage for the LOB they are reading from being deallocated. That lock is held until commit. A held LOB locator also causes the LOB lock and LOB table space lock to be held past commit.

In summary, the main purpose of LOB locks is for managing the space used by LOBs and to ensure that LOB readers do not read partially updated LOBs. Applications need to free held locators so that the space can be reused.

Table 16 shows the relationship between the action that is occurring on the LOB value and the associated LOB table space and LOB locks that are acquired.

Table 16. Locks that are acquired for operations on LOBs. This table does not account for gross locks that can be taken because of LOCKSIZE TABLESPACE, the LOCK TABLE statement, or because of lock escalation.

Action on LOB value	LOB table space lock	LOB lock	Comment
Read (including UR)	IS	S	Prevents storage from being reused while the LOB is being read or while locators are referencing the LOB
Insert	IX	X	Prevents other processes from seeing a partial LOB
Delete	IS	S	To hold space in case the delete is rolled back. (The X is on the base table row or page.) Storage is not reusable until the delete is committed and there are no other readers of the LOB.
Update	IS->IX	Two LOB locks: an S-lock for the delete and an X-lock for the insert.	Operation is a delete followed by an insert.
Update the LOB to null or zero-length	IS	S	No insert, just a delete.
Update a null or zero-length LOB to a value	IX	X	No delete, just an insert.

UR readers: When an application is reading rows using uncommitted read or lock avoidance, no page or row locks are taken on the base table. Therefore, these

readers must take an S LOB lock to ensure that they are not reading a partial LOB or a LOB value that is inconsistent with the base row.

Hierarchy of LOB locks

Just as there is a hierarchical relationship between page locks (or row locks) and table space locks, there is a hierarchical relationship between LOB locks and locks on LOB table spaces. If the LOB table space is locked with a gross lock, then LOB locks are not acquired. In a data sharing environment, the lock on the LOB table space is used to determine whether the lock on the LOB must be propagated beyond the local IRLM.

LOB lock modes

The modes of locks that can be acquired on LOB table spaces and on the LOBs in those table spaces are listed below.

Modes of LOB locks

The following LOB lock modes are possible:

S (SHARE) The lock owner and any concurrent processes can read, update, or delete the locked LOB. Concurrent processes can acquire an S lock on the LOB. The purpose of the S lock is to reserve the space used by the LOB.

X (EXCLUSIVE)
The lock owner can read or change the locked LOB. Concurrent processes cannot access the LOB.

Modes of LOB table space locks

The following lock modes are possible on the LOB table space:

IS (INTENT SHARE)
The lock owner can update LOBs to null or zero-length, or read or delete LOBs in the LOB table space. Concurrent processes can both read and change LOBs in the same table space. The lock owner acquires a LOB lock on any data that it reads or deletes.

IX (INTENT EXCLUSIVE)
The lock owner and concurrent processes can read and change data in the LOB table space. The lock owner acquires a LOB lock on any data it accesses.

S (SHARE) The lock owner and any concurrent processes can read and delete LOBs in the LOB table space. The lock owner does not need LOB locks.

SIX (SHARE with INTENT EXCLUSIVE)
The lock owner can read and change data in the LOB table space. If the lock owner is inserting (INSERT or UPDATE), the lock owner obtains a LOB lock. Concurrent processes can read or delete data in the LOB table space (or update to a null or zero-length LOB).

X (EXCLUSIVE)
The lock owner can read or change LOBs in the LOB table space. The lock owner does not need LOB locks.

Duration of locks

Duration of locks on LOB table spaces: Locks on LOB table spaces are acquired when they are needed; that is, the ACQUIRE option of BIND has no effect on when the table space lock on the LOB table space is taken. The table space lock is released according to the value specified on the RELEASE option of BIND (except when a cursor is defined WITH HOLD or if there is a held LOB locator).

Duration of LOB locks: Locks on LOBs are taken when they are needed and are usually released at commit. However, if that LOB value is assigned to a LOB locator, the S lock remains until the application commits.

If the application uses HOLD LOCATOR, the locator (and the LOB lock) is not freed until the first commit operation after a FREE LOCATOR statement is issued, or until the thread is deallocated.

A note about held cursors: If a cursor is defined WITH HOLD, LOB locks are held through commit operations.

A note about INSERT with subselect: Because LOB locks are held until commit, it is possible that a statement such as an INSERT with a subselect that involves LOB columns can accumulate many more locks than a similar statement that does not involve LOB columns. To prevent system problems caused by too many locks, you can:

- Ensure that you have lock escalation enabled for the LOB table spaces that are involved in the INSERT. In other words, make sure that LOCKMAX is non-zero for those LOB table spaces.
- Alter the LOB table space to change the LOCKSIZE to TABLESPACE before executing the INSERT with subselect.
- Use the LOCK TABLE statement to lock the LOB table space.

When locks on the LOB table space are not taken

A lock might not be acquired on a LOB table space at all. For example, if a row is deleted from a table, and the value of the LOB column is null, the LOB table space associated with that LOB column is not locked. DB2 does not access the LOB table space if the application:

- Selects a LOB that is null or zero length
- Deletes a row where the LOB is null or zero length
- Inserts a null or zero length LOB
- Updates a null or zero-length LOB to null or zero-length

Controlling the number of locks

This section describes how you can control the number of LOB locks that are taken.

Controlling the number of LOB locks that are acquired for a user: LOB locks are counted toward the total number of locks allowed per user. Control this number by the value you specify on the LOCKS PER USER field of installation panel DSNTIPJ. The number of LOB locks that are acquired during a unit of work is reported in IFCID 0020.

Controlling LOB lock escalation: As with any table space, use the LOCKMAX clause of the CREATE or ALTER TABLESPACE statement to control the number of LOB locks that are acquired within a particular LOB table space before the lock is escalated. When the number of LOB locks reaches the maximum you specify in the LOCKMAX clause, the LOB locks escalate to a gross lock on the LOB table space, and the LOB locks are released.

Information about LOB locks and lock escalation is reported in IFCID 0020.

The LOCK TABLE statement

The reasons for using LOCK TABLE on an auxiliary table are somewhat different than that for regular tables.

- You can use LOCK TABLE to control the number of locks acquired on the auxiliary table.
- You can use LOCK TABLE IN SHARE MODE to prevent other applications from inserting LOBs.

  With auxiliary tables, LOCK TABLE IN SHARE MODE does not prevent any changes to the auxiliary table. The statement does prevent LOBs from being inserted into the auxiliary table, but it does not prevent deletes. Updates are generally restricted also, except where the LOB is updated to a null value or a zero-length string.

- You can use LOCK TABLE IN EXCLUSIVE MODE to prevent other applications from accessing LOBs.

  With auxiliary tables, LOCK TABLE IN EXCLUSIVE MODE also prevents access from uncommitted readers.
- Either statement eliminates the need for lower-level LOB locks.

The LOCKSIZE clause for LOB table spaces

The LOCKSIZE TABLE, PAGE, and ROW options are not valid for LOB table spaces. The other options act as follows:

LOCKSIZE TABLESPACE
A process acquires no LOB locks.

LOCKSIZE ANY
DB2 chooses the size of the lock. For a LOB table space, this is usually LOCKSIZE LOB.

LOCKSIZE LOB
If a LOB must be accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX).

Recovering table spaces that contain LOBs

In general, planning for LOB recovery is similar to planning for recovery of databases with referential constraints— a *relationship* exists between a table with a LOB column and the associated LOB table space. A LOB table space and its associated base table space are parts of a table space set that should be recovered together.

The biggest consideration in planning for recovery of LOBs is that LOBs can be very large. You need to take this fact into account when you create your LOB table spaces. The CREATE LOB TABLESPACE statement contains the LOG parameter that gives you the choice of whether to log changes to the LOB table space. (The changes to the indicator column in the base table are always logged, along with the rest of the data in the base table.)

If you specify LOG YES, performance of your database system can degrade significantly when you insert or update LOBs. However, with LOG NO, database writes are forced to DASD at commit points, which can also have a severe impact on performance.

If you specify LOG NO, and you need to recover the LOB table space, changes to the LOB values after the most recent image copy are lost. If you specify GBPCACHE CHANGED and LOG NO, the changed pages are written to the group buffer pool. See “Procedure for recovering invalid LOBs” on page 142 for information on what you can do to recover the LOB data.

Recovering to a prior point in time

You can recover a LOB table space to a prior point in time by using any of the following methods:

- Use TOLOGPOINT, TORBA, or TOCOPY options of the RECOVER utility.
- Run the QUIESCE utility with the TABLESPACESET option and recover to a common quiesce point.

- Take an image copy with SHRLEVEL REFERENCE and then run the RECOVER utility.

If you do not have copies of the LOB table space and the related base table space at the quiesce point, DB2 puts the table spaces in a check pending state.

If the LOB table space is defined with LOG NO, then any changes to the LOB table space between its last image copy and the recovery point have not been logged. If DB2 discovers any missing log records during the recovery process, it marks the individual LOB value as invalid and sets the Aux Warning status (AUXW) on the LOB table space. Run the CHECK LOB utility to find any invalid LOBs.

Recovering to the current point in time

Recovery to the current point in time for a base table that contains a LOB column is the same as the recovery of any other kind of table. DB2 applies the appropriate image copy to the base table space and then applies log records from the date of the image copy to the present. The same is true for the LOB table space. However, if the LOB table space is defined with LOG NO, then any changes to the LOB table space between its last image copy and the recovery point have not been logged. If DB2 discovers any missing log records during the recovery process, it marks the individual LOB value as invalid and sets the Aux Warning status (AUXW) on the LOB table space. Other LOB values remain accessible.

Recovering LOB pages on the logical page list

You recover logical page list (LPL) entries for a LOB table space the same as for any other table space: Execute the START DATABASE command with the SPACENAM option. If the LOB table space is defined with LOG NO and missing log data is needed for the LPL recovery, DB2 sets the Aux Warning status on the LOB table space. You then need to run the CHECK LOB utility to identify which LOBs are invalid.

Procedure for recovering invalid LOBs

Unless your LOBs are fairly small, specifying LOG NO for LOB objects is recommended. The performance cost of logging exceeds the benefits you can receive from logging such large amounts of data. If no changes are made to LOB data, this is not an issue. However, you should make image copies of the LOB table space to prepare for failures. The frequency with which you make image copies is based on how often you update LOB data.

If you need to recover LOB data that changed after your last image copy, follow this procedure:

1. Run the RECOVER utility as you do for other table spaces:

   ```
   RECOVER TABLESPACE dbname.lobts
   ```

   If changes were made after the image copy, DB2 puts the table space in *Aux Warning* status. The purpose of this status is let you know that some of your LOBs are invalid. Applications that try to retrieve the values of those LOBs will receive SQLCODE -904. Applications can still access other LOBs in the LOB table space.

2. Get a report of the invalid LOBs by running CHECK LOB on the LOB table space:

   ```
   CHECK LOB TABLESPACE dbname.lobts
   ```

   DB2 generates messages like the following one:

```
LOB WITH ROWID = 'xxxxxxx' VERSION = n  IS INVALID
```

3. Fix the invalid LOBs, by updating the LOBs or setting them to the null value. For example, suppose you determine from the CHECK LOB utility that the row of the EMP_PHOTO_RESUME table with ROWID X'C1BDC4652940D40A81C201AA0A28' has an invalid value for column RESUME. If host variable hvlob contains the correct value for RESUME, you can use this statement to correct the value:

```
UPDATE DSN8610.EMP_PHOTO_RESUME
  SET RESUME = :hvlob
  WHERE EMP_ROWID = ROWID(X'C1BDC4652940D40A81C201AA0A28')
;
```

Creating and using user-defined functions

A user-defined function (UDF) is an extension to the SQL language. A user-defined function is similar to a host language subprogram or function. However, a user-defined function is often the better choice for an SQL application because you can invoke a user-defined function in an SQL statement.

This chapter presents the following information about user-defined functions:

- "Overview of user-defined function definition, implementation, and invocation"
- "Defining a user-defined function" on page 146
- "Implementing an external user-defined function" on page 150
- "Invoking a user-defined function" on page 192

Overview of user-defined function definition, implementation, and invocation

The two types of user-defined functions are:

- *Sourced* user-defined functions, which are based on existing built-in functions or user-defined functions
- *External* user-defined functions, which a programmer writes in a host language

User-defined functions can also be categorized as *user-defined scalar functions* or a *user-defined table functions*:

- A user-defined scalar function returns a single-value answer each time it is invoked.
- A user-defined table function returns a table to the SQL statement that references it.

External user-defined functions can be user-defined scalar functions or user-defined table functions. Sourced user-defined functions cannot be user-defined table functions.

Creating and using a user-defined function involves these steps:

- Setting up the environment for user-defined functions

  A system administrator probably performs this step. The user-defined function environment is shown in Figure 22 on page 144. The steps for setting up and maintaining the user-defined function environment are the same as for setting up and maintaining the environment for stored procedures in WLM-established address spaces.

- Writing and preparing the user-defined function

  This step is necessary only for an external user-defined function.

  The person who performs this step is called the user-defined function *implementer*.
- Defining the user-defined function to DB2

  The person who performs this step is called the user-defined function *definer*.
- Invoking the user-defined function from an SQL application

  The person who performs this step is called the user-defined function *invoker*.

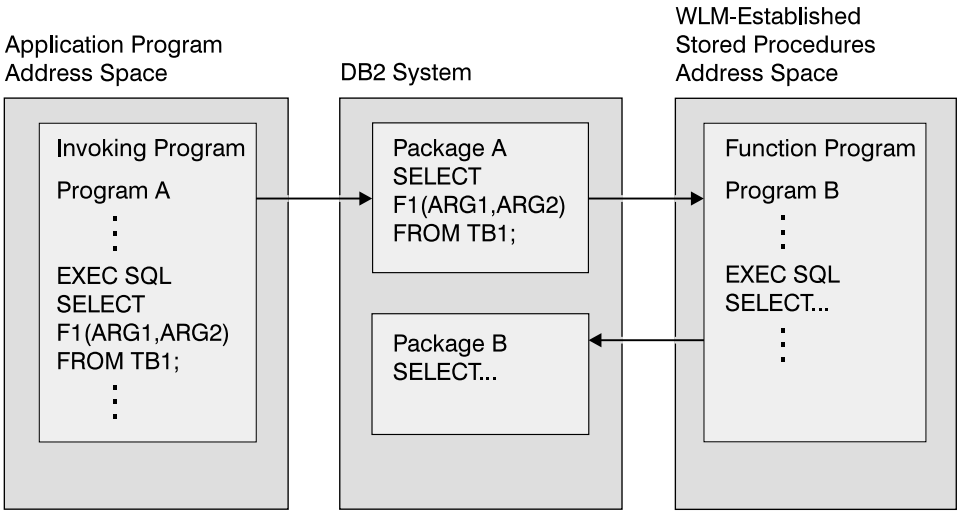


Figure 22. The user-defined function environment

Example of creating and using a user-defined scalar function

Suppose that your organization needs a user-defined scalar function that calculates the bonus that each employee receives. All employee data, including salaries, commissions, and bonuses, is kept in the employee table, EMP. The input fields for the bonus calculation function are the values of the SALARY and COMM columns. The output from the function goes into the BONUS column. Because this function gets its input from a DB2 table and puts the output in a DB2 table, the most convenient method to manipulate the data is through a user-defined function.

The user-defined function's definer and invoker determine that this new user-defined function should have these characteristics:

- The user-defined function name is CALC_BONUS.
- The two input fields are of type DECIMAL(9,2).
- The output field is of type DECIMAL(9,2).
- The user-defined function is written in COBOL, and the load module is named CBONUS.

Because no built-in or user-defined function exists on which to build a sourced user-defined function, the function implementer must write an external user-defined function. The implementer performs the following steps:

- Writes the user-defined function, which is a COBOL program
- Precompiles, compiles, and links the program

- Binds a package if the user-defined function contains SQL statements
- Tests the program thoroughly
- Grants execute authority on the user-defined function package to the definer

The user-defined function definer executes this CREATE FUNCTION statement to register CALC_BONUS to DB2:

```
CREATE FUNCTION CALC_BONUS(DECIMAL(9,2),DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  EXTERNAL NAME 'CBONUS'
  PARAMETER STYLE DB2SQL
  LANGUAGE COBOL;
```

The definer then grants execute authority on CALC_BONUS to all invokers.

User-defined function invokers write and prepare application programs that invoke CALC_BONUS. An invoker might write a statement like this, which uses the user-defined function to update the BONUS field in the employee table:

```
 UPDATE EMP
   SET BONUS = CALC_BONUS(SALARY,COMM);
```

An invoker can execute this statement either statically or dynamically.

User-defined function samples shipped with DB2

To assist you in defining, implementing, and invoking your user-defined functions, DB2 provides a number of sample user-defined functions. All user-defined function code is in data set DSN610.SDSNSAMP.

Table 17 summarizes the characteristics of the sample user-defined functions.

Table 17 (Page 1 of 2). User-defined function samples shipped with DB2

User-defined function name	Language	Member that contains source code	Purpose
ALTDATE[1]	C	DSN8DUAD	Converts the current date to a user-specified format
ALTDATE[2]	C	DSN8DUCD	Converts a date from one format to another
ALTTIME[3]	C	DSN8DUAT	Converts the current time to a user-specified format
ALTTIME[4]	C	DSN8DUCT	Converts a time from one format to another
DAYNAME	C++	DSN8EUDN	Returns the day of the week for a user-specified date
MONTHNAME	C++	DSN8EUMN	Returns the month for a user-specified date
CURRENCY	C	DSN8DUCY	Formats a floating-point number as a currency value
TABLE_NAME	C	DSN8DUTI	Returns the unqualified table name for a table, view, or alias

Table 17 (Page 2 of 2). User-defined function samples shipped with DB2

User-defined function name	Language	Member that contains source code	Purpose
TABLE_SCHEMA	C	DSN8DUTI	Returns the schema for a table, view, or alias
TABLE_LOCATION	C	DSN8DUTI	Returns the location for a table, view, or alias
WEATHER	C	DSN8DUWC	Returns a table of weather information from an EBCDIC data set

Notes to Table 17 on page 145:

1. This version of ALTDATE has one input parameter, of type VARCHAR(13).
2. This version of ALTDATE has three input parameters, of type VARCHAR(17), VARCHAR(13), and VARCHAR(13).
3. This version of ALTTIME has one input parameter, of type VARCHAR(14).
4. This version of ALTTIME has three input parameters, of type VARCHAR(11), VARCHAR(14), and VARCHAR(14).

Member DSNTEJ2U shows you how to define and prepare the sample user-defined functions.

Defining a user-defined function

Before you can define a user-defined function to DB2, you must determine the characteristics of the user-defined function, such as the user-defined function name, schema (qualifier), and number and data types of the input parameters and types of the values returned. Then you execute a CREATE FUNCTION statement to register the information in the DB2 catalog. If you discover after you define the function that any of these characteristics is not appropriate for the function, you can use an ALTER FUNCTION statement to change information in the definition. For more information about the options supported by the ALTER FUNCTION and CREATE FUNCTION statements, see Chapter 6 of *DB2 SQL Reference*.

Components of a user-defined function definition

The characteristics you include in a CREATE FUNCTION or ALTER FUNCTION statement depend on whether the user-defined function is external or sourced. Table 18 lists the characteristics of a user-defined function, the corresponding parameters in the CREATE FUNCTION and ALTER FUNCTION statements, and which parameters are valid for sourced and external user-defined functions.

Table 18 (Page 1 of 3). Characteristics of a user-defined function

Characteristic	CREATE FUNCTION or ALTER FUNCTION parameter	Valid in sourced function?	Valid in external function?
User-defined function name	FUNCTION	Yes	Yes
Input parameter types	FUNCTION	Yes	Yes

Table 18 (Page 2 of 3). Characteristics of a user-defined function

Characteristic	CREATE FUNCTION or ALTER FUNCTION parameter	Valid in sourced function?	Valid in external function?
Output parameter types	RETURNS RETURNS TABLE[1]	Yes	Yes
Specific name	SPECIFIC	Yes	Yes
External name	EXTERNAL NAME	No	Yes
Language	LANGUAGE ASSEMBLE LANGUAGE C LANGUAGE COBOL LANGUAGE PLI	No	Yes
Deterministic or not deterministic	NOT DETERMINISTIC DETERMINISTIC	No	Yes
Types of SQL statements in the function	NO SQL CONTAINS SQL READS SQL DATA MODIFIES SQL DATA	No	Yes[2]
Name of source function	SOURCE	Yes	No
Parameter style	PARAMETER STYLE DB2SQL	No	Yes
Address space for user-defined functions	FENCED	No	Yes
Call with null input	RETURNS NULL ON NULL INPUT CALLED ON NULL INPUT	No	Yes
External actions	EXTERNAL ACTION NO EXTERNAL ACTION	No	Yes
Scratchpad specification	NO SCRATCHPAD SCRATCHPAD *length*	No	Yes
Call function after SQL processing	NO FINAL CALL FINAL CALL	No	Yes
Consider function for parallel processing	ALLOW PARALLEL DISALLOW PARALLEL	No	Yes[2]
Package collection	NO COLLID COLLID *collection-id*	No	Yes
WLM environment	WLM ENVIRONMENT *name* WLM ENVIRONMENT *name,**	No	Yes
CPU time for a function invocation	ASUTIME NO LIMIT ASUTIME LIMIT *integer*	No	Yes
Load module stays in memory	STAY RESIDENT NO STAY RESIDENT YES	No	Yes
Program type	PROGRAM TYPE MAIN PROGRAM TYPE SUB	No	Yes
Security	SECURITY DB2 SECURITY USER SECURITY DEFINER	No	Yes
Run-time options	RUN OPTIONS *options*	No	Yes
Pass DB2 environment information	NO DBINFO DBINFO	No	Yes

Table 18 (Page 3 of 3). Characteristics of a user-defined function

Characteristic	CREATE FUNCTION or ALTER FUNCTION parameter	Valid in sourced function?	Valid in external function?
Expected number of rows returned	CARDINALITY *integer*	No	Yes[1]

Notes to Table 18 on page 146:

1. RETURNS TABLE and CARDINALITY are valid only for user-defined table functions.
2. MODIFIES SQL DATA and ALLOW PARALLEL are not valid for user-defined table functions.

For a complete explanation of the parameters in a CREATE FUNCTION or ALTER FUNCTION statement, see Chapter 6 of *DB2 SQL Reference*.

Examples of user-defined function definitions

Example: Definition for an external user-defined scalar function: A programmer has written a user-defined function that searches for a string of maximum length 200 in a CLOB value whose maximum length is 500 KB. The output from the user-defined function is of type float, but users require integer output for their SQL statements. The user-defined function is written in C and contains no SQL statements. This CREATE FUNCTION statement defines the user-defined function:

```
CREATE FUNCTION FINDSTRING (CLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC FINDSTRINCLOB
  EXTERNAL NAME 'FINDSTR'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  FENCED;
```

Example: Definition for an external user-defined scalar function that overloads an operator: A programmer has written a user-defined function that overloads the built-in SQL division operator (/). That is, this user-defined function is invoked when an application program executes a statement like either of the following:

```
UPDATE TABLE1 SET INTCOL1=INTCOL2/INTCOL3;

UPDATE TABLE1 SET INTCOL1="/"(INTCOL2,INTCOL3);
```

The user-defined function takes two integer values as input. The output from the user-defined function is of type integer. The user-defined function is in the MATH schema, is written in assembler, and contains no SQL statements. This CREATE FUNCTION statement defines the user-defined function:

```
CREATE FUNCTION MATH."/" (INT, INT)
  RETURNS INTEGER
  SPECIFIC DIVIDE
  EXTERNAL NAME 'DIVIDE'
  LANGUAGE ASSEMBLE
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  FENCED;
```

Suppose you want the FINDSTRING user-defined function to work on BLOB data types, as well as CLOB types. You can define another instance of the user-defined function that specifies a BLOB type as input:

```
CREATE FUNCTION FINDSTRING (BLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC FINDSTRINBLOB
  EXTERNAL NAME 'FNDBLOB'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  FENCED;
```

Each instance of FINDSTRING uses a different application program to implement the user-defined function.

Example: Definition for a sourced user-defined function: Suppose you need a user-defined function that finds a string in a value with a distinct type of BOAT. BOAT is based on a BLOB type. User-defined function FINDSTRING, which takes a BLOB type and performs the required function, has already been defined. You can therefore define a sourced user-defined function that is based on FINDSTRING to do the string search on values of type BOAT. This CREATE FUNCTION statement defines the sourced user-defined function:

```
CREATE FUNCTION FINDSTRING (BOAT, VARCHAR(200))
  RETURNS INTEGER
  SPECIFIC FINDSTRINBOAT
  SOURCE SPECIFIC FINDSTRINBLOB;
```

Example: Definition for a user-defined table function: An application programmer has written a user-defined function that receives two values and returns a table. The two input values are:

- A character string of maximum length 30 that describes a subject
- A character string of maximum length 255 that contains text to search for

The user-defined function scans documents on the subject for the search string and returns a list of documents that match the search criteria. The list is in the form of a one-column table. The column is a character column of length 16 that contains document IDs.

The user-defined function is written in COBOL, uses SQL only to perform queries, always produces the same output for given input, and should not execute as a parallel task. The program is reentrant, and successive invocations of the

user-defined function share information. You expect an invocation of the user-defined function to return about 20 rows.

The following CREATE FUNCTION statement defines the user-defined function:

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
  RETURNS TABLE (DOC_ID CHAR(16), DOC_ABSTRACT VARCHAR(5000))
   EXTERNAL NAME 'DOCMTCH'
  LANGUAGE COBOL
  PARAMETER STYLE DB2SQL
  READS SQL DATA
  DETERMINISTIC
  NO EXTERNAL ACTION
  FENCED
  SCRATCHPAD
  FINAL CALL
  DISALLOW PARALLEL
  CARDINALITY 20;
```

Implementing an external user-defined function

This section discusses these steps in implementing an external user-defined function:

- "Writing a user-defined function"
- "Preparing a user-defined function for execution" on page 188
- "Testing a user-defined function" on page 190

Writing a user-defined function

A user-defined function is similar to any other SQL program. When you write a user-defined function, you can include static or dynamic SQL statements, IFI calls, and DB2 commands that are issued through IFI calls.

Your user-defined function can also access remote data using the following methods:

- DB2 private protocol access using three-part names or aliases for three-part names
- DRDA access using three-part names or aliases for three-part names
- DRDA access using CONNECT or SET CONNECTION statements

The user-defined function and the application that calls it can access the same remote site if both use the same protocol.

You can write an external user-defined function in assembler, C, C++, COBOL, or PL/I. User-defined functions that are written in COBOL can include object-oriented extensions, just as other DB2 COBOL programs can.

The following sections include additional information that you need when you write a user-defined function:

- "Restrictions on user-defined function programs" on page 151
- "Writing your user-defined function as a main program or as a subprogram" on page 151
- "Parallelism considerations" on page 151
- "Passing parameter values to and from a user-defined function" on page 153
- "Examples of passing parameters in a user-defined function" on page 166
- "Using special registers in a user-defined function" on page 179

- "Using a scratchpad in a user-defined function" on page 181
- "Accessing transition tables in a user-defined function" on page 182

Restrictions on user-defined function programs: Observe these restrictions when you write a user-defined function:

- Because DB2 uses the Recoverable Resource Manager Services attachment facility (RRSAF) as its interface with your user-defined function, you must not include RRSAF calls in your user-defined function. DB2 rejects any RRSAF calls that it finds in a user-defined function.
- If your user-defined function is not defined with parameters SCRATCHPAD or EXTERNAL ACTION, the user-defined function is not guaranteed to execute under the same task each time it is invoked.
- You cannot execute COMMIT or ROLLBACK statements in your user-defined function.
- You must close all open cursors in a user-defined scalar function. DB2 returns an SQL error if a user-defined scalar function does not close all cursors before it completes.
- When you choose the language in which to write a user-defined function program, be aware of restrictions on the number of parameters that can be passed to a routine in that language. User-defined table functions in particular can require large numbers of parameters. Consult the programming guide for the language in which you plan to write the user-defined function for information on the number of parameters that can be passed.

Writing your user-defined function as a main program or as a subprogram: You can write your user-defined function as either a main program or a subprogram. The way in which you write your program must agree with the way in which you defined the user-defined function: with the PROGRAM TYPE MAIN parameter or the PROGRAM TYPE SUB parameter. The main difference is that when a main program starts, Language Environment allocates the application program storage that the external user-defined function uses. When a main program ends, Language Environment closes files and releases dynamically allocated storage.

If you write your user-defined function as a subprogram and manage the storage and files yourself, you can get better performance. The user-defined function should always free any allocated storage before it exits. To keep data between invocations of the user-defined function, use a scratchpad.

Requirement: You must write a user-defined table function that accesses external resources as a subprogram. Also ensure that the definer specifies the EXTERNAL ACTION parameter in the CREATE FUNCTION or ALTER FUNCTION statement. Program variables for a subprogram persist between invocations of the user-defined function, and use of the EXTERNAL ACTION parameter ensures that the user-defined function stays in the same address space from one invocation to another.

Parallelism considerations: If the definer specifies the parameter ALLOW PARALLEL in the definition of a user-defined scalar function, and the invoking SQL statement runs in parallel, the function can run under a parallel task. DB2 executes a separate instance of the user-defined function for each parallel task. When you

write your function program, you need to understand how the following parameter values interact with ALLOW PARALLEL so that you can avoid unexpected results:

- SCRATCHPAD

  When an SQL statement invokes a user-defined function that is defined with the ALLOW PARALLEL parameter, DB2 allocates one scratchpad for each parallel task of each reference to the function. This can lead to unpredictable or incorrect results.

  For example, suppose that the user-defined function uses the scratchpad to count the number of times it is invoked. If a scratchpad is allocated for each parallel task, this count is the number of invocations done by the *parallel task* and not for the entire SQL statement, which is not the desired result.

- FINAL CALL

  If a user-defined function performs an external action, such as sending a note, for each final call to the function, one note is sent for each parallel task instead of once for the function invocation.

- EXTERNAL ACTION

  Some user-defined functions with external actions can receive incorrect results if the function is executed by parallel tasks.

  For example, if the function sends a note for each initial call to the function, one note is sent for each parallel task instead of once for the function invocation.

- NOT DETERMINISTIC

  A user-defined function that is not deterministic can generate incorrect results if it is run under a parallel task.

  For example, suppose you execute the following query under parallel tasks:

  ```
  SELECT * FROM T1 WHERE C1 = COUNTER();
  ```

  COUNTER is a user-defined function that increments a variable in the scratchpad every time it is invoked. Counter is nondeterministic because the same input does not always produce the same output. Table T1 contains one column, C1, that has these values:

  ```
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
  ```

  When the query is executed with no parallelism, DB2 invokes COUNTER once for each row of table T1, and there is one scratchpad for counter, which DB2 initializes the first time that COUNTER executes. COUNTER returns 1 the first time it executes, 2 the second time, and so on. The result table for the query is therefore:

```
1
2
3
4
5
6
7
8
9
10
```

Now suppose that the query is run with parallelism, and DB2 creates three parallel tasks. DB2 executes the predicate WHERE C1 = COUNTER() for each parallel task. This means that each parallel task invokes its own instance of the user-defined function and has its own scratchpad. DB2 initializes the scratchpad to zero on the first call to the user-defined function for each parallel task.

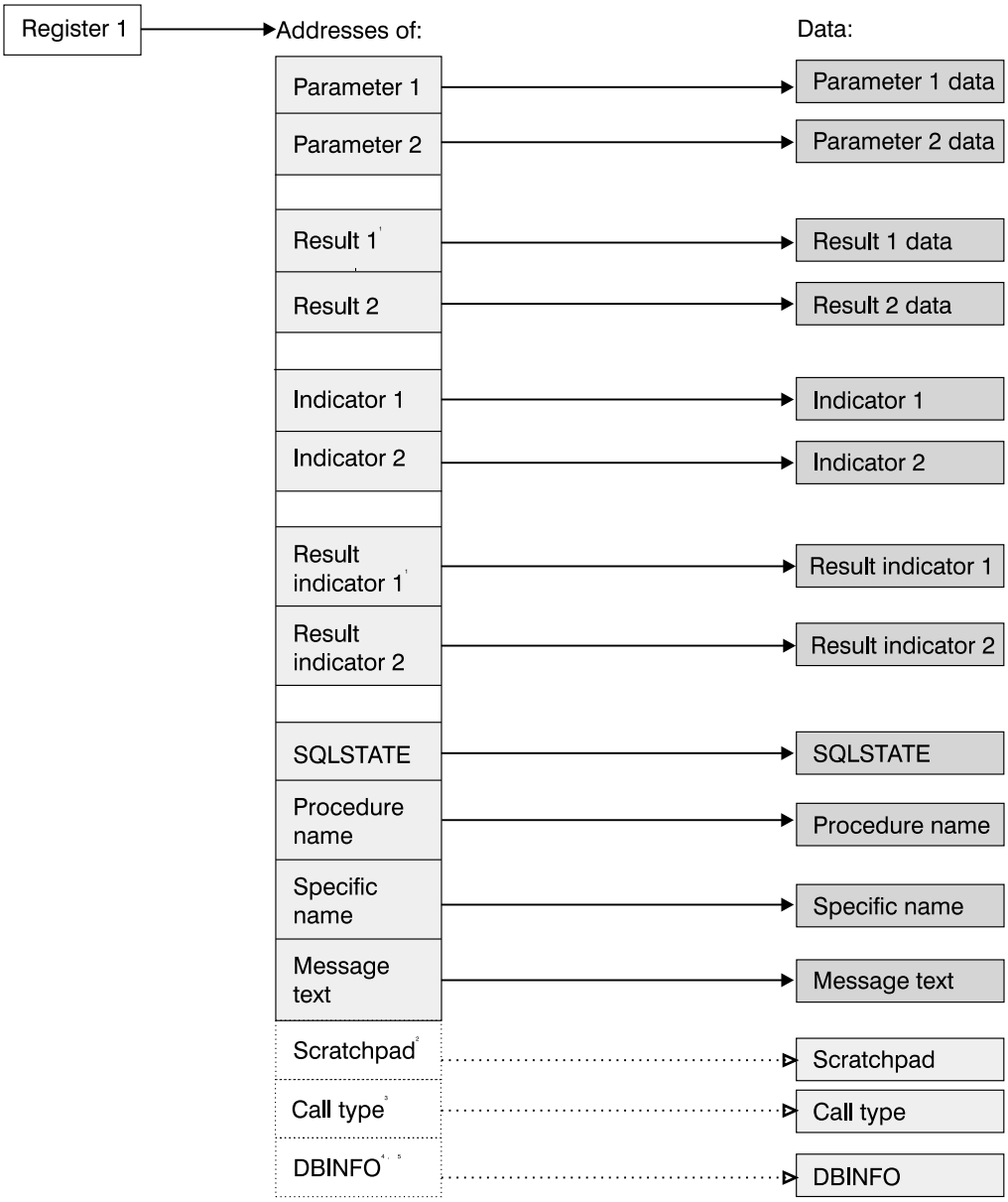
If parallel task 1 processes rows 1 to 3, parallel task 2 processes rows 4 to 6, and parallel task 3 processes rows 7 to 10, the following results occur:

- When parallel task 1 executes, C1 has values 1, 2, and 3, and COUNTER returns values 1, 2, and 3, so the query returns values 1, 2, and 3.
- When parallel task 2 executes, C1 has values 4, 5, and 6, but COUNTER returns values 1, 2, and 3, so the query returns no rows.
- When parallel task 3, executes, C1 has values 7, 8, 9, and 10, but COUNTER returns values 1, 2, 3, and 4, so the query returns no rows.

Thus, instead of returning the 10 rows that you might expect from the query, DB2 returns only 3 rows.

Passing parameter values to and from a user-defined function: To receive parameters from and pass parameters to a function invoker, you must understand the structure of the parameter list, the meaning of each parameter, and whether DB2 or your user-defined function sets the value of each parameter. This section explains the parameters and gives examples of how a user-defined function in each host language receives the parameter list.

Figure 23 on page 154 shows the structure of the parameter list that DB2 passes to a user-defined function. An explanation of each parameter follows.



[1] For a user-defined scalar function, only one result and one result indicator are passed.
[2] Passed if the SCRATCHPAD option is specified in the user-defined function definition.
[3] Passed if the FINAL CALL option is specified in a user-defined scalar function definition; always passed for a user-defined table function.
[4] For PL/I, this value is the address of a pointer to the DBINFO data..
[5] Passed if the DBINFO option is specified in the user-defined function definition

Figure 23. Parameter conventions for a user-defined function

Input parameter values: DB2 obtains the input parameters from the invoker's parameter list, and your user-defined function receives those parameters according to the rules of the host language in which the user-defined function is written. The number of input parameters is the same as the number of parameters in the user-defined function invocation. If one of the parameters in the function invocation is an expression, DB2 evaluates the expression and assigns the result of the expression to the parameter. To determine the data types to specify when you receive input parameters, check the input parameter data types in the user-defined

function definition. Then use Table 19 on page 155 and Table 20 on page 157 and to find the host data type that is compatible with each input parameter data type.

You can also retrieve information about the data types from the SYSPARMS catalog table if you have appropriate authorization.

Table 19 (Page 1 of 2). Compatible C and COBOL declarations for user-defined function parameters

SQL data type	C	COBOL
SMALLINT	short int	PIC S9(4) USAGE COMP
INTEGER	long int	PIC S9(9) USAGE COMP
DECIMAL(*p*,*s*)	decimal (*x*,*y*)[1]	PIC S9(p-s)V9(s) USAGE COMP-3
REAL	float	USAGE COMP-1
FLOAT	double	USAGE COMP-2
CHAR(1)	char	PIC X(1)
CHAR(n)	char var [n+1][2,3]	PIC X(n)
VARCHAR(n) FOR BIT DATA	struct {short int var_len; char var_data[n]; } var;[3]	01 var. 49 var-LEN PIC 9(4) USAGE COMP. 49 var-TEXT PIC X(n).
VARCHAR(n)	char var [n+1][2,3]	01 var. 49 var-LEN PIC 9(4) USAGE COMP. 49 var-TEXT PIC X(n).
GRAPHIC(1)	wchar_t var	PIC G(1) USAGE DISPLAY-1. or PIC N(1).
GRAPHIC(n)	wchar_t var [n+1][2]	PIC G(n) USAGE DISPLAY-1. PIC N(n).
VARGRAPHIC(n)	struct {short int var_len; wchar_t var_data[n]; } var;	01 var. 49 var-LEN PIC 9(4) USAGE COMP. 49 var-TEXT PIC G(n) USAGE DISPLAY-1. or 01 var. 49 var-LEN PIC 9(4) USAGE COMP. 49 var-TEXT PIC N(n).
TIME	char var [8+1][2,3]	PIC X(8)
DATE	char var [10+1][2,3]	PIC X(10)
TIMESTAMP	char var [26+1][2,3]	PIC X(26)
ROWID	struct {short int length; char data[40];} var;[3]	01 var. 49 var-LEN PIC 9(4) USAGE COMP. 49 var-DATA PIC X(40).
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	unsigned long[4]	01 var PIC S9(9) USAGE IS BINARY.[4]

Table 19 (Page 2 of 2). Compatible C and COBOL declarations for user-defined function parameters

SQL data type	C	COBOL
BLOB(n)	struct {unsigned long length; char data[n]; } var;	If n <= 32767: 01 var. 49 var-LENGTH PIC 9(9) USAGE COMP. 49 var-DATA PIC X(n). If length > 32767: 01 var. 02 var-LENGTH PIC S9(9) USAGE COMP. 02 var-DATA. 49 FILLER PIC X(32767). 49 FILLER PIC X(32767). ⋮ 49 FILLER PIC X(mod(n,32767)).
CLOB(n)	struct {unsigned long length; char data[n]; } var;	If n <= 32767: 01 var. 49 var-LENGTH PIC 9(9) USAGE COMP. 49 var-DATA PIC X(n). If length > 32767: 01 var. 02 var-LENGTH PIC S9(9) USAGE COMP. 02 var-DATA. 49 FILLER PIC X(32767). 49 FILLER PIC X(32767). ⋮ 49 FILLER PIC X(mod(n,32767)).
DBCLOB(n)	struct {unsigned long length; wchar_t data[n]; } var;	If n <eq; 32767: 01 var. 49 var-LENGTH PIC 9(9) USAGE COMP. 49 var-DATA PIC G(n) USAGE DISPLAY-1. If length > 32767: 01 var. 02 var-LENGTH PIC S9(9) USAGE COMP. 02 var-DATA. 49 FILLER PIC G(32767) USAGE DISPLAY-1. 49 FILLER PIC G(32767). USAGE DISPLAY-1. ⋮ 49 FILLER PIC G(mod(n,32767)) USAGE DISPLAY-1.

Notes to Table 19 on page 155:

1. *x* is the total number of digits. In SQL, this is the precision of the number; in C, it is the size of the number. *y* is the number of digits to the right of the decimal

separator. In SQL, this is the scale of the number; in C, it is the precision of the number.

2. The length of character string variables is n+1, because DB2 places a null character at the end of character strings that it passes to external user-defined functions written in C.
3. Special rules apply to character string parameters of user-defined functions that are written in C:
   - CHAR(*n*) corresponds to a NUL-terminated character string variable with a length of *n*+1 that is defined according to the ANSI/ISO SQL standard of 1992.
   - VARCHAR(*n*) corresponds to a C NUL-terminated character string variable with a length of *n*+1.
   - VARCHAR(*n*) FOR BIT DATA corresponds to the VARCHAR structured form of the character string variable (the simulated VARCHAR form that can include X'00' because it is not NUL-terminated).
   - ROWID corresponds to the ROWID structured form of the row ID variable (like the simulated VARCHAR form that can include X'00' because it is not NUL-terminated).

   NUL-terminated variables of length *n*+1 that are defined according to the ANSI/ISO SQL standard of 1992 differ from C NUL-terminated variables only when they are the target of an assignment for which the length of the source string is less than *n.*

   - When the target is a NUL-terminated variable defined according to the ANSI/ISO SQL standard of 1992, DB2 pads the string on the right with blanks and the NUL is in the last byte of the variable. This is the same rule that DB2 uses to assign the value of a fixed-length string column to a NUL-terminated output variable.
   - When the target is a C NUL-terminated variable, the string is assigned to the variable and the NUL is in the next byte. This is the same rule that DB2 uses to assign the value of a varying-length string column to a NUL-terminated output variable.
4. When you pass a locator value to a user-defined function. The user-defined function must contain a host language declaration of a 4-byte binary integer to receive the locator value. Before you can use the locator in SQL statements in the user-defined function, you must assign its value to a locator variable that is declared in the user-defined function.

Table 20 (Page 1 of 3). Compatible PL/I and assembler declarations for user-defined function parameters

SQL data type	PL/I	Assembler
SMALLINT	BIN FIXED(15)	DS HL2
INTEGER	BIN FIXED(31)	DS FL4
DECIMAL(*p,s*)	DEC FIXED(p,s)	DS PLn'value'[1]
REAL	BIN FLOAT(21)	DS EL4[2] DS EHL4
FLOAT	BIN FLOAT(53)	DS DL8 DS DHL8
CHAR(1)	CHAR(1)	DS CL1
CHAR(n)	CHAR(n)	DS CLn

Table 20 (Page 2 of 3). Compatible PL/I and assembler declarations for user-defined function parameters

SQL data type	PL/I	Assembler
VARCHAR(n) FOR BIT DATA	CHAR(n) VAR	DS HL2,CLn
VARCHAR(n)	CHAR(n) VAR	DS HL2,CLn
GRAPHIC(1)	GRAPHIC(1)	DS GL2
GRAPHIC(n)	GRAPHIC(n)	DS GLm[3]
VARGRAPHIC(n)	GRAPHIC(n) VAR	DS HL2,GLm[3]
TIME	CHAR(8)	DS CL8
DATE	CHAR(10)	DS CL10
TIMESTAMP	CHAR(26)	DS CL26
ROWID	CHAR(40) VAR	DS HL2,CL40
TABLE LOCATOR BLOB LOCATOR CLOB LOCATOR DBCLOB LOCATOR	BIN FIXED(31)[4]	DS FL4[4]
BLOB(n)	If n <= 32767: 01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA CHAR(n); If n > 32767: 01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) CHAR(32767), 03 var_DATA2 CHAR(mod(n,32767));	If n <= 65535: var DS 0FL4 var_length DS FL4 var_data DS CLn If n > 65535: var DS 0FL4 var_length DS FL4 var_data DS CL65535 ORG var_data+(n-65535)
CLOB(n)	If n <= 32767: 01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA CHAR(n); If n > 32767: 01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) CHAR(32767), 03 var_DATA2 CHAR(mod(n,32767));	If n <= 65535: var DS 0FL4 var_length DS FL4 var_data DS CLn If n > 65535: var DS 0FL4 var_length DS FL4 var_data DS CL65535 ORG var_data+(n-65535)

Table 20 (Page 3 of 3). Compatible PL/I and assembler declarations for user-defined function parameters

SQL data type	PL/I	Assembler
DBCLOB(n)	If n <eq; 16383: 01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA GRAPHIC(n); If n > 16383: 01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) GRAPHIC(16383), 03 var_DATA2 GRAPHIC(mod(n,16383));	If m (=2*n) <= 65534: var DS 0FL4 var_length DS FL4 var_data DS CLm If m > 65534: var DS 0FL4 var_length DS FL4 var_data DS CL65534 ORG var_data+(m-65534)

Notes to Table 20 on page 157:

1. You must use L *n*, *value* or both. If you use L *n*, the precision is 2 *n*-1; otherwise, it is the number of digits in *value*. If you use *value*, the scale is the number of digits to the right of the decimal point in *value*; otherwise it is 0.
2. IEEE floating point host variables are not supported in user-defined functions.
3. *m* is expressed in bytes (2 times *n*).
4. When you pass a locator value to a user-defined function, the user-defined function must contain a host language declaration of a 4-byte binary integer to receive the locator value. Before you can use the locator in SQL statements in the user-defined function, you must assign its value to a locator variable that is declared in the user-defined function.

Result parameters: Set these values in your user-defined function before exiting. For a user-defined scalar function, you return one result parameter. For a user-defined table function, you return the same number of parameters as columns in the RETURNS TABLE clause of the CREATE FUNCTION statement. DB2 allocates a buffer for each result parameter value and passes the buffer address to the user-defined function. Your user-defined function places each result parameter value in its buffer. You must ensure that the length of the value you place in each output buffer does not exceed the buffer length. Use the SQL data type and length in the user-defined function definition to determine the buffer length.

Use Table 19 on page 155 and Table 20 on page 157 to determine the host data type to use for each result parameter value. If the CREATE FUNCTION statement contains a CAST FROM clause, use a data type that corresponds to the SQL data type in the CAST FROM clause. Otherwise, use a data type that corresponds to the SQL data type in the RETURNS or RETURNS TABLE clause.

To improve performance for user-defined table functions that return many columns, you can pass values for a subset of columns to the invoker. For example, a user-defined table function might be defined to return 100 columns, but the invoker needs values for only two columns. Use the DBINFO parameter to indicate to DB2 the columns for which you will return values. Then return values for only those columns. See the explanation of DBINFO below for information on how to indicate the columns of interest.

Input parameter indicators: These are SMALLINT values, which DB2 sets before it passes control to the user-defined function. You use the indicators to determine whether the corresponding input parameters are null. The number and order of the indicators are the same as the number and order of the input parameters. On entry to the user-defined function, each indicator contains one of these values:

0 The input parameter value is not null.

negative The input parameter value is null.

Code the user-defined function to check all indicators for null values unless the user-defined function is defined with RETURNS NULL ON NULL INPUT. A user-defined function defined with RETURNS NULL ON NULL INPUT executes only if all input parameters are not null.

Result indicators: These are SMALLINT values, which you must set before the user-defined function ends to indicate to the invoking program whether each result parameter value is null. A user-defined scalar function has one result indicator. A user-defined table function has the same number of result indicators as the number of result parameters. The order of the result indicators is the same as the order of the result parameters. Set each result indicator to one of these values:

0 or positive The result parameter is not null.

negative The result parameter is null.

SQLSTATE value: This is a CHAR(5) value, which you must set before the user-defined function ends. The user-defined function can return one of these SQLSTATE values:

00000 Use this value to indicate that the user-defined function executed without any warnings or errors.

01H*xx* Use these values to indicate that the user-defined function detected a warning condition. *xx* can be any two single-byte alphanumeric characters. DB2 returns SQLCODE +462 if the user-defined function sets the SQLSTATE to 01H*xx*.

02000 Use this value to indicate that no more rows are to be returned from a user-defined table function.

38*yxx* Use these values to indicate that the user-defined function detected an error condition. *y* can be any single-byte alphanumeric character except 5. *xx* can be any two single-byte alphanumeric characters. However, if an SQL statement in the user-defined function returns one of the following SQLSTATEs, passing that SQLSTATE back to the invoker is recommended.

38001 The user-defined function attempted to execute an SQL statement, but the user-defined function is not defined with NO SQL. DB2 returns SQLCODE -487 with this SQLSTATE. The user-defined function attempted to execute an SQL statement, but the user-defined function is defined with NO SQL. DB2 returns SQLCODE -487 with this SQLSTATE.

38002	The user-defined function attempted to execute an SQL statement that requires that the user-defined function is defined with MODIFIES SQL DATA, but the user-defined function is not defined with MODIFIES SQL DATA. DB2 returns SQLCODE -577 with this SQLSTATE.
38003	The user-defined function executed a COMMIT or ROLLBACK statement, which are not permitted in a user-defined function. DB2 returns SQLCODE -751 with this SQLSTATE.
38004	The user-defined function attempted to execute an SQL statement that requires that the user-defined function is defined with READS SQL DATA or MODIFIES SQL DATA, but the user-defined function is not defined with either of these options. DB2 returns SQLCODE -579 with this SQLSTATE.

When your user-defined function returns an SQLSTATE of 38*yxx* other than one of the four listed above, DB2 returns SQLCODE -443.

If the user-defined function returns an SQLSTATE that is not permitted for a user-defined function, DB2 replaces that SQLSTATE with 39001 and returns SQLCODE -463.

If both the user-defined function and DB2 set an SQLSTATE value, DB2 returns its SQLSTATE value to the invoker.

User-defined function name: DB2 sets this value in the parameter list before the user-defined function executes. This value is VARCHAR(137): 8 bytes for the schema name, 1 byte for a period, and 128 bytes for the user-defined function name. If you use the same code to implement multiple versions of a user-defined function, you can use this parameter to determine which version of the function the invoker wants to execute.

Specific name: DB2 sets this value in the parameter list before the user-defined function executes. This value is VARCHAR(128) and is either the specific name from the CREATE FUNCTION statement or a specific name that DB2 generates. If you use the same code to implement multiple versions of a user-defined function, you can use this parameter to determine which version of the function the invoker wants to execute.

Diagnostic message: This is a VARCHAR(70) value, which your user-defined function can set before exiting. Use this area to pass descriptive information about an error or warning to the invoker.

DB2 allocates a 70-byte buffer for this area and passes you the buffer address in the parameter list. Ensure that you do not write more than 70 bytes to the buffer. At least the first 17 bytes of the value you put in the buffer appear in the SQLERRMC field of the SQLCA that is returned to the invoker. The exact number of bytes depends on the number of other tokens in SQLERRMC. Do not use X'FF' in your diagnostic message. DB2 uses this value to delimit tokens.

Scratchpad: If the definer specified SCRATCHPAD in the CREATE FUNCTION statement, DB2 allocates a buffer for the scratchpad area and passes its address to

the user-defined function. Before the user-defined function is invoked for the first time in an SQL statement, DB2 sets the length of the scratchpad in the first 4 bytes of the buffer and then sets the scratchpad area to X'00'. DB2 does not reinitialize the scratchpad between invocations of a correlated subquery.

You must ensure that your user-defined function does not write more bytes to the scratchpad than the scratchpad length.

Call type: For a user-defined scalar function, if the definer specified FINAL CALL in the CREATE FUNCTION statement, DB2 passes this parameter to the user-defined function. For a user-defined table function, DB2 always passes this parameter to the user-defined function.

On entry to a *user-defined scalar function,* the call type parameter has one of the following values:

-1 This is the *first call* to the user-defined function for the SQL statement. For a first call, all input parameters are passed to the user-defined function. In addition, the scratchpad, if allocated, is set to binary zeros.

0 This is a *normal call*. For a normal call, all the input parameters are passed to the user-defined function. If a scratchpad is also passed, DB2 does not modify it.

1 This is a *final call*. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, DB2 does not modify it.

This type of final call occurs when the invoking application explicitly closes a cursor. When a value of 1 is passed to a user-defined function, the user-defined function can execute SQL statements.

255 This is a *final call*. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, DB2 does not modify it.

This type of final call occurs when the invoking application executes a COMMIT or ROLLBACK statement, or when the invoking application abnormally terminates. When a value of 255 is passed to the user-defined function, the user-defined function cannot execute any SQL statements, except for CLOSE CURSOR. If the user-defined function executes any close cursor statements during this type of final call, the user-defined function should tolerate SQLCODE -501 because DB2 might have already closed cursors before the final call.

During the first call, your user-defined scalar function should acquire any system resources it needs. During the final call, the user-defined scalar function should release any resources it acquired during the first call. The user-defined scalar function should return a result value only during normal calls. DB2 ignores any results that are returned during a final call. However, the user-defined scalar function can set the SQLSTATE and diagnostic message area during the final call.

If an invoking SQL statement contains more than one user-defined scalar function, and one of those user-defined functions returns an error SQLSTATE, DB2 invokes all of the user-defined functions for a final call, and the invoking SQL statement receives the SQLSTATE of the first user-defined function with an error.

On entry to a *user-defined table function,* the call type parameter has one of the following values:

-2 This is the *first call* to the user-defined function for the SQL statement. A first call occurs only if the FINAL CALL keyword is specified in the user-defined function definition. For a first call, all input parameters are passed to the user-defined function. In addition, the scratchpad, if allocated, is set to binary zeros.

-1 This is the *open call* to the user-defined function by an SQL statement. If FINAL CALL is not specified in the user-defined function definition, all input parameters are passed to the user-defined function, and the scratchpad, if allocated, is set to binary zeros during the open call. If FINAL CALL is specified for the user-defined function, DB2 does not modify the scratchpad.

0 This is a *fetch call* to the user-defined function by an SQL statement. For a fetch call, all input parameters are passed to the user-defined function. If a scratchpad is also passed, DB2 does not modify it.

1 This is a *close call*. For a close call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, DB2 does not modify it.

2 This is a *final call*. This type of final call occurs only if FINAL CALL is specified in the user-defined function definition. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, DB2 does not modify it.

This type of final call occurs when the invoking application executes a CLOSE CURSOR statement.

255 This is a *final call*. For a final call, no input parameters are passed to the user-defined function. If a scratchpad is also passed, DB2 does not modify it.

This type of final call occurs when the invoking application executes a COMMIT or ROLLBACK statement, or when the invoking application abnormally terminates. When a value of 255 is passed to the user-defined function, the user-defined function cannot execute any SQL statements, except for CLOSE CURSOR. If the user-defined function executes any close cursor statements during this type of final call, the user-defined function should tolerate SQLCODE -501 because DB2 might have already closed cursors before the final call.

If a user-defined table function is defined with FINAL CALL, the user-defined function should allocate any resources it needs during the first call and release those resources during the final call that sets a value of 2.

If a user-defined table function is defined with NO FINAL CALL, the user-defined function should allocate any resources it needs during the open call and release those resources during the close call.

During a fetch call, the user-defined table function should return a row. If the user-defined function has no more rows to return, it should set the SQLSTATE to 02000.

During the close call, a user-defined table function can set the SQLSTATE and diagnostic message area.

If a user-defined table function is invoked from a subquery, the user-defined table function receives a CLOSE call for each invocation of the subquery within the higher level query, and a subsequent OPEN call for the next invocation of the subquery within the higher level query.

DBINFO: If the definer specified DBINFO in the CREATE FUNCTION statement, DB2 passes the DBINFO structure to the user-defined function. DBINFO contains information about the environment of the user-defined function caller. DBINFO contains the following fields, in the order shown:

Location name length
An unsigned 2-byte integer field. It contains the length of the location name in the next field.

Location name
A128-byte character field. It contains the name of the location to which the invoker is currently connected.

Authorization ID length
An unsigned 2-byte integer field. It contains the length of the authorization ID in the next field.

Authorization ID
A 128-byte character field. It contains the authorization ID of the application from which the user-defined function is invoked, padded on the right with blanks. If this user-defined function is nested within other user-defined functions, this value is the authorization ID of the application that invoked the highest-level user-defined function.

Subsystem code page
A 48-byte structure that consists of seven integer fields and 20 bytes of reserved space. These fields provide information about the CCSIDs and encoding scheme of the subsystem from which the user-defined function is invoked. The seven integer fields contain:

- ASCII SBCS CCSID
- ASCII MIXED CCSID
- ASCII DBCS CCSID
- EBCDIC SBCS CCSID
- EBCDIC MIXED CCSID
- EBCDIC DBCS CCSID
- Encoding scheme

Table qualifier length
An unsigned 2-byte integer field. It contains the length of the table qualifier in the next field. If the table name field is not used, this field contains 0.

Table qualifier
A 128-byte character field. It contains the qualifier of the table that is specified in the table name field.

Table name length
An unsigned 2-byte integer field. It contains the length of the table name in the next field. If the table name field is not used, this field contains 0.

Table name
A 128-byte character field. This field contains the name of the table that the UPDATE or INSERT modifies if the reference to the user-defined function in the invoking SQL statement is in one of the places shown in the list that follows. Otherwise, this field is blank.

- The right side of a SET clause in an UPDATE statement
- In the VALUES list of an INSERT statement

Column name length
An unsigned 2-byte integer field. It contains the length of the column name in the next field. If no column name is passed to the user-defined function, this field contains 0.

Column name
A 128-byte character field. This field contains the name of the column that the UPDATE or INSERT modifies if the reference to the user-defined function in the invoking SQL statement is in one of the places shown in the list that follows. Otherwise, this field is blank.

- The right side of a SET clause in an UPDATE statement
- In the VALUES list of an INSERT statement

Product information
An 8-byte character field that defines the product on which the user-defined function executes. The field has the form *pppvvrrm*, where:

- *ppp* is a 3-byte product code:

  DSN DB2 for OS/390

  ARI DB2 Server for VSE & VM

  QSQ DB2 for AS/400

  SQL DB2 Universal Database

- *vv* is a 2-digit version identifier.
- *rr* is a 2-digit release identifier.
- *m* is a 1-digit modification level identifier.

Operating system
A 4-byte integer field. It identifies the operating system on which the program that invokes the user-defined function runs. The value is one of these:

0 Unknown

1 OS/2

3 Windows

4 AIX

5 Windows NT

6 HP-UX

7 Solaris

8 OS/390

13 Siemens Nixdorf

15 Windows 95

16 SCO Unix

Number of entries in table function column list
An unsigned 2-byte integer field.

Reserved area
A reserved area that is 24 bytes in length.

Table function column list pointer
If a table function is defined, this field is a pointer to an array that contains 1000 2-byte integers. DB2 dynamically allocates the array. If a table function is not defined, this pointer is null.

Only the first *n* entries, where *n* is the value in the field entitled number of entries in table function column list, are of interest. *n* is greater than or equal to 0 and less than or equal to the number result columns defined for the user-defined function in the RETURNS TABLE clause of the CREATE FUNCTION statement. The values correspond to the numbers of the columns that the invoking statement needs from the table function. A value of 1 means the first defined result column, 2 means the second defined result column, and so on. The values can be in any order. If *n* is equal to 0, the first array element is 0. This is the case for a statement like the following one, where the invoking statement needs no column values.

```
SELECT COUNT(*) FROM TABLE(TF(...)) AS QQ
```

This array represents an opportunity for optimization. The user-defined function does not need to return all values for all the result columns of the table function. Instead, the user-defined function can return only those columns that are needed in the particular context, which you identify by number in the array. However, if this optimization complicates the user-defined function logic enough to cancel the performance benefit, you might choose to return every defined column.

Unique application identifier
This field is a pointer to a string that uniquely identifies the application's connection to DB2. The string is regenerated at for each connection to DB2.

The string is the LUWID, which consists of a fully-qualified LU network name followed by a period and an LUW instance number. The LU network name consists of a 1- to 8-character network ID, a period, and a 1- to 8-character network LU name. The LUW instance number consists of 12 hexadecimal characters that uniquely identify the unit of work.

Reserved area
This area is 20 bytes.

See the following section for examples of declarations of passed parameters in each language. If you write your user-defined function in C or C++, you can use the declarations in member SQLUDF of DSN610.SDSNSAMP for many of the passed parameters. To include SQLUDF, put this statement in your program:

```
#include <sqludf.h>
```

Examples of passing parameters in a user-defined function: The following examples show how a user-defined function that is written in each of the supported host languages receives the parameter list that is passed by DB2.

These examples assume that the user-defined function is defined with the SCRATCHPAD, FINAL CALL, and DBINFO parameters.

Assembler: Figure 24 shows the parameter conventions for a user-defined scalar function that is written as a main program that receives two parameters and returns one result. For an assembler language user-defined function that is a subprogram, the conventions are the same. You must include the CEEENTRY and CEEEXIT macros.

```
MYMAIN   CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
         USING PROGAREA,R13

         L     R7,0(R1)            GET POINTER TO PARM1
         MVC   PARM1(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF PARM1
         L     R7,4(R1)            GET POINTER TO PARM2
         MVC   PARM1(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF PARM2
         L     R7,12(R1)           GET POINTER TO INDICATOR 1
         MVC   F_IND1(2),0(R7)  MOVE PARM1 INDICATOR TO LOCAL STORAGE
         LH    R7,F_IND1        MOVE PARM1 INDICATOR INTO R7
         LTR   R7,R7               CHECK IF IT IS NEGATIVE
         BM    NULLIN              IF SO, PARM1 IS NULL
         L     R7,16(R1)           GET POINTER TO INDICATOR 2
         MVC   F_IND2(2),0(R7)  MOVE PARM2 INDICATOR TO LOCAL STORAGE
         LH    R7,F_IND2        MOVE PARM2 INDICATOR INTO R7
         LTR   R7,R7               CHECK IF IT IS NEGATIVE
         BM    NULLIN              IF SO, PARM2 IS NULL

  ⋮
         L     R7,8(R1)            GET ADDRESS OF AREA FOR RESULT
NULLIN   MVC   0(9,R7),RESULT      MOVE A VALUE INTO RESULT AREA
         L     R7,20(R1)           GET ADDRESS OF AREA FOR RESULT IND
         MVC   0(2,R7),=H'0'       MOVE A VALUE INTO INDICATOR AREA

  ⋮
         CEETERM  RC=0
*********************************************************************
*  VARIABLE DECLARATIONS AND EQUATES                                *
*********************************************************************
R1       EQU   1                   REGISTER 1
R7       EQU   7                   REGISTER 7
PPA      CEEPPA  ,                 CONSTANTS DESCRIBING THE CODE BLOCK
         LTORG ,                   PLACE LITERAL POOL HERE
PROGAREA DSECT
         ORG   *+CEEDSASZ          LEAVE SPACE FOR DSA FIXED PART
PARM1    DS    F                   PARAMETER 1
PARM2    DS    F                   PARAMETER 2
RESULT   DS    CL9                 RESULT
F_IND1   DS    H                   INDICATOR FOR PARAMETER 1
F_IND2   DS    H                   INDICATOR FOR PARAMETER 2
F_INDR   DS    H                   INDICATOR FOR RESULT

PROGSIZE EQU   *-PROGAREA
         CEEDSA  ,                 MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA  ,                 MAPPING OF THE COMMON ANCHOR AREA
         END   MYMAIN
```

Figure 24. How an assembler language user-defined function receives parameters

C or C++: For C or C++, the conventions for passing parameters are different for main programs and subprograms. Therefore, for C or C++ user-defined functions, the conventions for main programs are also different from those for subprograms.

For subprograms, you pass the parameters directly. For main programs, you use the standard argc and argv variables to access the input and output parameters:

- The argv variable contains an array of pointers to the parameters that are passed to the user-defined function. All string parameters that are passed back to DB2 must be null terminated.
  - argv[0] contains the address of the load module name for the user-defined function.
  - argv[1] through argv[*n*] contain the addresses of parameters 1 through *n*.
- The argc variable contains the number of parameters that are passed to the external user-defined function, including argv[0].

Figure 25 on page 169 shows the parameter conventions for a user-defined scalar function that is written as a main program that receives two parameters and returns one result.

```
#include <stdlib.h>
#include <stdio.h>

main(argc,argv)
  int argc;
  char *argv[];
 {
   /*************************************************/
   /* Assume that the user-defined function invocation*/
   /* included 2 input  parameters in the parameter   */
   /* list.  Also assume that the definition includes */
   /* the SCRATCHPAD, FINAL CALL, and DBINFO options, */
   /* so DB2 passes the scratchpad, calltype, and     */
   /* dbinfo parameters.                              */
   /* The argv vector contains these entries:         */
   /*      argv[0]         1    load module name      */
   /*      argv[1-2]       2    input parms           */
   /*      argv[3]         1    result parm           */
   /*      argv[4-5]       2    null indicators       */
   /*      argv[6]         1    result null indicator */
   /*      argv[7]         1    SQLSTATE variable     */
   /*      argv[8]         1    qualified func name   */
   /*      argv[9]         1    specific func name    */
   /*      argv[10]        1    diagnostic string     */
   /*      argv[11]        1    scratchpad            */
   /*      argv[12]        1    call type             */
   /*      argv[13]      + 1    dbinfo                */
   /*                    ------                       */
   /*                      14   for the argc variable */
   /*************************************************/
   if argc<>14
   {

⋮
     /*********************************************************/
     /* This section would contain the code executed if the    */
     /* user-defined function is invoked with the wrong number */
     /* of parameters.                                         */
     /*********************************************************/
   }
```

Figure 25 (Part 1 of 2). How a C or C++ user-defined function that is written as a main program receives parameters

```
  /*************************************************/
  /* Assume the first parameter is an integer.     */
  /* The code below shows how to copy the integer  */
  /* parameter into the application storage.       */
  /*************************************************/
  int parm1;
  parm1 = *(int *) argv[1];

  /*************************************************/
  /* Access the null indicator for the first       */
  /* parameter on the invoked user-defined function*/
  /* as follows:                                   */
  /*************************************************/
  short int ind1;
  ind1 = *(short int *) argv[4];

  /*************************************************/
  /* Use the expression below to assign            */
  /* 'xxxxx' to the SQLSTATE returned to caller on */
  /* the SQL statement that contains the invoked   */
  /* user-defined function.                        */
  /*************************************************/
  strcpy(argv[7],"xxxxx/0");

  /*************************************************/
  /* Obtain the value of the qualified function    */
  /* name with this expression.                    */
  /*************************************************/
  char f_func[28];
  strcpy(f_func,argv[8]);
  /*************************************************/
  /* Obtain the value of the specific function     */
  /* name with this expression.                    */
  /*************************************************/
  char f_spec[19];
  strcpy(f_spec,argv[9]);

  /*************************************************/
  /* Use the expression below to assign            */
  /* 'yyyyyyyy' to the diagnostic string returned  */
  /* in the SQLCA associated with the invoked      */
  /* user-defined function.                        */
  /*************************************************/
  strcpy(argv[10],"yyyyyyyy/0");

  /*************************************************/
  /* Use the expression below to assign the        */
  /* result of the function.                       */
  /*************************************************/
  char l_result[11];
  strcpy(argv[3],l_result);


  .
  .
  .
 }
```

Figure 25 (Part 2 of 2). How a C or C++ user-defined function that is written as a main program receives parameters

Figure 26 on page 171 shows the parameter conventions for a user-defined scalar function that is written as a C subprogram that receives two parameters and returns one result.

```
#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

struct sqludf_scratchpad
{
  unsigned long  length;     /* length of scratchpad data */
  char  data[SQLUDF_SCRATCHPAD_LEN];   /* scratchpad data */
};
struct sqludf_dbinfo
{
  unsigned short dbnamelen;   /* database name length      */
  unsigned char  dbname[128]; /* database name             */
  unsigned short authidlen;   /* appl auth id  length      */
  unsigned char  authid[128]; /* appl authorization ID     */
  unsigned long  ascii_sbcs;   /* ASCII SBCS CCSID      */
  unsigned long  ascii_dbcs;   /* ASCII MIXED CCSID     */
  unsigned long  ascii_mixed;  /* ASCII DBCS CCSID      */
  unsigned long  ebcdic_sbcs;  /* EBCDIC SBCS CCSID     */
  unsigned long  ebcdic_dbcs;  /* EBCDIC MIXED CCSID    */
  unsigned long  ebcdic_mixed; /* EBCDIC DBCS CCSID     */
  unsigned long  encode;          /* UDF encode scheme     */
  unsigned char  reserv1[20];     /* reserved for later use*/
  unsigned short tbqualiflen;     /* table qualifier length */
  unsigned char  tbqualif[128];   /* table qualifer name   */
  unsigned short tbnamelen;       /* table name length     */
  unsigned char  tbname[128];     /* table name            */
  unsigned short colnamelen;      /* column name length    */
  unsigned char  colname[128];    /* column name           */
  unsigned char  relver[8]; /* Database release & version  */
  unsigned long  platform;  /* Database platform           */
  unsigned short numtfcol;  /* # of Tab Fun columns used   */
  unsigned char  reserv1[24];  /* reserved                 */
  unsigned short *tfcolnum;    /* table fn column list     */
  unsigned short *appl_id; /* LUWID for DB2 connection    */
  unsigned char  reserv2[20];  /* reserved                 */
};

void myfunc(long *parm1, char parm2[11], char result[11],
            short *f_ind1, short *f_ind2, short *f_indr,
            char udf_sqlstate[6], char udf_fname[138],
            char udf_specname[129], char udf_msgtext[71],
            struct sqludf_scratchpad *udf_scratchpad,
            long *udf_call_type,
            struct sql_dbinfo *udf_dbinfo);
```

Figure 26 (Part 1 of 2). How a C language user-defined function that is written as a subprogram receives parameters

```
{
  /*************************************************/
  /* Declare local copies of parameters            */
  /*************************************************/
  int l_p1;
  char l_p2[11];
  short int l_ind1;
  short int l_ind2;
  char ludf_sqlstate[6];     /* SQLSTATE              */
  char ludf_fname[138];      /* function name         */
  char ludf_specname[129];   /* specific function name */
  char ludf_msgtext[71]      /* diagnostic message text*/
  sqludf_scratchpad *ludf_scratchpad; /* scratchpad    */
  long *ludf_call_type;      /* call type             */
  sqludf_dbinfo *ludf_dbinfo /* dbinfo                */
  /*************************************************/
  /* Copy each of the parameters in the parameter    */
  /* list into a local variable to demonstrate       */
  /* how the parameters can be referenced.           */
  /*************************************************/

  l_p1 = *parm1;
  strcpy(l_p2,parm2);
  l_ind1 = *f_ind1;
  l_ind1 = *f_ind2;
  strcpy(ludf_sqlstate,udf_sqlstate);
  strcpy(ludf_fname,udf_fname);
  strcpy(ludf_specname,udf_specname);
  l_udf_call_type = *udf_call_type;
  strcpy(ludf_msgtext,udf_msgtext);
  memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
  memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));

 .
 .
 .
}
```

Figure 26 (Part 2 of 2). How a C language user-defined function that is written as a subprogram receives parameters

Figure 27 on page 173 shows the parameter conventions for a user-defined scalar function that is written as a C++ subprogram that receives two parameters and returns one result. This example demonstrates that you must use an external "C" modifier to indicate that you want the C++ subprogram to receive parameters according to the C linkage convention. This modifier is necessary because the CEEPIPI CALL_SUB interface, which DB2 uses to call the user-defined function, passes parameters using the C linkage convention.

```
#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>
struct sqludf_scratchpad
{
  unsigned long  length;     /* length of scratchpad data */
  char  data[SQLUDF_SCRATCHPAD_LEN];   /* scratchpad data */
};
struct sqludf_dbinfo
{
  unsigned short dbnamelen;   /* database name length      */
  unsigned char  dbname[128]; /* database name             */
  unsigned short authidlen;   /* appl auth id  length      */
  unsigned char  authid[128]; /* appl authorization ID     */
  unsigned long  ascii_sbcs;   /* ASCII SBCS CCSID      */
  unsigned long  ascii_dbcs;   /* ASCII MIXED CCSID     */
  unsigned long  ascii_mixed;  /* ASCII DBCS CCSID      */
  unsigned long  ebcdic_sbcs;  /* EBCDIC SBCS CCSID     */
  unsigned long  ebcdic_dbcs;  /* EBCDIC MIXED CCSID    */
  unsigned long  ebcdic_mixed; /* EBCDIC DBCS CCSID     */
  unsigned long  encode;          /* UDF encode scheme     */
  unsigned char  reserv1[20];     /* reserved for later use*/
  unsigned short tbqualiflen;     /* table qualifier length */
  unsigned char  tbqualif[128];   /* table qualifer name   */
  unsigned short tbnamelen;       /* table name length     */
  unsigned char  tbname[128];     /* table name            */
  unsigned short colnamelen;      /* column name length    */
  unsigned char  colname[128];    /* column name           */
  unsigned char  relver[8]; /* Database release & version  */
  unsigned long  platform;  /* Database platform           */
  unsigned short numtfcol;  /* # of Tab Fun columns used   */
  unsigned char  reserv1[24];  /* reserved                 */
  unsigned short *tfcolnum;    /* table fn column list     */
  unsigned short *appl_id; /* LUWID for DB2 connection    */
  unsigned char  reserv2[20];  /* reserved                 */
};
extern "C" void myfunc(long *parm1, char parm2[11],
            char result[11], short *f_ind1, short *f_ind2, short *f_indr,
            char udf_sqlstate[6], char udf_fname[138],
            char udf_specname[129], char udf_msgtext[71],
            struct sqludf_scratchpad *udf_scratchpad,
            long *udf_call_type,
            struct sql_dbinfo *udf_dbinfo);
```

Figure 27 (Part 1 of 2). How a C++ user-defined function that is written as a subprogram receives parameters

```
{
  /************************************************/
  /* Define local copies of parameters.           */
  /************************************************/
  int l_p1;
  char l_p2[11];
  short int l_ind1;
  short int l_ind2;
  char ludf_sqlstate[6];     /* SQLSTATE              */
  char ludf_fname[138];      /* function name         */
  char ludf_specname[129];   /* specific function name */
  char ludf_msgtext[71]      /* diagnostic message text*/
  sqludf_scratchpad *ludf_scratchpad; /* scratchpad    */
  long *ludf_call_type;      /* call type             */
  sqludf_dbinfo *ludf_dbinfo /* dbinfo                */
  /************************************************/
  /* Copy each of the parameters in the parameter    */
  /* list into a local variable to demonstrate       */
  /* how the parameters can be referenced.           */
  /************************************************/
  l_p1 = *parm1;
  strcpy(l_p2,parm2);
  l_ind1 = *f_ind1;
  l_ind1 = *f_ind2;
  strcpy(ludf_sqlstate,udf_sqlstate);
  strcpy(ludf_fname,udf_fname);
  strcpy(ludf_specname,udf_specname);
  l_udf_call_type = *udf_call_type;
  strcpy(ludf_msgtext,udf_msgtext);
  memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
  memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));

 ⋮
}
```

Figure 27 (Part 2 of 2). How a C++ user-defined function that is written as a subprogram receives parameters

COBOL: Figure 28 on page 175 shows the parameter conventions for a user-defined table function that is written as a main program that receives two parameters and returns two results. For a COBOL user-defined function that is a subprogram, the conventions are the same.

```
CBL APOST,RES,RENT
      IDENTIFICATION DIVISION.

 ⋮
      DATA DIVISION.

 ⋮
      LINKAGE SECTION.
     ***********************************************************
     * Declare each of the parameters                          *
     ***********************************************************
      01  UDFPARM1 PIC S9(9) USAGE COMP.
      01  UDFPARM2 PIC X(10).

 ⋮
     ***********************************************************
     * Declare these variables for result parameters           *
     ***********************************************************
      01  UDFRESULT1 PIC X(10).
      01  UDFRESULT2 PIC X(10).

 ⋮
     ***********************************************************
     * Declare a null indicator for each parameter             *
     ***********************************************************
      01  UDF-IND1 PIC S9(4) USAGE COMP.
      01  UDF-IND2 PIC S9(4) USAGE COMP.

 ⋮
     ***********************************************************
     * Declare a null indicator for result parameter           *
     ***********************************************************
      01  UDF-RIND1 PIC S9(4) USAGE COMP.
      01  UDF-RIND2 PIC S9(4) USAGE COMP.

 ⋮
     ***********************************************************
     * Declare the SQLSTATE that can be set by the             *
     * user-defined function                                   *
     ***********************************************************
      01  UDF-SQLSTATE PIC X(5).
```

Figure 28 (Part 1 of 4). How a COBOL user-defined function receives parameters

```
*********************************************************
 01  UDF-CALL-TYPE PIC 9(9) USAGE BINARY.
* Declare the qualified function name                   *
*********************************************************
 01  UDF-FUNC.
     49 UDF-FUNC-LEN PIC 9(4) USAGE BINARY.
     49 UDF-FUNC-TEXT PIC X(137).
*********************************************************
* Declare the specific function name                    *
*********************************************************
 01  UDF-SPEC.
     49 UDF-SPEC-LEN PIC 9(4) USAGE BINARY.
     49 UDF-SPEC-TEXT PIC X(128).
*********************************************************
* Declare SQL diagnostic message token                  *
*********************************************************
 01  UDF-DIAG.
     49 UDF-DIAG-LEN PIC 9(4) USAGE BINARY.
     49 UDF-DIAG-TEXT PIC X(70).
*********************************************************
* Declare the scratchpad                                *
*********************************************************
 01  UDF-SCRATCHPAD.
     49 UDF-SPAD-LEN PIC 9(9) USAGE BINARY.
     49 UDF-SPAD-TEXT PIC X(100).
*********************************************************
* Declare the call type                                 *
*********************************************************
```

Figure 28 (Part 2 of 4). How a COBOL user-defined function receives parameters

```
* Declare the DBINFO structure
  01  UDF-DBINFO.
 *       Location length and name
       02 UDF-DBINFO-LOCATION.
          49 UDF-DBINFO-LLEN PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-LOC  PIC X(128).
 *       Authorization ID length and name
       02 UDF-DBINFO-AUTHORIZATION.
          49 UDF-DBINFO-ALEN PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-AUTH PIC X(128).
 *       CCSIDs for DB2 for OS/390
       02 UDF-DBINFO-CCSID PIC X(48).
       02 UDF-DBINFO-CCSID-REDEFINE REDEFINES UDF-DBINFO-CCSID.
          03 UDF-DBINFO-ESBCS   PIC 9(9) USAGE BINARY.
          03 UDF-DBINFO-EMIXED  PIC 9(9) USAGE BINARY.
          03 UDF-DBINFO-EDBCS   PIC 9(9) USAGE BINARY.
          03 UDF-DBINFO-ASBCS   PIC 9(9) USAGE BINARY.
          03 UDF-DBINFO-AMIXED  PIC 9(9) USAGE BINARY.
          03 UDF-DBINFO-ADBCS   PIC 9(9) USAGE BINARY.
          03 UDF-DBINFO-ENCODE  PIC 9(9) USAGE BINARY.
          03 UDF-DBINFO-RESERV0 PIC X(20).
 *       Schema length and name
       02 UDF-DBINFO-SCHEMA0.
          49 UDF-DBINFO-SLEN PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-SCHEMA PIC X(128).
 *       Table length and name
       02 UDF-DBINFO-TABLE0.
          49 UDF-DBINFO-TLEN PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-TABLE  PIC X(128).
 *       Column length and name
       02 UDF-DBINFO-COLUMN0.
          49 UDF-DBINFO-CLEN PIC 9(4) USAGE BINARY.
          49 UDF-DBINFO-COLUMN  PIC X(128).
 *       DB2 release level
       02 UDF-DBINFO-VERREL PIC X(8).
 *       Unused
       02 FILLER            PIC X(2).
      *       Database Platform
           02 UDF-DBINFO-PLATFORM PIC 9(9) USAGE BINARY.
      *       # of entries in Table Function column list
           02 UDF-DBINFO-NUMTFCOL PIC 9(4) USAGE BINARY.
      *       reserved
           02 UDF-DBINFO-RESERV1 PIC X(24).
      *       Unused
           02 FILLER            PIC X(2).
      *       Pointer to Table Function column list
           02 UDF-DBINFO-TFCOLUMN PIC 9(9) USAGE BINARY.
      *       Pointer to Application ID
           02 UDF-DBINFO-APPLID  PIC 9(9) USAGE BINARY.
      *       reserved
           02 UDF-DBINFO-RESERV2 PIC X(20).
      *
```

Figure 28 (Part 3 of 4). How a COBOL user-defined function receives parameters

```
       PROCEDURE DIVISION USING UDFPARM1, UDFPARM2, UDFRESULT1,
                                UDFRESULT2, UDF-IND1, UDF-IND2,
                                UDF-RIND1, UDF-RIND2,
                                UDF-SQLSTATE, UDF-FUNC, UDF-SPEC,
                                UDF-DIAG, UDF-SCRATCHPAD,
                                UDF-CALL-TYPE, UDF-DBINFO.

  ⋮
```

Figure 28 (Part 4 of 4). How a COBOL user-defined function receives parameters

PL/I: Figure 29 shows the parameter conventions for a user-defined scalar function that is written as a main program that receives two parameters and returns one result. For a PL/I user-defined function that is a subprogram, the conventions are the same.

```
*PROCESS SYSTEM(MVS);
 MYMAIN: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
              UDF_IND1, UDF_IND2, UDF_INDR,
              UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
              UDF_DIAG_MSG, UDF_SCRATCHPAD,
              UDF_CALL_TYPE, UDF_DBINFO)
         OPTIONS(MAIN NOEXECOPS REENTRANT);

 DCL UDF_PARM1 BIN FIXED(31);   /* first parameter             */
 DCL UDF_PARM2 CHAR(10);       /* second parameter            */
 DCL UDF_RESULT CHAR(10);       /* result parameter            */
 DCL UDF_IND1 BIN FIXED(15);    /* indicator for 1st parm      */
 DCL UDF_IND2 BIN FIXED(15);    /* indicator for 2nd parm      */
 DCL UDF_INDR BIN FIXED(15);    /* indicator for result        */
 DCL UDF_SQLSTATE CHAR(5);      /* SQLSTATE returned to DB2    */
 DCL UDF_NAME CHAR(137) VARYING; /* Qualified function name     */
 DCL UDF_SPEC_NAME CHAR(128) VARYING; /* Specific function name  */
 DCL UDF_DIAG_MSG CHAR(70) VARYING; /* Diagnostic string       */
```

Figure 29 (Part 1 of 2). How a PL/I user-defined function receives parameters

```
DCL 01 UDF_SCRATCHPAD          /* Scratchpad                   */
     03 UDF_SPAD_LEN BIN FIXED(31),
     03 UDF_SPAD_TEXT CHAR(100);
DCL UDF_CALL_TYPE BIN FIXED(31); /* Call Type                  */
DCL DBINFO PTR;
DCL 01 UDF_DBINFO BASED(DBINFO),           /* Dbinfo           */
      03 UDF_DBINFO_LLEN BIN FIXED(15),    /* location length  */
      03 UDF_DBINFO_LOC  CHAR(128),        /* location name    */
      03 UDF_DBINFO_ALEN BIN FIXED(15),    /* auth ID  length  */
      03 UDF_DBINFO_AUTH CHAR(128),        /* authorization ID */
      03 UDF_DBINFO_CCSID,          /* CCSIDs for DB2 for OS/390*/
       05 R1                BIN FIXED(15), /* Reserved         */
       05 UDF_DBINFO_ASBCS  BIN FIXED(15), /* ASCII SBCS CCSID */
       05 R2                BIN FIXED(15), /* Reserved         */
       05 UDF_DBINFO_AMIXED BIN FIXED(15), /* ASCII MIXED CCSID */
       05 R3                BIN FIXED(15), /* Reserved         */
       05 UDF_DBINFO_ADBCS  BIN FIXED(15), /* ASCII DBCS CCSID */
       05 R4                BIN FIXED(15), /* Reserved         */
       05 UDF_DBINFO_ESBCS  BIN FIXED(15), /* EBCDIC SBCS CCSID */
       05 R5                BIN FIXED(15), /* Reserved         */
       05 UDF_DBINFO_EMIXED BIN FIXED(15), /* EBCDIC MIXED CCSID*/
       05 R6                BIN FIXED(15), /* Reserved         */
       05 UDF_DBINFO_EDBCS  BIN FIXED(15), /* EBCDIC DBCS CCSID */
       05 UDF_DBINFO_ENCODE BIN FIXED(31), /* UDF encode scheme */
       05 UDF_DBINFO_RESERV0 CHAR(20),     /* reserved         */
      03 UDF_DBINFO_SLEN BIN FIXED(15),    /* schema length    */
      03 UDF_DBINFO_SCHEMA CHAR(128),      /* schema name      */
      03 UDF_DBINFO_TLEN BIN FIXED(15),    /* table length     */
      03 UDF_DBINFO_TABLE  CHAR(128),      /* table name       */
      03 UDF_DBINFO_CLEN BIN FIXED(15),    /* column length    */
      03 UDF_DBINFO_COLUMN  CHAR(128),     /* column name      */
      03 UDF_DBINFO_RELVER  CHAR(8),       /* DB2 release level */
      03 UDF_DBINFO_PLATFORM BIN FIXED(31), /* database platform*/
      03 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # of TF cols used*/
      03 UDF_DBINFO_RESERV1 CHAR(24),       /* reserved        */
      03 UDF_DBINFO_TFCOLUMN PTR,      /* -> table fun col list */
      03 UDF_DBINFO_APPLID   PTR,      /* -> application id    */
      03 UDF_DBINFO_RESERV2 CHAR(20);       /* reserved        */

  ⋮
```

Figure 29 (Part 2 of 2). How a PL/I user-defined function receives parameters

Using special registers in a user-defined function: You can use all special registers in a user-defined function. However, you can modify only some of those special registers. After a user-defined function completes, DB2 restores all special registers to the values they had before invocation.

Table 21 on page 180 shows information you need when you use special registers in a user-defined function.

Table 21. Characteristics of special registers in a user-defined function

Special register	Initial value	Function can use SET statement to modify?
CURRENT DATE	New value for each SQL statement in the user-defined function package[1]	Not applicable[4]
CURRENT DEGREE	Inherited from invoker[2]	Yes
CURRENT LOCALE LC_TYPE	Inherited from invoker	Yes
CURRENT OPTIMIZATION HINT	The value of bind option OPTHINT for the user-defined function package or inherited from invoker[5]	Yes
CURRENT PACKAGESET	Inherited from invoker[3]	Yes
CURRENT PATH	The value of bind option PATH for the user-defined function package or inherited from invoker[5]	Yes
CURRENT PRECISION	Inherited from invoker	Yes
CURRENT RULES	Inherited from invoker	Yes
CURRENT SERVER	Inherited from invoker	Yes
CURRENT SQLID	The primary authorization ID of the application process or inherited from invoker[6]	Yes[7]
CURRENT TIME	New value for each SQL statement in the user-defined function package[1]	Not applicable[4]
CURRENT TIMESTAMP	New value for each SQL statement in the user-defined function package[1]	Not applicable[4]
CURRENT TIMEZONE	Inherited from invoker	Not applicable[4]
CURRENT USER	Primary authorization ID of the application process	Not applicable[4]

Notes on Table 21:

1. If the function is invoked within the scope of a trigger, DB2 uses the timestamp for the triggering SQL statement as the timestamp for all SQL statements in the function package.
2. DB2 allows parallelism at only one level of a nested SQL statement. If you set the value of the CURRENT DEGREE special register, and parallelism is disabled, DB2 ignores the CURRENT DEGREE value.
3. If the user-defined function definer specifies a value for COLLID in the CREATE FUNCTION statement, DB2 sets CURRENT PACKAGESET to the value of COLLID.
4. Not applicable because no SET statement exists for the special register.
5. If a program within the scope of the invoking program issues a SET statement for the special register before the user-defined function is invoked, the special

register inherits the value from the SET statement. Otherwise, the special register contains the value that is set by the bind option for the user-defined function package.

6. If a program within the scope of the invoking program issues a SET CURRENT SQLID statement before the user-defined function is invoked, the special register inherits the value from the SET statement. Otherwise, CURRENT SQLID contains the authorization ID of the application process.
7. If the user-defined function package has a dynamic SQL statement behavior other than run behavior, the SET CURRENT SQLID statement can be executed but does not affect the authorization ID that is used for the dynamic SQL statements in the user-defined function package. The dynamic SQL statement behavior determines the authorization ID that is used for dynamic SQL statements. See the discussion of DYNAMICRULES in Chapter 2 of *DB2 Command Reference* for more information about dynamic SQL statement behavior.

Using a scratchpad in a user-defined function: You can use a scratchpad to save information between invocations of a user-defined function. To indicate that a scratchpad should be allocated when the user-defined function executes, the function definer specifies the SCRATCHPAD parameter in the CREATE FUNCTION statement.

The scratchpad consists of a 4-byte length field, followed by the scratchpad area. The definer can specify the length of the scratchpad area in the CREATE FUNCTION statement. The specified length does not include the length field. The default size is 100 bytes. DB2 initializes the scratchpad for each function to binary zeros at the beginning of execution for each subquery of an SQL statement and does not examine or change the content thereafter. On each invocation of the user-defined function, DB2 passes the scratchpad to the user-defined function. You can therefore use the scratchpad to preserve information between invocations of a reentrant user-defined function.

Figure 30 on page 182 demonstrates how to enter information in a scratchpad for a user-defined function defined like this:

```
CREATE FUNCTION COUNTER()
  RETURNS INT
  SCRATCHPAD
  FENCED
  NOT DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'UDFCTR';
```

The scratchpad length is not specified, so the scratchpad has the default length of 100 bytes, plus 4 bytes for the length field. The user-defined function increments an integer value and stores it in the scratchpad on each execution.

```
#pragma linkage(ctr,fetchable)
#include <stdlib.h>
#include <stdio.h>
/* Structure scr defines the passed scratchpad for function ctr */
  struct scr {
    long len;
    long countr;
    char not_used[96];
  };
/*************************************************************/
/* Function ctr:  Increments a counter and reports the value   */
/*                from the scratchpad.                         */
/*                                                             */
/*     Input:  None                                            */
/*     Output: INTEGER out      the value from the scratchpad  */
/*************************************************************/
void ctr(
  long *out,                        /* Output answer (counter)  */
  short *outnull,                   /* Output null indicator    */
  char *sqlstate,                   /* SQLSTATE                 */
  char *funcname,                   /* Function name            */
  char *specname,                   /* Specific function name   */
  char *mesgtext,                   /* Message text insert      */
  struct scr *scratchptr)           /* Scratchpad               */
{
  *out = ++scratchptr->countr;      /* Increment counter and    */
                                    /* copy to output variable  */
  *outnull = 0;                     /* Set output null indicator*/
   return;
}
/* end of user-defined function ctr */
```

Figure 30. Example of coding a scratchpad in a user-defined function

Accessing transition tables in a user-defined function: When you write a user-defined function that is to be invoked from a trigger, you might need access to transition tables for the trigger. Transition tables are the sets of rows that a triggering SQL statement modifies. For more information on transition tables, see "Parts of a trigger" on page 213. To access transition tables in a user-defined function, use table locators, which are pointers to the transition tables. You declare table locators as input parameters in the CREATE FUNCTION statement using the TABLE LIKE *table-name* AS LOCATOR clause. See Chapter 6 of *DB2 SQL Reference* for more information about the CREATE FUNCTION statement.

The five basic steps to accessing transition tables in a user-defined function are:

1. Declare input parameters to receive table locators. You must define each parameter that receives a table locator as an unsigned 4-byte integer.
2. Declare table locators. You can declare table locators in assembler, C, C++, COBOL, and PL/I. The syntax for declaring table locators in each language is described below.

   Table locator declaration in assembler

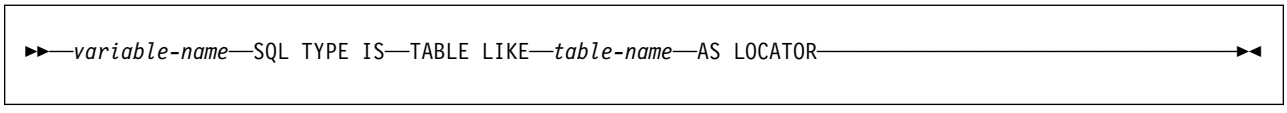


Table locator declaration in C

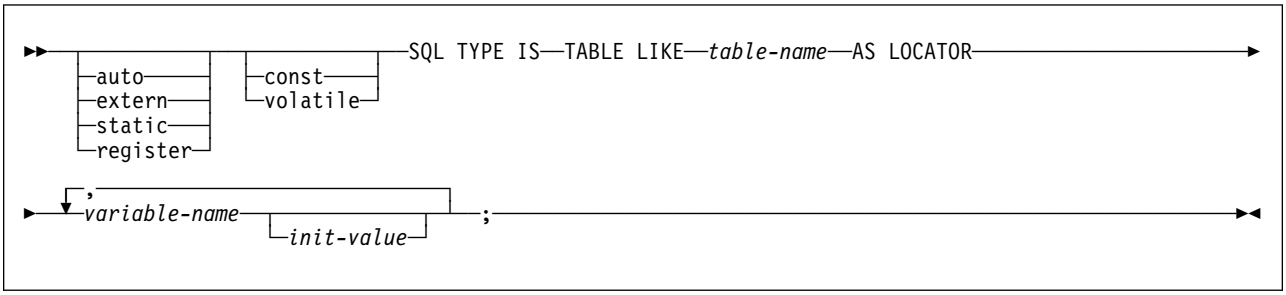


Table locator declaration in COBOL

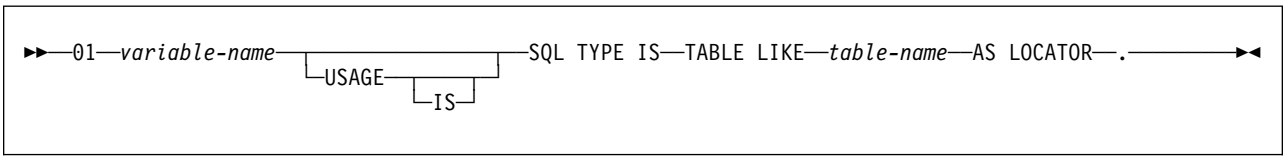
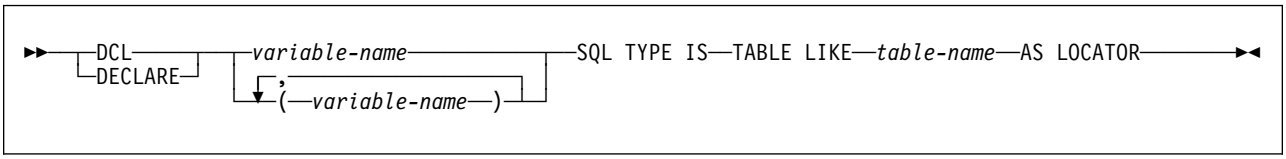


Table locator declaration in PL/I



3. Declare a cursor to access the rows in each transition table.
4. Assign the input parameter values to the table locators.
5. Access rows from the transition tables using the cursors that are declared for the transition tables.

The following examples show how a user-defined function that is written in each of the supported host languages accesses a transition table for a trigger. The transition table, NEWEMP, contains modified rows of the employee sample table. The trigger is defined like this:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES (CHECKEMP(TABLE NEWEMPS));
  END;
```

The user-defined function definition looks like this:

```
CREATE FUNCTION CHECKEMP(TABLE LIKE EMP AS LOCATOR)
  RETURNS INTEGER
  EXTERNAL NAME 'CHECKEMP'
  PARAMETER STYLE DB2SQL
  LANGUAGE language;
```

Assembler: Figure 31 on page 184 shows how an assembler program accesses rows of transition table NEWEMPS.

```
CHECKEMP CSECT
         SAVE  (14,12)             ANY SAVE SEQUENCE
         LR    R12,R15             CODE ADDRESSABILITY
         USING CHECKEMP,R12        TELL THE ASSEMBLER
         LR    R7,R1               SAVE THE PARM POINTER
         USING PARMAREA,R7         SET ADDRESSABILITY FOR PARMS
         USING SQLDSECT,R8         ESTABLISH ADDRESSIBILITY TO SQLDSECT
         L     R6,PROGSIZE         GET SPACE FOR USER PROGRAM
         GETMAIN R,LV=(6)          GET STORAGE FOR PROGRAM VARIABLES
         LR    R10,R1              POINT TO THE ACQUIRED STORAGE
         LR    R2,R10              POINT TO THE FIELD
         LR    R3,R6               GET ITS LENGTH
         SR    R4,R4               CLEAR THE INPUT ADDRESS
         SR    R5,R5               CLEAR THE INPUT LENGTH
         MVCL  R2,R4               CLEAR OUT THE FIELD
         ST    R13,FOUR(R10)       CHAIN THE SAVEAREA PTRS
         ST    R10,EIGHT(R13)      CHAIN SAVEAREA FORWARD
         LR    R13,R10             POINT TO THE SAVEAREA
         USING PROGAREA,R13        SET ADDRESSABILITY
         ST    R6,GETLENTH         SAVE THE LENGTH OF THE GETMAIN

  ⋮
************************************************************
* Declare table locator host variable TRIGTBL              *
************************************************************
TRIGTBL SQL TYPE IS TABLE LIKE EMPLOYEE AS LOCATOR
************************************************************
* Declare a cursor to retrieve rows from the transition    *
* table                                                    *
************************************************************
         EXEC SQL DECLARE C1 CURSOR FOR                              X
               SELECT LASTNAME FROM TABLE(:TRIGTBL LIKE EMP)         X
              WHERE SALARY > 100000
```

Figure 31 (Part 1 of 2). How an Assembler user-defined function accesses a transition table

```
*************************************************************
* Copy table locator for trigger transition table          *
*************************************************************
         L     R2,TABLOC           GET ADDRESS OF LOCATOR
         L     R2,0(0,R2)          GET LOCATOR VALUE
          ST    R2,TRIGTBL
         EXEC SQL OPEN C1
         EXEC SQL FETCH C1 INTO :NAME

  ⋮
         EXEC SQL CLOSE C1
  ⋮
PROGAREA DSECT                     WORKING STORAGE FOR THE PROGRAM
SAVEAREA DS    18F                 THIS ROUTINE'S SAVE AREA
GETLENTH DS    A                   GETMAIN LENGTH FOR THIS AREA
  ⋮
NAME     DS    CL24
  ⋮
         DS    0D
PROGSIZE EQU   *-PROGAREA          DYNAMIC WORKAREA SIZE
PARMAREA DSECT
TABLOC   DS    A                   INPUT PARAMETER FOR TABLE LOCATOR
  ⋮
         END   CHECKEMP
```

Figure 31 (Part 2 of 2). How an Assembler user-defined function accesses a transition table

C or C++: Figure 32 on page 186 shows how a C or C++ program accesses rows of transition table NEWEMPS.

```
int CHECK_EMP(int trig_tbl_id
)
{

  ⋮
  /*********************************************************/
  /* Declare table locator host variable trig_tbl_id  */
  /*********************************************************/
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS TABLE LIKE EMP AS LOCATOR trig_tbl_id;
    char name[25];
  EXEC SQL END DECLARE SECTION;

  ⋮
  /*********************************************************/
  /* Declare a cursor to retrieve rows from the transition  */
  /* table                                                 */
  /*********************************************************/
  EXEC SQL DECLARE C1 CURSOR FOR
    SELECT NAME FROM TABLE(:trig_tbl_id LIKE EMPLOYEE)
    WHERE SALARY > 100000;
  /*********************************************************/
  /* Fetch a row from transition table                     */
  /*********************************************************/
  EXEC SQL OPEN C1;
  EXEC SQL FETCH C1 INTO :name;

  ⋮
  EXEC SQL CLOSE C1;

  ⋮
}
```

Figure 32. How a C or C++ user-defined function accesses a transition table

COBOL: Figure 33 on page 187 shows how a COBOL program accesses rows of transition table NEWEMPS.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CHECKEMP.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      ********************************************************
      * Declare table locator host variable TRIG-TBL-ID      *
      ********************************************************
       01  TRIG-TBL-ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR.
       01  NAME   PIC X(24).

  ⋮
       LINKAGE SECTION.

  ⋮
       PROCEDURE DIVISION USING TRIG-TBL-ID.

  ⋮
      ********************************************************
      * Declare cursor to retrieve rows from transition table *
      ********************************************************
       EXEC SQL DECLARE C1 CURSOR FOR
         SELECT NAME FROM TABLE(:TRIG-TBL-ID LIKE EMP)
         WHERE SALARY > 100000 END-EXEC.
      ********************************************************
      * Fetch a row from transition table                    *
      ********************************************************
       EXEC SQL OPEN C1 END-EXEC.
       EXEC SQL FETCH C1 INTO :NAME END-EXEC.

  ⋮
       EXEC SQL CLOSE C1 END-EXEC.

  ⋮
       PROG-END.
       GOBACK.
```

Figure 33. How a COBOL user-defined function accesses a transition table

PL/I: Figure 34 on page 188 shows how a PL/I program accesses rows of transition table NEWEMPS.

```
CHECK_EMP: PROC(TRIG_TBL_ID) RETURNS(BIN FIXED(31))
           OPTIONS(MAIN NOEXECOPS REENTRANT);
/*****************************************************/
/* Declare table locator host variable TRIG_TBL_ID  */
/*****************************************************/
DECLARE TRIG_TBL_ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR;
DECLARE NAME CHAR(24);

 ⋮
/*****************************************************/
/* Declare a cursor to retrieve rows from the       */
/* transition table                                  */
/*****************************************************/
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT NAME FROM TABLE(:TRIG_TBL_ID LIKE EMP)
  WHERE SALARY > 100000;
/*****************************************************/
/* Retrieve rows from the transition table          */
/*****************************************************/
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 INTO :NAME;

 ⋮
EXEC SQL CLOSE C1;

 ⋮
END CHECK_EMP;
```

Figure 34. How a PL/I user-defined function accesses a transition table

Preparing a user-defined function for execution

To prepare a user-defined function for execution, perform these steps:

1. Precompile the user-defined function program and bind the DBRM into a package.

   You need to do this only if your user-defined function contains SQL statements. You do not need to bind a plan for the user-defined function.

2. Compile the user-defined function program and link-edit it with Language Environment and RRSAF.

   You must compile the program with a compiler that supports Language Environment and link-edit the appropriate Language Environment components with the user-defined function. You must also link-edit the user-defined function with RRSAF.

   For the minimum compiler and Language Environment requirements for user-defined functions, see Appendix G, “Prerequisites of Version 6 of DB2 for OS/390” on page 301.

   The program preparation JCL samples DSNHASM, DSNHC, DSNHCPP, DSNHICOB, and DSNHPLI show you how to precompile, compile, and link-edit assembler, C, C++, COBOL, and PL/I DB2 programs. If your DB2 subsystem works with Language Environment, you can use this sample JCL when you prepare your user-defined functions. For object-oriented programs in C++ or COBOL, see JCL samples DSNHCPP2 and DSNHICB2 for program preparation hints.

3. If your user-defined function contains SQL statements, grant EXECUTE authority on the user-defined function package to the function definer.

Making a user-defined function reentrant

Compiling and link-editing your user-defined function as reentrant is recommended. (For an assembler program, you must also code the user-defined function to be reentrant.) Reentrant user-defined functions have the following advantages:

- The operating system does not need to load the user-defined function into storage every time the user-defined function is called.
- Multiple tasks in a WLM-established stored procedures address space can share a single copy of the user-defined function. This decreases the amount of virtual storage that is needed for code in the address space.

Preparing user-defined functions that contain multiple programs: If your user-defined function consists of several programs, you must bind each program that contains SQL statements into a separate package. The definer of the user-defined function must have EXECUTE authority for all packages that are part of the user-defined function.

When the primary program of a user-defined function calls another program, DB2 uses the CURRENT PACKAGESET special register to determine the collection to search for the called program's package. The primary program can change this collection ID by executing the statement SET CURRENT PACKAGESET.

Determining the authorization ID for user-defined function invocation

If you invoke your user-defined function statically, the authorization ID under which that function is invoked is the owner of the package that contains the user-defined function invocation.

If you invoke your user-defined function dynamically, the authorization ID under which the user-defined function is invoked depends on the dynamic SQL statement behavior for the package that contains the user-defined function invocation.

Dynamic SQL statement behavior influences a number of features of an application program. See Section 6 of *DB2 Application Programming and SQL Guide* for more information. on dynamic SQL statement behavior.

For more information on the authorization required to invoke and execute SQL statements in a user-defined function, see Chapter 6 of *DB2 SQL Reference*.

Preparing user-defined functions to run concurrently

Multiple user-defined functions and stored procedures can run concurrently, each under its own OS/390 task (TCB).

To maximize the number of user-defined functions and stored procedures that can run concurrently, follow these preparation recommendations:

- Ask the system administrator to set the region size parameter in the startup procedures for the WLM-established stored procedures address spaces to REGION=0. This lets an address space obtain the largest possible amount of storage below the 16-MB line.
- Limit storage required by application programs below the 16-MB line by:

  - Link-editing programs with the AMODE(31) and RMODE(ANY) attributes
  - Compiling COBOL programs with the RES and DATA(31) options
- Limit storage that is required by Language Environment by using these run-time options:

HEAP(,,ANY)	Allocates program heap storage above the 16-MB line
STACK(,,ANY,)	Allocates program stack storage above the 16-MB line
STORAGE(,,,4K)	Reduces reserve storage area below the line to 4 KB
BELOWHEAP(4K,,)	Reduces the heap storage below the line to 4 KB
LIBSTACK(4K,,)	Reduces the library stack below the line to 4 KB
ALL31(ON)	Causes all programs that are contained in the external user-defined function to execute with AMODE(31) and RMODE(ANY)

  The definer can list these options as values of the RUN OPTIONS parameter of CREATE FUNCTION, or the system administrator can establish these options as defaults during Language Environment installation.

  For example, the RUN OPTIONS option parameter could contain:

  ```
  H(,,ANY),STAC(,,ANY,),STO(,,,4K),BE(4K,,),LIBS(4K,,),ALL31(ON)
  ```

- Ask the system administrator to set the NUMTCB parameter for WLM-established stored procedures address spaces to a value greater than 1. This lets more than one TCB run in an address space. Be aware that setting NUMTCB to a value greater than 1 also reduces your level of application program isolation. For example, a bad pointer in one application can overwrite memory that is allocated by another application.

Testing a user-defined function

Some commonly used debugging tools, such as TSO TEST, are not available in the environment where user-defined functions run. This section describes some alternative testing strategies.

CODE/370: You can use the CoOperative Development Environment/370 licensed program, which works with Language Environment, to test DB2 for OS/390 user-defined functions written in any of the supported languages. You can use CODE/370 either interactively or in batch mode.

Using CODE/370 interactively: To test a user-defined function interactively using CODE/370, you must use the CODE/370 PWS Debug Tool on a workstation. You must also have CODE/370 installed on the OS/390 system where the user-defined function runs. To debug your user-defined function using the PWS Debug Tool:

1. Compile the user-defined function with the TEST option. This places information in the program that the Debug Tool uses.
2. Invoke the Debug Tool. One way to do that is to specify the Language Environment run-time TEST option. The TEST option controls when and how the Debug Tool is invoked. The most convenient place to specify run-time

options is with the RUN OPTIONS parameter of CREATE FUNCTION or ALTER FUNCTION. See “Components of a user-defined function definition” on page 146 for more information on the RUN OPTIONS parameter.

For example, assume that you code this option:

```
TEST(ALL,*,PROMPT,JBJONES%SESSNA:)
```

The parameter values cause the following things to happen:

ALL
The Debug Tool gains control when an attention interrupt, abend, or program or Language Environment condition of severity 1 and above occurs.

***** Debug commands will be entered from the terminal.

PROMPT
The Debug Tool is invoked immediately after Language Environment initialization.

JBJONES%SESSNA:
CODE/370 initiates a session on a workstation identified to APPC/MVS as JBJONES with a session ID of SESSNA.

3. If you want to save the output from your debugging session, issue a command that names a log file. For example:

   ```
   SET LOG ON FILE dbgtool.log;
   ```

   This command starts logging to a file on the workstation called dbgtool.log. This should be the first command that you enter from the terminal or include in your commands file.

Using CODE/370 in batch mode: To test your user-defined function in batch mode, you must have the CODE/370 Mainframe Interface (MFI) Debug Tool installed on the OS/390 system where the user-defined function runs. To debug your user-defined function in batch mode using the MFI Debug Tool:

1. If you plan to use the Language Environment run-time TEST option to invoke CODE/370, compile the user-defined function with the TEST option. This places information in the program that the Debug Tool uses during a debugging session.
2. Allocate a log data set to receive the output from CODE/370. Put a DD statement for the log data set in the start-up procedure for the stored procedures address space.
3. Enter commands in a data set that you want CODE/370 to execute. Put a DD statement for that data set in the start-up procedure for the stored procedures address space. To define the data set that contains Debug Tool commands to CODE/370, specify the data set name or DD name in the TEST run-time option. For example, this option tells CODE/370 to look for the commands in the data set that is associated with DD name TESTDD:

   ```
   TEST(ALL,TESTDD,PROMPT,*)
   ```

   The first command in the commands data set should be:

   ```
   SET LOG ON FILE ddname;
   ```

   This command directs output from your debugging session to the log data set you defined in step 2. For example, if you defined a log data set with DD name

INSPLOG in the start-up procedure for the stored procedures address space, the first command should be:

```
SET LOG ON FILE INSPLOG;
```

4. Invoke the Debug Tool. Two possible methods are:
   - Specify the run-time TEST option. The most convenient place to do that is in the RUN OPTIONS parameter of CREATE FUNCTION or ALTER FUNCTION.
   - Put CEETEST calls in the user-defined function source code. If you use this approach for an existing user-defined function, you must recompile, link edit again, and bind it, and then issue the STOP FUNCTION SPECIFIC and START FUNCTION SPECIFIC commands to reload the user-defined function.

   You can combine the run-time TEST option with CEETEST calls. For example, you might want to use TEST to name the commands data set but use CEETEST calls to control when the Debug Tool takes control.

For more information on CODE/370, see *CoOperative Development Environment/370: Debug Tool*.

Route debugging messages to SYSPRINT: You can include simple print statements in your user-defined function code that you route to the SYSPRINT data set. Then use System Display and Search Facility (SDSF) to examine the SYSPRINT contents while the WLM-established stored procedure address space is running. You can serialize I/O by running the WLM-established stored procedure address space with NUMTCB=1.

Driver applications: You can write a small driver application that calls the user-defined function as a subprogram and passes the parameter list for the user-defined function. You can then test and debug the user-defined function as a normal DB2 application under TSO. You can then use TSO TEST and other commonly used debugging tools.

SQL insert: You can use SQL to insert debugging information into a DB2 table. This allows other machines in the network (such as a workstation) to easily access the data in the table using DRDA access.

DB2 discards the debugging information if the application executes the ROLLBACK statement. To prevent the loss of the debugging data, code the calling application so that it retrieves the diagnostic data before executing the ROLLBACK statement.

Invoking a user-defined function

You can invoke a sourced or external user-defined scalar function in an SQL statement wherever you use an expression. For a table function, you can invoke the user-defined function only in the FROM clause of a SELECT statement. The invoking SQL statement can be in a stand-alone program, a stored procedure, a trigger body, or another user-defined function.

See the following sections for details you should know before you invoke a user-defined function:

- “Syntax for user-defined function invocation” on page 193

- "Ensuring that DB2 executes the intended user-defined function" on page 193
- "Casting of user-defined function arguments" on page 199
- "What happens when a user-defined function abnormally terminates" on page 200
- "Other considerations for user-defined function invocation" on page 200

Syntax for user-defined function invocation

Use the syntax shown in Figure 35 when you invoke a user-defined scalar function:

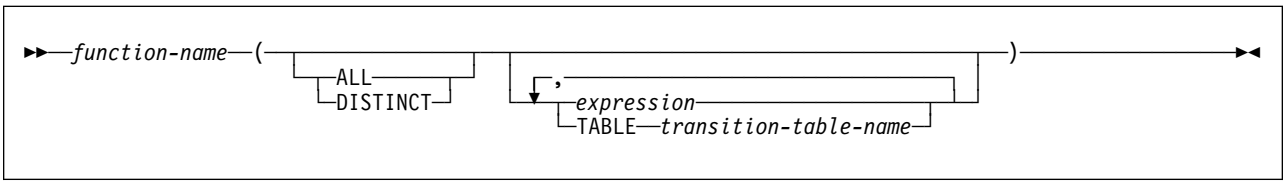


Figure 35. Syntax for user-defined scalar function invocation

Use the syntax shown in Figure 36 when you invoke a table function:

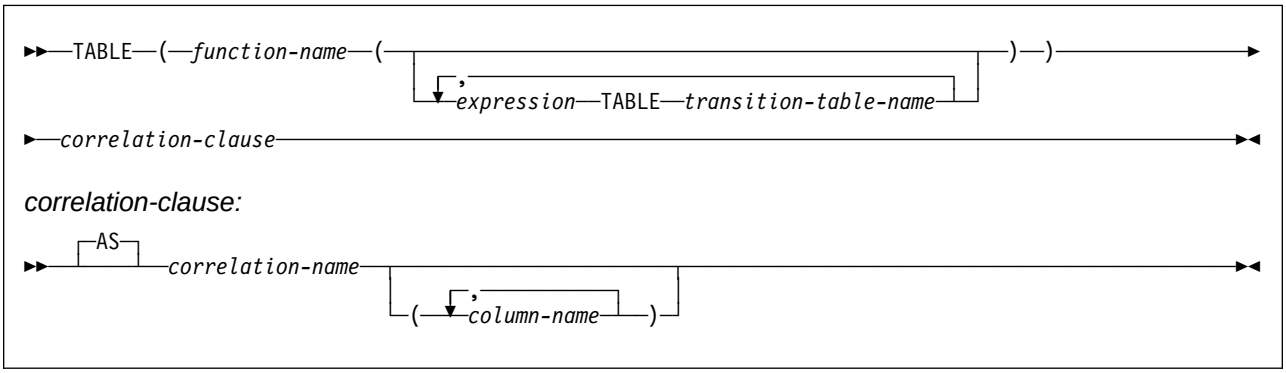


Figure 36. Syntax for table function invocation

Ensuring that DB2 executes the intended user-defined function

Several user-defined functions with the same name but different numbers or types of parameters can exist in a DB2 subsystem. Several user-defined functions with the same name can have the same number of parameters, as long as the data types of any of the first 30 parameters are different. In addition, several user-defined functions might have the same name as a built-in function. When you invoke a user-defined function, DB2 must determine which user-defined function or built-in function to execute. This process is known as *function resolution*. You need to understand DB2's function resolution process to ensure that you invoke the user-defined function that you want to invoke.

DB2 performs these steps for function resolution:

1. Determines which function instances are candidates for execution. If no candidates exist, DB2 issues an SQL error message.
2. Compares the data types of the input parameters to determine which candidates best fit the invocation.

   For a qualified function invocation, the result of the data type comparison is one best fit. That best fit is the choice for execution.

For an unqualified function invocation, DB2 might find multiple best fits because the same function name with the same input parameters can exist in different schemas.

3. Chooses, if two or more candidates fit the unqualified function invocation equally well, the user-defined function whose schema name is earliest in the SQL path.

   For example, suppose functions SCHEMA1.X and SCHEMA2.X fit a function invocation equally well. Assume the SQL path is:

   ```
   "SCHEMA2", "SYSPROC", "SYSIBM", "SCHEMA1", "SYSFUN"
   ```

   In this case, DB2 chooses function SCHEMA2.X.

The remainder of this section discusses details of the function resolution process and gives suggestions on how you can ensure that DB2 picks the right function.

How DB2 chooses candidate functions

An instance of a user-defined function is a candidate for execution only if it meets all of the following criteria:

- If the function name is qualified in the invocation, the schema of the function instance matches the schema in the function invocation.

  If the function name is unqualified in the invocation, the schema of the function instance matches a schema in the invoker's SQL path.

- The name of the function instance matches the name in the function invocation.
- The number of input parameters in the function instance matches the number of input parameters in the function invocation.
- The function invoker is authorized to execute the function instance.
- The type of each of the input parameters in the function invocation matches or is *promotable* to the type of the corresponding parameter in the function instance.

  For a function invocation that passes a transition table, the data type, length, precision, and scale of each column in the transition table must match the data type, length, precision, and scale of each column of the table that is named in the function instance definition. For information on transition tables, see “Using triggers for active data” on page 211.

- The bind timestamp for the user-defined function must be older than the bind timestamp for the package or plan in which the user-defined function is invoked.

  If DB2 authorization checking is in effect, and DB2 performs an automatic rebind on a plan or package that contains a user-defined function invocation, any user-defined functions that were created after the original BIND or REBIND of the invoking plan or package are not candidates for execution.

  If you use an access control authorization exit routine, some user-defined functions that were not candidates for execution before the original BIND or REBIND of the invoking plan or package might become candidates for execution during the automatic rebind of the invoking plan or package. See Appendix B (Volume 2) of *DB2 Administration Guide* for information about function resolution with access control authorization exit routines.

If a user-defined function is invoked during an automatic rebind, and that user-defined function is invoked from a trigger body and receives a transition table, then the form of the invoked function that DB2 uses for function selection includes only the columns of the transition table that existed at the time of the original BIND or REBIND of the package or plan for the invoking program.

To determine whether a data type is promotable to another data type, see Table 22. The first column lists data types in function invocations. The second column lists data types to which the types in the first column can be promoted, in order from best fit to worst fit. For example, suppose that in this statement, the data type of A is SMALLINT:

```
SELECT USER1.ADDTWO(A) FROM TABLEA;
```

Two instances of USER1.ADDTWO are defined: one with an input parameter of type INTEGER and one with an input parameter of type DECIMAL. Both function instances are candidates for execution because the SMALLINT type is promotable to either INTEGER or DECIMAL. However, the instance with the INTEGER type is a better fit because INTEGER is higher in the list than DECIMAL.

Table 22. Promotion of data types

Data type in function invocation	Possible fits (in best-to-worst order)
CHAR or GRAPHIC	CHAR or GRAPHIC VARCHAR or VARGRAPHIC CLOB or DBCLOB
VARCHAR or VARGRAPHIC	VARCHAR or VARGRAPHIC CLOB or DBCLOB
CLOB or DBCLOB[1]	CLOB or DBCLOB
BLOB[1]	BLOB
SMALLINT	SMALLINT INTEGER DECIMAL REAL DOUBLE
INTEGER	INTEGER DECIMAL REAL DOUBLE
DECIMAL	DECIMAL REAL DOUBLE
REAL[2]	REAL DOUBLE
DOUBLE[3]	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
ROWID	ROWID
Distinct type	Distinct type with same name

Notes to Table 22:

1. This promotion also applies if the parameter type in the invocation is a LOB locator for a LOB with this data type.
2. The FLOAT type with a length of less than 22 is equivalent to REAL.
3. The FLOAT type with a length of greater than or equal to 22 is equivalent to DOUBLE.

How DB2 chooses the best fit among candidate functions

More than one function instance might be a candidate for execution. In that case, DB2 determines which function instances are the best fit for the invocation by comparing parameter data types.

If the data types of all parameters in a function instance are the same as those in the function invocation, that function instance is a best fit. If no exact match exists, DB2 compares data types in the parameter lists from left to right, using this method:

1. DB2 compares the data types of the first parameter in the function invocation to the data type of the first parameter in each function instance.
2. For the first parameter, if one function instance has a data type that fits the function invocation better than the data types in the other instances, that function is a best fit. Table 22 on page 195 shows the possible fits for each data type, in best-to-worst order.
3. If the data types of the first parameter are the same for all function instances, DB2 repeats this process for the next parameter. DB2 continues this process for each parameter until it finds a best fit.

For example, suppose that a qualified function invocation has three parameters whose data types are VARCHAR, SMALLINT, and DECIMAL. Two function instances meet the criteria in "How DB2 chooses candidate functions" on page 194 and are therefore candidates for execution. Candidate function 1 has parameters with data types VARCHAR, INTEGER, and DOUBLE. Candidate function 2 has parameters with data types VARCHAR, REAL, and DOUBLE. For the first parameter, the data types for both candidate functions fit the function invocation equally well. However, for the second parameter, the data type of candidate function 1 (INTEGER) fits the data type in the invocation (SMALLINT) better than the data type of candidate function 2 (REAL). Therefore, candidate function 1 is the choice for execution.

How you can simplify function resolution

When you invoke a function, use the qualified name. This causes DB2 to search for functions only in the schema you specify. This has two advantages:

- DB2 is less likely to choose a function that you did not intend to use. Several functions might fit the invocation equally well. DB2 picks the function whose schema name is earliest in the SQL path, which might not be the function you want.
- The number of candidate functions is smaller, so DB2 takes less time for function resolution.
- Cast parameters in a user-defined function invocation to the types in the user-defined function definition. For example, if an input parameter for user-defined function FUNC is defined as DECIMAL(13,2), and the value you

want to pass to the user-defined function is an integer value, cast the integer value to DECIMAL(13,2):

```
SELECT FUNC(CAST (INTCOL AS DECIMAL(13,2))) FROM T1;
```

- Avoid defining user-defined function parameters as CHAR, GRAPHIC, SMALLINT or REAL. Use VARCHAR, VARGRAPHIC, INTEGER or DOUBLE instead. An invocation of a user-defined function defined with parameters of type CHAR, GRAPHIC, SMALLINT, or REAL must use parameters of the same types. For example, if user-defined function FUNC is defined with a parameter of type SMALLINT, only an invocation with a parameter of type SMALLINT resolves correctly. An invocation like this does not resolve to FUNC because the constant 123 is of type INTEGER, not SMALLINT:

  ```
  SELECT FUNC(123) FROM T1;
  ```

  # If you must define parameters for a user-defined function as CHAR, and you
  # call the user-defined function from a C program or SQL procedure, you need to
  # cast the corresponding parameter values in the user-defined function invocation
  # to CHAR to ensure that DB2 invokes the correct function. For example,
  # suppose that a C program calls user-defined function CVRTNUM, which takes
  # one input parameter of type CHAR(6). Also suppose that you declare host
  # variable empnumbr as `char empnumbr[6]`. When you invoke CVRTNUM, cast
  # empnumbr to CHAR:

  ```
  UPDATE EMP
   SET EMPNO=CVRTNUM(CHAR(:empnumbr))
   WHERE EMPNO = :empnumbr;
  ```

Using DSN_FUNCTION_TABLE to see how DB2 resolves a function

You can use DB2's EXPLAIN tool to obtain information about how DB2 resolves functions. DB2 stores the information in a table called DSN_FUNCTION_TABLE, which you create. DB2 puts a row in DSN_FUNCTION_TABLE for each function that is referenced in an SQL statement when one of the following events occurs:

- You execute the SQL EXPLAIN statement on an SQL statement that contains user-defined function invocations.
- You run a program whose plan is bound with EXPLAIN(YES), and the program executes an SQL statement that contains user-defined function invocations.

Before you use EXPLAIN to obtain information about function resolution, create DSN_FUNCTION_TABLE. The table definition looks like this:

```
CREATE TABLE DSN_FUNCTION_TABLE
  (QUERYNO            INTEGER      NOT NULL WITH DEFAULT,
   QBLOCKNO           INTEGER      NOT NULL WITH DEFAULT,
   APPLNAME           CHAR(8)      NOT NULL WITH DEFAULT,
   PROGNAME           CHAR(8)      NOT NULL WITH DEFAULT,
   COLLID             CHAR(18)     NOT NULL WITH DEFAULT,
   GROUP_MEMBER       CHAR(8)      NOT NULL WITH DEFAULT,
   EXPLAIN_TIME       TIMESTAMP    NOT NULL WITH DEFAULT,
   SCHEMA_NAME        CHAR(8)      NOT NULL WITH DEFAULT,
   FUNCTION_NAME      CHAR(18)     NOT NULL WITH DEFAULT,
   SPEC_FUNC_NAME     CHAR(18)    NOT NULL WITH DEFAULT,
   FUNCTION_TYPE      CHAR(2)      NOT NULL WITH DEFAULT,
   VIEW_CREATOR       CHAR(8)      NOT NULL WITH DEFAULT,
   VIEW_NAME          CHAR(18)     NOT NULL WITH DEFAULT,
   PATH               VARCHAR(254) NOT NULL WITH DEFAULT,
   FUNCTION_TEXT      VARCHAR(254) NOT NULL WITH DEFAULT);
```

Columns QUERYNO, QBLOCKNO, APPLNAME, PROGNAME, COLLID, and GROUP_MEMBER have the same meanings as in the PLAN_TABLE. The meanings of the other columns are:

EXPLAIN_TIME
Timestamp when the EXPLAIN statement was executed.

SCHEMA_NAME
Schema name of the function that is invoked in the explained statement.

FUNCTION_NAME
Name of the function that is invoked in the explained statement.

SPEC_FUNC_NAME
Specific name of the function that is invoked in the explained statement.

FUNCTION_TYPE
The type of function that is invoked in the explained statement. Possible values are:

SU Scalar function

TU Table function

VIEW_CREATOR
The creator of the view if the function that is specified in the FUNCTION_NAME column is referenced in a view definition. Otherwise, this field is blank.

VIEW_NAME
The name of the view if the function that is specified in the FUNCTION_NAME column is referenced in a view definition. Otherwise, this field is blank.

PATH
The value of the SQL path when DB2 resolved the function reference.

FUNCTION_TEXT
The text of the function reference (the function name and parameters). If the function reference exceeds 100 bytes, this column contains the first 100 bytes.

For a function specified in infix notation, FUNCTION_TEXT contains only the function name. For example, suppose a user-defined function named / is in the function reference A/B. Then FUNCTION_TEXT contains only /, not A/B.

Casting of user-defined function arguments

Whenever you invoke a user-defined function, DB2 assigns your input parameter values to parameters with the data types and lengths in the user-defined function definition.

When you invoke a user-defined function that is sourced on another function, DB2 casts your parameters to the data types and lengths of the sourced function.

The following example demonstrates what happens when the parameter definitions of a sourced function differ from those of the function on which it is sourced.

Suppose that external user-defined function TAXFN1 is defined like this:

```
CREATE FUNCTION TAXFN1(DEC(6,0))
  RETURNS DEC(5,2)
  PARAMETER STYLE DB2SQL
  LANGUAGE C
  EXTERNAL NAME TAXPROG;
```

Sourced user-defined function TAXFN2, which is sourced on TAXFN1, is defined like this:

```
CREATE FUNCTION TAXFN2(DEC(8,2))
  RETURNS DEC(5,0)
  SOURCE TAXFN1;
```

You invoke TAXFN2 using this SQL statement:

```
UPDATE TB1
  SET SALESTAX2 = TAXFN2(PRICE2);
```

TB1 is defined like this:

```
CREATE TABLE TB1
  (PRICE1    DEC(6,0),
   SALESTAX1 DEC(5,2),
   PRICE2    DEC(9,2),
   SALESTAX2 DEC(7,2));
```

Now suppose that PRICE2 has the DECIMAL(9,2) value 0001234.56. DB2 must first assign this value to the data type of the input parameter in the definition of TAXFN2, which is DECIMAL(8,2). The input parameter value then becomes 001234.56. Next, DB2 casts the parameter value to a source function parameter, which is DECIMAL(6,0). The parameter value then becomes 001234. (When you cast a value, that value is truncated, rather than rounded.)

Now, if TAXFN1 returns the DECIMAL(5,2) value 123.45, DB2 casts the value to DECIMAL(5,0), which is the result type for TAXFN2, and the value becomes 00123. This is the value that DB2 assigns to column SALESTAX2 in the UPDATE statement.

Casting of parameter markers: If you use a parameter marker in a function invocation, you must cast the parameter to the correct type. For example, if function FX is defined with one parameter of type INTEGER, an invocation of FX with a parameter marker looks like this:

```
  SELECT FX(CAST(? AS INTEGER)) FROM T1;
```

What happens when a user-defined function abnormally terminates

When an external user-defined function abnormally terminates, your program receives SQLCODE -430 for the invoking statement, and DB2 places the unit of work that contains the invoking statement in a must-rollback state. You should include code in your program to check for a user-defined function abend and to roll back the unit of work that contains the user-defined function invocation.

Other considerations for user-defined function invocation

Invoke user-defined functions with external actions from SELECT lists: Invoke functions with external actions from SELECT lists, rather than predicates. The access path that DB2 chooses for a predicate determines whether a user-defined function in that predicate is executed. To ensure that DB2 executes the user-defined function (and therefore executes the external action) for each row of the result set, put the user-defined function invocation in the SELECT list.

Invoke user-defined functions defined as NOT DETERMINISTIC from SELECT lists: It is best to invoke nondeterministic user-defined functions from the SELECT list, rather than in a predicate. The following example demonstrates that invoking a nondeterministic user-defined function in a predicate can yield undesirable results.

Suppose that you execute this query:

```
SELECT COUNTER(), C1, C2 FROM T1 WHERE COUNTER() = 2;
```

Table T1 looks like this:

```
C1  C2
--  --
 1   b
 2   c
 3   a
```

COUNTER is a user-defined function that increments a variable in the scratchpad each time it is invoked.

DB2 invokes an instance of COUNTER in the predicate 3 times. Assume that COUNTER is invoked for row 1 first, for row 2 second, and for row 3 third. Then COUNTER returns 1 for row 1, 2 for row 2, and 3 for row 3. Therefore, row 2 satisfies the predicate WHERE COUNTER()=2, so DB2 evaluates the SELECT list for row 2. DB2 uses a different instance of COUNTER in the SELECT list from the instance in the predicate. Because the instance of COUNTER in the SELECT list is invoked only once, it returns a value of 1. Therefore, the result of the query is:

```
COUNTER()  C1 C2
---------  -- --
    1       2  c
```

This is not the result you might expect.

The results can differ even more, depending on the order in which DB2 retrieves the rows from the table. Suppose that an ascending index is defined on column C2. Then DB2 retrieves row 3 first, row 1 second, and row 2 third. This means that row 1 satisfies the predicate WHERE COUNTER()=2. The value of COUNTER in the SELECT list is again 1, so the result of the query in this case is:

```
COUNTER()  C1 C2
---------  -- --
    1       1  b
```

Figure DB2 cost information for accessing user-defined functions:
User-defined table functions add additional access cost to the execution of an SQL statement. For DB2 to factor in the effect of user-defined table functions in the selection of the best access path for an SQL statement, the total cost of the user-defined table function must be determined.

The total cost of a table function consists of the following three components:

- The initialization cost that results from the first call processing
- The cost that is associated with acquiring a single row
- The final call cost that performs the clean up processing

These costs, though, are not known to DB2 when I/O costs are added to the CPU cost.

To assist DB2 in determining the cost of user-defined table functions, you can use four fields in SYSIBM.SYSROUTINES. Use the following fields to provide cost information:

- IOS_PER_INVOC for the estimated number of I/Os per row
- INSTS_PER_INVOC for the estimated number of instructions
- INITIAL_IOS for the estimated number of I/Os performed the first and last time the function is invoked
- INITIAL_INSTS for the estimated number of instructions for the first and last time the function is invoked

These values, along with the CARDINALITY value of the table being accessed, are used by DB2 to determine the cost. The results of the calculations can influence such things as the join sequence for a multi-table join and the cost estimates generated for and used in predictive governing.

Determine values for the four fields by examining the source code for the table function. Estimate the I/Os by examining the code executed during the FIRST call and FINAL call. Look for the code executed during the OPEN, FETCH, and CLOSE calls. The costs for the OPEN and CLOSE calls can be amortized over the expected number of rows returned. Estimate the I/O cost by providing the number of I/Os that will be issued. Include the I/Os for any file access.

Figure the instruction cost by counting the number of high level instructions executed in the user-defined table function and multiplying it by a factor of 20. For assembler programs, the instruction cost is the number of assembler instructions.

If SQL statements are issued within the user-defined table function, use DB2 Estimator to determine the number of instructions and I/Os for the statements. Examining the JES job statistics for a batch program doing equivalent functions can also be helpful. For all fields, a precise number of instructions is not required. Because DB2 already accounts for the costs of invoking table functions, these costs should not be included in the estimates.

The following example shows how these fields can be updated. The authority to update is the same authority as that required to update any catalog statistics column.

```
UPDATE SYSIBM.SYSROUTINES SET
   IOS_PER_INVOC   = 0.0,
   INSTS_PER_INVOC = 4.5E3,
   INITIAL_IOS     = 2.0
   INITIAL_INSTS   = 1.0E4,
   CARDINALITY     = 5E3
WHERE
   SCHEMA = 'SYSADM' AND
   SPECIFICNAME = 'FUNCTION1' AND
   ROUTINETYPE = 'F';
```

Creating and using distinct types

A distinct type is a data type that you define using the CREATE DISTINCT TYPE statement. Each distinct type has the same internal representation as a built-in data type. You can use distinct types in the same way that you use built-in data types, in any type of SQL application except for a DB2 private protocol application.

This section presents the following information about distinct types:

- "Introduction to distinct types"
- "Creating a distinct type"
- "Using distinct types in applications" on page 203

Introduction to distinct types

The main reason to use distinct types is because DB2 enforces *strong typing* for distinct types. Strong typing ensures that only functions, procedures, comparisons, and assignments that are defined for a data type can be used.

For example, if you have defined a user-defined function to convert U.S. dollars to euro currency, you do not want anyone to use this same function to convert Japanese Yen to euros because the wrong amount is returned. Suppose you define three distinct types:

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL(9,2) WITH COMPARISONS;
CREATE DISTINCT TYPE EURO AS DECIMAL(9,2) WITH COMPARISONS;
CREATE DISTINCT TYPE JAPANESE_YEN AS DECIMAL(9,2) WITH COMPARISONS;
```

If a conversion function is defined that takes an input parameter of type US_DOLLAR as input, DB2 returns an error if you try to execute the function with an input parameter of type JAPANESE_YEN.

Creating a distinct type

Suppose you want to define some audio and video data in a DB2 table. You can define columns for both types of data as BLOB, but you might want to use a data type that more specifically describes the data. To do that, define distinct types. You can then use those types when you define columns in a table or manipulate the data in those columns. For example, you can define distinct types for the audio and video data like this:

```
CREATE DISTINCT TYPE AUDIO AS BLOB (1M);
CREATE DISTINCT TYPE VIDEO AS BLOB (1M);
```

Then, your CREATE TABLE statement might look like this:

```
CREATE TABLE VIDEO_CATALOG
  (VIDEO_NUMBER  CHAR(6)  NOT NULL,
   VIDEO_SOUND   AUDIO,
   VIDEO_PICS    VIDEO,
   ROW_ID        ROWID NOT NULL GENERATED ALWAYS);
```

You must define a column of type ROWID in the table because tables with any type of LOB columns require a ROWID column, and internally, the VIDEO_CATALOG table contains two LOB columns. For more information on LOB data, see “Working with large objects (LOBs)” on page 124. For syntax information for the CREATE DISTINCT TYPE statement, see *DB2 SQL Reference*.

After you define distinct types and columns of those types, you can use those data types in the same way you use built-in types. You can use the data types in assignments, comparisons, function invocations, and stored procedure calls. However, when you assign one column value to another or compare two column values, those values must be of the same distinct type. For example, you must assign a column value of type VIDEO to a column of type VIDEO, and you can compare a column value of type AUDIO only to a column of type AUDIO. When you assign a host variable value to a column with a distinct type, you can use any host data type that is compatible with the source data type of the distinct type. For example, to receive an AUDIO or VIDEO value, you can define a host variable like this:

```
SQL TYPE IS BLOB (1M) HVAV;
```

When you use a distinct type as an argument to a function, a version of that function must accept that distinct type. For example, if function SIZE takes a BLOB type as input, you cannot automatically use a value of type AUDIO as input. However, you can create a sourced user-defined function that takes the AUDIO type as input. For example:

```
CREATE FUNCTION SIZE(AUDIO)
  RETURNS INTEGER
  SOURCE SIZE(BLOB(1M));
```

Using distinct types in applications

You can use distinct types in applications to enforce strong typing, which means that functions, comparisons, and assignments that are defined for a data type can be executed. To use distinct types in applications, follow these steps:

1. Determine which distinct types you want to use. This determination depends on where and how you choose to enforce strong typing in your applications.
2. Create each distinct type based on one of DB2's built-in data types, such as INTEGER, CHARACTER, or DATE. See “Creating a distinct type” on page 202 for an example.
3. Create the DB2 tables you need that include columns of the distinct types you created. See “Creating a distinct type” on page 202 for an example.
4. Create user-defined functions to manipulate columns with distinct types. DB2 automatically creates cast functions between each distinct type and its base type. See “Example: Casting constants and host variables to distinct types to invoke a user-defined function” on page 204 for an example.

Invoking functions with distinct types

DB2 enforces strong typing when you pass arguments to a function. This means that you can pass arguments that have distinct types to a function if either of the following conditions is true:

- A version of the function that accepts those distinct types is defined.

  This also applies to infix operators. If you want to use one of the five built-in infix operators with your distinct types, you must define a version of that operator that accepts the distinct types.

- You can cast your distinct types to the argument types of the function.

  If you pass arguments to a function that accepts only distinct types, the arguments you pass must have the same distinct types as in the function definition. If the types are different, you must cast your arguments to the distinct types in the function definition. If you pass constants or host variables to a function that accepts only distinct types, you must cast the constants or host variables to the distinct types that the function accepts.

The following examples demonstrate how to use distinct types as arguments in function invocations.

Example: Defining a function with distinct types as arguments: Suppose you want to invoke the built-in function HOUR with a distinct type that is defined like this:

```
CREATE DISTINCT TYPE FLIGHT_TIME AS TIME WITH COMPARISONS;
```

The HOUR function takes only the TIME or TIMESTAMP data type as an argument, so you need a sourced function that is based on the HOUR function that accepts the FLIGHT_TIME data type. You might declare a function like this:

```
CREATE FUNCTION HOUR(FLIGHT_TIME)
  RETURNS INTEGER
  SOURCE SYSIBM.HOUR(TIME);
```

Example: Casting function arguments to acceptable types: You can invoke the HOUR function using the FLIGHT_TIME distinct type by listing all of the flights that depart at a certain hour of the day. This example casts a host variable of type CHAR to the TIME data type, then casts TIME to the FLIGHT_TIME distinct type, then invokes the HOUR function.

```
SELECT HOUR(CAST(TIME(:charhostvar) AS FLIGHT_TIME)) FROM FLIGHT_INFO;
```

Example: Using an infix operator with distinct type arguments: Suppose you want to add two values of type US_DOLLAR. Before you can do this, you must define a version of the + function that accepts values of type US_DOLLAR as operands:

```
CREATE FUNCTION "+"(US_DOLLAR,US_DOLLAR)
  RETURNS US_DOLLAR
  SOURCE SYSIBM."+"(DECIMAL(9,2),DECIMAL(9,2));
```

Because the US_DOLLAR type is based on the DECIMAL(9,2) type, the source function must be the version of + with arguments of type DECIMAL(9,2).

Example: Casting constants and host variables to distinct types to invoke a user-defined function: Suppose function EURO_TO_US is defined like this:

```
CREATE FUNCTION EURO_TO_US(EURO)
  RETURNS US_DOLLAR
  EXTERNAL NAME 'EUROCVT'
  PARAMETER STYLE DB2SQL
  LANGUAGE C;
```

This means that EURO_TO_US accepts only the EURO type as input. Therefore, if you want to call EURO_TO_US with a constant or host variable argument, you must cast that argument to distinct type EURO:

```
SELECT * FROM US_SALES
  WHERE TOTAL = EURO_TO_US(EURO(:H1));

SELECT * FROM US_SALES
  WHERE TOTAL = EURO_TO_US(EURO(10000));
```

Comparing distinct types

The basic rule for comparisons is that the data types of the operands must be compatible. The compatibility rule defines, for example, that all numeric types (SMALLINT, INTEGER, FLOAT, and DECIMAL) are compatible. That is, you can compare an INTEGER value with a value of type FLOAT. However, you cannot compare an object of a distinct type to an object of a different type. You can compare an object with a distinct type only to an object with exactly the same distinct type.

DB2 does not let you compare data of a distinct type directly to data of its source type. However, you can compare a distinct type to its source type by using a cast function.

For example, suppose you want to know which products sold more than US $100,000.00 in the U.S. in July of 1992. Because you cannot compare data of type US_DOLLAR with instances of data of the source type of US_DOLLAR (DECIMAL) directly, you must use a cast function to cast data from DECIMAL to US_DOLLAR or from US_DOLLAR to DECIMAL. Whenever you create a distinct type, DB2 creates two cast functions, one to cast from the source type to the distinct type and the other to cast from the distinct type to the source type. For distinct type US_DOLLAR, DB2 creates a cast function called DECIMAL and a cast function called US_DOLLAR. When you compare an object of type US_DOLLAR to an object of type DECIMAL, you can use one of those cast functions to make the data types identical for the comparison. Suppose table US_SALES is defined like this:

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         US_DOLLAR);
```

Then you can cast DECIMAL data to US_DOLLAR like this:

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > US_DOLLAR(100000.00)
  AND    MONTH = 7
  AND    YEAR  = 1992;
```

This satisfies the requirement that the compared data types are identical.

You cannot use host variables in statements that you prepare for dynamic execution; however, you can substitute parameter markers for host variables when

you prepare a statement, and then use host variables when you execute the statement.

If you use a parameter marker in a predicate of a query, and the column to which you compare the value represented by the parameter marker is of a distinct type, you must cast the parameter marker to the distinct type, or cast the column to its source type.

For example, suppose distinct type CNUM is defined like this:

```
CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISONS;
```

Table CUSTOMER is defined like this:

```
CREATE TABLE CUSTOMER
  (CUST_NUM     CNUM NOT NULL,
   FIRST_NAME   CHAR(30) NOT NULL,
   LAST_NAME    CHAR(30) NOT NULL,
   PHONE_NUM    CHAR(20) WITH DEFAULT,
   PRIMARY KEY  (CUST_NUM));
```

In an application program, you prepare a SELECT statement that compares the CUST_NUM column to a parameter marker. Because CUST_NUM is of a distinct type, you must cast the distinct type to its source type:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER
  WHERE CAST(CUST_NUM AS INTEGER) = ?
```

Alternatively, you can cast the parameter marker to the distinct type:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER
  WHERE CUST_NUM=CAST(? AS CNUM)
```

Making assignments involving distinct types

For assignments from columns to columns or from constants to columns for distinct types, the type of that value to be assigned must match the type of the object to which the value is assigned, or you must be able to cast one type to the other.

If you need to assign a value of one distinct type to a column of another distinct type, a function must exist that converts the value from one type to another. Because DB2 provides cast functions only between distinct types and their source types, you must write the function to convert from one distinct type to another.

Example: Assigning column values to columns with different distinct types: Suppose tables JAPAN_SALES and JAPAN_SALES_98 are defined like this:

```
CREATE TABLE JAPAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         JAPANESE_YEN);

CREATE TABLE JAPAN_SALES_98
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR);
```

You need to insert values from the TOTAL column in JAPAN_SALES into the TOTAL column of JAPAN_SALES_98. Because INSERT statements follow assignment rules, DB2 does not let you insert the values directly from one column to the other because the columns are of different distinct types. Suppose that a

user-defined function called US_DOLLAR has been written that accepts values of type JAPANESE_YEN as input and returns values of type US_DOLLAR. You can then use this function to insert values into the JAPAN_SALES_98 table:

```
INSERT INTO JAPAN_SALES_98
  SELECT PRODUCT_ITEM, US_DOLLAR(TOTAL)
  FROM JAPAN_SALES
  WHERE YEAR = 1998;
```

Example: Assigning column values with distinct types to host variables: The rules for assigning distinct types to host variables or host variables to columns of distinct types differ from the rules for constants and columns.

You can assign a column value of a distinct type to a host variable if you can assign a column value of the distinct type's source type to the host variable. In the following example, you can assign SIZECOL1 and SIZECOL2, which has distinct type SIZE, to host variables of type double and short because the source type of SIZE, which is INTEGER, can be assigned to host variables of type double or short.

```
EXEC SQL BEGIN DECLARE SECTION;
  double hv1;
  short  hv2;
EXEC SQL END DECLARE SECTION;
CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
⋮
SELECT SIZECOL1, SIZECOL2
  INTO :hv1, :hv2
  FROM TABLE1;
```

Example: Assigning host variable values to columns with distinct types: When you assign a value in a host variable to a column with a distinct type, the type of the host variable must be castable to the distinct type. For a table of base data types and the base data types to which they can be cast, see Table 22 on page 195.

In this example, values of host variable hv2 can be assigned to columns SIZECOL1 and SIZECOL2, because C data type short is equivalent to DB2 data type SMALLINT, and SMALLINT is promotable to data type INTEGER. However, values of hv1 cannot be assigned to SIZECOL1 and SIZECOL2, because C data type double, which is equivalent to DB2 data type DOUBLE, is not promotable to data type INTEGER.

```
EXEC SQL BEGIN DECLARE SECTION;
  double hv1;
  short  hv2;
EXEC SQL END DECLARE SECTION;
CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
⋮
INSERT INTO TABLE1
  VALUES (:hv1,:hv1);     /* Invalid statement */
INSERT INTO TABLE1
  VALUES (:hv2,:hv2);     /* Valid statement   */
```

Using distinct types in UNION

As with comparisons, DB2 enforces strong typing of distinct types in UNIONs. When you use a UNION to combine column values from several tables, the combined columns must be of the same types. For example, suppose you create a view that combines the values of the US_SALES, EUROPEAN_SALES, and JAPAN_SALES tables. The TOTAL columns in the three tables are of different distinct types, so before you combine the table values, you must convert the types of two of the TOTAL columns to the type of the third TOTAL column. Assume that the US_DOLLAR type has been chosen as the common distinct type. Because DB2 does not generate cast functions to convert from one distinct type to another, two user-defined functions must exist:

- A function that converts values of type EURO to US_DOLLAR
- A function that converts values of type JAPANESE_YEN to US_DOLLAR

Assume that these functions exist, and that both are called US_DOLLAR. Then you can execute a query like this to display a table of combined sales:

```
SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL
FROM US_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR(TOTAL)
FROM EUROPEAN_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR(TOTAL)
FROM JAPAN_SALES;
```

Because the result type of both US_DOLLAR functions is US_DOLLAR, you have satisfied the requirement that the distinct types of the combined columns are the same.

Combining distinct types with user-defined functions and LOBs

The example in this section demonstrates the following concepts:

- Creating a distinct type based on a LOB data type
- Defining a user-defined function with a distinct type as an argument
- Creating a table with a distinct type column that is based on a LOB type
- Defining a LOB table space, auxiliary table, and auxiliary index
- Inserting data from a host variable into a distinct type column based on a LOB column
- Executing a query that contains a user-defined function invocation
- Casting a LOB locator to the input data type of a user-defined function

Suppose you keep electronic mail documents that are sent to your company in a DB2 table. The DB2 data type of an electronic mail document is CLOB, but you define it as a distinct type so that you can control the types of operations that are performed on the electronic mail. The distinct type is defined like this:

```
CREATE DISTINCT TYPE E_MAIL AS CLOB(5M);
```

You have also defined and written user-defined functions to search for and return the following information about an electronic mail document:

- Subject
- Sender
- Date sent
- Message content

- Indicator of whether the document contains a user-specified string

The user-defined function definitions look like this:

```
CREATE FUNCTION SUBJECT(E_MAIL)
  RETURNS VARCHAR(200)
  EXTERNAL NAME 'SUBJECT'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;

CREATE FUNCTION SENDER(E_MAIL)
  RETURNS VARCHAR(200)
  EXTERNAL NAME 'SENDER'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;

CREATE FUNCTION SENDING_DATE(E_MAIL)
  RETURNS DATE
  EXTERNAL NAME 'SENDDATE'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;

CREATE FUNCTION CONTENTS(E_MAIL)
  RETURNS CLOB(1M)
  EXTERNAL NAME 'CONTENTS'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;

CREATE FUNCTION CONTAINS(E_MAIL, VARCHAR (200))
  RETURNS INTEGER
  EXTERNAL NAME 'CONTAINS'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION;
```

The table that contains the electronic mail documents is defined like this:

```
CREATE TABLE DOCUMENTS
  (LAST_UPDATE_TIME  TIMESTAMP,
   DOC_ROWID      ROWID NOT NULL GENERATED ALWAYS,
   A_DOCUMENT        E_MAIL);
```

Because the table contains a column with a source data type of CLOB, the table requires a ROWID column and an associated LOB table space, auxiliary table, and index on the auxiliary table. Use statements like this to define the LOB table space, the auxiliary table, and the index:

```
CREATE LOB TABLESPACE DOCTSLOB
  LOG YES
  GBPCACHE SYSTEM;

CREATE AUX TABLE DOCAUX_TABLE
  IN DOCTSLOB
  STORES DOCUMENTS COLUMN A_DOCUMENT;

CREATE INDEX A_IX_DOC ON DOCAUX_TABLE;
```

To populate the document table, you write code that executes an INSERT statement to put the first part of a document in the table, and then executes multiple UPDATE statements to concatenate the remaining parts of the document. For example:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hv_current_time[26];
  SQL TYPE IS CLOB (1M) hv_doc;
EXEC SQL END DECLARE SECTION;
/* Determine the current time and put this value */
/* into host variable hv_current_time.           */
/* Read up to 1 MB of document data from a file  */
/* into host variable hv_doc.                    */
⋮
/* Insert the time value and the first 1 MB of   */
/* document data into the table.                 */
EXEC SQL INSERT INTO DOCUMENTS
  VALUES(:hv_current_time, DEFAULT, E_MAIL(:hv_doc));

/* While there is more document data in the      */
/* file, read up to 1 MB more of data, and then  */
/* use an UPDATE statement like this one to      */
/* concatenate the data in the host variable     */
/* to the existing data in the table.            */
EXEC SQL UPDATE DOCUMENTS
  SET A_DOCUMENT = A_DOCUMENT || E_MAIL(:hv_doc)
  WHERE LAST_UPDATE_TIME = :hv_current_time;
```

Now that the data is in the table, you can execute queries to learn more about the documents. For example, you can execute this query to determine which documents contain the word “performance”:

```
SELECT SENDER(A_DOCUMENT), SENDING_DATE(A_DOCUMENT), SUBJECT(A_DOCUMENT)
  FROM DOCUMENTS
  WHERE CONTAINS(A_DOCUMENT,'performance') = 1;
```

Because the electronic mail documents can be very large, you might want to use LOB locators to manipulate the document data instead of fetching all of a document into a host variable. You can use a LOB locator on any distinct type that is defined on one of the LOB types. The following example shows how you can cast a LOB locator as a distinct type, and then use the result in a user-defined function that takes a distinct type as an argument:

```
EXEC SQL BEGIN DECLARE SECTION
  long hv_len;
  char hv_subject[200];
  SQL TYPE IS CLOB_LOCATOR hv_email_locator;
EXEC SQL END DECLARE SECTION
⋮
/* Select a document into a CLOB locator.        */
EXEC SQL SELECT A_DOCUMENT, SUBJECT(A_DOCUMENT)
  INTO :hv_email_locator, :hv_subject
  FROM DOCUMENTS
  WHERE LAST_UPDATE_TIME = :hv_current_time;
⋮
/* Extract the subject from the document.  The   */
/* SUBJECT function takes an argument of type    */
/* E_MAIL, so cast the CLOB locator as E_MAIL.   */
EXEC SQL SET :hv_subject =
  SUBJECT(CAST(:hv_email_locator AS E_MAIL));
⋮
```

Using triggers for active data

Triggers are sets of SQL statements that execute when a certain event occurs in a DB2 table. Like constraints, triggers can be used to control changes in DB2 databases. Triggers are more powerful, however, because they can monitor a broader range of changes and perform a broader range of actions than constraints can. For example, a constraint can disallow an update to the salary column of the employee table if the new value is over a certain amount. A trigger can monitor the amount by which the salary changes, as well as the salary value. If the change is above a certain amount, the trigger might substitute a valid value and call a user-defined function to send a notice to an administrator about the invalid update.

Triggers also move application logic into DB2, which can result in faster application development and easier maintenance. For example, you can write applications to control salary changes in the employee table, but each application program that changes the salary column must include logic to check those changes. A better method is to define a trigger that controls changes to the salary column. Then DB2 does the checking for any application that modifies salaries.

This section presents the following information about triggers:

- "Example of creating and using a trigger" on page 212
- "Parts of a trigger" on page 213
- "Invoking stored procedures and user-defined functions from triggers" on page 218
- "Trigger cascading" on page 219
- "Ordering of multiple triggers" on page 220
- "Interactions among triggers and referential constraints" on page 221
- "Creating triggers to obtain consistent results" on page 223

Example of creating and using a trigger

Triggers automatically execute a set of SQL statements whenever a specified event occurs. These SQL statements can perform tasks such as validation and editing of table changes, reading and modifying tables, or invoking functions or stored procedures that perform operations both inside and outside DB2.

You create triggers using the CREATE TRIGGER statement. Figure 37 shows an example of a CREATE TRIGGER statement.

```
                  1
CREATE TRIGGER REORDER
    2     3                                4
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
                         5
  REFERENCING NEW AS N_ROW
       6
  FOR EACH ROW MODE DB2SQL
  7
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  8
  BEGIN ATOMIC
  CALL ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                                N_ROW.ON_HAND,
                                N_ROW.PARTNO);
  END
```

Figure 37. Example of a trigger

The parts of this trigger are:

1. Trigger name (REORDER)
2. Trigger activation time (AFTER)
3. Triggering event (UPDATE)
4. Triggering table name (PARTS)
5. New transition variable correlation name (N_ROW)
6. Granularity (FOR EACH ROW)
7. Trigger condition (WHEN...)
8. Trigger body (BEGIN ATOMIC...END;)

When you execute this CREATE TRIGGER statement, DB2 creates a trigger package called REORDER and associates the trigger package with table PARTS. DB2 records the timestamp when it creates the trigger. If you define other triggers on the PARTS table, DB2 uses this timestamp to determine which trigger to activate first. The trigger is now ready to use.

After DB2 updates columns ON_HAND or MAX_STOCKED in each row of table PARTS, trigger REORDER is activated. The trigger calls a stored procedure called ISSUE_SHIP_REQUEST if, after a row is updated, the quantity of parts on hand is less than 10% of the maximum quantity stocked. In the trigger condition, N_ROW represents the value of a modified row after the triggering event.

When you no longer want to use trigger REORDER, you can delete the trigger by executing the statement:

```
DROP TRIGGER REORDER RESTRICT;
```

Executing this statement drops trigger REORDER and its associated trigger package named REORDER.

If you drop table PARTS, DB2 also drops trigger REORDER and its trigger package.

Parts of a trigger

This section gives you the information you need to code each of the trigger parts:

- Trigger name
- Triggering table
- Trigger activation time
- Triggering event
- Granularity
- Transition variables
- Transition tables
- Triggered action, which consists of a *trigger condition* and *trigger body*

Trigger name: Use a short, ordinary identifier to name your trigger. You can use a qualifier or let DB2 determine the qualifier. When DB2 creates a trigger package for the trigger, it uses the qualifier for the collection ID of the trigger package. DB2 uses these rules to determine the qualifier:

- If you use static SQL to execute the CREATE TRIGGER statement, DB2 uses the authorization ID in the bind option QUALIFIER for the plan or package that contains the CREATE TRIGGER statement. If the bind command does not include the QUALIFIER option, DB2 uses the owner of the package or plan.
- If you use dynamic SQL to execute the CREATE TRIGGER statement, DB2 uses the authorization ID in special register CURRENT SQLID.

Triggering table: When you perform an insert, update, or delete operation on this table, the trigger is activated. You must name a local table in the CREATE TRIGGER statement. You cannot define a trigger on a catalog table or on a view.

Trigger activation time: The two choices for trigger activation time are NO CASCADE BEFORE and AFTER. NO CASCADE BEFORE means that the trigger is activated before DB2 makes any changes to the triggering table, and that the triggered action does not activate any other triggers. AFTER means that the trigger is activated after DB2 makes changes to the triggering table and can activate other triggers. Triggers with an activation time of NO CASCADE BEFORE are known as before triggers. Triggers with an activation time of AFTER are known as after triggers.

Triggering event: Every trigger is associated with an event. A trigger is activated when the triggering event occurs in the triggering table. The triggering event is one of the following SQL operations:

- INSERT
- UPDATE
- DELETE

A triggering event can also be an update or delete operation that occurs as the result of a referential constraint with ON DELETE SET NULL or ON DELETE CASCADE.

Triggers are not activated as the result of updates made to tables by DB2 utilities.

When the triggering event for a trigger is an update operation, the trigger is called an update trigger. Similarly, triggers for insert operations are called insert triggers, and triggers for delete operations are called delete triggers.

The SQL statement that performs the triggering SQL operation is called the triggering SQL statement.

The following example shows a trigger that is defined with an INSERT triggering event:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END
```

Each triggering event is associated with one triggering table and one SQL operation. If the triggering SQL operation is an update operation, the event can be associated with specific columns of the triggering table. In this case, the trigger is activated only if the update operation updates any of the specified columns. For example, the following trigger, PAYROLL1, is activated only if an update operation is performed on columns SALARY or BONUS of table PAYROLL:

```
CREATE TRIGGER PAYROLL1
  AFTER UPDATE OF SALARY, BONUS ON PAYROLL
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(PAYROLL_LOG(USER, 'UPDATE', CURRENT TIME, CURRENT DATE));
  END
```

Granularity: The triggering SQL statement might modify multiple rows in the table. The granularity of the trigger determines whether the trigger is activated only once for the triggering SQL statement or once for every row that the SQL statement modifies. The granularity values are:

- FOR EACH ROW

  The trigger is activated once for each row that DB2 modifies in the triggering table. If the triggering SQL statement modifies no rows, the trigger is not activated. However, if the triggering SQL statement updates a value in a row to the same value, the trigger is activated. For example, if an UPDATE trigger is defined on table COMPANY_STATS, the following SQL statement will activate the trigger.

  ```
  UPDATE COMPANY_STATS SET NBEMP = NBEMP;
  ```

- FOR EACH STATEMENT

  The trigger is activated once when the triggering SQL statement executes. The trigger is activated even if the triggering SQL statement modifies no rows.

Triggers with a granularity of FOR EACH ROW are known as row triggers. Triggers with a granularity of FOR EACH STATEMENT are known as statement triggers. Statement triggers can only be after triggers.

The following statement is an example of a row trigger:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END
```

Trigger NEW_HIRE is activated once for every row inserted into the employee table.

Transition variables: When you code a row trigger, you might need to refer to the values of columns in each updated row of the triggering table. To do this, specify transition variables in the REFERENCING clause of your CREATE TRIGGER statement. The two types of transition variables are:

- Old transition variables, specified with the OLD *transition-variable* clause, capture the values of columns before the triggering SQL statement updates them. You can define old transition variables for update and delete triggers.
- New transition variables, specified with the NEW *transition-variable* clause, capture the values of columns after the triggering SQL statement updates them. You can define new transition variables for update and insert triggers.

The following example uses a new transition variable to capture an employee's salary after it is updated:

```
CREATE TRIGGER BIGPAY AFTER UPDATE OF SALARY ON EMP
  REFERENCING NEW AS MODIFIED
  FOR EACH ROW MODE DB2SQL WHEN (MODIFIED.SALARY > 99999)
  BEGIN ATOMIC
    CALL BIGPAY_LIST(MODIFIED.EMPNO, MODIFIED.FIRSTNME,
                     MODIFIED.MIDINIT, MODIFIED.LASTNAME,
                     MODIFIED.SALARY);
  END
```

Transition tables: If you want to refer to the entire set of rows that a triggering SQL statement modifies, rather than to individual rows, use a transition table. Like transition variables, transition tables can appear in the REFERENCING clause of a CREATE TRIGGER statement. Transition tables are valid for both row triggers and statement triggers. The two types of transition tables are:

- Old transition tables, specified with the OLD TABLE *transition-table* clause, capture the values of columns before the triggering SQL statement updates them. You can define old transition tables for update and delete triggers.
- New transition tables, specified with the NEW TABLE *transition-table* clause, capture the values of columns after the triggering SQL statement updates them. You can define new transition variables for update and insert triggers.

The scope of old and new transition table names is the trigger body. If another table exists that has the same name as a transition table, any unqualified reference to that name in the trigger body points to the transition table. To reference the other table in the trigger body, you must use the fully qualified table name.

The following example uses a new transition table to capture the set of rows that are inserted into the INVOICE table:

```
CREATE TRIGGER LRG_ORDR
  AFTER INSERT ON INVOICE
  REFERENCING NEW TABLE AS N_TABLE
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    SELECT LARGE_ORDER_ALERT(CUST_NO, TOTAL_PRICE, DELIVERY_DATE)
      FROM N_TABLE WHERE TOTAL_PRICE > 10000;
  END
```

The SELECT statement in LRG_ORDER causes user-defined function LARGE_ORDER_ALERT to execute for each row in transition table N_TABLE that satisfies the WHERE clause (TOTAL_PRICE > 10000).

Triggered action: When a trigger is activated, a triggered action occurs. Every trigger has one triggered action, which consists of two parts: the *trigger condition* and the *trigger body.*

Trigger condition: If you want the triggered action to occur only when certain conditions are true, code a trigger condition. A trigger condition is similar to a predicate in a SELECT, except that the trigger condition begins with WHEN, rather than WHERE. If you do not include a trigger condition in your triggered action, the trigger body executes every time the trigger is activated.

For a row trigger, DB2 evaluates the trigger condition once for each modified row of the triggering table. For a statement trigger, DB2 evaluates the trigger condition once for each execution of the triggering SQL statement.

The following example shows a trigger condition that causes the trigger body to execute only when the number of ordered items is greater than the number of available items:

```
CREATE TRIGGER CK_AVAIL
  NO CASCADE BEFORE INSERT ON ORDERS
  REFERENCING NEW AS NEW_ORDER
  FOR EACH ROW MODE DB2SQL
  WHEN (NEW_ORDER.QUANTITY >
    (SELECT ON_HAND FROM PARTS
     WHERE NEW_ORDER.PARTNO=PARTS.PARTNO))
     BEGIN ATOMIC
       VALUES(ORDER_ERROR(NEW_ORDER.PARTNO, NEW_ORDER.QUANTITY));
     END
```

Trigger body: In the trigger body, you code the SQL statements that you want to execute whenever the trigger condition is true. The trigger body begins with BEGIN ATOMIC and ends with END. You cannot include host variables or parameter markers in your trigger body.

The statements you can use in a trigger body depend on the activation time of the trigger. Table 23 summarizes which SQL statements you can use in which types of triggers.

Table 23 (Page 1 of 2). Valid SQL statements for triggers and trigger activation times

	Valid for activation time	
SQL statement	**Before**	**After**
SELECT	Yes	Yes

Table 23 (Page 2 of 2). Valid SQL statements for triggers and trigger activation times

	Valid for activation time	
SQL statement	**Before**	**After**
VALUES	Yes	Yes
CALL	Yes	Yes
SIGNAL SQLSTATE	Yes	Yes
SET *transition-variable*	Yes	No
INSERT	No	Yes
UPDATE	No	Yes
DELETE	No	Yes

The following list provides more information about the SQL statements that are valid for triggers:

- *SELECT, VALUES, and CALL:* Use the SELECT or VALUES statement in a trigger body to conditionally or unconditionally invoke a user-defined function. Use the CALL statement to invoke a stored procedure. See “Invoking stored procedures and user-defined functions from triggers” on page 218 for more information on invoking user-defined functions and stored procedures from triggers.
- *SET transition-variable:* Because before triggers operate on rows of a table before those rows are modified, you cannot perform operations in the body of a before trigger that directly modify the triggering table. You can, however, use the SET *transition-variable* statement to modify the values in a row before those values go into the table. For example, this trigger uses a new transition variable to fill in today's date for the new employee's hire date:

  ```
  CREATE TRIGGER HIREDATE
    NO CASCADE BEFORE INSERT ON EMP
    REFERENCING NEW AS NEW_VAR
    FOR EACH ROW MODE DB2SQL
    BEGIN ATOMIC
      SET NEW_VAR.HIRE_DATE = CURRENT_DATE;
    END
  ```

- *SIGNAL SQLSTATE:* Use the SIGNAL SQLSTATE statement in the trigger body to report an error condition and back out any changes that are made by the trigger, as well as actions that result from referential constraints on the triggering table. When DB2 executes the SIGNAL SQLSTATE statement, it returns an SQLCA to the application with SQLCODE -438. The SQLCA also includes the following values, which you supply in the SIGNAL SQLSTATE statement:
  - A five-character value that DB2 uses as the SQLSTATE
  - An error message that DB2 places in the SQLERRMC field

  In the following example, the SIGNAL SQLSTATE statement causes DB2 to return an SQLCA with SQLSTATE 75001 and terminate the salary update operation if an employee's salary increase is over 20%:

```
CREATE TRIGGER SAL_ADJ
  BEFORE UPDATE OF SALARY ON EMP
  REFERENCING OLD AS OLD_EMP
  NEW AS NEW_EMP
  FOR EACH ROW MODE DB2SQL
  WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY * 1.20))
  BEGIN ATOMIC
    SIGNAL SQLSTATE '75001'
      ('Invalid Salary Increase - Exceeds 20%');
  END
```

- *INSERT, UPDATE, and DELETE:* Because you can include INSERT, UPDATE, and DELETE statements in your trigger body, execution of the trigger body might cause activation of other triggers. See “Trigger cascading” on page 219 for more information.
- *Failures during trigger execution:* If any SQL statement in the trigger body fails during trigger execution, DB2 rolls back all changes that are made by the triggering SQL statement and the triggered SQL statements. However, if the trigger body executes actions that are outside of DB2's control or are not under the same commit coordination as the DB2 subsystem in which the trigger executes, DB2 cannot undo those actions. Examples of external actions that are not under DB2's control are:
  - Performing updates that are not under RRS commit control
  - Sending an electronic mail message

  If the trigger executes external actions that are under the same commit coordination as the DB2 subsystem under which the trigger executes, and an error occurs during trigger execution, DB2 places the application process that issued the triggering statement in a must-rollback state. The application must then execute a rollback operation to roll back those external actions. Examples of external actions that are under the same commit coordination as the triggering SQL operation are:
  - Executing a distributed update operation
  - From a user-defined function or stored procedure, executing an external action that affects an external resource manager that is under RRS commit control

Invoking stored procedures and user-defined functions from triggers

A trigger body can include only SQL statements. Therefore, if you want the trigger to perform actions or use logic that is not available in SQL, you need to write a user-defined function or stored procedure and invoke that function or stored procedure from the trigger body.

Because a before trigger must not modify any table, functions and procedures that you invoke from a trigger cannot include INSERT, UPDATE, or DELETE statements that modify the triggering table.

To invoke a user-defined function from a trigger, code a SELECT statement or VALUES statement. Use a SELECT statement to execute the function conditionally. The number of times the user-defined function executes depends on the number of rows in the result set of the SELECT statement. For example, in this trigger, the SELECT statement causes user-defined function LARGE_ORDER_ALERT to execute for each row in transition table N_TABLE with an order of more than 10000:

```
CREATE TRIGGER LRG_ORDR
  AFTER INSERT ON INVOICE
  REFERENCING NEW TABLE AS N_TABLE
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    SELECT LARGE_ORDER_ALERT(CUST_NO, TOTAL_PRICE, DELIVERY_DATE)
      FROM N_TABLE WHERE TOTAL_PRICE > 10000;
  END
```

Use the VALUES statement to execute a function unconditionally; that is, once for each execution of a statement trigger or once for each row in a row trigger. In this example, user-defined function PAYROLL_LOG executes every time an update operation occurs that activates trigger PAYROLL1:

```
CREATE TRIGGER PAYROLL1
  AFTER UPDATE ON PAYROLL
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(PAYROLL_LOG(USER, 'UPDATE',
      CURRENT TIME, CURRENT DATE));
  END
```

To invoke a stored procedure from a trigger, use the CALL statement. The parameters of this stored procedure call must be literals, transition variables, table locators, or expressions.

Passing transition tables to user-defined functions and stored procedures

When you call a user-defined function or stored procedure from a trigger, you might want to give the function or procedure access to the entire set of modified rows. That is, you want to pass a pointer to the old or new transition table. You do this using table locators.

Most of the coding work for using a table locator occurs in the function or stored procedure that receives the locator. “Accessing transition tables in a user-defined function” on page 182 explains how a function defines a table locator and uses it to receive a transition table. To pass the transition table from a trigger, specify the parameter TABLE *transition-table* when you invoke the function or stored procedure. For example, this trigger passes a table locator for a new transition table to stored procedure CHECKEMP:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    CALL (CHECKEMP(TABLE NEWEMPS));
  END
```

Trigger cascading

An SQL operation that a trigger performs might modify the triggering table or other tables with triggers, so DB2 also activates those triggers. A trigger that is activated as the result of another trigger can be activated at the same level as the original trigger or at a *different* level. Two triggers, A and B, are activated at different levels if trigger B is activated after trigger A is activated and completes before trigger A completes. If trigger B is activated after trigger A is activated and completes after

trigger A completes, then the triggers are at the same level. For example, in these cases, trigger A and trigger B are activated at the same level:

- Table X has two triggers defined on it, A and B. A is a before trigger and B is an after trigger. An update to table X causes both trigger A and trigger B to activate.
- Trigger A updates table X, which has a referential constraint with table Y, which has trigger B defined on it. The referential constraint causes table Y to be updated, which activates trigger B.

In these cases, trigger A and trigger B are activated at different levels:

- Trigger A is defined on table X, and trigger B is defined on table Y. Trigger B is an update trigger. An update to table X activates trigger A, which contains an UPDATE statement on table B in its trigger body. This UPDATE statement activates trigger B.
- Trigger A calls a stored procedure. The stored procedure contains an INSERT statement for table X, which has insert trigger B defined on it. When the INSERT statement on table X executes, trigger B is activated.

Trigger cascading occurs when triggers are activated at different levels. Trigger cascading can occur only for after triggers, because DB2 does not support cascading of before triggers.

To prevent the possibility of endless trigger cascading, DB2 supports only 16 levels of cascading of triggers, stored procedures, and user-defined functions. If a trigger, user-defined function, or stored procedure at the 17th level is activated, DB2 returns SQLCODE -724 and backs out all SQL changes in the 16 levels of cascading. However, as with any other SQL error that occurs during trigger execution, if any action occurs that is outside the control of DB2, that action is not backed out.

You can write a monitor program that issues IFI READS requests to collect DB2 trace information about the levels of cascading of triggers, user-defined functions, and stored procedures in your programs. See Appendixes (Volume 2) of *DB2 Administration Guide* for information on how to write a monitor program.

Ordering of multiple triggers

You can create multiple triggers for the same triggering table, event, and activation time. The order in which those triggers are activated is the order in which the triggers were created. DB2 records the timestamp when each CREATE TRIGGER statement executes. When an event occurs in a table that activates more than one trigger, DB2 uses the stored timestamps to determine which trigger to activate first.

DB2 always activates all before triggers defined on a table before the after triggers that are defined on that table, but within the set of before triggers, the activation order is by timestamp, and within the set of after triggers, the activation order is by timestamp.

In this example, triggers NEWHIRE1 and NEWHIRE2 have the same triggering event (INSERT), the same triggering table (EMP), and the same activation time (AFTER). Suppose that the CREATE TRIGGER statement for NEWHIRE1 is run before the CREATE TRIGGER statement for NEWHIRE2:

```
CREATE TRIGGER NEWHIRE1
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEWHIRE2
  AFTER INSERT ON EMP
  REFERENCING NEW AS N_EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE DEPTS SET NBEMP = NBEMP + 1
      WHERE DEPT_ID = N_EMP.DEPT_ID;
  END
```

When an insert operation occurs on table EMP, DB2 activates NEWHIRE1 first because NEWHIRE1 was created first. Now suppose that someone drops and recreates NEWHIRE1. NEWHIRE1 now has a later timestamp than NEWHIRE2, so the next time an insert operation occurs on EMP, NEWHIRE2 is activated before NEWHIRE1.

If two row triggers are defined for the same action, the trigger that was created earlier is activated first for all affected rows. Then the second trigger is activated for all affected rows. In the previous example, suppose that an INSERT statement with a subselect inserts 10 rows into table EMP. NEWHIRE1 is activated for all 10 rows, then NEWHIRE2 is activated for all 10 rows.

Interactions among triggers and referential constraints

When you create triggers, you need to understand the interactions among the triggers and constraints on your tables and the effect that the order of processing of those constraints and triggers can have on the results.

In general, the following steps occur when triggering SQL statement S1 performs an insert, update, or delete operation on table T1:

1. DB2 determines the rows of T1 to modify. Call that set of rows M1. The contents of M1 depend on the SQL operation:
   - For a delete operation, all rows that satisfy the search condition of the statement for a searched delete operation, or the current row for a positioned delete operation
   - For an insert operation, the row identified by the VALUES statement , or the rows identified by a SELECT clause
   - For an update operation, all rows that satisfy the search condition of the statement for a searched update operation, or the current row for a positioned update operation
2. DB2 processes all before triggers that are defined on T1, in order of creation.

   Each before trigger executes the triggered action once for each row in M1. If M1 is empty, the triggered action does not execute.

   If an error occurs when the triggered action executes, DB2 rolls back all changes made by S1.
3. DB2 makes the changes that are specified in statement S1 to table T1.

If an error occurs, DB2 rolls back all changes made by S1.

4. DB2 applies all the following constraints and checks that are defined on table T1 if M1 is not empty:
   - Referential constraints
   - Check constraints
   - Checks that are due to updates of the table through views that are defined WITH CHECK OPTION

   Application of referential constraints with rules of DELETE CASCADE or DELETE SET NULL are activated before delete triggers or before update triggers on the dependent tables.

   If any constraint is violated, DB2 rolls back all changes that are made by constraint actions or by statement S1.

5. DB2 processes all after triggers that are defined on T1, and all after triggers on tables modified as the result of referential constraint actions, in order of creation.

   Each after row trigger executes the triggered action once for each row in M1. If M1 is empty, the triggered action does not execute.

   Each after statement trigger executes the triggered action once for each execution of S1, even if M1 is empty.

If any triggered actions contain SQL insert, update, or delete operations, repeat steps 1 through 5 for each of those operations.

If an error occurs when the triggered action executes, or if a triggered action is at the 17th level of trigger cascading, DB2 rolls back all changes that are made in step 5 and all previous steps.

For example, table DEPT is a parent table of EMP, with these conditions:

- The DEPTNO column of DEPT is the primary key.
- The WORKDEPT column of EMP is the foreign key.
- The constraint is ON DELETE SET NULL.

Suppose the following trigger is defined on EMP:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(CHECKEMP(TABLE NEWEMPS));
  END
```

Also suppose that an SQL statement deletes the row with department number E21 from DEPT. Because of the constraint, DB2 finds the rows in EMP with a WORKDEPT value of E21 and sets WORKDEPT in those rows to null. This is equivalent to an update operation on EMP, which has update trigger EMPRAISE. Therefore, because EMPRAISE is an after trigger, EMPRAISE is activated after the constraint action sets WORKDEPT values to null.

Creating triggers to obtain consistent results

When you create triggers and write SQL statements that activate those triggers, you need to ensure that executing those statements on the same set of data always produces the same results. Two common reasons that you can get inconsistent results are:

- Positioned UPDATE or DELETE statements that use uncorrelated subqueries cause triggers to operate on a larger result set than you intended.
- DB2 does not always process rows in the same order, so triggers that propagate rows of a table can generate different result tables at different times.

The following examples demonstrate these situations.

Example: Effect of an uncorrelated subquery on a triggered action: Suppose that tables T1 and T2 look like this:

```
Table T1     Table T2
   A1           B1
   ==           ==
    1            1
    2            2
```

The following trigger is defined on T1:

```
CREATE TRIGGER TR1
  AFTER UPDATE OF T1
  FOR EACH ROW
  MODE DB2SQL
    BEGIN ATOMIC
      DELETE FROM T2 WHERE B1 = 2;
    END
```

Now suppose that an application executes the following statements to perform a positioned update operation:

```
EXEC SQL BEGIN DECLARE SECTION;
 long hv1;
EXEC SQL END DECLARE SECTION;
⋮
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT A1 FROM T1
  WHERE A1 IN (SELECT B1 FROM T2)
  FOR UPDATE OF A1;
⋮
EXEC SQL OPEN C1;
⋮
while(SQLCODE>=0 && SQLCODE!=100)
{
  EXEC SQL FETCH C1 INTO :hv1;
  UPDATE T1 SET A1=5 WHERE CURRENT OF C1;
}
```

When DB2 executes the FETCH statement that positions cursor C1 for the first time, DB2 evaluates the subselect, SELECT B1 FROM T2, to produce a result table that contains the two rows of column T2:

```
1
2
```

When DB2 executes the positioned UPDATE statement for the first time, trigger TR1 is activated. When the body of trigger TR1 executes, the row with value 2 is deleted from T2. However, because SELECT B1 FROM T2 is evaluated only once, when the FETCH statement is executed again, DB2 finds the second row of T1, even though the second row of T2 was deleted. The FETCH statement positions the cursor to the second row of T1, and the second row of T1 is updated. The update operation causes the trigger to be activated again, which causes DB2 to attempt to delete the second row of T2, even though that row was already deleted.

To avoid processing of the second row after it should have been deleted, use a correlated subquery in the cursor declaration:

```
DCL C1 CURSOR FOR
  SELECT A1 FROM T1 X
  WHERE EXISTS (SELECT B1 FROM T2 WHERE X.A1 = B1)
  FOR UPDATE OF A1;
```

In this case, the subquery, SELECT B1 FROM T2 WHERE X.A1 = B1, is evaluated for each FETCH statement. The first time that the FETCH statement executes, it positions the cursor to the first row of T1. The positioned UPDATE operation activates the trigger, which deletes the second row of T2. Therefore, when the FETCH statement executes again, no row is selected, so no update operation or triggered action occurs.

Example: Effect of row processing order on a triggered action: The following example shows how the order of processing rows can change the outcome of an after row trigger.

Suppose that tables T1, T2, and T3 look like this:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    1         (empty)      (empty)
    2
```

The following trigger is defined on T1:

```
CREATE TRIGGER TR1
  AFTER UPDATE ON T1
  REFERENCING NEW AS N
  FOR EACH ROW
  MODE DB2SQL
    BEGIN ATOMIC
      INSERT INTO T2 VALUES(N.C1);
      INSERT INTO T3 (SELECT B1 FROM T2);
    END
```

Now suppose that a program executes the following UPDATE statement:

```
UPDATE T1 SET A1 = A1 + 1;
```

The contents of tables T2 and T3 after the UPDATE statement executes depend on the order in which DB2 updates the rows of T1. If DB2 updates the first row of T1 first, after the UPDATE statement and the trigger execute for the first time, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    2            2            2
    2
```

After the second row of T1 is updated, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    2            2            2
    3            3            2
                              3
```

However, if DB2 updates the second row of T1 first, after the UPDATE statement and the trigger execute for the first time, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    1            3            3
    3
```

After the first row of T1 is updated, the values in the three tables are:

```
Table T1     Table T2     Table T3
   A1           B1           C1
   ==           ==           ==
    2            3            3
    3            2            3
                              2
```

DB2 Extenders™

You can easily leverage DB2' support for large multimedia objects by using the DB2 Extenders feature of DB2 for OS/390. Use DB2 Extenders to store and manipulate image, audio, video, and text objects.

The DB2 Extenders comprise a separate Image Extender, Audio Extender, Video Extender, and Text Extender. Each extender defines a distinct type, and a set of user-defined functions for use with objects of its distinct type. The extenders automatically capture and maintain a variety of attribute information about each object that you store. They also provide a rich body of APIs that can take your object-oriented applications to new of sophistication.

Chapter 7. Features of DB2 UDB Server for OS/390

The DB2 UDB Server for OS/390 offers a number of optional features that come with DB2 for OS/390 and make it easy for you to take advantage of the full potential of your DB2 subsystem.

The following no-charge features are shipped to you automatically when you order DB2 Universal Database for OS/390.

- DB2 Management Tools Package, which includes the following elements:
  - DB2 UDB Control Center
  - DB2 Stored Procedures Builder
  - DB2 Installer
  - DB2 Visual Explain
  - DB2 Estimator
- Net.Data for OS/390

Priced features are offered in a “Try and Buy” program that you can install and use for up to 90 days without paying license charges. To obtain unrestricted access for a priced feature, simply order the “Buy” component and install it onto your DB2 UDB server. The following features are provided in the “Try and Buy” program:

- Query Management Facility
- DB2 DataPropagator
- DB2 Performance Monitor
- DB2 Buffer Pool Tool
- DB2 Administration Tool

This section provides an overview of each feature. You can obtain additional information at the DB2 Universal Database for OS/390 Web site: http://www.software.ibm.com/data/db2/os390, or see *DB2 What's New?*.

DB2 is Tivoli ready and will work with Tivoli Global Enterprise Manager (GEM) which is available from IBM.

Control Center for DB2 UDB

Users of DB2 for OS/390 can now manage data in a new way.

The Control Center capability of IBM's DB2 Universal Database Version 6 for Windows, UNIX, and OS/2 now extends support to DB2 for OS/390 Version 6. As a Java-based tool, the Control Center can run on many types of workstations and on many different operating systems. Users can now use the same tool, with its easy-to-use graphical user interface (GUI), to manage DB2 databases on OS/390, as well as on workstation servers. The GUI supports DB2 for OS/390 SQL statements (such as CREATE and ALTER), DB2 commands (such as DISPLAY, START, and STOP), and utilities (such as REORG and LOAD).

The Control Center can run either as a Java application or as an application on your Web server that your Web browser can access. Because the Control Center requires DB2 Connect, the DB2 Management Tools Package provides a restricted-use copy of DB2 Connect Version 6 to satisfy this functional dependency. Control Center is packaged along with DB2 Connect on the same CD.

DB2 Stored Procedures Builder

The IBM DB2 Stored Procedure Builder (SPB), an element of the DB2 Management Tools Package, provides an easy-to-use development environment for creating, installing, and testing stored procedures. With the DB2 Stored Procedure Builder, you can focus on creating your stored procedure logic rather than on the details of registering, building, and installing stored procedures on a DB2 server. You can develop stored procedures on one operating system and deploy them on other server operating systems. The Stored Procedure Builder provides a single development environment that supports the entire DB2 family ranging from the workstation to System/390®.

Because the Stored Procedures Builder requires DB2 Connect, the DB2 Management Tools Package provides a restricted-use copy of DB2 Connect Version 6 to satisfy this functional dependency. The DB2 Stored Procedures Builder is packaged along with DB2 Connect on the same CD.

DB2 Installer

DB2 Installer enhances your productivity whether you are installing DB2 for the first time or are an experienced installer. From your workstation, you can install, migrate, or update DB2 for OS/390 from a graphical interface, customize your DB2 subsystem, and run your install-related jobs. Windows NT support and the migration and update options were added to Version 5 of DB2 Installer. In Version 6, several optional features that come with DB2 for OS/390 can be installed using DB2 Installer, including DB2 PM.

DB2 Visual Explain

DB2 Visual Explain graphically presents DB2 EXPLAIN output. You can use DB2 Visual Explain on Windows NT or OS/2. Many enhancements for Version 6 are from specific customer requests. These enhancements include:

- View statement cost in milliseconds and service units
- Filter explainable statements from multiple plans and packages
- Specify your own qualifier for the catalog and explain tables
- Generate customized reports quickly

DB2 Visual Explain works with a Version 5 or Version 6 plan table.

If you are using Control Center, you can launch Visual Explain directly from the Control Center.

Because DB2 Visual Explain requires DB2 Connect, the DB2 Management Tools Package provides a restricted-use copy of DB2 Connect Version 6 to satisfy this functional dependency.

DB2 Estimator

DB2 Estimator is an easy-to-use, stand-alone tool for estimating the performance of applications for Version 5 and Version 6 of DB2 for OS/390. From a simple table sizing to a detailed performance analysis of an entire DB2 application, DB2 Estimator saves time and lowers costs by letting you investigate the impact to your production system of new or modified applications before you implement them.

You can use DB2 Estimator for both Version 5 and Version 6 of DB2 for OS/390. It runs on any 16-bit or 32-bit Windows operating system. Use DB2 Estimator on any data that has been imported from DB2 for OS/390, or you can model an application for which none of the tables, SQL, transactions, or configuration exist.

In Version 6, DB2 Estimator expands support for utilities and SQL statements to help you approximate your working environment more closely.

DB2 Estimator is available for download at the DB2 Universal Database for OS/390 Web site: http://www.software.ibm.com/data/db2/os390/downloads.html

Net.Data for OS/390

Net.Data for OS/390 lets you access your business data from the World Wide Web. With Net.Data for OS/390, you can build high-performance Web applications that execute your business logic to create dynamic Web pages based on data stored within DB2 and other data sources within your enterprise. Net.Data provides continuous application availability in a scalable, secure, high performance environment. And Net.Data provides a powerful macro capability for robust Web application development.

Net.Data Version 2.2, which is a no-charge feature of DB2 for OS/390 Version 6, now uses the IBM and Domino Web server application programming interfaces, Internet Connection Application Interfaces (API), and Domino Go Web server API. Applications that use the Web server APIs perform more efficiently than Common Gateway Interface (CGI) applications.

With Net.Data for OS/390, your high-performance, business-critical Web applications can more efficiently use the data in your enterprise. Enhancements improve Net.Data's performance, scalability, and usability.

Query Management Facility

IBM Query Management Facility (QMF), is the tightly integrated, powerful, reliable query and reporting tool for IBM's DB2 family. New capabilities help you access and present mission-critical data better than ever before. QMF enables you to work with data all over the enterprise. Data can be anywhere from DB2 for OS/390 and DB2 for VSE and VM, to workstation servers that run OS/2, Windows NT™, AIX and other UNIX operating systems, and to large parallel processors. When coupled with IBM's DB2 DataJoiner® product, QMF allows access to non-relational and other vendor data sources.

The QMF family of integrated tools—QMF, QMF High Performance Option (HPO), and QMF for Windows—offers a total solution for you to access large amounts of

data, share central repositories of queries and enterprise reports, and implement tightly controlled distributed or client/server solutions.

- QMF HPO includes QMF for Windows, QMF HPO/Manager, and the QMF HPO/Compiler as one comprehensive feature. When you get QMF HPO Version 6, you always get all three components. QMF HPO is a multi-functional performance package that lets you ramp up the availability of on-demand information, while protecting production applications and operational data.
- QMF for Windows provides a Windows-based, point-and-click query tool for customers with DB2 databases of many sizes.

QMF Version 6 includes enhancements for end users, administrators, application developers, and the enterprise. Some highlights are the Euro currency symbol, formal QMF administrative authority, use of the DB2 Version 6 predictive governor with QMF for Windows, and DB2 for OS/390 server load balancing.

DataPropagator Relational

DB2 DataPropagator provides a powerful replication capability for the DB2 family of databases. Data replication has evolved as the key technology for harnessing the full power and potential of distributed database environments. DB2 DataPropagator, the core component of IBM's replication solution, lets you unite your distributed relational databases into a cohesive and integrated database solution. It automatically captures your data changes against a source database and propagates those changes to any specified target database, keeping the two consistent.

DB2 DataPropagator enables you to tailor data for maximum usability. You can:

- Manage replication with a graphical interface
- Minimize impact on production systems and networks
- Integrate mixed database environments

In Version 6, DB2 DataPropagator delivers many new and powerful functions and capabilities such as DB2 Catalog replication for speeding up of ODBC-based queries.

For more information about catalog replication is available at the DB2 Universal Database for OS/390 Web site:
http://www.software.ibm.com/data/db2/os390/downloads.html

DB2 Performance Monitor

DB2 Performance Monitor for OS/390 (DB2 PM) is IBM's strategic tool for analyzing, controlling, and tuning the performance of DB2 and DB2 applications. It includes a real-time online monitor, a wide variety of reports for in-depth analysis, and an Explain feature to analyze and optimize SQL statements. DB2 PM Version 6 supports full performance monitoring and problem analysis for all functions of DB2.

If you are using Control Center, you can launch DB2 PM directly from the Control Center.

The following components and functions are new or enhanced for DB2 PM Version 6:

- Workstation Online Monitor for Windows NT and OS/2
- Online monitoring of data sharing groups in a Parallel Sysplex environment
- Additional improvements that make performance monitoring even more efficient, such as more detailed DDF thread information.

DB2 Buffer Pool Tool

DB2 Buffer Pool Tool helps performance analysts evaluate tuning alternatives and achieve performance gains by tuning DB2 buffer pools. The tool collects performance data and provides information at the buffer pool and individual object levels.

DB2 Administration Tool

The DB2 Administration Tool (DB2 ADMIN) provides an easy way to perform queries of the DB2 catalog and manage DB2 objects. The tool runs under Interactive System Productivity Facility (ISPF) and uses SQL to access DB2 catalog tables. The tool provides comprehensive information for system administrators, database administrators, and application developers. Use DB2 ADMIN to obtain a quick overview of a database, discover problems, copy tables from one DB2 to another, and perform many administration tasks with ease.

Chapter 8. Planning for migration and fallback

This chapter contains considerations for migration and fallback between DB2 for OS/390 Version 5 and DB2 for OS/390 Version 6, and a directory of new and revised installation panels. See *DB2 Installation Guide* for complete, step-by-step instructions for installing, migrating, or falling back. This chapter contains these sections:

- "Migration considerations"
- "Release incompatibilities" on page 237
- "Release coexistence" on page 244
- "Preparing for fallback" on page 247
- "Installation changes" on page 250

Migration considerations

This section includes items to consider before migrating to DB2 for OS/390 Version 6. DB2 for OS/390 Version 5 is the only release from which you can migrate.

Make sure that your Version 5 subsystem is at the proper service level. See *IBM® Database 2™ Program Directory*, which is shipped with the product, for keyword specifications for preventive service planning (PSP). Check Information/Access or the ServiceLink facility of IBMLink™ for PSP information both before you migrate and monthly for access to the most current information about DB2.

In Version 6, the following items are no longer supported. You cannot migrate to Version 6 until these items are removed from your catalog.

- Type 1 indexes
- Data set passwords for all objects except the bootstrap data set (BSDS)
- Shared read-only data

The new job DSNTIJPM identifies these unsupported objects.

Type 2 indexes are required

Type 1 indexes are no longer supported. You must convert all indexes to type 2 before migrating to Version 6. Catalog migration will fail if any type 1 indexes are found.

Data set password protection is removed

You must remove data set passwords before migrating to Version 6. Use OS/390 Security Server or an equivalent security system to protect your data sets. Catalog migration will fail if any data set passwords are found.

Shared read-only data is removed

Data sharing is a more substantial and more usable function than shared read-only data. You can also use distributed data to share information. Catalog migration will fail if any shared read-only data is found.

Remove views on two catalog tables

Before running job DSNTIJTC (CATMAINT), remove all views on catalog tables SYSIBM.SYSCOLDIST and SYSIBM.SYSCOLDISTSTATS.

Private protocol function not enhanced

No enhancements are planned for distributed data using private protocol. Private-protocol support may be removed in a later release of DB2. Private protocol can no longer use type 2 inactive threads. Specify a non-zero value for MAXTYPE1 to use type 1 inactive threads for private protocol. To take advantage of the enhancements to stored procedures, TCP/IP, and new data types, you must use the DRDA protocol. You can use DRDA without changing your applications by rebinding with the DBPROTOCOL option set to DRDA. See Chapter 2 of *DB2 Command Reference* for more information.

More than 32 K databases are supported

The maximum number of databases is no longer 32K. The database identifier column of catalog table SYSIBM.SYSDATABASE can contain negative numbers, to indicate that there are more than 32K databases defined.

Log buffer size increased

The maximum log output buffer size is 100000 4-KB buffers (400 MB). The default input (read) buffer size is increased to 60 KB. You will probably experience better log read and log write performance with these increases.

Consider enlarging BSDS

The BSDS may need to be larger to accommodate the additional buffer pools. To avoid the BSDS going into secondary extents, change the record size of the primary allocation to 180 records. To increase the space allocation for the BSDS:

1. Rename existing BSDSs.
2. Define larger BSDSs with the original names.
3. Copy the renamed BSDSs into the new BSDSs.

You can do this using access method services. See the Version 6 installation job DSNTIJIN for the definition that uses the larger primary extent size.

Increase maximum number of data sets open

The maximum number of concurrently allocated data sets increases to 32767 for customers running OS/390 Version 2 Release 6. The practical limit for concurrently allocated data sets depends on virtual storage below the 16M line, OS/390 allocation control blocks and some DB2 storage. For more details, refer to Section 5 (Volume 2) of *DB2 Administration Guide*.

Customized DB2I defaults can be migrated

A DB2I TSO IPSF profile member from a prior release can be migrated to the current release. The DSNEMC01 CLIST uses the values specified on installation panel DSNTIPF and stores the results in the ISPF profile member DSNEPROF. Any customized DSNEPROF members can be migrated from Version 5 to Version 6. However, you need to examine any new or changed default panel values to ensure that your customized values are still valid.

DB2 online help reader not used

The DB2 online help reader is no longer needed. BookManager READ/MVS is included with OS/390 base. Before you install online help, you must first install BookManager READ/MVS. See *BookManager READ/MVS V1R3: Installation Planning & Customization* for information on how to do this.

Stored procedures

In earlier releases of DB2, the AUTHID and LUNAME columns of the SYSIBM.SYSPROCEDURES catalog table were used to uniquely identify multiple instances of a procedure. After migrating to Version 6, use the SCHEMA column in the SYSIBM.SYSROUTINES catalog table, the CURRENT PATH special register, and the PATH bind option to identify multiple instances of a procedure. During migration, DB2 generates CREATE PROCEDURE statements which populate SYSROUTINES and SYSPARMS. Rows in SYSIBM.SYSPROCEDURES that contain non-blank values for the AUTHID or LUNAME columns are not used to generate the CREATE PROCEDURE statements . You can identify those rows by using the following statement:

```
SELECT * FROM SYSIBM.SYSPROCEDURES WHERE AUTHID <> ' ' OR LUNAME <> ' ';
```

DB2 also copies information from the rows in SYSIBM.SYSPROCEDURES into the SYSIBM.SYSPARMS table and propagates information from the PARAMETER column of SYSIBM.SYSPROCEDURES into SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS.

Procedures that are migrated from Version 5 have no owner, because they were not created with the CREATE PROCEDURE statement. DB2 authorization treats these migrated procedures differently than procedures that are created with CREATE PROCEDURE. The authorization for migrated procedures is unchanged.

The SYSIBM.SYSPROCEDURES catalog table is not used in Version 6 to define stored procedures to DB2. The SYSCAT.PROCEDURES view of the SYSIBM.SYSPROCEDURES catalog table is not used to list the stored procedures that are registered in DB2. Although these tables are no longer used in Version 6, DB2 does not drop them during migration, because they are needed for fall back and data sharing coexistence.

You can enable the DB2-supplied stored procedures after you have completed the installation process. The two new jobs provided to assist you with enabling some of the DB2-supplied stored procedures are:

- DSNTIJCC defines the DB2-supplied stored procedures for DB2 UDB Control Center.
- DSNTIJSQ defines the SQL procedures support.

Both jobs require tailoring which is listed in the job prolog. For more details about these jobs, see *DB2 Installation Guide*.

For more information on stored procedures, refer to *DB2 Application Programming and SQL Guide*.

ALTER TABLE changes

A new clause called ALTER COLUMN was added to the ALTER TABLE statement. The new clause allows the definition of an existing VARCHAR column to be changed up to the maximum length allowed for the VARCHAR data type. For details on the changes see the ALTER TABLE statement in *DB2 SQL Reference*.

Utility enhancements

The online REORG performance has been enhanced. The performance improvements are explained in Section 5 (Volume 2) of *DB2 Administration Guide*. The enhancements use new calculations for the sort and unload data sets. You may need to increase the size of your unload and sort data sets. The new calculations are explained in *DB2 Utility Guide and Reference* under Defining work data sets.

Work file database size calculations

The migration job DSNTIJTC creates and updates indexes on catalog tables. These indexes are created and updated *sequentially* during migration. The work file database is used for the sort of each index; DB2 needs enough work file storage to sort the largest of the indexes that are listed in Table 24. The migration fails if you do not have enough storage, so ensure that you have enough space before you begin.

Table 24 shows the indexes that are new and changed for existing catalog tables.

Table 24. Indexes added or updated sequentially using the work file database

Catalog table name	Index name	Column names
SYSIBM.SYSINDEXPART	SYSIBM.DSNDRX02	STORNAME
SYSIBM.SYSCOLUMNS	SYSIBM.DSNDCX02	TYPESCHEMA, TYPENAME
SYSIBM.SYSPACKDEP	SYSIBM.DSNKDX03	BQUALIFIER, BNAME, BTYPE, DTYPE
SYSIBM.SYSTABLEPART	SYSIBM.DSNDPX02	STORNAME
SYSIBM.SYSVIEWDEP	SYSIBM.DSNGGX03	BSCHEMA, BNAME, BTYPE
SYSIBM.SYSTABAUTH	SYSIBM.DSNATX02	GRANTEE, TCREATOR, TTNAME, GRANTEETYPE, UPDATECOLS, ALTERAUTH, DELETEAUTH, INDEXAUTH, INSERTAUTH, SELECTAUTH, UPDATEAUTH, CAPTUREAUTH,REFERENCESAUTH, REFCOLS, TRIGGERAUTH
SYSIBM.SYSUSERAUTH	SYSIBM.DSNAUH01	GRANTEE, GRANTEDTS
SYSIBM.SYSINDEXES	SYSIBM.DSNDXX02	DBNAME, INDEXSPACE, COPY, COPYLRSN

Changes to Subsystem parameters

Subsystem parameter OJPERFEH is no longer used.

The subsystem parameter, EDMBFIT is added in macro DSN6SPRM. It is used to adjust the free chain search algorithm on systems with a large EDM pool which is greater than 40M.

The subsystem parameters, SMSCDFL and SMSCDIX in macro DSN6SPRM, specify whether SMS dataclass names are used for table spaces and indexes.

The subsystem parameter, NPGTHRSH in macro DSN6SPRM, controls whether DB2 uses special access path selection for tables under a given size.

The subsystem parameter, IMMEDWRI in macro DSN6GRP, controls at the DB2 member level when DB2 writes updated group buffer pool dependent pages. This option only applies to data sharing environments.

Several important subsystem parameters have been moved to installation panels (see Table 25). As a result, the values you choose for these parameters are used during migration to a new release of DB2.

Table 25. Subsystem parameters that are moved to installation panels

Subsystem parameter	Panel	Field name
BMPTOUT	DSNTIPI	IMS BMP TIMEOUT
DESCSTAT	DSNTIPF	DESCRIBE FOR STATIC
DLDFREQ	DSNTIPN	LEVELID UPDATE FREQ
DLITOUT	DSNTIPI	DL/I BATCH TIMEOUT
HOPAUTH	DSNTIP5	AUTH AT HOP SITE
PCLOSEN	DSNTIPN	RO SWITCH CHKPTS
PCLOSET	DSNTIPN	RO SWITCH TIME
RELCURHL	DSNTIP4	RELEASE LOCKS
RETLWAIT	DSNTIPI	RETAINED LOCK TIMEOUT

Release incompatibilities

Maintenance level requirements: The following enhancements require a higher maintenance level as described in DB2 Program Directory number GI10-8182-02 than the base Version 6 maintenance level as described in DB2 Program Directory number GI10-8182-01:

- External savepoints
- Identity columns
- Declared temporary tables
- Deferred define dataset
- Subselect on the SET clause of an UPDATE
- VARCHAR_FORMAT and TIMESTAMP_FORMAT built-in functions
- IFI consolidation

To determine whether a DB2 subsystem supports these functions, take the following actions:

- For a connection to a remote subsystem, after you execute a CONNECT statement, check for SQLERRP value in the SQLCA. The value must be DSN06011.

- For a connection through RRSAF or CAF, after you execute an IDENTIFY call (for RRSAF) or CONNECT call (for CAF), check the RIBCNUMB and RIBCINFO fields in the RIB. RIBCNUMB must be 1 and RIBCINFO must be DSN06011.

This section describes changes that might affect your DB2 operations after migrating to Version 6 of DB2 for OS/390.

Adjust application programs

You might need to modify your application programs because of the following release incompatibilities.

SQLCODE -101: After migrating to DB2 Version 6, you might receive SQLCODE -101 on long or complicated SQL statements that previously executed successfully. This is possible because SQL statements and DB2 internal structures are buffered in the same local storage, and release changes in the internal structures can result in less storage being available for the SQL statements.

Rewrite the unsuccessful SQL statements by using correlated references, breaking up UNIONs, or using OUTER JOINs.

SQLCODE +802: In Version 6, if an arithmetic overflow occurs when DB2 evaluates the arguments of a VALUE or COALESCE function, DB2 returns SQLCODE +802. In Version 5, DB2 returned SQLCODE 0 under those circumstances.

No colon on a host variable is an error: All host variable references must have the leading colon. If you neglect to use a colon to designate a host variable, the precompiler issues a DSNH104I message or interprets the host variable as an unqualified column name, which might lead to unintended results. The host variable without a colon is interpreted as a column name when the host variable is referenced in a context in which a column name can also be referenced.

Changed format for DSN messages: The DSN message identifiers can now be 8 or 9 characters long. The message identifier formats are DSN*cnnn*I and DSN*cnnnn*I. To accommodate the longer message identifier, the message text will begin in column 13 for all DSN messages. If you have applications that scan the text of DSN messages, you might have to modify them.

Changed format for message DSNU050I: Certain formatting problems in message DSNU050I have been corrected in Version 6. The message formats differently in Version 6 than it did in previous versions. If you have applications that scan the text of message DSNU050I, you might have to modify them.

SQL reserved words: Version 6 has several new SQL reserved words. Refer to *DB2 SQL Reference* for the list, and adjust your applications accordingly.

Using the Euro symbol: New CCSIDs that support the Euro symbol have been added to DB2. See Appendix A in *DB2 Installation Guide* for details on the coded character set identifiers.

Using aliases: Before Version 6, aliases were known locally. Now, if remote aliases are used in SQL, they should be defined at the server site. With SET

statements that assign a host variable with the value of a special register, the host variable is only assigned the local value.

QUIESCE return code: The QUIESCE utility command produces a return code of 4 for duplicate names in the list of table spaces that are to be quiesced. This is a change from a return code of 8. The QUIESCE processing continues, and the table space is quiesced only once.

DSNH message ID lengths: DSNH messages can be 8, 9, or 10 characters long. Existing messages in Version 6 will not be modified to use the longer length.

Positive SQLCODE from PREPARE: A return code greater than zero can be returned from a PREPARE statement if your migrated application is using predictive governing or query optimization hints.

Changed SQLSTATEs: For SQLCODE +2000 change the SQLSTATE to 01638. For SQLCODE +655 change the SQLSTATE to 01597. For SQLCODE +466 change the SQLSTATE to 0100D. For SQLCODE +464 change the SQLSTATE to 0100E. For SQLCODE -470 change the SQLSTATE to 39004. For SQLCODE -751 change the SQLSTATE to 38003. For SQLCODE -30081 change the SQLSTATE to 08001. For a complete listing of SQLSTATEs, see Appendix C of *DB2 Messages and Codes*.

New meaning for SQLCODE: SQLCODE -120 has been expanded to include a WHERE clause, SET clause, VALUES clause, or a SET ASSIGNMENT statement that includes a column function.

New DBPROTOCOL option: Applications may be affected by the new DBPROTOCOL option which is both a subsystem parameter and a bind parameter. If the DBPROTOCOL option is not coded on the BIND, the value listed in the subsystem parameter DBPROTCL will be used. For plans and packages to execute the same as they did in Version 5, they may need to be rebound depending on the DBPROTOCOL value selected. In prior releases, remote access by name resulted in the use of DB2 Private Protocol to communicate with a remote DB2 for OS/390 site. Remote access with the CONNECT statement used the DRDA database protocol. Users can now specify which protocol to use when communicating with the remote server.

Changed default for RELCURHL subsystem parameter: Use the RELCURHL subsystem parameter to indicate whether you want DB2 to release a data page or row lock at COMMIT for the row on which a cursor that is defined WITH HOLD is positioned. In Version 5, the default value was NO. Although this particular row or page lock is not necessary for maintaining cursor position, NO was provided as a default for existing applications that relied on that data lock. In Version 6, RELCURHL is a field on installation panel DSNTIP4. The field name is RELEASE LOCKS, and the default is changed to YES to provide for better concurrency. If you still require the lock, specify NO for RELEASE LOCKS.

Changed default for DYNAMICRULES(BIND) as specified in subsystem parameter DYNRULS: In earlier releases, the default for the DYNRULS subsystem parameter is NO. NO means that the values of the precompiler options are used for dynamic SQL statements in plans or packages bound with DYNAMICRULES(BIND). For Version 6, the default is YES. YES means the application programming (DSNHDECP) defaults will be used for dynamic SQL

| statements in plans or packages bound using DYNAMICRULES bind, define, or
| invoke behavior.

Changed default for PTASKROL subsystem parameter: In Version 5, the default for the PTASKROL subsystem parameter is NO, which means that each parallel
| task produces its own accounting trace record . For Version 6, the default is YES, which means that the originating task cuts an additional accounting trace record with all the roll up values from parallel tasks.

Using new column called CLUSTERRATIOF: The CLUSTERRATIOF column has been added to SYSINDEXES and SYSINDEXSTATS. After migration this column will have a default value until it is updated by RUNSTATS or some other application. If the optimizer sees the default value in CLUSTERRATIOF, it will use the value in CLUSTERRATIO.

| ***Support for large objects:*** The following incompatibilities result from the
| expansion of the SQLDA to support large objects:

- Because of the additional data types supported in Version 6, the host language statements generated by the precompiler for each SQL statement include a larger parameter list. As a result, the size of the object code that results from compiling the output of the precompiler increases. This increase varies according to the number of host variables that are specified in an SQL statement.
- INCLUDE SQLDA generates additional fields for applications written in assembler, PL/I, C, and C++ (see Appendix C of *DB2 SQL Reference* for descriptions of these fields). If an SQL application program written in one of those languages defines a name that matches any symbol in the additional fields, and the program also uses INCLUDE SQLDA, then that program might not operate, assemble, or compile correctly.

| ***A technique for reducing the number of matching columns no longer works:***
| Section 5 (Volume 2) of *DB2 Administration Guide* formerly described a technique for reducing the number of matching columns for an index by changing equal predicates to BETWEEN predicates. Due to enhancements to BETWEEN predicate processing, this technique no longer reduces the number of matching columns. If you have written queries that use this technique, consider using the alternative technique of discouraging the use of a particular index.

New reserved qualifer for tables, SESSION: DB2 uses a new implicit qualifier,
SESSION for a declared temporary table. All tables with a high level qualifier of
SESSION will be treated as a declared temporary table. You must modify any
existing tables with a high level qualifier of SESSION to use another qualifier.

Use the following query to identify existing tables, views, aliases, and synonyms
that will be inaccessible because of the 'SESSION' qualifier:

#

Product-sensitive Programming Interface

```
#                     SELECT 'TABLE', CREATOR, NAME
#                         FROM SYSIBM.SYSTABLES
#                         WHERE CREATOR = 'SESSION' AND TYPE IN ('T', 'G', 'X')
#                     UNION
#                       SELECT 'VIEW', CREATOR, NAME
#                          FROM SYSIBM.SYSTABLES
#                          WHERE CREATOR = 'SESSION' AND TYPE = 'V'
#                     UNION
#                       SELECT 'ALIAS', CREATOR, NAME
#                          FROM SYSIBM.SYSTABLES
#                          WHERE CREATOR = 'SESSION' AND TYPE  = 'A'
#                     UNION
#                       SELECT 'SYNONYM', CREATOR, NAME
#                          FROM SYSIBM.SYSSYNONYMS
#                          WHERE CREATOR = 'SESSION';
```

End of Product-sensitive Programming Interface

DB2 cannot know during the bind of the plan or package whether the static SQL
statement that references a table name qualified by SESSION is for a persistent
base table or for a declared temporary table. These static SQL statements will be
incrementally bound at run-time when the static SQL statement is issued.

Examine all new and changed values for DB2I panels

The DB2I default panels DSNEOP01 and DSNEOP02, will not be initialized with the values specified during the installation CLIST process during a migration. The DB2I panel variables in the ISPF profile from the previous release will be used on the current release. Any customized DSNEPROF members will be migrated from Version 5 to Version 6. However, you must examine any new or changed default panel values to ensure that your customized values are still valid.

Changes to the RLST

The RLST has new optional columns and a new search order for the index. You can use an RLST that has been defined from a release earlier then Version 6 (the RLST index is in ascending order, and no new columns exist), but you cannot do predictive governing, and you cannot take advantage of the performance improvements with the new descending index. You can alter an existing RLST to
| add the new columns and change the definition of the index. Migration job
| DSNTIJSG modifies your existing RLST.

SYSIBM.SYSPROCEDURES no longer used

The SYSIBM.SYSPROCEDURES catalog table is no longer used to define stored procedures to DB2. During catalog migration, all rows in SYSIBM.SYSPROCEDURES are automatically migrated to the new SYSIBM.SYSROUTINES catalog table and to SYSIBM.SYSPARMS.

In Version 6, you use the CREATE and ALTER PROCEDURE statements to define stored procedures. To prepare for fallback, or to run in a coexistence environment for data sharing, you must modify the SYSPROCEDURES catalog table to match the updates to SYSROUTINES and SYSPARMS that those CREATE statements make.

An 'X' plan in the PLAN_TABLE

If you rebind a parallel group that is capable of Sysplex query parallelism to Version 6, and only one DB2 data sharing member is active, a single-CEC parallel plan is developed, and an ***X*** is placed in column PARALLEL_MODE of table PLAN_TABLE instead of the ***C*** that was used in Version 5. If you do not want the ***X***, bind the plan or package on a member that is installed with COORDINATOR=NO. The ***X*** for column PARALLEL_MODE means that all available assisting DB2 members are to be used at run time. To take advantage of this change, you need to rebind all static plans that have a PARALLEL_MODE of X after migrating.

Limit backouts with system restarts

An incompatibility exists if you use the AUTO setting of LIMIT BACKOUT on installation panel DSNTIPN. The purpose of the setting is so that you can indicate that you want DB2 to postpone some of the back-out work that is usually performed during system restart. The default setting, AUTO, is recommended. However, be aware that after restart, some objects (table spaces, index spaces, or partitions) might be unavailable until the automatically started process completes the back-out work. In releases prior to Version 6, all of DB2 was unavailable until the back-out work was completed.

Changes to IFCID fields

The following IFCID fields are no longer set by DB2:

- QWACBSRB
- QWACESRB
- QWACASRB
- QW0148SB
- QW0148SE

See Appendix F, “New and changed IFCIDs” on page 295 for a complete listing of changes to IFCIDs.

DISPLAY BUFFERPOOL changes

The DISPLAY BUFFERPOOL command syntax and output have changed. The new command provides more data sharing information required for understanding the level of inter-system sharing taking place. The DISPLAY BUFFERPOOL output no longer issues messages DSNB450I, DSNB451I, and DSNB452I. Several new messages with expanded information are generated by this command. For details on the changes see the DISPLAY BUFFERPOOL command in *DB2 Command Reference*.

Index changes

Indexes that have been altered, rebuilt, reorganized, or newly created can increase in size by one page per nonpartitioned index or one page per index partition. The leaf distribution column (LEAFDIST) value in SYSINDEXPART may increase for indexes that have been altered, rebuilt, reorganized, or newly created.

ALTER INDEX syntax

In Version 6, ALTER INDEX is more flexible in allowing multiple PART clauses to be specified in a single ALTER INDEX statement. Existing ALTER INDEX statements that specify partition-level options before the PART keyword have different results in Version 6 than in previous releases. In Version 6, when these options are specified before the PART keyword, they apply to all partitions in the index unless you override them with a PART specification.

For example, assume that you have the following statement to change the FREEPAGE value to 10 for partition 1:

```
ALTER INDEX ixname
 FREEPAGE 10
 PART 1;
```

In Version 6, this same statement changes the FREEPAGE value to 10 for all index partitions. If you want to obtain the same results for this statement as you did in previous releases, change the statement as follows:

```
ALTER INDEX ixname
 PART 1
 FREEPAGE 10;
```

See *DB2 SQL Reference* for the correct syntax of ALTER INDEX.

RECOVER INDEX becomes REBUILD INDEX

Before migrating to Version 6, ensure that you have applied APAR PQ09842, which enables you to change your RECOVER INDEX jobs to use the new syntax REBUILD INDEX. In Version 6, RECOVER INDEX means to use a copy of the index and apply log records. Thus, your old RECOVER INDEX jobs are likely to fail. Change any existing RECOVER INDEX utility jobs to use REBUILD INDEX. See APAR PQ09842 for more information.

Work space formulas changed for utilities

Due to the larger table spaces and indexes available, the record header for several utility work data sets is lengthened by several bytes. The changed record headers which are used in several formulas for estimating work data set size affects the following utilities:

- CHECK DATA
- CHECK INDEX
- LOAD
- REBUILD INDEX
- REORG

The new calculations are explained in *DB2 Utility Guide and Reference* under the topic for defining work data sets for each utility.

Support for up to 150000 connections

To support up to 150000 connections, the token for displaying LUWIDs for DISPLAY THREAD is now a 6–digit decimal number.

Change to parameter in IRLMPROC startup procedure

The GROUP parameter in the IRLMPROC startup procedure has been changed to IRLMGRP. Be sure to use the IRLMGRP parameter instead of the GROUP parameter to prevent JCL errors during IRLM startup.

Release coexistence

This section describes changes that might affect your DB2 operations after migrating to Version 6 of DB2 for OS/390.

Coexistence in a distributed data environment

DB2 for OS/390 communicates in a distributed data environment with DB2 Version 2 Release 3 and later, using either DB2 private protocol access or DRDA access. However, the distributed enhancements and the new object data types and that are introduced in this release of DB2 for OS/390 can be used only when using DRDA access.

Other DRDA partners at DRDA level 4 can also take advantage of the functions introduced in Version 6 of DB2 for OS/390.

Coexistence in a data sharing environment

DB2 can support both Version 5 and Version 6 members in a data sharing group.

When developing your migration plan, keep in mind that new functions introduced in this release are not available to members of the group who have not yet migrated.

Coexistence considerations for BIND options

If DSN is under Version 6 and if the DB2 member that is named in the DSN command is Version 5, a BIND or REBIND subcommand that uses any of the following options is rejected:

- OPTHINT (non-blank) for BIND PLAN or PACKAGE
- OPTHINT (any value) for REBIND PLAN or PACKAGE
- DBPROTOCOL(DRDA) for BIND PLAN or PACKAGE
- DBPROTOCOL (any value) for REBIND PLAN or PACKAGE
- The DYNAMICRULES values of DEFINEBIND, DEFINERUN, INVOKEBIND, or INVOKERUN for BIND and REBIND PACKAGE
- PATH for BIND and REBIND PLAN or PACKAGE
- PATHDEFAULT for REBIND PLAN or PACKAGE
- REBIND TRIGGER PACKAGE

To avoid problems, make sure the DB2 subsystem named in the DSN subcommand matches the load libraries that are used for the DSN command.

If you chose a default value of DRDA for DATABASE PROTOCOL on panel
DSNTIP5, and your DB2 Version 6 subsystem is in a data sharing group with DB2
subsystems that are at previous release levels, you need to bind the plans for
DB2-supplied applications such as SPUFI and DCLGEN with the
DBPROTOCOL(PRIVATE) option so that those applications are accessible to
non-Version 6 members of the data sharing group. Edit the BIND commands for
DB2-supplied applications in job DSNTIJSG.

Coexistence considerations for specific capabilities

The purpose of this section is to clarify how some of the new Version 6 functions work (or not) when there are mixed releases in the data sharing group. These considerations are similar to those you have to understand for the fallback environment. Objects that are frozen in fallback are not accessible from a Version 5 subsystem.

New page sizes: Table spaces that use an 8-KB or 16-KB buffer pool are not available to Version 5 members. You cannot issue the ALTER or DISPLAY BUFFERPOOL commands for those buffer pools from a Version 5 system.

New built-in functions: DB2 uses the new CURRENT LOCALE LC_CTYPE special register when evaluating the TRANSLATE, UPPER, and LOWER scalar functions.

New GBPCACHE options: Table spaces or indexes that are defined with GBPCACHE NONE or GBPCACHE SYSTEM are not available to Version 5 members. Table spaces that are defined in a buffer pool defined with GBPCACHE NO are also not available to Version 5 members.

TRACKMOD options: If you try to use COPY from a Version 5 member for a table space defined with TRACKMOD NO, the result will always be a full image copy. If you change from TRACKMOD NO to TRACKMOD YES and then try to copy from a Version 5 member, the first copy will be a full image copy.

New resource limit facility columns for predictive governing: If you populate the new columns in the resource limit facility table, those columns are ignored on a Version 5 member. There is no predictive governing on a Version 5 members.

Populating DSN_STATEMNT_TABLE: You cannot use EXPLAIN to populate the DSN_STATEMNT table from a Version 5 member.

Duplexed group buffer pools: Do not start duplexing until all members of the group are at the level of DB2 and OS/390, and until the group buffer pool is allocated in a coupling facility with coupling facility control code level 5 or higher. No Version 5 members can connect to a duplexed group buffer pool. A Version 6 member that is not running on an OS/390 system at the correct level cannot connect to a duplexed group buffer pool.

If any downlevel members are connected to a group buffer pool, duplexing cannot start for that group buffer pool until all downlevel members are disconnected from the group buffer pool.

Postponed backout processing: A Version 5 member cannot request limited backout processing. However, if you enter a DISPLAY DATABASE command on a

Version 5 subsystem, it is possible for that display to show the advisory restart-pending (AREST) state of page sets that can result when Version 6 members use postponed backout processing.

Similarly, if you issue a DISPLAY GROUP command on a Version 5 subsystem, a member that is both active and has indoubt URs is displayed as ACTIVE (they are displayed as 'A I' on a Version 6 subsystem).

The meaning of the 'Q I' status is slightly different for Version 5 and Version 6 members in the display. For a Version 6 member, it can indicate that postponed abort URs are present.

Optimization hints: Although a Version 5 member can use a Version 6 PLAN_TABLE, it cannot use any hints that you've added to the PLAN_TABLE. If a static statement is bound using optimization hints, that statement can be executed on a Version 5 member as long as that statement has no other release dependencies. The static statement can also run on another Version 6 member that does not have optimization hints enabled.

For dynamic statements, hints are only used if the subsystem that the statement is running on has enabled optimization hints.

Changing partitioning values: You cannot issue the ALTER INDEX statement with the VALUES clause from a Version 5 member. Also, while a table space has any partitions in REORG pending, that entire table space cannot be accessed from a Version 5 member. However, you can run REORG TABLESPACE from the Version 5 member to turn off the REORG pending state and to rebalance the partitions.

Data sets that use extended addressability: The following objects are not available from a Version 5 member:

- Partitioned table spaces with a DSSIZE greater than 4 GB
- Partitioning indexes for those partitioned table spaces
- Nonpartitioning indexes created with a PIECESIZE value greater than 4 GB

Nonpartitioning indexes of greater than 128 pieces: As of Version 6, your nonpartitioning index can grow to 254 pieces (from a previous limit of 128). As long as that nonpartitioning index is not for an EA-enabled table space, you can still access a nonpartitioning index of more than 128 pieces.

Change RECOVER INDEX to REBUILD INDEX: As of Version 6, you must use the syntax REBUILD INDEX to build an index from data. RECOVER INDEX is used to recover an index from an image copy of the index and with log record updates. Install APAR PQ09842 on your Version 5 subsystems and change utility JCL to use the REBUILD INDEX syntax so that you can run the JCL on any subsystem on the group and to avoid confusion over the fact that RECOVER INDEX works differently on Version 5 than it does in Version 6.

Sysplex query parallelism: If a plan or package that uses Sysplex query parallelism is bound on Version 6, then DB2 will not distribute queries to Version 5 subsystems. If the plan or package is bound on Version 5, though, queries can be distributed to both Version 5 and Version 6 subsystems.

Revoking SYSADM or SYSCTRL in Version 5: If SYSADM or SYSCTRL grants privileges on new objects in Version 6, you cannot revoke SYSADM or SYSCTRL from a Version 5 subsystem. This operation is disallowed on Version 5 because the revoke cascades, and a DB2 Version 5 is not aware of the new objects or privileges in Version 6. You must issue the revoke from a Version 6 subsystem.

Preparing for fallback

You can fall back to DB2 for OS/390 Version 5 after migrating to Version 6. This section highlights fallback considerations for Version 6. If you experience a severe application or performance error and want to return to Version 5, follow the detailed step-by-step instructions in *DB2 Installation Guide*.

To avoid complications, do not use the new DB2 for OS/390 Version 6 facilities until you are certain that you will not need to fall back. You can save your Version 6 TSO LOGON procedures and JCL for remigration to Version 6.

To fall back to Version 5:

1. Run Phase 0 of the Version 6 installation verification procedure.
2. Stop DB2 Version 6 activity.
3. Reactivate Version 5.
4. Reconnect TSO, IMS, and CICS to Version 5.
5. Start DB2 Version 5.
6. Verify fallback by running the DB2 sample applications or your own applications.

After fallback to Version 5, plans or packages that were bound in Version 6 will be automatically rebound the first time they run in Version 5.

If you fallback and then try to use frozen plans or packages, the automatic rebind in Version 5 fails. To make the plans and packages that were not automatically rebound on Version 5 available, change the SQL statements or remove the reference to a frozen object, precompile the application programs, and explicitly BIND the plans and packages on Version 5.

Frozen objects

Falling back does not undo changes that were made to the catalog during migration to Version 6. The migrated catalog is used after fallback. Some objects in this catalog that have been affected by Version 6 function might become *frozen* objects after fallback. Frozen objects are unavailable, and they are marked with the release dependency marker I or J . If an object is marked with a release dependency, it is never unmarked. The release dependency marker is listed in the IBMREQD column of catalog tables.

In general, objects that depend on the new facilities of DB2 for OS/390 Version 6 are frozen after you fall back to Version 5 until you remigrate to Version 6. Table 26 on page 248 lists the objects that are frozen when falling back to Version 5. They are marked with the **release dependency marker, I or J.**

Table 26. Objects that are frozen when falling back to DB2 for OS/390 Version 5

RELEASE DEPENDENCY MARKER = I or J
• Plans that use any new syntax or objects(1)
• Packages that use any new syntax or objects(1)
• Tables and views with triggers
• 8 KB and 16 KB table spaces and associated indexes(2)
• Plans or packages that reference user-defined functions
• Plans or packages that use any of the new bind options such as DBPROTOCOL(DRDA)(3).
• Trigger packages
• Tables with LOB columns
• LOB table spaces
• Auxiliary tables
• Indexes on auxiliary tables
• Tables with columns using the ROWID data type
• Tables that are defined with a user-defined type
• Tables defined with identity columns
• Any table space with a DSSIZE greater that 4GB and their partitioning indexes and nonpartitioning indexes with PIECESIZE greater than 4GB.

Notes:

1. If a plan or package simply references a declared temporary table using the qualifier SESSION., but do not use a DELCARE GLOBAL TEMPORARY TABLE statement, the plan or package does not become frozen.
2. The table spaces are marked with a release dependency. The indexes are not marked with a release dependency but are unavailable after fallback due to their association with the table spaces.
3. The release dependency is marked whether the DBPROTOCOL(DRDA) option is specified on bind, rebind, or is the default defined in the subsystem parameter.

Plans and packages become frozen objects when they use new SQL syntax, new BIND options or attributes, or reference frozen objects. When plans and packages become frozen objects, the automatic rebind process is adversely affected.

Product-sensitive Programming Interface

While operating in Version 5, you can determine if any of your objects are frozen by issuing the following SELECT statements:

```
#                   SELECT NAME FROM SYSIBM.SYSPLAN
#                          WHERE IBMREQD = 'I' OR IBMREQD = 'J';
#                   SELECT LOCATION, COLLID, NAME, VERSION FROM SYSIBM.SYSPACKAGE
#                          WHERE IBMREQD = 'I' OR IBMREQD = 'J';
#                   SELECT CREATOR, NAME FROM SYSIBM.SYSVIEWS
#                          WHERE IBMREQD = 'I' OR IBMREQD = 'J';
#                   SELECT CREATOR, NAME FROM SYSIBM.SYSINDEXES
#                          WHERE IBMREQD = 'I' OR IBMREQD = 'J';
#                   SELECT CREATOR, NAME, TYPE FROM SYSIBM.SYSTABLES
#                          WHERE IBMREQD = 'I' OR IBMREQD = 'J';
#                   SELECT DBNAME, NAME FROM SYSIBM.SYSTABLESPACE
#                          WHERE IBMREQD = 'I' OR IBMREQD = 'J';
```

End of Product-sensitive Programming Interface

Other fallback considerations

Before you fall back to Version 5, you must be aware of the following considerations:

- You should remove all restrictive states from a table space before falling back.
- You should alter indexes with the COPY NO option before falling back to Version 5.

After you have fallen back to Version 5 and you are operating on that subsystem, you must be aware of the following operational considerations:

- Any stored procedures created with the Version 6 CREATE PROCEDURE statement will be unavailable when you fall back to Version 5.
- After you fall back to Version 5, you can not use any frozen objects as listed in 248.
- You can use the Version 6 RLST after falling back, but you cannot take advantage of the predictive governing features.
- If any log records are found for LOBs, the applicable pages for the LOB table space are placed in the LPL.
- You can use your Version 6 PLAN_TABLE after falling back, but you cannot take advantage of the optimization hints that you have added to the PLAN_TABLE.
- You must use the Version 5 COPY and RECOVER utility jobs for backup and recovery after fallback.
- Falling back causes postponed abort units of recovery (URs) to be treated as inabort or inflight.
- A CREATE or ALTER DATABASE or TABLESPACE that specifies an 8 KB or 16 KB buffer pool name does not work after falling back.
- If a buffer pool is defined as VPTYPE(DATASPACE) in Version 6, and you fallback to Version 5, this buffer pool becomes a primary type buffer pool and it is allocated with the specified VPSIZE unless the VPSIZE is larger than the allowable 1.6 GB in which case the default VPSIZE is used.
- When you fall back to Version 5, the value in CLUSTERRATIO will be used instead of the value in CLUSTERRATIOF.

- Changing the length of a column of an index in Version 6 may cause an error after falling back to Version 5. A resource unavailable message with RC '00C9009E' occurs if the APAR to enable immediate index access has not been applied.
- When you fallback to Version 5 and attempt to revoke SYSADM or SYSCTRL privileges for a grantor and that revoke cascades down to Version 6 objects or new privilege grants on Version 5 objects, the revoke will fail with message DSNT501I.
- When you fallback to Version 5, any Version 6 plan or package bound with IMMEDWRITE(YES | PH1) will use the default IMMEDWRITE(NO) behavior without having to be rebound.
- In Version 6, you can use the DEFINE NO option on the CREATE TABLESPACE or CREATE INDEX statements to defer creating the VSAM data set until it is used. If the underlying VSAM data sets have not been created when you fall back to Version 5, you will receive an -904 SQLCODE for any SQL statements that refer to the table spaces and index spaces with undefined data sets. Do not use the REPAIR DBD or STOSPACE utilities on the undefined data sets.

For more information on fallback considerations, refer to *DB2 Installation Guide*.

Installation changes

This section shows the panels used by the installation CLIST to customize the jobs you use to install or migrate to Version 6. This section also lists the changes to installation jobs, SMP/E jobs, and sample jobs.

In Version 6, you can also install DB2 for OS/390 from a Windows NT or OS/2 workstation with the DB2 Installer feature.

Version 6 panels

Table 27 shows the panels for DB2 for OS/390 Version 6 installation and migration. With the addition of the new functions in Version 6, several panels have been modified and new fields have been added. The new and modified panels have a 'yes' listed under the **Panel modified** column in Table 27.

Table 27 (Page 1 of 2). Version 6 installation and migration panels

Panel ID	Panel title	Panel modified
DSNTIPA0	Online Book Data Set Names	
DSNTIPA1	Main Panel	
DSNTIPA2	Data Parameters	
DSNTIPK	Define Group or Member (1)	
DSNTIPH	System Resource Data Set Names	
DSNTIPT	Data Set Names Panel 1	yes
DSNTIPU	Data Set Names Panel 2	yes
DSNTIPQ	Data Set Names Panel 3	yes
DSNTIPG	Data Set Names Panel 4	yes
DSNTIPW	Data Set Names Panel 5	yes
DSNTIPD	Sizes Panel 1	yes
DSNTIP7	Sizes Panel 2	yes
DSNTIPE	Thread Management	yes
DSNTIP1	Buffer Pool Sizes Panel 1	yes

Table 27 (Page 2 of 2). Version 6 installation and migration panels

Panel ID	Panel title	Panel modified
DSNTIP2	Buffer Pool Sizes Panel 2	yes
DSNTIP6	Buffer Pool Sizes Panel 3	yes
DSNTIPN	Tracing and Checkpoint Parameters	yes
DSNTIPO	Operator Functions	yes
DSNTIPF	Application Programming Defaults Panel 1	yes
DSNTIP4	Application Programming Defaults Panel 2	yes
DSNTIPI	IRLM Panel 1	yes
DSNTIPJ	IRLM Panel 2	yes
DSNTIPP	Protection	yes
DSNTIPM	MVS PARMLIB Updates	
DSNTIPL	Active Log Data Set Parameters	yes
DSNTIPA	Archive Log Data Set Parameters	
DSNTIPS	Databases and Spaces to Start Automatically	
DSNTIPR	Distributed Data Facility Panel 1	yes
DSNTIP5	Distributed Data Facility Panel 2	yes
DSNTIPX	Routine Parameters	yes
DSNTIPZ	Data Definition Control Support	
DSNTIPY	Job Editing	yes
DSNTIPC	DB2 CLIST Calculations Panel 1	yes
DSNTIPC1	DB2 CLIST Calculations Panel 2	
DSNTIPB	Update Selection Menu	yes

Notes:

1. DSNTIPK is for installing and migrating data sharing.

SMP/E changes

The SMP/E process for installing DB2 has been simplified. There is a separate optional job for defining SMP/E data sets and zones for DB2. All required FMIDs for DB2 UDB for OS/390 are consolidated into one set of SMP/E jobs for receive, apply and accept processing. The FMIDs that have been moved into the consolidated set of jobs include online help, and DB2I English panels.

Changes to installation jobs

Installation job DSNTIJSG is modified to define a new stored procedure, DSNTPSMP. If you have already installed DB2, you can use job DSNTIJCC to define stored procedures for the DB2 Control Center. You can use job DSNTIJSQ to define the SQL procedures support. See . for details on enabling stored procedures after installation. The new stored procedures are:

DSNACCQC
: DB2 UDB Control Center catalog query stored procedure

DSNACCAV
: DB2 UDB Control Center partition information stored procedure

DSNTPSMP
: SQL procedures processor stored procedure

Changes to sample jobs

With the addition of the new functions in Version 6, several existing sample jobs have been modified and several new jobs added. The new and changed sample jobs are listed in Table 28.

Table 28. New and modified sample jobs

Sample job	New or modified
DSNTEJ0	modified
DSNTEJ1L	new
DSNTEJ1P	modified
DSNTEJ2U	new
DSNTEJ6S	modified
DSNTEJ6T	modified
DSNTEJ63	new
DSNTEJ64	new
DSNTEJ65	new
DSNTEJ7	new
DSNTEJ71	new
DSNTEJ73	new
DSNTEJ75	new

Appendix A. Changes to commands

This appendix provides an overview of the new and changed commands in Version 6 of DB2 for OS/390:

"New commands"
Table 30 on page 254

The purpose of the appendix is to highlight the major changes. For complete information on all the changes, such as the syntax for new or changed commands see *DB2 Command Reference*.

New commands

"New commands" shows the new commands in Version 6.

Table 29. New commands

Command name	Description
DISPLAY FUNCTION SPECIFIC (DB2)	Displays statistics about external user-defined functions that DB2 applications access
DISPLAY LOG (DB2)	Displays log information and the status of the off-load task.
MODIFY irlmproc, DIAG, DELAY (MVS IRLM)	Initiates diagnostic dumps for IRLM subsystems in a data sharing group when responses to XES requests take longer than 45 seconds.
MODIFY irlmproc, SET (MVS IRLM)	Dynamically sets the maximum CSA allowed for IRLM and the number of trace buffers allowed for IRLM
REBIND TRIGGER PACKAGE (DSN)	Can be used to do the following: • Rebind a package that was created when DB2 executed a CREATE TRIGGER statement. • Change a limited subset of the default bind options that DB2 used when creating the package. • Re-optimize its SQL statements after you create a new index or use the RUNSTATS utility. • Rebind a package if it has been marked invalid because an index, or another object it was dependent on, was dropped.
RECOVER POSTPONED (DB2)	Completes back-out processing for units of recovery that are left incomplete during an earlier restart (POSTPONED ABORT units of recovery). Use this command when automatic resolution was not selected.
SET LOG (DB2)	Until restart, modifies the checkpoint frequency specified during installation. This command also overrules the value that was specified in a previous invocation of the SET LOG command.
START FUNCTION SPECIFIC (DB2)	Activates an external function that is stopped.
STOP FUNCTION SPECIFIC (DB2)	Prevents DB2 from accepting SQL statements with invocations of the specified functions.

Changed commands

As shown in Table 30, many existing commands have new and changed options.

Table 30 (Page 1 of 7). Changes to existing commands

Command	Description of enhancements and notes
ALTER BUFFERPOOL	**New option:** PGSTEAL **Changed options:** VPSIZE HPSIZE PGSTEAL specifies the page stealing algorithm DB2 uses for the virtual buffer pool (LRU, FIFO). VPSIZE, HPSIZE and (bpname) are changed to support 8-KB and 16-KB page sizes.
ALTER GROUPBUFFERPOOL	**New option:** GBPCACHE GBPCACHE specifies whether (gbpname) is used for both caching data and cross-invalidation, or just for cross-invalidation (YES, NO). (gbpname) has been change to support 8-KB and 16-KB page sizes.
BIND and REBIND PACKAGE	**New options:** DBPROTOCOL OPTHINT PATH PATHDEFAULT (REBIND only) **Changed option:** DYNAMICRULES DBPROTOCOL Directs DB2 to use either DRDA or PRIVATE protocol when it connects to a remote site that is identified by a three-part name statement. DYNAMICRULES accepts four new values, DEFINEBIND, DEFINERUN, INVOKEBIND, or INVOKERUN, to determine what rules apply to dynamic SQL statements issued within a stored procedure or user-defined function. OPTHINT controls whether query optimization hints are used for static SQL. PATH determines the SQL path that DB2 uses to resolve unqualified names for the following items: • Stored procedure names in CALL statements • User-defined distinct types • User-defined functions PATHDEFAULT resets the PATH to the default value (REBIND only).

Table 30 (Page 2 of 7). Changes to existing commands

Command	Description of enhancements and notes
BIND and REBIND PLAN	**New options:** DBPROTOCOL OPTHINT PATH PATHDEFAULT (REBIND only) DBPROTOCOL Directs DB2 to use either DRDA or PRIVATE protocol when it connects to a remote site that is identified by a three-part name statement. OPTHINT controls whether query optimization hints are used for static SQL. PATH determines the SQL path that DB2 uses to resolve unqualified names for the following items: • Stored procedure names in CALL statements • User-defined distinct types • User-defined functions PATHDEFAULT resets the PATH to the default value (REBIND only).
Bind Options	**New options:** DBPROTOCOL OPTHINT PATH PATHDEFAULT (REBIND only) **Changed option:** DYNAMICRULES See BIND and REBIND PLAN for a description of enhancements.
DISPLAY BUFFERPOOL	**New options:** GBPDEP CASTOWNR **Changed options:** LIST LSTATS GBPDEP indicates whether to restrict the list of data sets to those that are group-buffer-pool-dependent. CASTOWNR indicates to restrict the list of data sets to those for which this DB2 member is the castout owner. LIST lists the open index spaces and the table spaces associated with the buffer pools included in the report. Basic information is provided for non-data-sharing systems while more detail is provided if data-sharing is active. LSTATS lists data set statistics for the open index spaces and table spaces associated with the buffer pools included in the report. The statistics displayed are incremental since the last time they were displayed.

Table 30 (Page 3 of 7). Changes to existing commands

Command	Description of enhancements and notes
DISPLAY DATABASE	**New option:** ADVISORY **Changed options:** SPACENAM AFTER RESTRICT ADVISORY limits the display to indexes and table spaces to which read/write access is allowed, but for which some action is recommended. SPACENAM has been enhanced so that you can write *space-name* like *database-name* to designate a partial name, including a beginning or ending pattern-matching character (*), a pattern-matching character between two strings, or any combination of these. *Message DSNT392I status information:* The new status codes displayed by the DISPLAY DATABASE command and their respective descriptions are as follows: **ACHKP** Indicates an error in the LOB column of the base table space. The base table space is in the auxiliary CHECK pending restrictive state. **AUXW** Either the base table space in the auxiliary warning advisory state, indicating an error in the LOB column, or the LOB table space is in the auxiliary warning advisory state, indicating an invalid LOB. **CHKP** The object (a table space, a partition within a table space, or an index) is in the CHECK pending state. **ICOPY** The index space is in the informational COPY pending advisory state. **PSRBD** The index space is in page set rebuild pending status. **RBDP** The object (an index space, index partition or logical index partition) is in rebuild pending status. For logical partitions, the RBDP status can appear as RBDP*. **REORP** The data partition is in the REORG pending state. **RESTP** The table space or index space is in the restart pending state.

Table 30 (Page 4 of 7). Changes to existing commands

Command	Description of enhancements and notes
DISPLAY GROUPBUFFERPOOL	**Changed option:** TYPE(NOCACHE) TYPE(NOCACHE) displays group buffer pools that have the GBPCACHE attribute set to NO. New information in the Summary Report is as follows: **DUPLEX** Indicates the current option for the group buffer pool specified in the active CFRM policy. **DUPLEXING STATUS** Indicates the current state of the group buffer pool relative to duplexing. **CFNAME, CFLEVEL** Indicates the names and level of the coupling facility in which the group buffer pool is associated. If the group buffer pool is duplexed, this is the coupling facility name and level associated with the primary group buffer pool. **ICOPY** The index space is in the informational COPY pending advisory state. **REORP** The data partition is in the REORG pending state. **RESTP** The table space or index space is in the restart pending state. New information in the Group Detail Report is as follows: **DUPLEXING STATISTICS FOR GBP*n*** Indicates detailed duplexing statistics. New information in the Member Detail Report is as follows: **DUPLEXING STATISTICS FOR GBP0-SEC** Lists details of other interactions that the DB2 has with the group buffer pool. **DELETE NAME LIST** Indicates the number of Delete List requests to delete a set of pages from the secondary group buffer pool that have just been cast out from the primary buffer pool. **DELETE NAME** Indicates the number of Delete Name requests to delete orphaned data entries from the secondary group buffer pool. The DB2 member that is the group buffer pool structure owner issues these requests if it determines that garbage collection is necessary.

Table 30 (Page 5 of 7). Changes to existing commands

Command	Description of enhancements and notes
DISPLAY PROCEDURE	**New options:** *(schema.procedure-name)* *(schema.partial-name*)* **Changed options:** *(procedure-name)* *(partial-name*)* (*.*) *(schema.procedure-name)* displays the specified stored procedure in the specified schema. *(schema.partial-name*)* displays a set of stored procedures in the specified schema that have been accessed by DB2 applications since DB2 was started. The names of all procedures in the set begin with *partial-name* and can end with an string, including the empty string. *(procedure-name)* displays one or more specific stored procedure names in the SYSPROC schema. I *(partial-name*)* displays information for a set of stored procedures in the SYSPROC schema that have been accessed by DB2 applications since DB2 was started. (*.*) Displays information for all stored procedures in all schemas that have been accessed by DB2 applications since DB2 was started. New output columns are as follows: **MAXQUE** The maximum number of threads that have waited concurrently for the procedure to be scheduled since DB2 was started. DB2 resets this value to 0 each time you execute the START PROCEDURE command. **TIMEOUT** The number of times an SQL CALL statement timed out while waiting for a request for the procedure to be scheduled. DB2 resets this value to 0 each time you execute the START PROCEDURE command.
DISPLAY THREAD	**Changed option:** TYPE(POSTPONED) TYPE(POSTPONED) displays information about units of work whose back-out processing has been postponed.
DSNH (TSO CLIST) and DB2I panels	**New DSNH CLIST parameters:** PDBPROTOCAL PDYNAMICRULES POPTHINT PPATH **New DSN BIND PACKAGE subcommand keywords:** DBPROTOCOL DYNAMICRULES OPTHINT PATH

Table 30 (Page 6 of 7). Changes to existing commands

Command	Description of enhancements and notes
START DATABASE	**Changed options:** SPACENAM ACCESS(FORCE) SPACENAM has been enhances so you can write *space-name* like *database-name* to designate: • The name of a single table space or index space • A range of names • A partial name, including a beginning or ending pattern-matching character (*), a pattern-matching character (*) between two strings, or any combination of these uses. You cannot use a partial name or range of names with the ACCESS(FORCE) option. ACCESS(FORCE) has been changed so that FORCE cannot be used to reset the restart pending (RESTP) state. START DATABASE ACCESS(FORCE) will not execute as long as postponed abort or indoubt units of recovery exist, and the command will fail issuing SQLCODE -904. The START DATABASE command can be used to start LOB table spaces and indexes on auxiliary tables. LOB table spaces are started independently of the base table space with which the LOB table space is associated.
START PROCEDURE	**New options:** *(schema.procedure-name)* *(schema.partial-name*)* (*.*) **Changed options:** *(procedure-name)* *(partial-name*)* You can now qualify stored procedures with schema name. *(schema.procedure-name)* starts the specified stored procedure in the specified schema. *(schema.partial-name*)* starts a set of stored procedures in the specified schema. The names of all procedures in the set begin with *partial-name* and can end with an string, including the empty string. (*.*) marks all stored procedures in all schemas as available to be called.*procedure-name* marks one or more specific stored procedure as available to be called. *partial-name** marks a set of stored procedures in the SYSPROC schema as available to be called.

Table 30 (Page 7 of 7). Changes to existing commands

Command	Description of enhancements and notes
STOP DATABASE	**Changed options:** SPACENAM AT(COMMIT) SPACENAM has been enhances so you can write *space-name* like *database-name* to designate: • The name of a single table space or index space • A range of names • A partial name, including a beginning or ending pattern-matching character (*), a pattern-matching character (*) between two strings, or any combination of these uses. AT(COMMIT) marks the specified object as being in STOPP status to prevent access from new requesters. The object is actually stopped and put in the STOP status when all jobs release their claims on it and all utilities release their drain locks on it. An object might remain in the STOPP status if the STOP DATABASE command did not successfully complete processing. The STOP DATABASE command can be used to stop LOB table spaces and indexes on auxiliary tables. LOB table spaces are stopped independently of the base table space with which the LOB table space is associated.
STOP PROCEDURE	**New options:** *(schema.procedure-name)* *(schema.partial-name*)* **Changed options:** *(procedure-name)* *(partial-name*)* (*.*) A stopped procedure does not remain stopped if DB2 is stopped and restarted. To disable a stored procedure you can drop the procedure using the DROP PROCEDURE SQL statement. *(schema.procedure-name)* identifies the fully-qualified procedure name that is stopped. *(schema.partial-name*)* stops a set of stored procedures in the specified schema. The names of all procedures in the set begin with *partial-name* and can end with an string, including the empty string. *procedure-name* indicates one or more specific stored procedure names to be stopped. The procedure name is implicitly qualified with the SYSPROC schema name.*partial-name** stops a set of stored procedures in the SYSPROC schema. The names of all procedures in the set begin with *partial-name* and can end with an string, including the empty string. (*.*) Stops access to all stored procedures in all schemas, including procedure definitions that have not yet been accessed by DB2 applications.

Appendix B. Changes to utilities

This chapter summarizes the numerous changes to utilities in Version 6 of DB2 for OS/390:

"New utilities"
"Changed utilities"
"Other utility changes" on page 268

New utilities

Table 31 shows the new utilities.

Table 31. Overview of new utilities

Utility name	Description
CHECK LOB	The CHECK LOB online utility can be run against a LOB table space to identify any structural defects in the LOB table space and any invalid LOB values. Run the CHECK LOB online utility against a LOB table space that is marked CHECK pending (CHKP) to identify structural defects. If none is found, the CHECK LOB utility turns the CHKP status off. Run the CHECK LOB online utility against a LOB table space that is in auxiliary warning (AUXW) status to identify invalid LOBs. If none exists, the CHECK LOB utility turns AUXW status off. Run CHECK LOB after a conditional restart or a point-in-time recovery on all table spaces where LOB table spaces might not be synchronized.
REBUILD INDEX	REBUILD INDEX recreates indexes from the table that they reference. **Important:** You must modify all RECOVER INDEX jobs from a previous release of DB2 to use REBUILD INDEX instead. The REBUILD INDEX utility supports the following enhancements: • Collect inline statistics for an index using the STATISTICS keyword. • Rebuild indexes in parallel using the SORTKEYS keyword. • Logically reset DB2-managed data sets using the REUSE keyword.

Changed utilities

As shown in Table 32 on page 262, many existing DB2 for OS/390 utilities have new and changed options.

Table 32 (Page 1 of 7). New and changed utility options

Utility name	Description of enhancements and notes
CHECK DATA	**New options:** AUXERROR LOG **Changed option:** SCOPE Use the AUXERROR option of the CHECK DATA utility to detect, identify, and invalidate LOB column errors. If you specify the LOG NO option, CHECK DATA does not log any records that are deleted during the REPORTCK phase. The SCOPE option now supports checking LOB columns in a base table space for errors, or excluding these from being checked.
COPY	**New options:** CHECKPAGE INDEX INDEXSPACE PARALLEL **Changed option:** CHANGELIMIT Specify the new CHECKPAGE keyword to check each page in the table space or index space for validity. You can now take a full image copy of indexes and index spaces that were defined with the COPY YES attribute. To improve performance for image copies on DASD devices, use the new PARALLEL option to copy any of the following objects in parallel: • Table space • Table space partition • Data set of a linear table space • Index space • Index space partition Specifying objects in a list is useful for copying a complete set of referentially related table spaces after running QUIESCE. The CHANGELIMIT option now accepts either an integer or a decimal value from 0 to 100.
DSN1COMP	**New options:** DSSIZE PAGESIZE **Changed option:** LARGE The new DSSIZE option allows you to specify the size, in gigabytes, of the input data set. Use the PAGESIZE option to specify the size of the input data set that is defined by SYSUT1. If you specify LARGE, then DB2 assumes that the data set has a 4-GB boundary. The preferred method of specifying a table space defined with LARGE is **DSSIZE(4G)**.

Table 32 (Page 2 of 7). New and changed utility options

Utility name	Description of enhancements and notes
DSN1COPY	**New options:** DSSIZE LOB PAGESIZE **Changed options:** FULLCOPY LARGE The new DSSIZE option allows you to specify the size, in gigabytes, of the input data set. The new LOB option specifies that the SYSUT1 data set is a LOB table space. Use the PAGESIZE option to specify the size of the input data set that is defined by SYSUT1. DSN1COPY now accepts an index image copy as input when you specify the FULLCOPY option. If you specify LARGE, then DB2 assumes that the data set has a 4-GB boundary. The preferred method of specifying a table space defined with LARGE is **DSSIZE(4G)**.
DSN1PRNT	**New options:** DSSIZE LOB PAGESIZE **Changed options:** FULLCOPY LARGE The new DSSIZE option allows you to specify the size, in gigabytes, of the input data set. The new LOB option specifies that the SYSUT1 data set is a LOB table space. Use the PAGESIZE option to specify the size of the input data set that is defined by SYSUT1. DSN1PRNT now accepts an index image copy as input when you specify the FULLCOPY option. If you specify LARGE, then DB2 assumes that the data set has a 4-GB boundary. The preferred method of specifying a table space defined with LARGE is **DSSIZE(4G)**.

Table 32 (Page 3 of 7). New and changed utility options

Utility name	Description of enhancements and notes
LOAD	**New options:** FLOAT NOCOPYPEND REUSE STATISTICS **New options for data types in a field specification:** BLOB CLOB DBCLOB ROWID **Changed option:** SORTKEYS Specify the new FLOAT keyword to indicate whether floating point numbers are provided in System/390 hexadecimal Floating Point (HPF) format, or in IEEE Binary Floating Point (BFP) format. NOCOPYPEND specifies that LOAD is not to set the table space in the COPY pending status, even though LOG NO was specified. A NOCOPYPEND specification will not turn on or change any informational COPY pending (ICOPY) status for indexes. Logically reset DB2-managed data sets with the new REUSE keyword. Collect inline statistics for a table or index using the new STATISTICS keyword. Specify any of the following new data types in a field specification: • Binary large object (BLOB) • Character large object (CLOB) • Double-byte character large object (DBCLOB) • A row ID (ROWID) Build multiple indexes in parallel using the SORTKEYS keyword.
QUIESCE	**New option:** TABLESPACESET The QUIESCE utility now supports specifying a set of referentially-related table spaces using the TABLESPACESET keyword. You can also specify a list of table spaces and table space sets to QUIESCE, within the limits of available memory.

Table 32 (Page 4 of 7). New and changed utility options

Utility name	Description of enhancements and notes
RECOVER	**New options:** INDEXSPACE PARALLEL REUSE ***Compatibility with prior releases:*** In previous releases of DB2, REBUILD INDEX was called RECOVER INDEX. You must modify all utility control statements from previous releases to use REBUILD INDEX if you want to continue recovering the indexes via a scan of the data. However, if you want to recover the indexes from a full image copy, change those control statements to use the new RECOVER INDEX syntax. Only indexes that were defined with the COPY YES attribute can be copied and subsequently recovered; see "Backup and recover indexes using image copies" on page 30 for more information about copying and recovering indexes. Specify the new INDEXSPACE keyword to recover an index space for which a full image copy exists. The new PARALLEL keyword allows you to recover a list of objects in parallel. If you use the RECOVER utility to recover a table space set, then you must ensure that you recover the entire set of table spaces, including all indexes on auxiliary tables, to a common quiesce point or to a SHRLEVEL REFERENCE copy. The new REUSE keyword specifies that RECOVER logically resets and reuses DB2-managed data sets without deleting and redefining them.
REORG INDEX	**New options:** DRAIN LEAFDISTLIMIT REPORTONLY REUSE STATISTICS TIMEOUT DRAIN specifies drain behavior at the end of the log phase after the MAXRO threshold is reached and when the last iteration of the log is to be applied.You can determine when to run REORG INDEX by using the LEAFDISTLIMIT catalog query option. Specify the REPORTONLY option to produce a report that indicates if a REORG is recommended; a REORG is not performed. Specify the REUSE keyword to enable logical reset of DB2-managed data sets. Logically resetting data sets is faster than a delete and redefine. STATISTICS collects inline statistics, updating the catalog tables during a reorganization. TIMEOUT specifies the action to be taken (ABEND or TERM) if the REORG utility gets a time out condition while trying to drain objects in either the LOG or the SWITCH phase.

Table 32 (Page 5 of 7). New and changed utility options

Utility name	Description of enhancements and notes
REORG TABLESPACE	**New options:** DISCARD DISCARDDN DRAIN INDREFLIMIT NOPAD OFFPOSLIMIT PART *m*:*n* PUNCHDDN REPORTONLY REUSE STATISTICS TIMEOUT UNLOAD EXTERNAL **Changed option:** SORTKEYS Specify the DISCARD option to delete rows during the REORG. The optional DISCARDDN keyword specifies a discard data set to hold copies of records that meet the DISCARD FROM TABLE ... WHEN criteria. DRAIN specifies drain behavior at the end of the log phase after the MAXRO threshold is reached and when the last iteration of the log is to be applied. You can determine when to run REORG for non-LOB table spaces and indexes by using the OFFPOSLIMIT and INDREFLIMIT catalog query options. Specify the REPORTONLY option to produce a report that indicates if a REORG is recommended; a REORG is not performed. The new NOPAD option specifies that the variable-length columns in unloaded or discarded records occupy the actual data length without additional padding. The unloaded or discarded records may have varying lengths. You can reorganize a single partition of a partitioned table space using the PART keyword. You can also rebalance a range of partitions that are in REORG pending status using the PART *m*:*n* keyword. Specify the PUNCHDDN keyword to designate a data set to receive the LOAD utility control statements that are generated by UNLOAD EXTERNAL or DISCARD FROM TABLE ... WHEN. Specify the REUSE keyword to enable logical reset of DB2-managed data sets. Logically resetting data sets is faster than a delete and redefine. STATISTICS collects inline statistics, updating the catalog tables during a reorganization. You can maximize availability during an online REORG with SHRLEVEL CHANGE by using the new TIMEOUT option. You can use the UNLOAD EXTERNAL option to unload data in a format that is acceptable to the LOAD utility of any DB2 subsystem. The SORTKEYS option provides improved performance by sorting index keys and building indexes in parallel.

Table 32 (Page 6 of 7). New and changed utility options

Utility name	Description of enhancements and notes
REPAIR	**New SET TABLESPACE statement options:** NOAUXCHKP NOAUXWARN The SET TABLESPACE statement resets the auxiliary warning (AUXW) or auxiliary CHECK pending (ACHKP) status for a table space or data set. **New SET INDEX statement options:** NOCOPYPEND NOCHECKPEND The SET INDEX statement resets the informational COPY pending (ICOPY) or CHECK pending status for an index. **New LOCATE TABLESPACE statement options:** PART ROWID VERSION **Changed LOCATE TABLESPACE statement option:** PAGE Using the LOCATE block statement, you can now specify the partition (PART) and relative page number (PAGE) as an *integer* value within a table space or partitioned table space. Alternatively, you can specify PAGE as a hexadecimal value. ROWID or VERSION specifies that the data to be located is a LOB in a LOB table space. ROWID *byte-string* is the row ID that identifies the LOB column. VERSION *byte-string* is the version number that identifies the version of the LOB column. **New LOCATE INDEX statement option:** PART **Changed LOCATE INDEX statement option:** PAGE Using the LOCATE block statement, you can now specify the partition (PART) and relative page number (PAGE) as an *integer* value within a partitioning index on a partitioned table space. Alternatively, you can specify PAGE as a hexadecimal value. **New DUMP statement options:** DATA MAP Use the DUMP statement to dump LOB map pages and data pages, or specify the new MAP or DATA options to dump one or the other.

Table 32 (Page 7 of 7). New and changed utility options

Utility name	Description of enhancements and notes
REPORT	**New options:** ARCHLOG INDEX INDEXSPACE You can now use REPORT RECOVERY to find information that is necessary for recovering a table space, index, or a table space and all related indexes. REPORT RECOVERY output also includes information about any indexes that are in informational COPY pending (ICOPY) status, because this affects the recoverability of an index. You can use the new ARCHLOG option to designate which archive log data sets are reported. The REPORT utility also gives the LOB table spaces associated with a base table space.
RUNSTATS	**Changed option:** STATISTICS Use the STATISTICS keyword with LOAD, REBUILD INDEX, and REORG jobs to eliminate the need to execute RUNSTATS for updating catalog statistics. If you restart a LOAD or REBUILD INDEX job that uses the STATISTICS keyword, inline statistics collection does not occur. To update catalog statistics, run the RUNSTATS utility after the restarted utility job completes. You can specify that a LOB table space is to have space statistics collected so you can determine when the LOB table space should be reorganized. No statistics on the LOB table space affect access path selection. Distribution statistics are now collected for uniform distributions as well as non-uniform distributions. New space statistics are collected for the SPACE, PQTY, and SECQTYI catalog columns in SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART.

Other utility changes

Some other Version 6 functions impact utility operations as follows:

- The CATMAINT UPDATE utility abends if you still have type 1 indexes, shared read-only data, or data set passwords.
- The CHECK INDEX utility verifies that each LOB is represented by an index entry in the index on the auxiliary table, and that an index entry exists for every LOB.
- The MODIFY utility automatically removes the SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX recovery records that meet the AGE and DATE criteria for all indexes over the table space that were defined with the COPY YES attribute.

Appendix C. Changes to SQL

This appendix provides an overview of the new and changed SQL in Version 6 of DB2 for OS/390:

"New SQL statements"
"Changed SQL statements" on page 270
"New built-in functions" on page 279
"Changed built-in functions" on page 282
"Other SQL language changes" on page 282

The purpose of the appendix is to highlight the major changes. For complete information on all the changes, such as the syntax for new or changed SQL statements, comprehensive descriptions of keywords, and examples of usage, see *DB2 SQL Reference*.

New SQL statements

Table 33 shows the new SQL statements in Version 6. Many of these statements were introduced to provide support for object-relational extensions and active data, such as defining user-defined functions, distinct types, or triggers.

Table 33 (Page 1 of 2). New SQL statements

SQL statement	Description
ALTER FUNCTION	Changes the description of an external user-defined function
ALTER PROCEDURE	Changes the description of a stored procedure
CREATE AUXILIARY TABLE	Defines an auxiliary table for storing LOB data
CREATE DISTINCT TYPE	Defines a distinct type (user-defined data type)
CREATE FUNCTION	Defines a user-defined function
CREATE PROCEDURE	Defines a stored procedure
CREATE TRIGGER	Defines a trigger
# DECLARE GLOBAL TEMPORARY # TABLE	Defines a declared temporary table
DESCRIBE INPUT	Puts information about the input parameters (markers) of a prepared statement into a descriptor
FREE LOCATOR	Removes the association between a LOB locator variable and its value
GRANT (Distinct Type Privileges)	Grants the usage privilege on a distinct type (user-defined data type)
GRANT (Function or Procedure Privileges)	Grants privileges on a user-defined function or a stored procedure
GRANT (Schema Privileges)	Grants privileges on a schema
HOLD LOCATOR	Allows a LOB locator variable to retain its association with its value beyond a unit of work
# RELEASE SAVEPOINT #	Releases a savepoint and any subsequently set savepoints within a unit of recovery
REVOKE (Distinct Type Privileges)	Revokes the usage privilege on a distinct type (user-defined data type)
REVOKE (Function or Procedure Privileges)	Revokes privileges on a user-defined function or a stored procedure

Table 33 (Page 2 of 2). New SQL statements

SQL statement	Description
REVOKE (Schema Privileges)	Revokes privileges on a schema
# SAVEPOINT	Sets a savepoint within a unit of recovery
SET Assignment	Assigns values to host variables and transition variables
SET CURRENT LOCALE LC_CTYPE	Assigns a value to the CURRENT LOCALE LC_CTYPE special register
SET CURRENT OPTIMIZATION HINT	Assigns a value to the CURRENT OPTIMIZATION HINT special register
SET CURRENT PATH	Assigns a value to the CURRENT PATH special register
SIGNAL SQLSTATE	Signals an error with a user-specified SQLSTATE and description
VALUES	Provides a method to invoke a user-defined function from a trigger
VALUES INTO	Assigns values to host variables

Changed SQL statements

As shown in Table 34, many existing SQL statements have new and changed clauses.

Table 34 (Page 1 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
ALTER DATABASE	**New clause:** INDEXBP **Changed clause:** BUFFERPOOL BUFFERPOOL is changed to only identify the default buffer pool for table spaces and to support 8-KB and 16-KB page sizes. Use the new clause INDEXBP to specify the default buffer pool for indexes.

Table 34 (Page 2 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
ALTER INDEX	**New clauses:** COPY VALUES **Changed clauses:** GBPCACHE PIECESIZE **Deleted clauses:** CONVERT TO DSETPASS The new COPY clause specifies whether the COPY and RECOVER utilities are allowed on the index. For partitioning indexes, the statement is enhanced to allow more than one partition to be altered in a single ALTER statement, allowing you to alter some or all of the partitions in one statement. In addition, the VALUES clause is supported to change the index key for a partition. For nonpartitioning indexes defined on EA-enabled table spaces, changes to PIECESIZE enable a larger maximum piece size to be specified. In a data sharing environment, a new value of NONE for GBPCACHE indicates that pages are not cached to the group buffer pool and DB2 will use the group buffer pool only for cross-invalidation. Because DB2 now only allows type 2 indexes, an index cannot be altered to a type 1. Therefore, the CONVERT TO and associated TYPE 1, TYPE 2, and SUBPAGES clauses are no longer supported. You can also no longer use DSETPASS to specify a password; support for the clause is removed.
ALTER STOGROUP	**Deleted clause:** PASSWORD The PASSWORD clause is no longer supported. To protect your data sets, use OS/390 Security Server or an equivalent security system.

Table 34 (Page 3 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
ALTER TABLE	**New clauses:** ALTER COLUMN GENERATED **Changed clauses:** ADD column-definition DEFAULT With the new ALTER COLUMN clause, you can increase the length of any VARCHAR column that exists in the table. The length of a VARCHAR column can be increased only if the table is not defined with DATA CAPTURE CHANGE. Changing the length of a VARCHAR column invalidates all the plans and packages that refer to the table. # The ADD clause is enhanced to support adding a column that has a LOB data type (CLOB, DBCLOB, or BLOB), a row ID data type, or a distinct type (user-defined data type). A column that is being added can also be identified as an identity column. For additional details about adding these types of columns to a table and the GENERATED and DEFAULT clauses, see the description of changes for the CREATE TABLE statement in this table.
ALTER TABLESPACE	**New clauses:** LOG TRACKMOD **Changed clauses:** GBPCACHE LOCKSIZE PRIQTY SECQTY For LOB table spaces, the new LOG clause specifies whether changes to LOB columns in the table space are logged. A new value of LOB for the LOCKSIZE clause enables LOB locks to be acquired on a LOB table space. The new TRACKMOD clause specifies whether modified pages in the space map pages are tracked. You cannot specify TRACKMOD for LOB tables spaces. For the primary space allocation, the PRIQTY clause is extended to support minimum allocation values for 8-KB and 16-KB page sizes, in addition to the already supported minimums for 4-KB and 32-KB page sizes. For secondary space allocation, the maximum value that DB2 can allocate for any page size is increased to 4194304 kilobytes (for both LOB and non-LOB table spaces). For LOB table spaces, DB2 uses minimum allocation values for each page size that are larger than the minimum values that are used for non-LOB table spaces. For data sharing, the GBPCACHE clause has two new values. The value SYSTEM, which can only be specified for LOB table spaces, indicates that only changed system pages within the table space are to be cached to the group buffer pool. A system page is a space map page or any other page that does not contain actual data values. A value of NONE, which can be specified for any table space, indicates that pages are not cached to the group buffer pool and DB2 will use the group buffer pool only for cross-invalidation.

Table 34 (Page 4 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
ASSOCIATE LOCATORS	**Changed clause:** WITH PROCEDURE The procedure name that is identified in the WITH PROCEDURE clause must be specified the same way that it was specified on the CALL statement. The names must have the same number of parts. However, if a three-part name was used in the CALL statement and the current server is the same as the location name specified, the location name can be omitted in the ASSOCIATE LOCATORS statement.
CALL	To support the enhancements for stored procedures, the CALL statement is revised to invoke procedures that are qualified by schema names. The statement also supports the ability to pass more types of parameters. For example, any expression that does not contain a column name, labeled duration, column function or user-defined function that is based on a column function can be passed. If the expression is a single host variable, the host variable can identify a structure. In addition, if the CALL statement is used within the triggered action of a trigger, a table can be passed as a parameter.
COMMENT ON	**New clauses:** DISTINCT TYPE FUNCTION SPECIFIC FUNCTION PROCEDURE TRIGGER You can use the new clauses in the COMMENT ON statement to add or replace comments in the descriptions of distinct types, user-defined functions, stored procedures, and triggers in the DB2 catalog.
# COMMIT #	When the COMMIT statement ends a unit of recovery, the statement will also release any active savepoints that were set within the unit of recovery.
CREATE ALIAS	If an application program use three-part alias names for remote objects and DRDA access to the remote site, the application must be bound at each location that is specified in the three-part names. Also, each alias must defined at the remote site as well as the local site. An application uses DRDA access for three-part names when the plan or package that contains the query to distributed data is bound with bind option DBPROTOCOL(DRDA), or the value of installation panel field DATABASE PROTOCOL is DRDA and the bind option DBPROTOCOL was not specified when the plan or package was bound.

Table 34 (Page 5 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
CREATE DATABASE	**New clause:** INDEXBP **Changed clause:** AS BUFFERPOOL BUFFERPOOL is changed to only identify the default buffer pool for table spaces and to support 8-KB and 16-KB page sizes. Use the new clause INDEXBP to specify the default buffer pool for indexes. The AS clause is enhanced to support TEMP as an additional keyword. Specifying AS TEMP indicates that the database is for declared temporary tables only. If you are using declared temporary tables, you must create a database that is defined AS TEMP (the TEMP database) because declared temporary tables can only reside in that database. A DB2 subsystem or data sharing member can have no more than one TEMP database.
CREATE INDEX	**New clause:** COPY DEFINE **Changed clauses:** GBPCACHE ON PIECESIZE **Deleted clauses:** DSETPASS SUBPAGES TYPE 1 The new COPY clause specifies whether the COPY and RECOVER utilities are allowed on the index. For DB2-managed data sets, you can use the new DEFINE clause to delay the physical creation of the underlyng data sets for the index until data is first inserted into the index. Auxiliary tables can have indexes; therefore, you can identify an auxiliary table in the ON clause. For nonpartitioning indexes defined on EA-enabled table spaces, changes to PIECESIZE enable a larger maximum piece size to be specified. In a data sharing environment, a new value of NONE for the GBPCACHE clause indicates that pages are not cached to the group buffer pool and DB2 will use the group buffer pool only for cross-invalidation. Because DB2 now only allows type 2 indexes, a type 1 index cannot be created. Therefore, the TYPE 1 and SUBPAGES clauses are no longer supported. You can also no longer use DSETPASS to specify a password; support for the clause is removed.
CREATE STOGROUP	**Deleted clause:** PASSWORD The PASSWORD clause is no longer supported. To protect your data sets, use OS/390 Security Server or an equivalent security system.

Table 34 (Page 6 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
CREATE TABLE	**New clause:** GENERATED AS IDENTITY **Changed clauses:** data-type DEFAULT LIKE The CREATE TABLE statement is enhanced to support the new data types: LOBs (CLOB, DBCLOB, or BLOB), row IDs, and distinct types (user-defined data types). A table can have only one column with a row ID data type (or distinct type based on a row ID type). When creating a table with a ROWID column, you must use the new GENERATED clause to specify whether DB2 will generate a value for the column when a row is inserted into the table regardless of whether a value is specified. You cannot use the DEFAULT clause to specify a default value for a ROWID column. When creating a table with a distinct type column, you can use one of the cast functions that DB2 generated when the distinct type was created as the value for the DEFAULT clause. If you define a table with a LOB column (or distinct type based on a LOB data type), you must also define a ROWID column for the table. Unless DB2 implicitly creates the LOB table space, auxiliary table to store the actual values of the LOB column, and index on the auxiliary table, you need to create these objects using the CREATE TABLESPACE, CREATE AUXILIARY TABLE and CREATE INDEX statements. DB2 implicitly creates the objects if bind option SQLRULES(STD) is in effect for the plan or package. If you define a table with a LOB column in a partitioned table space, there must be one auxiliary table defined for each partition of the base table space. You can use the new AS IDENTITY clause in conjunction with the GENERATED clause to define a column with an exact numeric type as the identity column of the table. When a table has an identity column, DB2 generates a unique, sequential numeric value for each row that is inserted into the table, making the column suitable for the task of generating unique primary key values. A table can have only one identity column. As with a ROWID column, the GENERATED clause for an identity column indicates whether DB2 will generate the value for the column regardless of whether a value is specified. You cannot use the DEFAULT clause to specify a default value for an identity column. When you are creating a table like another table that has an identity column, you can use an optional clause with the LIKE clause to indicate that the corresponding column of the new table is the identity column for the table. The new INCLUDING IDENTITY COLUMN ATTRIBUTES clause causes the new column to inherit all of the identity attributes of the identity column in the table identified in the LIKE clause.

Table 34 (Page 7 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
CREATE TABLESPACE	**New clauses:** # DEFINE DSSIZE LOB LOG TRACKMOD **Changed clauses:** GBPCACHE LOCKSIZE PRIQTY SECQTY # For DB2-managed data sets, the new DEFINE clause can delay the physical # creation of the underlyng data sets for the table space until data is first inserted into # the table space. You can use the new DSSIZE clause to create a table space that holds up to 16 terabytes of data. The value specified in DSSIZE indicates the maximum size in gigabytes of each partition (or for LOB table spaces, each data set). You can use the new LOB clause to indicate that the table space is to be a LOB table space. LOB table spaces hold auxiliary tables, which hold the actual LOB values for base tables that are defined with LOB columns. For LOB table spaces, you can specify whether changes to LOB columns in the table space are logged with the new LOG clause. A new value of LOB for the LOCKSIZE clause enables LOB locks to be acquired on a LOB table space. The new TRACKMOD clause specifies whether modified pages in the space map pages are tracked. You cannot specify TRACKMOD for LOB tables spaces. For the primary space allocation, the PRIQTY clause is extended to support minimum allocation values for 8-KB and 16-KB page sizes, in addition to the already supported minimums for 4-KB and 32-KB page sizes. For secondary space allocation, the maximum value that DB2 can allocate for any page size is increased to 4194304 kilobytes (for both LOB and non-LOB table spaces). For LOB table spaces, DB2 uses minimum allocation values for each page size that are larger than the minimum values that are used for non-LOB table spaces. For data sharing, the GBPCACHE clause has two new values. The value SYSTEM, which can only be specified for LOB table spaces, indicates that only changed system pages within the table space are to be cached to the group buffer pool. A system page is a space map page or any other page that does not contain actual data values. A value of NONE, which can be specified for any table space, indicates that pages are not cached to the group buffer pool and DB2 will use the group buffer pool only for cross-invalidation.
DECLARE TABLE	A table that is defined with a column that has any data type, including any one of the new built-in data types (CLOB, DBCLOB, BLOB, or row ID) or a distinct type, can be documented.

Table 34 (Page 8 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
DELETE	**New clause:** QUERYNO Rows cannot be explicitly deleted from an auxiliary table. When a row that contains a LOB column is deleted from a base table, the appropriate row that holds the actual LOB data in the corresponding auxiliary table row is deleted. You can use the new QUERYNO clause to assign a particular number to the QUERYNO column of the plan table for information that is added to the plan table for the statement.
DESCRIBE	When allocating the SQLDA, additional occurrences of the SQLVAR might be needed if the information that is returned describes LOB columns or distinct type columns. The number of SQLVAR occurrences that are required per column depends on whether both labels and names of the columns are returned. The first SQLVAR occurrence per column, which is always present, is called the base SQLVAR entry. When a second or third SQLVAR occurrence is required for a column, an extended SQLVAR entry is used. (The *DB2 SQL Reference* contains a complete description of the these two types of SQLVARs.)
DESCRIBE PROCEDURE	The procedure name that is identified in the WITH PROCEDURE clause must be specified the same way that it was specified on the CALL statement. The names must have the same number of parts. However, if a three-part name was used in the CALL statement and the current server is the same as the location name specified, the location name can be omitted in the DESCRIBE PROCEDURE statement.
DROP	**New clauses:** DISTINCT TYPE FUNCTION SPECIFIC FUNCTION PROCEDURE TRIGGER You can use the new clauses in the DROP statement to drop distinct types, user-defined functions, stored procedures, and triggers.
EXPLAIN	The plan table has three new columns. For the format of the table and descriptions of the new and changed columns, see Appendix E, “EXPLAIN table changes” on page 293. The EXPLAIN statement is also extended to insert information into two new tables. The new DSN_FUNCTION_TABLE is useful for finding out information about function resolution, and the new DSN_STATEMENT_TABLE is useful for finding out the estimated cost of SQL statements. Unlike the plan table, neither the function table nor the statement table has to exist to use EXPLAIN. For more details on these new tables, see “Using DSN_FUNCTION_TABLE to see how DB2 resolves a function” on page 197 and “Creating a statement table” on page 80.
EXECUTE	As described under the changes for the DESCRIBE statement, when allocating the SQLDA, additional occurrences of the SQLVAR might be needed.
FETCH	As described under the changes for the DESCRIBE statement, when allocating the SQLDA, additional occurrences of the SQLVAR might be needed.

Table 34 (Page 9 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
GRANT (table and view privileges)	**New clause:** TRIGGER The new TRIGGER clause grants the privilege to use the CREATE TRIGGER statement to create a trigger on a table. For an auxiliary table, only the INDEX privilege can be granted. DELETE, INSERT, SELECT, and UPDATE privileges on the base table that is associated with the auxiliary table extend to the auxiliary table.
INSERT	**New clause:** OVERRIDING USER VALUE QUERYNO **Changed clause:** VALUES subselect # Some types of columns, such as a ROWID column or an identity column, can be defined with the new attribute GENERATED ALWAYS. This attribute indicates that DB2 will always generate a value for the column when a row is inserted into the table. If you try to provide a value for such a column on the INSERT statement, the statement will fail unless you specify the new OVERRIDING USER VALUE clause. OVERRIDING USER VALUE specifies that DB2 will ignore the user-specified value and use a system-generated value instead. You can use the new QUERYNO clause to assign a particular number to the QUERYNO column of the plan table for information that is added to the plan table for the statement. # # # # The VALUES clause is enhanced to support any type of expression in the VALUES clause. In addition, a value of DEFAULT can be specified in the VALUES clause to indicate that the default value of the column will be assigned. You can specify DEFAULT only for columns that have an assigned default value, ROWID columns, and identity columns. When DEFAULT is specifiedfor a ROWID or identity column, DB2 will generate a unique value for the column. The rules for the *subselect* now allow the target table of the INSERT to be identified as a table in the *subselect*. With the ability to retrieve data from the table into which it is to be inserted, you can quickly replicate rows in a table or certain data fields.
LOCK TABLE	An auxiliary table can be locked. The effect of locking an auxiliary table is to lock the LOB table space that contains the auxiliary table. For information on the reasons to lock an auxiliary table, see “The LOCK TABLE statement” on page 140. For general guidance information on locking LOBs, see “Locking LOBs” on page 136.
OPEN	As described under the changes for the DESCRIBE statement, when allocating the SQLDA, additional occurrences of the SQLVAR might be needed.
PREPARE	As described under the changes for the DESCRIBE statement, when allocating the SQLDA, additional occurrences of the SQLVAR might be needed.
RENAME	An auxiliary table can be renamed. However, unlike plans and packages that refer to other tables that are renamed, plans and packages that refer to the auxiliary table are not invalidated.
REVOKE (table and view privileges)	**New clause:** TRIGGER The new TRIGGER clause revokes the privilege to use the CREATE TRIGGER statement to create a trigger on a table.

Table 34 (Page 10 of 10). Changes to existing SQL statements

SQL statement	Description of enhancements and notes
ROLLBACK	**New clause:** TO SAVEPOINT Instead of rolling back all the database changes that were made in the unit of recovery, the ROLLBACK statement can be used to perform a partial rollback to a savepoint within the unit of recovery. You can specify the new TO SAVEPOINT clause to back out only those changes made after a savepoint was set. The unit of recovery is not ended when a partial rollback is performed. In addition, these items are not rolled back when the TO SAVEPOINT clause is specified: the opening and closing of cursors, changes in cursor positioning, the acquisition and release of locks, and the caching of the rolled back statements.
SELECT INTO	**New clauses:** WHERE GROUP BY QUERYNO The SELECT INTO statement is enhanced to support the WHERE and a GROUP BY clauses, which allow you to specify a search condition and group results when selecting values for the result table. You can also use the new QUERYNO clause to assign a particular number to the QUERYNO column of the plan table for information that is added to the plan table for the statement.
UPDATE	**New clause:** QUERYNO **Changed clause:** SET You can use the new QUERYNO clause to assign a particular number to the QUERYNO column of the plan table for information that is added to the plan table for the statement. The SET clause is enhanced to support subselects. When specifying the new values for the columns to be updated, you can use a subselect that returns a single row on the right side of the SET clause. The values in a ROWID column, an identity column that is defined as GENERATED ALWAYS, or an auxiliary table cannot be updated.

New built-in functions

Table 35 shows the new built-in functions in Version 6, which improve the power of the SQL language.

Table 35 (Page 1 of 3). New built-functions

Function name	Description
Column functions	
COUNT_BIG	Same as COUNT, except the result can be greater than the maximum value of an integer
STDDEV	Returns the standard deviation of a set of numbers
VARIANCE	Returns the variance of a set of numbers
Scalar functions	

Table 35 (Page 2 of 3). New built-functions

Function name	Description
ABS or ABSVAL	Returns the absolute value of its argument
ACOS	Returns the arcosine of an argument as an angle, expressed in radians
ASIN	Returns the arcsine of an argument as an angle, expressed in radians
ATAN	Returns the arctangent of an argument as an angle, expressed in radians
ATANH	Returns the hyperbolic arctangent of an argument as an angle, expressed in radians
ATAN2	Returns the arctangent of *x* and *y* coordinates as an angle, expressed in radians
BLOB	Returns a BLOB representation of its argument
CEIL or CEILING	Returns the smallest integer greater than or equal to the argument
CLOB	Returns a CLOB representation of its argument
CONCAT	Returns the concatenation of two strings
COS	Returns the cosine of an argument that is expressed as an angle in radians
COSH	Returns the hyperbolic cosine of an argument that is expressed as an angle in radians
DAYOFMONTH	Similar to DAY
DAYOFWEEK	Returns an integer in the range of 1 to 7, where 1 represents Sunday
DAYOFYEAR	Returns an integer in the range of 1 to 366, where 1 represents January 1
DBCLOB	Returns a DBCLOB representation of its argument
DEGREES	Returns the number of degrees for an argument that is expressed in radians
DOUBLE or DOUBLE-PRECISION	Returns a double precision floating-point representation of its argument
EXP	Returns the exponential function of an argument
FLOOR	Returns the largest integer that is less than or equal to the argument
GRAPHIC	Returns a GRAPHIC representation of its argument
IFNULL	Returns the first argument in a set of two arguments that is not null
INSERT	Returns a string that is composed of an argument inserted into another argument at the same position where some number of bytes have been deleted
JULIAN_DAY	Returns an integer that represents the number of days from January 1, 4712 B.C.
LCASE or LOWER	Returns a string with the characters converted to lowercase
LEFT	Returns a string that consists of the specified number of leftmost bytes of a string
LOCATE	Returns the position at which the first occurrence of an argument starts within another argument
LOG or LN	Returns the natural logarithm of an argument
LOG10	Returns the base 10 logarithm of an argument
LTRIM	Returns the characters of a string with the leading blanks removed
MIDNIGHT_SECONDS	Returns an integer in the range of 0 to 86400 that represents the number of seconds between midnight and the argument
MOD	Returns the remainder of one argument divided by second argument

Table 35 (Page 3 of 3). New built-functions

Function name	Description
POSSTR	Returns the position of the first occurrence of an argument within another argument
POWER	Returns the value of one argument raised to the power of a second argument
QUARTER	Returns an integer in the range of 1 to 4 that represents the quarter of the year for the date specified in the argument
RADIANS	Returns the number of radians for an argument that is expressed in degrees
RAISE_ERROR	Raises an error in the SQLCA with the specified SQLSTATE and error description
RAND	Returns a double precision floating-point random number
REAL	Returns a single precision floating-point representation of its argument
REPEAT	Returns a character string composed of an argument repeated a specified number of times
REPLACE	Returns a string in which all occurrences of an argument within a second argument are replaced with a third argument
RIGHT	Returns a string that consists of the specified number of rightmost bytes of a string
ROUND	Returns a number rounded to the specified number of places to the right or left of the decimal place
ROWID	Returns a row ID representation of its argument
RTRIM	Returns the characters of an argument with the trailing blanks removed
SECOND	Returns the second part of its argument
SIGN	Returns the sign of an argument
SIN	Returns the sine of an argument that is expressed as an angle in radians
SINH	Returns the hyperbolic sine of an argument that is expressed as an angle in radians
SMALLINT	Returns a small integer representation of its argument
SPACE	Returns a string that consists of the number of blanks the argument specifies
SQRT	Returns the square root of its argument
SUBSTR	Returns a substring of a string
TAN	Returns the tangent of an argument that is expressed as an angle in radians
TANH	Returns the hyperbolic tangent of an argument that is expressed as an angle in radians
# TIMESTAMP_FORMAT #	Returns a timestamp for a character string, using a specified format to interpret the string
TRANSLATE	Returns a string with one or more characters translated
TRUNCATE	Returns a number truncated to the specified number of spaces to the right or left of the decimal point
UCASE or UPPER	Returns a string with the characters converted to uppercase
VARCHAR	Returns the varying-length character string representation of its argument
# VARCHAR_FORMAT #	Returns a varying-length character string representation of a timestamp, with the string in a specified format
WEEK	Returns an integer that represents the week of the year

Changed built-in functions

As shown in Table 36, many of the existing built-in functions are enhanced to support arguments with additional data types, such as the new large object (LOB) and row ID data types.

Table 36. Changes to existing built-functions

Function name	Newly supported argument types
Column functions	
COUNT	Argument values can have a LOB (CLOB, DBCLOB, and BLOB) or row ID data type
Scalar functions	
CHAR	The argument can be any character string that is 255 bytes or less (including a CLOB), an integer, a floating-point number, or a row ID value
COALESCE	The arguments can have a LOB or row ID data type
INT	The argument can be any character string that is 255 bytes or less (excluding a CLOB)
LENGTH	The arguments can have a LOB or row ID data type
MICROSECOND	The argument can be a valid character string representation of a timestamp
MINUTE	The argument can be a valid character string representation of a time or timestamp
MONTH	The argument can be a valid character string representation of a date or timestamp
NULLIF	The arguments can have a row ID type
SECOND	The argument can be a valid character string representation of a time or timestamp
SUBSTR	The argument from which the substring is derived can be a LOB data type (CLOB, DBCLOB, and BLOB)
TIME	The argument can be a character string representation of a time or timestamp
VARGRAPHIC	The argument can be an EBCDIC-encoded character string

Other SQL language changes

In addition to the many new SQL statements and built-in functions in Version 6, Table 37 on page 283 shows some of the other enhancements to the SQL language.

Table 37. Other changes to SQL language

Item	Description
Special registers	There are three new special registers: • CURRENT LOCALE LC_CTYPE specifies the LC_CTYPE locale that will be used to execute SQL statements that use a function that references a locale. Built-in functions LCASE, UCASE, and TRANSLATE (with a single argument) refer to the locale when they are executed. You can change the value of the register by executing the statement SET CURRENT LOCALE LC_CTYPE. • CURRENT OPTIMIZATION HINT specifies the user-defined optimization hint that DB2 should use to generate the access path for dynamic statements. (DB2 must be installed with optimization hints enabled for hints to be used.) You can change the value of the special register by executing the statement SET CURRENT OPTIMIZATION HINT. • CURRENT PATH specifies the SQL path that is used to resolve unqualified data type names (both built-in and distinct type) and function names in dynamically prepared SQL statements. It is also used to resolve unqualified procedure names that are specified as host variables in SQL CALL statements (CALL *host-variable*). You can change the value of the special register by executing the statement SET CURRENT PATH.
Expressions	The types of expression that can be specified is extended to support user-defined functions (scalar and sourced functions only) and CAST specifications. A CAST specification changes the data type of a value into another data type.
Predicates	The non-select form of the IN predicate is extended to support any type of expression in the second operand. The LIKE predicate is enhanced to support more types of expression for its arguments.
Queries	In a subselect, a table function can be used to specify the intermediate result table in a FROM clause. Also, when tables are joined to form an intermediate result table, the join condition for inner, left outer, and right outer joins can be any search condition that does not contain a subquery. The join condition for a full join continues to be a search condition in which the predicates can be combined only with AND. In a select statement, the new QUERYNO clause can be specified. The clause specifies the number to assign to the QUERYNO column of the plan table for information that is added to the plan table for the statement. If optimization hints are being given to DB2 for access path selection when the statement is executed, the clause also specifies the value of the QUERYNO column in the plan table to use for the hints.

Appendix D. Catalog changes

This appendix provides an overview of the changes to the catalog for Version 6 of DB2 for OS/390:

"New catalog tables"
"Changed catalog tables" on page 286
"New indexes" on page 288
"Revised indexes" on page 290

For a complete description of the columns of the new and changed tables, see *DB2 SQL Reference*.

New catalog tables

Table 38 shows the nine new catalog tables.

Table 38. New catalog tables

Catalog table name	Description
SYSIBM.SYSAUXRELS	Contains one row for each auxiliary table created for a LOB column. A base table space that is partitioned must have one auxiliary table for each partition of each LOB column
SYSIBM.SYSCONSTDEP	Records dependencies on check constraints or user-defined defaults for a column
SYSIBM.SYSDATATYPES	Contains one row for each distinct type defined to the system
SYSIBM.SYSLOBSTATS	Contains one row for each LOB table space
SYSIBM.SYSPARMS	Contains a row for each parameter of a routine or multiple rows for table parameters (one row for each column of the table)
SYSIBM.SYSROUTINEAUTH	Records the privileges that are held by users on routines (A routine can be a user-defined function, cast function, or stored procedure.)
SYSIBM.SYSROUTINES	Contains a row for every routine (A routine can be a user-defined function, cast function, or stored procedure.)
SYSIBM.SYSSCHEMAAUTH	Contains one or more rows for each user that is granted a privilege on a particular schema in the database
# SYSIBM.SYSSEQUENCEDEP	# Records the dependencies of identity columns on tables
# SYSIBM.SYSSEQUENCES	Contains one row for each identity column
SYSIBM.SYSTRIGGERS	Contains one row for each trigger

SYSLOBSTATS resides in existing table space DSNDB06.SYSTATS. The other
new tables are in new table spaces. SYSEQUENCES is in DSNDB06.SYSSEQ;
SYSEQUENCDEP is in DSNDB06.SYSSEQ2; and the remaining tables are in DSNDB06.SYSOBJ.

Changed catalog tables

Many existing catalog tables were changed in Version 6. Table 39 shows a list of the new columns that were added and the existing columns that were revised.

Table 39 (Page 1 of 3). Summary of new and revised catalog table columns

Catalog table name	New column	Revised column
SYSCOLDIST	-	COLVALUE FREQUENCYF
SYSCOLDISTSTATS	-	COLVALUE FREQUENCYF
SYSCOLSTATS	-	HIGHKEY HIGH2KEY LOWKEY LOW2KEY STATSTIME In addition, all columns can be inserted, updated and deleted
SYSCOLUMNS	COLSTATUS LENGTH2 DATATYPEID SOURCETYPEID TYPESCHEMA TYPENAME CREATEDTS	COLTYPE LENGTH HIGH2KEY LOW2KEY UPDATES DEFAULT STATSTIME DEFAULTVALUE COLCARDF
SYSCOPY	OTYPE LOWDSNUM HIGHDSNUM	TSNAME DSNUM ICTYPE DSNAME STYPE
SYSDATABASE	INDEXBP	BPOOL DBID ROSHARE TIMESTAMP TYPE ENCODING_SCHEME SBCS_CCSID DBCS_CCSID MIXED_CCSID
SYSINDEXES	COPY COPYLRSN CLUSTERRATIOF	UNIQUERULE CLUSTERED DSETPASS IBMREQD CLUSTERRATIO INDEXTYPE
SYSINDEXPART	SECQTYI IPREFIX ALTEREDTS	SQTY SPACE GBPCACHE FAROFFPOSF NEAROFFPOSF CARDF

Table 39 (Page 2 of 3). Summary of new and revised catalog table columns

Catalog table name	New column	Revised column
SYSINDEXSTATS	FIRSTKEYCARDF FULLKEYCARDF KEYCOUNTF CLUSTERRATIOF	All columns can be inserted, updated, and deleted
SYSPACKAGE	PATHSCHEMAS TYPE DBPROTOCOL FUNCTIONTS OPTHINT	COLLID HOSTLANG IBMREQD VERSION DYNAMICRULES
SYSPACKDEP	DOWNER DTYPE	BTYPE
SYSPACKSTMT	ACCESSPATH STMTNOI SECTNOI	STMTNO SECTNO
SYSPLAN	PATHSCHEMAS DBPROTOCOL FUNCTIONTS OPTHINT	IBMREQD
SYSPLANDEP	-	BTYPE
SYSRESAUTH	-	QUALIFIER NAME OBTYPE IBMREQD
SYSSTMT	ACCESSPATH STMTNOI SECTNOI	STMTNO SECTNO
SYSSTOGROUP	-	VPASSWORD
SYSTABAUTH	TRIGGERAUTH	IBMREQD
SYSTABLEPART	TRACKMOD EPOCH SECQTYI CARDF IPREFIX ALTEREDTS	SQTY CARD FARINDREF NEARINDREF PERCACTIVE PERCDROP SPACE GBPCACHE
SYSTABLES	TABLESTATUS	TYPE NPAGES PCTPAGES IBMREQD RECLENGTH STATUS PCTROWCOMP CARDF

Table 39 (Page 3 of 3). Summary of new and revised catalog table columns

Catalog table name	New column	Revised column
SYSTABLESPACE	LOG NACTIVEF DSSIZE	LOCKRULE DSETPASS IBMREQD LOCKMAX TYPE ENCODING_SCHEME SBCS_CCSID DBCS_CCSID MIXED_CCSID
SYSTABSTATS	CARDF	All columns can be inserted, updated, and deleted
SYSVIEWDEP	BSCHEMA	BNAME BCREATOR BTYPE
SYSVIEWS	PATHSCHEMAS	IBMREQD

The changes made to SYSPACKDEP and SYSTABAUTH were to extend their support for triggers. For example, SYSPACKDEP now also records the dependencies of packages on triggers, and SYSTABAUTH records the privileges that users hold on triggers.

New indexes

Table 40 shows the new indexes in Version 6.

Table 40 (Page 1 of 3). New indexes

Table space DSNDB06. ...	Catalog table SYSIBM. ...	Index		Key column
SYSDBASE	SYSCOLUMNS	DSNDCX02	Non-unique index	TYPESCHEMA TYPENAME
	SYSINDEXPART	DSNDRX02	Non-unique index	STORNAME
	SYSTABLEPART	DSNDPX02	Non-unique index	STORNAME
SYSOBJ	SYSAUXRELS	DSNOXX01	Non-unique clustering index	TBOWNER TBNAME
		DSNOXX02	Non-unique index	AUXTBOWNER AUXTBNAME
	SYSCONSTDEP	DSNCCX01	Non-unique clustering index	BSCHEMA BNAME BTYPE
		DSNCCX02	Non-unique index	DTBCREATOR DTBNAME
	SYSDATATYPES	DSNODX01	Primary unique clustering index	SCHEMA NAME
		DSNODX02	Unique index (descending)	DATATYPEID
	SYSPARMS	DSNOPX01	Unique clustering index	SCHEMA SPECIFICNAME ROUTINETYPE ROWTYPE ORDINAL

Table 40 (Page 2 of 3). New indexes

Table space DSNDB06. ...	Catalog table SYSIBM. ...	Index		Key column
		DSNOPX02	Non-unique index	TYPESCHEMA TYPENAME ROUTINETYPE CAST_FUNCTION OWNER SCHEMA SPECIFICNAME
		DSNOPX03	Non-unique index	TYPESCHEMA TYPENAME
	SYSROUTINEAUTH	DSNOAX01	Non-unique index	GRANTOR SCHEMA SPECIFICNAME ROUTINETYPE GRANTEETYPE EXECUTEAUTH
		DSNOAX02	Non-unique clustering index	GRANTEE SCHEMA SPECIFICNAME ROUTINETYPE GRANTEETYPE EXECUTEAUTH GRANTEDTS
		DSNOAX03	Non-unique index	SCHEMA SPECIFICNAME ROUTINETYPE
	SYSROUTINES	DSNOFX01	Unique clustering index	NAME PARM_COUNT PARM_SIGNATURE ROUTINETYPE SCHEMA PARM1 PARM2 PARM3 PARM4 PARM5 PARM6 PARM7 PARM8 PARM9 PARM10 PARM11 PARM12 PARM13 PARM14 PARM15 PARM16 PARM17 PARM18 PARM19 PARM20 PARM21 PARM22 PARM23 PARM24 PARM25 PARM26 PARM27 PARM28 PARM29 PARM30

Table 40 (Page 3 of 3). New indexes

Table space DSNDB06. ...	Catalog table SYSIBM. ...	Index		Key column
		DSNOFX02	Primary unique index	SCHEMA SPECIFICNAME ROUTINETYPE
		DSNOFX03	Non-unique index	NAME SCHEMA CAST_FUNCTION PARM_COUNT PARM_SIGNATURE PARM1
		DSNOFX04	Unique index (descending)	ROUTINE_ID
		DSNOFX05	Non-unique index	SOURCESCHEMA SOURCESPECIFIC ROUTINETYPE
		DSNOFX06	Non-unique index	SCHEMA NAME ROUTINETYPE PARM_COUNT
	SYSSCHEMAAUTH	DSNSKX01	Non-unique clustering index	GRANTEE SCHEMANAME
		DSNSKX02	Non-unique index	GRANTOR
	SYSTRIGGERS	DSNOTX01	Unique clustering index	SCHEMA NAME SEQNO
		DSNOTX02	Non-unique index	TBOWNER TBNAME
SYSPKAGE	SYSPACKDEP	DSNKDX03	Non-unique index	BQUALIFIER BNAME BTYPE DTYPE
# SYSSEQ #	SYSSEQUENCES	DSNSQX01	Unique index	SCHEMA NAME
# #		DSNSQX02	Unique index (descending)	SEQUENCEID
# SYSSEQ2 # #	SYSSEQUENCEDEP	DSNSRX01	Unique index	DCREATOR DNAME DCOLNAME
SYSSTATS	SYSLOBSTATS	DSNLNX01	Unique clustering index	DBNAME NAME
SYSVIEWS	SYSVIEWDEP	DSNGGX03	Non-unique index	BSCHEMA BNAME BTYPE

Revised indexes

Table 41 on page 291 shows the revised indexes in Version 6. Index DSNDXX02 is now a non-unique index. Columns were added to DSNATX02 and DSNAUH01.

Table 41. Revised indexes

Table space DSNDB06. ...	Catalog table SYSIBM. ...	Index		Key column
SYSDBASE	SYSINDEXES	DSNDXX02	Non-unique index	TBCREATOR TBNAME CREATOR NAME
	SYSTABAUTH	DSNATX02	Non-unique index	GRANTEE TCREATOR TTNAME GRANTEETYPE UPDATECOLS ALTERAUTH DELETEAUTH INDEXAUTH INSERTAUTH SELECTAUTH UPDATEAUTH CAPTUREAUTH REFERENCESAUTH REFCOLS TRIGGERAUTH
SYSUSER	SYSUSERAUTH	DSNAUH01	Non-unique index	GRANTEE GRANTEDTS

Appendix E. EXPLAIN table changes

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information, as defined in Appendix H, “Notices” on page 313.

This appendix includes the complete definitions for a DB2 PLAN_TABLE and a description of the PLAN_TABLE columns that are new for Version 6 of DB2 for OS/390.

Other tables that can be used with EXPLAIN are the DSN_FUNCTION_TABLE, which is useful for finding out information about function resolution and the DSN_STATEMNT_TABLE, which you can use to find out the estimated cost of your SQL statements. See “Using DSN_FUNCTION_TABLE to see how DB2 resolves a function” on page 197 for information about the new DSN_FUNCTION_TABLE. See “Creating a statement table” on page 80 information about the DSN_STATEMNT_TABLE.

Before you can use EXPLAIN, you must create a table called PLAN_TABLE to hold the results of EXPLAIN. If you have an existing PLAN_TABLE, you can alter it to add the new columns. The format of the PLAN_TABLE is shown in Figure 38. The content of each of the new or changed columns for Version 6 is shown in Table 42 on page 294.

Format of the Version 6 PLAN_TABLE

The Version 6 PLAN_TABLE has three additional columns, giving it a total of 49 columns.

```
QUERYNO          INTEGER       NOT NULL
QBLOCKNO         SMALLINT      NOT NULL
APPLNAME         CHAR(8)       NOT NULL
PROGNAME         CHAR(8)       NOT NULL
PLANNO           SMALLINT      NOT NULL
METHOD           SMALLINT      NOT NULL
CREATOR          CHAR(8)       NOT NULL
TNAME            CHAR(18)      NOT NULL
TABNO            SMALLINT      NOT NULL
ACCESSTYPE       CHAR(2)       NOT NULL
MATCHCOLS        SMALLINT      NOT NULL
ACCESSCREATOR    CHAR(8)       NOT NULL
ACCESSNAME       CHAR(18)      NOT NULL
INDEXONLY        CHAR(1)       NOT NULL
SORTN_UNIQ       CHAR(1)       NOT NULL
SORTN_JOIN       CHAR(1)       NOT NULL
SORTN_ORDERBY    CHAR(1)       NOT NULL
SORTN_GROUPBY    CHAR(1)       NOT NULL
SORTC_UNIQ       CHAR(1)       NOT NULL
SORTC_JOIN       CHAR(1)       NOT NULL
SORTC_ORDERBY    CHAR(1)       NOT NULL
SORTC_GROUPBY    CHAR(1)       NOT NULL
TSLOCKMODE       CHAR(3)       NOT NULL
TIMESTAMP        CHAR(16)      NOT NULL
REMARKS          VARCHAR(254)  NOT NULL
    -------25 column format --------
PREFETCH  CHAR(1)  NOT NULL WITH DEFAULT
COLUMN_FN_EVAL  CHAR(1)  NOT NULL WITH DEFAULT
MIXOPSEQ        SMALLINT NOT NULL WITH DEFAULT
    -------28 column format --------
VERSION         VARCHAR(64) NOT NULL WITH DEFAULT
COLLID          CHAR(18)  NOT NULL WITH DEFAULT
    -------30 column format --------
ACCESS_DEGREE     SMALLINT
ACCESS_PGROUP_ID  SMALLINT
JOIN_DEGREE       SMALLINT
JOIN_PGROUP_ID    SMALLINT
    -------34 column format --------
SORTC_PGROUP_ID   SMALLINT
SORTN_PGROUP_ID   SMALLINT
PARALLELISM_MODE  CHAR(1)
MERGE_JOIN_COLS   SMALLINT
CORRELATION_NAME  CHAR(18)
PAGE_RANGE        CHAR(1) NOT NULL WITH DEFAULT
JOIN_TYPE         CHAR(1) NOT NULL WITH DEFAULT
GROUP_MEMBER      CHAR(8) NOT NULL WITH DEFAULT
IBM_SERVICE_DATA  VARCHAR(254)  NOT NULL WITH DEFAULT
    ------43 column format --------
WHEN_OPTIMIZE     CHAR(1)  NOT NULL WITH DEFAULT
QBLOCK_TYPE       CHAR(6)  NOT NULL WITH DEFAULT
BIND_TIME         TIMESTAMP NOT NULL WITH DEFAULT
  ------46 column format -----------
OPTHINT            CHAR(8) NOT NULL WITH DEFAULT
HINT_USED           CHAR(8) NOT NULL WITH DEFAULT
PRIMARY_ACCESSTYPE CHAR(1) NOT NULL WITH DEFAULT
  -------49 column format-----------
```

Figure 38. Format of PLAN_TABLE

Descriptions of new and changed columns

You can now set a query number for non-EXPLAIN statements using the QUERYNO clause. This changes the description of the QUERYNO column. The new columns include OPTHINT, HINT_USED, and PRIMARY_ACCESSTYPE.

Table 42. Descriptions of new and changed columns in PLAN_TABLE

Column Name	Description
QUERYNO	A number intended to identify the statement being explained. For a row produced by an EXPLAIN statement, you can specify the number in the QUERYNO clause. For a row produced by non-EXPLAIN statements, you can specify the number using the QUERYNO clause, which is the optional part of the SELECT, INSERT, UPDATE, and DELETE statement syntax. Otherwise, DB2 assigns a number based on the line number of the SQL statement in the source program. FETCH statements do not each have an individual QUERYNO assigned to them. Instead, DB2 uses the QUERYNO of the DECLARE CURSOR statement for all corresponding FETCH statements for that cursor.
OPTHINT	A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.
HINT_USED	If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPTHINT) in this column.
PRIMARY_ACCESSTYPE	Indicates whether direct row access will be attempted first: **D** DB2 will try to use direct row access. If DB2 cannot use direct row access at runtime, it uses the access path described in the ACCESSTYPE column of PLAN_TABLE. **blank** DB2 will not try to use direct row access.

Appendix F. New and changed IFCIDs

The information in this appendix is Product-sensitive Programming Interface and Associated Guidance Information, as defined in Appendix H, “Notices” on page 313.

This appendix briefly describes the new IFCIDs and the changes to the existing IFCIDs for each new function. The new IFCIDs are described in Table 43; the changes to existing IFCIDs are described in Table 44. For a detailed description of the fields in each IFCID record, refer to the mapping macros data set library *prefix*.SDSNMACS.

New IFCIDs

Table 43 lists the new IFCIDs.

Table 43. New IFCIDs

IFCID	Trace	Class	Mapping macro	Description
Data set statistics for buffer pools				
0199	STATISTICS	8	DSNDQW02	Records data set information that is provided in the -DISPLAY BUFFERPOOL LSTATS command
Data sharing enhancements				
0329	ACCOUNTING	3	DSNDQW04	Records asynchronous wait times for IXLCACHE and IXLFCOMP requests
Triggers				
0325	PERFORMANCE	8	DSNDQW04	Records the start or end of trigger activation
User-defined functions				
0324	PERFORMANCE	3	DSNDQW04	Records function resolution information
Other enhancements				
0330	STATISTICS	3	DSNDQW04	Records an active log shortage condition

Changed IFCIDs

Table 44 gives an overview of changes to existing IFCIDs. Changes to IFCID 0106, the system parameters record, are not included.

Table 44 (Page 1 of 6). Changed IFCIDs

IFCID	Description of changes
Dynamic SQL enhancements	
0002, 0003, 0124, 0147, 0148	Add a field to record DESCRIBE INPUT statements
Data sharing enhancments	
0254	Add a field to record the number of cross invalidate (XI) signals due to explicit XI requests from DB2 for page sets that are define with GBPCACHE NONE or SYSTEM or group buffer pools that are defined with GBPCACHE(NO).

Table 44 (Page 2 of 6). Changed IFCIDs

IFCID	Description of changes
0256	Add fields to record: • The GBPCACHE value before an ALTER GROUPBUFFERPOOL command is issued • The GBPCACHE value after an ALTER GROUPBUFFERPOOL command is issued
# 0263	Add fields to record information for CFLEVEL 7.
DDF inactive connection support	
0001	Add fields to record: • Number of connections that DB2 terminates instead of making them type 1 inactive • Current and maximum number of type 2 inactive threads • Number of queued receive requests for type 2 inactive threads • Number of queued type 2 inactive threads • Current and maximum number of active database access thread slots that are not in use
DRDA support for three-part names	
0108, 0177	Add a value for bind parameter DBPROTOCOL
0112, 0113, 0177	Add a field to indicate the protocol that is used for sending three-part names
Group buffer pool duplexing	
0002	Add fields to record: • The number of group buffer pool rebuilds in which a member participated • For writes to the secondary group buffer pool, the number of writes of changed pages that failed • The number of completion checks for writes to the secondary group buffer pool that were suspended because the write had not completed • The number of IXLCACHE requests to the secondary group buffer pool
0003	Add fields to record: • The number of writes of changed pages to the secondary group buffer pool for duplexing • The number of completion checks for writes to the secondary group buffer pool that were suspended because the write had not completed
0230	Add fields to record: • Whether the group buffer pool is duplexed • If the group buffer pool is duplexed, the size, number of allocated directory entries, and number of allocated data entries for the secondary group buffer pool
0250	Add a value for a connect to secondary group buffer pool
0252	Add a value for connect to secondary group buffer pool
0254	Add fields to record these counters for the secondary group buffer pool: • Number of successful coupling facility write requests for changed pages • Number of coupling facility write requests that did not complete because of a lack of resources • Number of allocated directory entries • Number of allocated data entries • Number of allocated data entries in the changed state
0263	Add a field to record the number of delete name list requests for the secondary group buffer pool

Table 44 (Page 3 of 6). Changed IFCIDs

IFCID	Description of changes
0267, 0268	Add values for: • Rebuild started to establish group buffer pool duplexing • Group buffer pool duplexing stopped • Dynamic expand or contract initiated on secondary group buffer pool
Inline statistics	
0023, 0024, 0025	Add the RUNSTATS phase for REORG TABLESPACE, REORG INDEX, LOAD, and REBUILD INDEX
LOBs	
0002	Add fields to record: • Maximum storage that is used for LOB values • The number of CREATE AUXILIARY TABLE, HOLD LOCATOR, and FREE LOCATOR statements that are executed
0003	Add fields to record: • Maximum storage that is used for LOB values • Number of log records that are written • Number of bytes of log records that are written
0058	Add fields to record: • Additional pages that are scanned in a LOB table space • Number of LOB data pages that are updated by an INSERT or UPDATE statement
0020	Enhance existing fields to record: • Maximum number of page, row, and LOB locks held • Maximum number of page, row, or LOB locks held for the thread
0021, 0044, 0150, 0172, 0196	Add LOB locks
0023, 0024, 0025	Add the CHECKLOB phase for the CHECK LOB utility
0062	Add values for CREATE AUXILIARY TABLE, HOLD LOCATOR, and FREE LOCATOR
0141	Add values for a LOB table space and an auxiliary table
Optimization hints	
0022, 0108, 0112, 0113, 0177	Add a field to record the query optimization hints
Parallelism for COPY and RECOVER	
# 0023, 0024, 0025	# • The number of records of these types and the sequence of these records differ when COPY or RECOVER run with parallelism and when they run without parallelism
#	# • Use an existing field to record the number of subtasks that are starteds.
Parallel index build	
0023, 0024, 0025	Add values to record: • The phases of parallel index build • Whether a main task or a subtask performs the work
Predictive governing	

Table 44 (Page 4 of 6). Changed IFCIDs

IFCID	Description of changes
0022	Add fields to record: • The estimated processor cost, in milliseconds, for an SQL statement • The estimator processor cost, in service units, for an SQL statement • The reasons that the value of COST_CATEGORY is B • The cost estimate and cost category of an SQL statement
0112, 0113, 0177	Add a field to indicate the protocol used for sending three-part names
ROWIDs	
0002, 0003, 0148	Add fields to record: • The number of times that DB2 used direct row access • The number of times that DB2 tried to use direct row access but used an index instead • The number of times that DB2 tried to use direct row access but used a table space scan instead
0022	Add a value that indicates that a statement is a candidate for direct row access
Stored procedures enhancements	
0002, 0003, 0147, 0148	Add fields to record the number of CREATE PROCEDURE, ALTER PROCEDURE, and DROP PROCEDURE statements that are executed
Triggers	
0002, 0003, 0147, 0148	Add fields to record: • The number of times a statement trigger is activated • The number of times a row trigger is activated • The number of times an error occurs during trigger execution • Maximum level of nesting of user-defined functions, stored procedures, and triggers
0003, 0147, 0148	Add fields to record: • TCB time spent executing a trigger, or a user-defined function or stored procedure under a trigger • Elapsed time spent executing a trigger, or a user-defined function or stored procedure under a trigger
0016, 0017, 0018	Add: • Values for updating or scanning a transition table • A field to record the trigger level
0311	Add values for: • A temporary table that is a transition table • A cursor on a transition table
0124	Add values for: • CREATE TRIGGER • DROP TRIGGER • SET *transition variable* • SIGNAL SQLSTATE
0140	Add a value for a trigger authorization check
0148	Add a field to record the name and schema name of a trigger
User-defined functions	

Table 44 (Page 5 of 6). Changed IFCIDs

IFCID	Description of changes
0002, 0003	Add fields to record: • Authorization checks for user-defined functions • Maximum level of nesting of user-defined functions, stored procedures, and triggers • Number of user-defined functions that are executed • Number of times a user-defined function abnormally terminated or timed out • Number of times a user-defined function was rejected
0003	Add fields to record: • Accumulated TCB time to satisfy user-defined function requests • Accumulated TCB time that user-defined functions spend under triggers • Number of SQL entry and exit events for user-defined functions • Elapsed time spent in user-defined functions and by user-defined functions processing SQL • Time spent waiting for a user-defined function to be scheduled
0022	Add fields to record: • The length and contents of the CURRENT PATH special register • The text of a user-defined function invocation • The position, schema name, name, and specific name of a user-defined function
0108, 0177	Add values DEFINEBIND, INVOKEBIND, DEFINERUN, and INVOKERUN for bind option DYNAMICRULES
8-KB and 16-KB page sizes	
0002, 0003, 0006, 0008, 0010, 0127, 0201, 0226, 0251, 0259	Add values for 8-KB and 16 KB buffer pools
0002, 0003	Add a field to record the number of explicit cross-invalidations
16-TB table spaces	
0006, 0127, 0128, 0223, 0226, 0227, 0305	Add values for: • Table space with a 4-byte RID • Table space with a 5-byte RID that is not EA-enabled • Table space with a 5-byte RID that is EA-enabled
0018, 0058	Extend the size of fields for the number of rows that are processed, looked at, qualified, inserted, updated, or deleted
Miscellaneous changes	
0002, 0003, 0148	Add two counters to indicate the number of times that a parallel group's parallel plan was rebalanced.
0003, 0147, 0148	Make the following changes to fields for wait times: • Separate the wait for log write I/O field into the following fields: – Wait for log write I/O – Wait for database I/O • Separate the wait for service task field into the following fields: – Wait for Commit phase 2 or abort – Wait for open/close service task, which includes the wait for HSM recall – Wait for SYSLGRNG recording service task – Wait for data set extend/delete/define service task – Wait for other service tasks • Move service task wait fields to a new accounting section
0024	When a range of partitions is reorganized, an IFCID 0024 record is generated for each partition.

Table 44 (Page 6 of 6). Changed IFCIDs

IFCID	Description of changes
0202	Add fields for the VPTYPE and PGSTEAL buffer pool attributes.

Appendix G. Prerequisites of Version 6 of DB2 for OS/390

This chapter identifies prerequisites and optional programs of DB2 for OS/390 Version 6 and for the features that are delivered with it. See “Prerequisites of features of DB2 for OS/390 Version 6” on page 308 for information about the requirements for the features of DB2 for OS/390 Version 6. This chapter identifies the minimum level of hardware or software that is required; unless otherwise noted, subsequent versions and releases of these products are acceptable.

DB2 for OS/390 Version 6 prerequisites

DB2 for OS/390 Version 6 has requirements for processors, other programs, and virtual storage.

Hardware requirements

DB2 for OS/390 Version 6 operates on any processor that is supported by OS/390 Version 1 Release 3 or subsequent releases. The processor must have enough real storage to satisfy the combined requirements of DB2, OS/390, the appropriate Data Facility Product, the appropriate access methods, telecommunications, batch, and other customer-required applications.

OS/390 runs on the hardware listed below:

- All models of the S/390® Parallel Enterprise Servers or S/390 Parallel Transaction Servers (IBM 9672)
- All models of the IBM ES/9000® Processor Unit 9021, the 9121, or the 9221
- PC Server S/390 or RS/6000® with S/390 Server-on-Board
- S/390 Multiprise® 2000

The configuration must include sufficient I/O devices to support the requirements for system output, system residence, and system data sets. Sufficient direct access storage (DASD) must be available to satisfy the user's information storage requirements and can consist of any direct-access facility that is supported by the system configuration and the programming system.

In addition to listing auxiliary storage and data communications devices, this section identifies function-dependent hardware requirements and virtual storage requirements.

Auxiliary storage

DB2 is independent of both DASD and tape device type. You can use any magnetic, optical, or tape device that is supported by the data facilities component of DFSMS/MVS® for the DB2 data sets. See Table 45 on page 302 for a list of device types that are supported for DB2 data sets.

Table 45. Auxiliary storage

Data set type	Device type
Active recovery log data sets	DASD
Archive recovery log data sets	DASD, tape
Image copy data sets	DASD, tape
Bootstrap data set	DASD
User data sets	DASD
DB2 catalog data sets	DASD
Work data sets (for utilities)	DASD, tape

If these data sets are on DASD that is shared with other OS/390 systems, you should use global resource serialization to prevent concurrent access by more than one OS/390 system.

The minimum DASD space requirement, based on installing DB2 using the panel default values, is approximately 600 MB. Users need additional DASD space for their data.

If you use dual logging and tape for the log archiving device, you need at least two tape drives.

Data communication devices

Control DB2 operations from:

- The system console
- Authorized IMS/ESA Transaction Manager terminals
- Authorized CICS terminals
- TSO terminals (by authorized users)

For information about the data communication devices that are supported by IMS/ESA Transaction Manager, CICS, and TSO, see the documentation for these products.

Function-dependent hardware requirements

DB2 has the following function-dependent requirements.

COPY and RECOVER INDEX: Use of the COPY utility to take DFSMS™ concurrent copies of indexes requires a 9390, a 3990 Model 6, or a 3990 Model 3 with the Extended Platform installed or equivalent function.

Use of the RECOVER utility to restore DFSMS concurrent copies of indexes also requires a 9390, a 3990 Model 6, or a 3990 Model 3 3390 Model 3 or 3990 Model 6 controller at the with the Extended Platform installed or equivalent function.

DB2 table spaces that are larger than 1 terabyte: DB2 table spaces that are larger than 1 terabyte require one of the following products:

- 3990 Model 3 or Model 6 controllers with extended support
- 9340 DASD array

Program requirements and optional programs

This section lists licensed programs that are required in the DB2 for OS/390 environment. You can use subsequent versions or releases of these programs, unless the description for a given program states otherwise. Check the RETAIN® Preventative Service Planning (PSP) Facility for the most current information about APARs you must install to run DB2 for OS/390 and its optional features.

This section also identifies the requirements associated with specific DB2 capabilities, as well as optional programs that you can use with DB2 for OS/390 Version 6.

Operating system and support programs

For an OS/390 environment, DB2 requires the function that is provided by the following licensed programs or their equivalents; subsequent versions or releases of any product are acceptable.

- OS/390 Version 1 Release 3 Base Services (5645-001)
- OS/390 Version 1 Release 3 Application Enablement optional feature for DFSORT™

Function-dependent program requirements

DB2 for OS/390 has the following function-specific program requirements. For specific software requirements for the required products, refer to the respective product announcements.

User-defined functions: User-defined functions require that you include Language Environment mapping macros and macros that generate a Language Environment-conforming prolog and epilog. Language Environment is part of the OS/390 Version 1 Release 3 Application Enablement base feature.

See *OS/390 Language Environment for OS/390 & VM Programming Guide* for details. See "Programming languages" on page 307 for specific levels of Assembler, COBOL, C, C++, or PL/I.

SQL procedures: SQL procedures have the following requirements:

- Execution of SQL procedures requires Language Environment.

  Language Environment is part of the OS/390 Version 1 Release 3 Application Enablement base feature.
- Preparation of SQL procedures for execution using JCL requires the following compiler language levels, or higher:
  - IBM C/C++ for MVS/ESA Version 3 Release 2 (5655-121)
  - IBM SAA AD/Cycle C/370™ Version 1 Release 2 (5688-194)
- Preparation of SQL procedures for execution using the SQL procedure processor requires the following products:
  - The following compiler language levels, or higher:
    - IBM C/C++ for MVS/ESA Version 3 Release 2 (5655-121)
    - IBM SAA AD/Cycle C/370™ Version 1 Release 2 (5688-194)
  - IBM DATABASE 2 Universal Database Server for OS/390 REXX Language Support (5645-DB2)

Predictive governing: To take full advantage of DB2 for OS/390 Version 6 predictive governing capabilities, DB2 Connect workstation clients require DB2 Connect Version 5.2 (with appropriate service applied).

Language Environment built-in functions: Use of any of the following built-in functions requires OS/390 Version 2 Release 4 Application Enablement base element with APARs.

- Language Environment built-in functions, including the following advanced math functions:
  - DEGREES and RADIANS
  - RAND
  - EXP and POWER
  - LOG10, LOG, and LN
  - Trigonometry functions (ACOS, ASIN, ATAN, ATANH, ATAN2, COS, COSH, SIN, SINH, TAN, TANH)
- DB2 built-in functions UPPER, LOWER, and TRANSLATE with locale

Language Environment character conversion: Use of Language Environment character conversions requires OS/390 Version 2 Release 9.

More than 10000 open data sets***:*** Use of more than 10 000 open data sets requires OS/390 Version 2 Release 6.

DB2 table spaces that are larger than 1 terabyte: DB2 table spaces that are larger than 1 terabyte require one of the following products:

- VSAM Extended Addressability Linear Data Sets in OS/390 Version 2 Release 7 (5647-A01)
- DFSMS Version 1 Release 5 (5695-DF1)

Group buffer pool duplexing: Group buffer pool duplexing has the following requirements:

- A minimum coupling facility architectural level of CFLEVEL=5.
- OS/390 Version 2 Release 6, or OS/390 Release 3, 4, or 5 with APAR; OS/390 Version 2 Release 6 and above do not require the APAR.

Open Database Connectivity: Open Database Connectivity (ODBC) functions have the following requirements:

- Execution of ODBC component in the application address space requires OS/390 Version 1 Release 3 Application Enablement optional feature for C/C++.
- Customer applications are supported in the following compiler language levels, or higher:
  - IBM C/C++ for MVS/ESA Version 3 Release 2 (5655-121)
  - IBM SAA AD/Cycle C/370™ Version 1 Release 2 (5688-194)

Java Database Connectivity: Java Database Connectivity (JDBC) requires Java for OS/390 (5655-A46).

Java stored procedures: Java stored procedures require Enterprise ToolKit for OS/390, which is part of VisualAge for Java, Enterprise Edition for OS/390 (5655-JAV).

| ***DB2 Extenders:*** Use of the DB2 Extenders requires OS/390 Version 2 Release 4 (5647-801).

| To write extender applications, C or C++ for OS/390 is required. Use of DB2 Extenders does not require C or C++.

| In addition, use of the DB2 Text Extender requires the IBM Text Search Engine Version 2 Release 1 (FMID HIMN210) or higher. If you are running OS/390 Version 2 (Releases 4 through 7), download and install the IBM Text Search Engine in SMP/E format from the following Web site:

http://www.ibm.com/software/iminer/fortext/

| OS/390 Release 8 includes the IBM Text Search Engine as a base element of that operating system.

| ***Sysplex workload balancing:*** For Sysplex workload balancing, workstation clients require one of the following products:

- DB2 Connect Version 6 (strongly recommended)
- DB2 Connect Version 5 Release 2

| ***IEEE floating point:*** For DB2 for OS/390 to accept IEEE floating point (also called binary floating point) values in SQL statements or LOAD utility jobs, OS/390 Version 2 Release 6 is required.

31-digit decimal support in COBOL: DB2 COBOL application program use of 31-digit decimal variables requires IBM COBOL for OS/390 and VM Version 2 (5648-A25).

Optional programs

You can use the following optional licensed programs with DB2 for OS/390 Version 6. Unless otherwise specified, the release shown for a product and any subsequent release are acceptable. In some cases, earlier versions or releases of IBM licensed programs may also work with DB2 for OS/390, but IBM may not have tested them at the time this document was published. If you have questions, please check with your IBM representative.

Connectivity: In addition to any DRDA-compliant database management systems, DB2 for OS/390 Version 6 supports the following IBM relational database products:

- IBM DB2 Universal Database Extended Enterprise Edition Version 5 with the DB2 Connect component installed
- IBM DB2 Universal Database Enterprise Edition Version 5 with the DB2 Connect component installed
- IBM DB2 Universal Database for AS/400 Version 4 Release 2
- IBM Operating System/400® (OS/400®) Version 4 Release 1 with DB2 for AS/400 (5769-SS1)
- IBM DB2 Server for VM and VSE Version 5 (5648-158)
- DB2 DataJoiner Version 2 Release 1.1 (5231-200)

Net.Data for OS/390, a feature of DB2 for OS/390 Version 6, provides connectivity to DB2 from the Web.

Capacity planning: DB2 Estimator, a feature of DB2 for OS/390 Version 6, works with DB2 data to estimate application feasibility, to model application cost and performance, and to estimate required CPU and I/O capacity.

Transaction management: The following transaction management products work with DB2:

- Information Management System (IMS)
  - Information Management System/ESA (IMS/ESA) Version 6 (5655-158)
  - Information Management System/ESA (IMS/ESA) Version 5 (5695-176)
- Customer Information Control System (CICS)
  - CICS Transaction Server for OS/390 Release 1 (5655-147)
  - CICS/ESA Version 4 (5655-018)

Query support: The following query products work with DB2:

- Query Management Facility (QMF), a feature of DB2 for OS/390 Version 6
- QMF for Windows, a feature of DB2 for OS/390 Version 6
- QMF High Performance Option, a feature of DB2 for OS/390 Version 6
- The DB2 Extenders capability of DB2 for OS/390 Version 6

Application development: The following application development programs work with DB2:

- C/C++ Productivity Tools for OS/390 (5655-B85)
- IBM VisualAge® Generator Version 3 (see software announcement 297-395)
- IBM VisualAge for Java, Enterprise Edition for OS/390, Version 2 (5655-JAV)
- IBM VisualAge COBOL Version 2 (see software announcement 297-371)
- IBM VisualAge PL/I Version 2 (see software announcement 297-372)
- Application System (AS) Version 4 Release 2 (5648-092)

Operational support: The following programs provide operational support for DB2:

- IBM DATABASE 2 Performance Monitor (DB2 PM), an optional feature of DB2 for OS/390 Version 6
- OS/390 Version 1 Release 3 System Services optional feature for DFSMS features
- OS/390 Version 1 Release 3 Security Server optional feature for RACF
- NetView® Version 3 for MVS/ESA™ (5655-007), or NetView Version 2 Release 4 (5685-111)
- Tivoli™ Performance Reporter for OS/390 Version 1 Release 4 (569510100) or Performance Reporter for MVS Version 1 Release 2 (5695-101)
- Library Readers included on the CD-ROMs for BookManager® books

Replication support: The following programs provide replication support for DB2:

- DB2 DataPropagator, an optional feature of DB2 for OS/390 Version 6; or one or both of the following products:
  - DataPropagator Relational Apply for MVS Version 5 Release 1 (5655-A22)
  - DataPropagator Relational Capture for MVS Version 5 Release 1 (5655-A23)
- DataPropagator NonRelational MVS/ESA Version 2 (5696-705)

- IBM DataRefresher Version 1 (5696-703)

Database administration and systems management support: The following programs support database systems management for DB2:

- DB2 Automated Utility Generator (DB2AUG) Version 1 Release 2 (5695-077)
- DB2 Administration Tool, a feature of DB2 for OS/390 Version 6
- DB2 Buffer Pool Tool, a feature of DB2 for OS/390 Version 6
- DB2 Control Center, an element of the DB2 Management Tools Package
- DB2 Installer, an element of the DB2 Management Tools Package
- Visual Explain, an element of the DB2 Management Tools Package
- DB2 Estimator, an element of the DB2 Management Tools Package

The DB2 Management Tools Package is a feature of DB2 for OS/390 Version 6.

Programming languages: You can use the following programming languages (in addition to High-Level Assembler, which is part of OS/390) to develop application programs for DB2 for OS/390 Version 6:

C	IBM AD/Cycle® C/370 Compiler Version 1 Release 2 (5688-216); IBM C/370 Library Version 2 Release 2 (5688-188)
C++	OS/390 Version 1 Release 3 Application Enablement optional feature for C/C++
COBOL	IBM COBOL for MVS and VM Version 1 Release 2 (5688-197), or IBM COBOL for OS/390 and VM Version 2 Release 1 (5648-A25), or VS COBOL II Compiler and Library Version 1 Release 4 (5668-958)
Fortran	VS Fortran Compiler, Library, and Interactive Debugger Version 2 Release 6 (5668-806)
Java	Java for OS/390 (5655-A46)
PL/I	IBM PL/I for MVS and VM Version 1 Release 1 (5688-235), or OS PL/I Compiler, Library, and Interactive Test Facility Version 2 Release 3 (5668-909)
REXX	IBM TSO Extensions for MVS REXX, which is part of OS/390
Smalltalk	IBM VisualAge Smalltalk Version 4 Release 5 (5655-B14 or 5802-AAR)

Virtual storage requirements

The amount of space needed for the common service area (CSA) below the 16-MB line is less than 40 KB for each DB2 subsystem and 24 KB for each IRLM when APAR PQ12390 and prior service is applied. High concurrent activity, parallelism, or high contention can require more CSA.

Most of the DB2 common data resides in the extended common service area (ECSA). Most modules, control blocks, and buffers reside in the extended private area. A DB2 subsystem with 200 concurrent users and 2000 open data sets should need less than 2 MB of virtual storage below the 16-MB line.

Prerequisites of features of DB2 for OS/390 Version 6

DB2 for OS/390 Version 6 includes many features, some of which have requirements of their own, above and beyond what DB2 for OS/390 Version 6 requires. This section identifies the requirements for using these features with DB2 for OS/390 Version 6, but it does not repeat those DB2 for OS/390 requirements that apply to the features. Also, some of these features can be used with prior releases of DB2; these requirements are not included in this section, but rather in the detailed installation information for the specific feature.

You can use subsequent versions or releases of the products mentioned in this section, unless otherwise noted.

Recommendation: Before using these features, refer to the installation information for these features to ensure that you have all required and recommended products.

DB2 Installer requirements

DB2 Installer is an element of the DB2 Management Tools Package, which is a feature of DB2 for OS/390 Version 6. DB2 Installer has hardware and program requirements.

Hardware requirements

DB2 Installer requires:

- 20 MB disk memory on the target drive and 2 MB of disk space for each subsystem that is defined
- A monitor that is capable of displaying 1024-by-768 resolution

Program requirements

DB2 Installer can run in either of the following environments, each of which has its own requirements:

- Windows NT, which requires Microsoft™ Windows NT Version 4.0
- OS/2, which requires:
  - OS/2 Version 4
  - TCP/IP on OS/2, Version 3.0

Both environments require TCP/IP in any of the following circumstances:

- To run jobs from the workstation
- To use the copy-jobs-to-host function from the workstation
- To use all the functions of DB2 Installer

If you don't have TCP/IP, you can use DB2 Installer to customize your installation jobs, but you need to use a method outside of DB2 Installer to move jobs from the workstation to OS/390 for execution.

Visual Explain requirements

The Visual Explain element of the DB2 Management Tools Package has hardware and program requirements.

Hardware requirements

The Visual Explain element of the DB2 Management Tools Package requires:

- A workstation with either OS/2 Version 4 or Windows NT 4.0
- Approximately 12 MB of hard disk space
- A high-resolution monitor

Program requirements

One of the following products must be installed on the DB2 Visual Explain workstation:

- DB2 Connect Personal Edition Version 5.2
- DB2 Client Application Enablers™ (CAE) Version 5.2, connected to a server that is running DB2 Connect Enterprise Edition Version 5.2

In addition, DB2 Visual Explain requires one of the following communication protocols:

- TCP/IP
- SNA communications using a product such as Communication Server 5.0, SNA Server Version 4.0, or the integrated SNA support in DB2 Universal Database Personal Edition

Visual Explain includes a browser that lets users view current values of subsystem parameters. To use this browser, your DB2 subsystem must have:

- Stored procedures capability
- The DSNWZP stored procedure enabled

DB2 Estimator requirements

DB2 Estimator has hardware and program requirements.

Hardware requirements

DB2 Estimator, which is an element of the DB2 Management Tools Package, requires:

- A workstation with one of the operating systems listed under “Program requirements.”
- Approximately 12 MB of hard disk space

Program requirements

The DB2 Estimator operates in the following environments:

- Windows 3.1 and subsequent releases
- Windows 98
- Windows 95
- Windows NT 4

Net.Data requirements

Net.Data for OS/390 Version 2 Release 2 requires an HTTP Web server that is installed on the same server as Net.Data and DB2 for OS/390. The Web server can be another HTTP-compliant Web server or one of the following servers:

- IBM Internet Connection Secure Server for OS/390 Version 2 Release 2 (5697-B14)

- Domino Go Webserver Version 4 Release 6 Modification 1 for OS/390 (5697-C58)

In addition, Net.Data servlets have the following requirements:

- OS/390 Version 2 Release 5 (5647-A01)
- Domino Go Webserver Version 5 (5697-D43) with Servlet Express

Net.Data for OS/390 has no hardware requirements.

QMF requirements

QMF, which is a feature of DB2 for OS/390 Version 6, has hardware and program requirements.

Hardware requirements

The following QMF features have hardware dependencies:

- QMF for OS/390 requires a display station that is supported by GDDM®.
- QMF High Performance Option (HPO) requires a display station that is supported by ISPF.
- QMF for Windows requires a workstation that supports:
  - A Windows (16-bit or 32-bit) operating system
  - Network connectivity

Program requirements

The following QMF features have program dependencies.

- QMF for OS/390 requires Graphical Data Display Manager (GDDM) Version 2 Release 3 (5665-356). (This requirement is satisfied by OS/390 Version 2.)

  Use of QMF forms calculations requires a Windows 32-bit operating system and IBM Object REXX Interpreter Edition Version 1.0 (5639-B73).
- QMF for Windows requires:
  - A Windows (16-bit or 32-bit) operating system
  - Network communication software on each user machine, plus one or both of the following programs:
    - An SNA product that provides a CPI-C interface
    - A TCP/IP product that provides a WinSock Version 1.1 interface

  Use of QMF for Windows with English Wizard natural language query requires English Wizard Release 3.1, from Linguistic Technologies.

  The QMF for Windows Administrator module requires a Windows 32-bit operating system.

DB2 Performance Monitor requirements

DB2 Performance Monitor (DB2 PM), which is a feature of DB2 for OS/390 Version 6, has hardware and program requirements.

Hardware requirements

DB2 Performance Monitor has the following dependencies.

- For the host-based Online Monitor, a display station that is supported by Interactive System Productivity Facility (ISPF)
- For the host-based graphics facility, an IBM color graphics display station, or equivalent, that is supported by Graphical Data Display Manager (GDDM)

The DB2 PM Workstation-Based Online Monitor has the following dependencies:

- A high-resolution monitor
- A workstation that supports OS/2 Version 3 or Windows NT Version 4.0
- Approximately 20 MB of hard disk space

Program requirements

For the host-based Online Monitor and host-based graphics facility, DB2 Performance Monitor has no functional dependencies if you are monitoring DB2 for OS/390 Version 6. The DB2 Performance Monitor feature supports an environment of multiple DB2 releases, namely instrumentation, catalog, and PLAN_TABLE data from:

- DB2 UDB for OS/390 Version 6 (5645-DB2)
- DB2 for OS/390 Version 5 (5655-DB2)
- DB2 for MVS/ESA Version 4 (5695-DB2)

Refer to the software requirements for the specific version of DB2.

Workstation-Based Analysis and Tuning

The Workstation-Based Analysis and Tuning feature of DB2 for OS/390 Version 6 has hardware and program requirements.

Hardware requirements

Workstation-Based Analysis and Tuning has the following hardware requirements:

- A high-resolution monitor
- A workstation that supports OS/2 Version 3 or Windows NT Version 4.0

Program requirements

Workstation-Based Analysis and Tuning requires one of the following operating systems:

- OS/2, which requires both:
  - OS/2 Version 3
  - Personal Communications AS/400 and 3270 V4.1 (5622-972)
- Windows NT Version 4.0. Windows NT uses TCP/IP to communicate with the programs that run on OS/390, which requires TCP/IP for MVS Version 3 Release 2 (5655-HAL).

Appendix H. 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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 publication 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 as your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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 is for planning purposes only. The information herein is subject to change before the products described become available.

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.

Programming interface information

This book is intended to help you to use the new commands and options of Version 6 of DB2 for OS/390 and write programs that contain the new SQL statements and clauses of Version 6 of DB2 for OS/390. This book primarily documents General-use Programming Interface and Associated Guidance Information provided by IBM DATABASE 2 Universal Database Server for OS/390 (DB2 for OS/390).

General-use programming interfaces allow the customer to write programs that obtain the services of DB2 for OS/390.

However, this book also documents Product-sensitive Programming Interface and Associated Guidance Information.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces might need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, by the following marking:

```
┌──────────────── Product-sensitive Programming Interface ────────────────┐

Product-sensitive Programming Interface and Associated Guidance Information ...

└──────────────── End of Product-sensitive Programming Interface ──────────┘
```

Trademarks

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

AD/Cycle
AIX
APL2
AS/400
BookManager
CICS
CICS/ESA
CICS/MVS
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Client Application Enablers
DB2 Connect
DB2 Universal Database
DFSMS
DFSMSdfp
DFSMShsm
DFSMS/MVS
DFSORT
Distributed Relational Database Architecture
DRDA
ES/3090
ES/9000
GDDM
IBM
IBMLink
IMS
IMS/ESA
Language Environment
Multiprise
MVS/ESA
Net.Data
Operating System/400
OS/390
OS/400
OS/2
Parallel Sysplex
QMF
RACF
RETAIN
RMF
RS/6000
System/390
S/390
VisualAge
VTAM

Tivoli™ and NetView are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Lotus®, Domino, and Go Webserver are trademarks of Lotus Development Corporation in the United States, other countries, or both.

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

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

UNIX® is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

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

Glossary

The following terms and abbreviations are defined as they are used in the DB2 library. If you do not find the term you are looking for, refer to the index or to *IBM Dictionary of Computing*.

A

after trigger. A trigger that is defined with the trigger activation time AFTER.

auxiliary index. An index on an auxiliary table in which each index entry refers to a LOB.

auxiliary table. A table that stores columns outside the table in which they are defined. Contrast with *base table*.

B

base table. (1) A table that is created by the SQL CREATE TABLE statement and that holds persistent data. Contrast with *result table* and *temporary table*.

(2) A table containing a LOB column definition. The actual LOB column data is not stored with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB columns. Contrast with *auxiliary table*.

base table space. A table space that contains base tables.

before trigger. A trigger that is defined with the trigger activation time BEFORE.

binary large object (BLOB). A sequence of bytes, where the size of the value ranges from 0 bytes to 2 GB - 1. Such a string does not have an associated CCSID.

binary string. A sequence of bytes that is not associated with a CCSID. For example, the BLOB data type is a binary string.

BLOB. Binary large object.

built-in function. A function that DB2 supplies. Contrast with *user-defined function*.

C

cast function. A function that is used to convert instances of a (source) data type into instances of a different (target) data type. In general, a cast function has the name of the target data type. It has one single argument whose type is the source data type; its return type is the target data type.

character large object (CLOB). A sequence of bytes representing single-byte characters or a mixture of single- and double-byte characters where the size of the value can be up to 2 GB - 1. In general, character large object values are used whenever a character string might exceed the limits of the VARCHAR type.

CLOB. Character large object.

column function. An SQL operation that derives its result from a collection of values across one or more rows. Contrast with *scalar function*.

cost category. A category into which DB2 places cost estimates for SQL statements at the time the statement is bound. A cost estimate can be placed in either of the following cost categories:

- A: Indicates that DB2 had enough information to make a cost estimate without using default values.
- B: Indicates that some condition exists for which DB2 was forced to use default values for its estimate.

The cost category is externalized in the COST_CATEGORY column of DSN_STATEMNT_TABLE when a statement is explained.

created temporary table. A table that holds temporary data and is defined with the SQL statement CREATE GLOBAL TEMPORARY TABLE. Information about created temporary tables is stored in the DB2 catalog, so this kind of table is persistent and can be shared across application processes. Contrast with *declared temporary table*. See also *temporary table*.

D

data space. A range of up to 2 GB of contiguous virtual storage addresses that a program can directly manipulate. Unlike an address space, a data space can hold only data; it does not contain common areas, system data, or programs.

DBCLOB. Double-byte character large object.

declared temporary table. A table that holds temporary data and is defined with the SQL statement DECLARE GLOBAL TEMPORARY TABLE. Information about declared temporary tables is not stored in the DB2 catalog, so this kind of table is not persistent and can only be used by the application process that issued the DECLARE statement. Contrast with *created temporary table*. See also *temporary table*.

delete trigger. A trigger that is defined with the triggering SQL operation DELETE.

deterministic function. A user-defined function whose result is dependent on the values of the input arguments. That is, successive invocations with the same input values produce the same answer. Sometimes referred to as a *not-variant* function. Contrast this with an *not-deterministic function* (sometimes called a *variant function*), which might not always produce the same result for the same inputs.

distinct type. A user-defined data type that is internally represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

double-byte character large object (DBCLOB). A sequence of bytes representing double-byte characters where the size of the values can be up to 2 GB. In general, double-byte character large object values are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

E

EA-enabled table space. A table space or index space that is enabled for extended addressability and that contains individual partitions (or pieces, for LOB table spaces) that are greater than 4 GB.

external function. A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with *sourced function* and *built-in function*.

F

function. A specific purpose of an entity or its characteristic action such as a column function or scalar function. (See also *column function* and *scalar function*.)

Functions can be user-defined, built-in, or generated by DB2. (See *built-in function*, *cast function*, *external function*, *sourced function*, and *user-defined function*.)

function definer. The authorization ID of the owner of the schema of the function that is specified in the CREATE FUNCTION statement.

function implementer. The authorization ID of the owner of the function program and function package.

function package. A package that results from binding the DBRM for a function program.

function package owner. The authorization ID of the user who binds the function program's DBRM into a function package.

function resolution. The process, internal to the DBMS, by which a function invocation is bound to a particular function instance. This process uses the function name, the data types of the arguments, and a list of the applicable schema names (called the *SQL path*) to make the selection. This process is sometimes called *function selection*.

function selection. See *function resolution*.

function signature. The logical concatenation of a fully qualified function name with the data types of all of its parameters.

G

group buffer pool duplexing. The ability to write data to two instances of a group buffer pool structure: a *primary group buffer pool* and a *secondary group buffer pool*. OS/390 publications refer to these instances as the 'old' (for primary) and 'new' (for secondary) structures.

I

identity column. A column that provides a way for DB2 to automatically generate a guaranteed-unique numeric value for each row that is inserted into the table. Identity columns are defined with the AS IDENTITY clause. A table can have no more than one identity column.

indicator column. A 4-byte value that is stored in a base table in place of a LOB column.

inheritance. The passing of class resources or attributes from a parent class downstream in the class hierarchy to a child class.

inoperative package. A package that cannot be used because one or more user-defined functions that the package depends on were dropped. Such a package must be explicitly rebound. Contrast with *invalid package.*

insert trigger. A trigger that is defined with the triggering SQL operation INSERT.

invalid package. A package that depends on an object (other than a user-defined function) that is dropped. Such a package is implicitly rebound on invocation. Contrast with *inoperative package.*

invariant character set. (1) A character set, such as the syntactic character set, whose code point assignments do not change from code page to code page. (2) A minimum set of characters that is available as part of all character sets.

L

large object (LOB). A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB - 1 byte in length. See also *BLOB*, *CLOB*, and *DBCLOB*.

LOB. Large object.

LOB locator. A mechanism that allows an application program to manipulate a large object value in the database system. A LOB locator is a fullword integer value that represents a single LOB value. An application program retrieves a LOB locator into a host variable and can then apply SQL operations to the associated LOB value using the locator.

LOB lock. A lock on a LOB value.

LOB table space. A table space that contains all the data for a particular LOB column in the related base table.

locale. The definition of a subset of a user's environment that combines characters that are defined for a specific language and country, and a CCSID.

M

materialize. (1) The process of putting rows from a view or nested table expression into a work file for additional processing by a query.

(2) The placement of a LOB value into contiguous storage. Because LOB values can be very large, DB2 avoids materializing LOB data until doing so becomes absolutely necessary.

N

not-deterministic function. A user-defined function whose result is not solely dependent on the values of the input arguments. That is, successive invocations with the same argument values can produce a different answer. this type of function is sometimes called a *variant* function. Contrast this with a *deterministic function* (sometimes called a *not-variant function*), which always produces the same result for the same inputs.

not-variant function. See *deterministic function*.

O

overloaded function. A function name for which multiple function instances exist.

P

path. See *SQL path*.

postponed abort UR. A unit of recovery that was inflight or in-abort, was interrupted by system failure or cancellation, and did not complete backout during restart.

primary group buffer pool. For a duplexed group buffer pool, the structure used to maintain the coherency of cached data. This structure is used for page registration and cross-invalidation. The OS/390 equivalent is *old* structure. Compare with *secondary group buffer pool*.

R

REORG pending (REORP). A condition that restricts SQL access and most utility access to an object that must be reorganized.

REORP. REORG pending.

restart pending (RESTP). A restrictive state of a page set or partition that indicates that restart (backout) work needs to be performed on the object. All access to the page set or partition is denied except for access by the:

- RECOVER POSTPONED command
- Automatic online backout (which DB2 invokes after restart if the system parameter LBACKOUT=AUTO)

RESTP. Restart pending.

result table. The set of rows that are specified by a SELECT statement.

ROWID. Row identifier.

row identifier (ROWID). A value that uniquely identifies a row. This value is stored with the row and never changes.

row trigger. A trigger that is defined with the trigger granularity FOR EACH ROW.

S

savepoint. A named entity that represents the state of data and schemas at a particular point in time within an agent's transaction. Special SQL statements exist to create savepoints, destroy savepoints, and restore data and schemas to the states that the savepoints represent. The restoration of data and schemas to a savepoint is usually referred to as rolling back to a savepoint.

scalar function. An SQL operation that produces a single value from another value and is expressed as a function name, followed by a list of arguments that are enclosed in parentheses. Contrast with *column function*.

schema. A logical grouping for user-defined functions, distinct types, triggers, and stored procedures. When an object of one of these types is created, it is assigned to one schema, which is determined by the name of the object. For example, the following statement creates a distinct type T in schema C:

```
CREATE DISTINCT TYPE C.T ...
```

secondary group buffer pool. For a duplexed group buffer pool, the structure that is used to back up changed pages that are written to the primary group buffer pool. No page registration or cross-invalidation occurs using the secondary group buffer pool. The OS/390 equivalent is *new* structure.

sourced function. A function that is implemented by another built-in or user-defined function that is already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with *external function* and *built-in function*.

source type. An existing type that is used to internally represent a distinct type.

specific function name. A particular user-defined function that is known to the database manager by its specific name. Many specific user-defined functions can have the same function name. When a user-defined function is defined to the database, every function is assigned a specific name that is unique within its schema. Either the user can provide this name, or a default name is used.

SQL path. An ordered list of schema names that are used in the resolution of unqualified references to user-defined functions, distinct types, and stored procedures. In dynamic SQL, the current path is found in the CURRENT PATH special register. In static SQL, it is defined in the PATH bind option.

statement trigger. A trigger that is defined with the trigger granularity FOR EACH STATEMENT.

strong typing. A process that guarantees that only user-defined functions and operations that are defined on a distinct type can be applied to that type. For example, you cannot directly compare two currency types, such as Canadian dollars and US dollars. But you can provide a user-defined function to convert one currency to the other and then do the comparison.

syntactic character set. A set of 81 graphic characters that are registered in the IBM registry as character set 00640. This set was originally recommended to the programming language community to be used for syntactic purposes toward maximizing portability and interchangeability across systems and country boundaries. It is contained in most of the primary registered character sets, with a few exceptions. See also *invariant character set*.

T

table function. A function that receives a set of arguments and returns a table to the SQL statement that references the function. A table function can only be referenced in the FROM clause of a subselect.

table locator. A mechanism that allows access to trigger transition tables in the FROM clause of SELECT statements, the subselect of INSERT statements, or from within user-defined functions. A table locator is a fullword integer value that represents a transition table.

table space set. A set of table spaces and partitions that should be recovered together for one of these reasons:

- Each of them contains a table that is a parent or descendent of a table in one of the others.
- The set contains a base table and associated auxiliary tables.

A table space set can contain both types of relationships.

temporary table. A table that holds temporary data; for example, temporary tables are useful for holding or sorting intermediate results from queries that contain a large number of rows. The two kinds of temporary table, which are created by different SQL statements, are the created temporary table and the declared temporary table. Contrast with *result table*. See also *created temporary table* and *declared temporary table*.

transition table. A temporary table that contains all the affected rows of the triggering table in their state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the table of changed rows in the old state or the new state.

transition variable. A variable that contains a column value of the affected row of the triggering table in its state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the set of old values or the set of new values.

trigger. A set of SQL statements that are stored in a DB2 database and executed when a certain event occurs in a DB2 table.

trigger activation. The process that occurs when the trigger event that is defined in a trigger definition is executed. Trigger activation consists of the evaluation of the triggered action condition and conditional execution of the triggered SQL statements.

trigger activation time. An indication in the trigger definition of whether the trigger should be activated before or after the triggered event.

trigger body. The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true.

trigger cascading. The process that occurs when the triggered action of a trigger causes the activation of another trigger.

triggered action. The SQL logic that is performed when a trigger is activated. The triggered action consists of an optional triggered action condition and a set of triggered SQL statements that are executed only if the condition evaluates to true.

triggered action condition. An optional part of the triggered action. This Boolean condition appears as a WHEN clause and specifies a condition that DB2 evaluates to determine if the triggered SQL statements should be executed.

triggered SQL statements. The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. Triggered SQL statements are also called the *trigger body*.

trigger granularity. A characteristic of a trigger, which determines whether the trigger is activated:

Only once for the triggering SQL statement
Once for each row that the SQL statement modifies

trigger package. A package that is created when a CREATE TRIGGER statement is executed. The package is executed when the trigger is activated.

triggering event. The specified operation in a trigger definition that causes the activation of that trigger. The triggering event is comprised of a triggering operation (INSERT, UPDATE, or DELETE) and a triggering table on which the operation is performed.

triggering SQL operation. The SQL operation that causes a trigger to be activated when performed on the triggering table.

triggering table. The table for which a trigger is created. When the defined triggering event occurs on this table, the trigger is activated.

typed parameter marker. A parameter marker that is specified along with its target data type. It has the general form:

```
CAST(? AS data-type)
```

U

UDF. User-defined function.

UDT. User-defined data type. In DB2 for OS/390, the term *distinct type* is used instead of user-defined function.

untyped parameter marker. A parameter marker that is specified without its target data type. It has the form of a single question mark (?).

update trigger. A trigger that is defined with the triggering SQL operation UDPATE.

user-defined data type (UDT). See *distinct type*.

user-defined function (UDF). A function that is defined to DB2 using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be either an *external function* or a *sourced function*. Contrast with *built-in function*.

V

variant function. See *not-deterministic function*.

Bibliography

DB2 Universal Database Server for OS/390 Version 6 Product Libraries:

DB2 Universal Database for OS/390

- *DB2 Administration Guide, SC26-9003*
- *DB2 Application Programming and SQL Guide, SC26-9004*
- *DB2 Application Programming Guide and Reference for Java™, SC26-9018*
- *DB2 ODBC Guide and Reference, SC26-9005*
- *DB2 Command Reference, SC26-9006*
- *DB2 Data Sharing: Planning and Administration, SC26-9007*
- *DB2 Data Sharing Quick Reference Card, SX26-3843*
- *DB2 Diagnosis Guide and Reference, LY36-3736*
- *DB2 Diagnostic Quick Reference Card, LY36-3737*
- *DB2 Image, Audio, and Video Extenders Administration and Programming, SC26-9650*
- *DB2 Installation Guide, GC26-9008*
- *DB2 Licensed Program Specifications, GC26-9009*
- *DB2 Messages and Codes, GC26-9011*
- *DB2 Master Index, SC26-9010*
- *DB2 Reference for Remote DRDA Requesters and Servers, SC26-9012*
- *DB2 Reference Summary, SX26-3844*
- *DB2 Release Planning Guide, SC26-9013*
- *DB2 SQL Reference, SC26-9014*
- *DB2 Text Extender Administration and Programming, SC26-9651*
- *DB2 Utility Guide and Reference, SC26-9015*
- *DB2 What's New? GC26-9017*
- *DB2 Program Directory, GI10-8182*

DB2 Administration Tool

- *DB2 Administration Tool for OS/390 User's Guide, SC26-9847*

DB2 Buffer Pool Tool

- *DB2 Buffer Pool Tool for OS/390 User's Guide and Reference, SC26-9306*

DB2 DataPropagator

- *DB2 Replication Guide and Reference, SC26-9642*

Net.Data for OS/390

The following books are available at
http://www.ibm.com/software/net.data/library.html:

- *Net.Data Library: Administration and Programming Guide for OS/390*
- *Net.Data Library: Language Environment Interface Reference*
- *Net.Data Library: Messages and Codes*
- *Net.Data Library: Reference*

DB2 PM for OS/390

- *DB2 PM for OS/390 Batch User's Guide, SC26-9167*
- *DB2 PM for OS/390 Command Reference, SC26-9166*
- *DB2 PM for OS/390 General Information, GC26-9172*
- *DB2 PM for OS/390 Installation and Customization, SC26-9171*
- *DB2 PM for OS/390 Messages, SC26-9169*
- *DB2 PM for OS/390 Online Monitor User's Guide, SC26-9168*
- *DB2 PM for OS/390 Report Reference Volume 1, SC26-9164*
- *DB2 PM for OS/390 Report Reference Volume 2, SC26-9165*
- *DB2 PM for OS/390 Using the Workstation Online Monitor, SC26-9170*
- *DB2 PM for OS/390 Program Directory, GI10-8183*

Query Management Facility

- *Query Management Facility: Developing QMF Applications, SC26-9579*
- *Query Management Facility: Getting Started with QMF on Windows, SC26-9582*
- *Query Management Facility: High Peformance Option User's Guide for OS/390, SC26-9581*
- *Query Management Facility: Installing and Managing QMF on OS/390, GC26-9575*
- *Query Management Facility: Installing and Managing QMF on Windows, GC26-9583*
- *Query Management Facility: Introducing QMF, GC26-9576*
- *Query Management Facility: Messages and Codes, GC26-9580*
- *Query Management Facility: Reference, SC26-9577*
- *Query Management Facility: Using QMF, SC26-9578*

Ada/370

- *IBM Ada/370 Language Reference, SC09-1297*
- *IBM Ada/370 Programmer's Guide, SC09-1414*
- *IBM Ada/370 SQL Module Processor for DB2 Database Manager User's Guide, SC09-1450*

APL2

- *APL2 Programming Guide, SH21-1072*
- *APL2 Programming: Language Reference, SH21-1061*
- *APL2 Programming: Using Structured Query Language (SQL), SH21-1057*

AS/400

- *DB2 for OS/400 SQL Programming, SC41-4611*
- *DB2 for OS/400 SQL Reference, SC41-4612*

BASIC

- *IBM BASIC/MVS Language Reference, GC26-4026*
- *IBM BASIC/MVS Programming Guide, SC26-4027*

BookManager READ/MVS

- *BookManager READ/MVS V1R3: Installation Planning & Customization, SC38-2035*

C/370

- *IBM SAA AD/Cycle C/370 Programming Guide, SC09-1841*
- *IBM SAA AD/Cycle C/370 Programming Guide for Language Environment/370, SC09-1840*
- *IBM SAA AD/Cycle C/370 User's Guide, SC09-1763*
- *SAA CPI C Reference, SC09-1308*

Character Data Representation Architecture

- *Character Data Representation Architecture Overview, GC09-2207*
- *Character Data Representation Architecture Reference and Registry, SC09-2190*

CICS/ESA

- *CICS/ESA Application Programming Guide, SC33-1169*
- *CICS for MVS/ESA Application Programming Reference, SC33-1170*
- *CICS for MVS/ESA CICS-RACF Security Guide, SC33-1185*
- *CICS for MVS/ESA CICS-Supplied Transactions, SC33-1168*
- *CICS for MVS/ESA Customization Guide, SC33-1165*
- *CICS for MVS/ESA Data Areas, LY33-6083*
- *CICS for MVS/ESA Installation Guide, SC33-1163*
- *CICS for MVS/ESA Intercommunication Guide, SC33-1181*
- *CICS for MVS/ESA Messages and Codes, GC33-1177*
- *CICS for MVS/ESA Operations and Utilities Guide, SC33-1167*
- *CICS/ESA Performance Guide, SC33-1183*
- *CICS/ESA Problem Determination Guide, SC33-1176*
- *CICS for MVS/ESA Resource Definition Guide, SC33-1166*
- *CICS for MVS/ESA System Definition Guide, SC33-1164*
- *CICS for MVS/ESA System Programming Reference, GC33-1171*

CICS/MVS

- *CICS/MVS Application Programmer's Reference, SC33-0512*
- *CICS/MVS Facilities and Planning Guide, SC33-0504*
- *CICS/MVS Installation Guide, SC33-0506*
- *CICS/MVS Operations Guide, SC33-0510*
- *CICS/MVS Problem Determination Guide, SC33-0516*
- *CICS/MVS Resource Definition (Macro), SC33-0509*
- *CICS/MVS Resource Definition (Online), SC33-0508*

IBM C/C++ for MVS/ESA

- *IBM C/C++ for MVS/ESA Library Reference, SC09-1995*
- *IBM C/C++ for MVS/ESA Programming Guide, SC09-1994*

IBM COBOL

- *IBM COBOL Language Reference, SC26-4769*
- *IBM COBOL for MVS & VM Programming Guide, SC26-4767*

Conversion Guide

- *IMS-DB and DB2 Migration and Coexistence Guide, GH21-1083*

Cooperative Development Environment

- *CoOperative Development Environment/370: Debug Tool, SC09-1623*

Data Extract (DXT)

- *Data Extract Version 2: General Information, GC26-4666*
- *Data Extract Version 2: Planning and Administration Guide, SC26-4631*

DataPropagator NonRelational

- *DataPropagator NonRelational MVS/ESA Administration Guide, SH19-5036*
- *DataPropagator NonRelational MVS/ESA Reference, SH19-5039*

Data Facility Data Set Services

- *Data Facility Data Set Services: User's Guide and Reference, SC26-4388*

Database Design

- *DB2 Design and Development Guide, Gabrielle Wiorkowski and David Kull, Addison Wesley, ISBN 0-20158-049-8*
- *Handbook of Relational Database Design, C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8*

DataHub

- *IBM DataHub General Information, GC26-4874*

DB2 Connect

- *DB2 Connect Enterprise Edition for OS/2 and Windows NT: Quick Beginnings, GC09-2828*
- *DB2 Connect Personal Edition Quick Beginnings, GC09-2830*
- *DB2 Connect User's Guide, SC09-2838*

DB2 Server for VSE & VM

- *DB2 Server for VM: DBS Utility, SC09-2394*
- *DB2 Server for VSE: DBS Utility, SC09-2395*

DB2 Universal Database (UDB)

- *DB2 UDB Administration Guide Volume 1: Design and Implementation, SC09-2839*
- *DB2 UDB Administration Guide Volume 2: Performance, SC09-2840*
- *DB2 UDB Administrative API Reference, SC09-2841*
- *DB2 UDB Application Building Guide, SC09-2842*
- *DB2 UDB Application Development Guide, SC09-2845*
- *DB2 UDB Call Level Interface Guide and Reference, SC09-2843*
- *DB2 UDB SQL Getting Started, SC09-2856*
- *DB2 UDB SQL Reference Volume 1, SC09-2847*
- *DB2 UDB SQL Reference Volume 2, SC09-2848*

Device Support Facilities

- *Device Support Facilities User's Guide and Reference, GC35-0033*

DFSMS/MVS

- *DFSMS/MVS: Access Method Services for the Integrated Catalog, SC26-4906*
- *DFSMS/MVS: Access Method Services for VSAM Catalogs, SC26-4905*
- *DFSMS/MVS: Administration Reference for DFSMSdss, SC26-4929*
- *DFSMS/MVS: DFSMShsm Managing Your Own Data, SH21-1077*
- *DFSMS/MVS: Diagnosis Reference for DFSMSdfp, LY27-9606*
- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26–3123*
- *DFSMS/MVS: Macro Instructions for Data Sets, SC26-4913*
- *DFSMS/MVS: Managing Catalogs, SC26-4914*
- *DFSMS/MVS: Program Management, SC26-4916*
- *DFSMS/MVS: Storage Administration Reference for DFSMSdfp, SC26-4920*
- *DFSMS/MVS: Using Advanced Services, SC26-4921*
- *DFSMS/MVS: Utilities, SC26-4926*
- *MVS/DFP: Using Data Sets, SC26-4749*

DFSORT

- *DFSORT Application Programming: Guide, SC33-4035*

Distributed Relational Database

- *Data Stream and OPA Reference, SC31-6806*
- *IBM SQL Reference, SC26-8416*
- *Open Group Technical Standard (the Open Group presently makes the following books available through its Web site at http://www.opengroup.org):*
  - *DRDA Volume 1: Distributed Relational Database Architecture (DRDA), ISBN 1-85912-295-7*
  - *DRDA Version 2 Volume 2: Formatted Data Object Content Architecture, available only on Web*
  - *DRDA Volume 3: Distributed Database Management (DDM) Architecture, ISBN 1-85912-206-X*

Domain Name System

- *DNS and BIND, Third Edition, Paul Albitz and Cricket Liu, O'Reilly, SR23-8771*

Education

- *IBM Dictionary of Computing, McGraw-Hill, ISBN 0-07031-489-6*
- *1999 IBM All-in-One Education and Training Catalog, GR23-8105*

Enterprise System/9000 and Enterprise System/3090

- *Enterprise System/9000 and Enterprise System/3090 Processor Resource/System Manager Planning Guide, GA22-7123*

High Level Assembler

- *High Level Assembler for MVS and VM and VSE Language Reference, SC26-4940*
- *High Level Assembler for MVS and VM and VSE Programmer's Guide, SC26-4941*

Parallel Sysplex Library

- *OS/390 Parallel Sysplex Application Migration, GC28-1863*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1862*
- *OS/390 Parallel Sysplex Overview: An Introduction to Data Sharing and Parallelism, GC28-1860*
- *OS/390 Parallel Sysplex Systems Management, GC28-1861*
- *OS/390 Parallel Sysplex Test Report, GC28-1963*
- *System/390 9672/9674 System Overview, GA22-7148*

ICSF/MVS

- *ICSF/MVS General Information, GC23-0093*

IMS/ESA

- *IMS Batch Terminal Simulator General Information, GH20-5522*
- *IMS/ESA Administration Guide: System, SC26-8013*
- *IMS/ESA Administration Guide: Transaction Manager, SC26-8731*
- *IMS/ESA Application Programming: Database Manager, SC26-8727*
- *IMS/ESA Application Programming: Design Guide, SC26-8016*
- *IMS/ESA Application Programming: Transaction Manager, SC26-8729*
- *IMS/ESA Customization Guide, SC26-8020*
- *IMS/ESA Installation Volume 1: Installation and Verification, SC26-8023*
- *IMS/ESA Installation Volume 2: System Definition and Tailoring, SC26-8024*
- *IMS/ESA Messages and Codes, SC26-8028*
- *IMS/ESA Operator's Reference, SC26-8030*
- *IMS/ESA Utilities Reference: System, SC26-8035*

ISPF

- *ISPF V4 Dialog Developer's Guide and Reference, SC34-4486*
- *ISPF V4 Messages and Codes, SC34-4450*
- *ISPF V4 Planning and Customizing, SC34-4443*
- *ISPF V4 User's Guide, SC34-4484*

Language Environment

- *Debug Tool User's Guide and Reference, SC09-2137*

National Language Support

- *National Language Support Reference Volume 2, SE09-8002*

NetView

- *NetView Installation and Administration Guide, SC31-8043*
- *NetView User's Guide, SC31-8056*

ODBC

- *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide, Microsoft Press, ISBN 1-55615-658-8*

OS/390

- *OS/390 C/C++ Programming Guide, SC09-2362*
- *OS/390 C/C++ Run-Time Library Reference, SC28-1663*
- *OS/390 C/C++ User's Guide, SC09-2361*
- *OS/390 eNetwork Communications Server: IP Configuration, SC31-8513*
- *OS/390 Hardware Configuration Definition Planning, GC28-1750*
- *OS/390 Information Roadmap, GC28-1727*
- *OS/390 Introduction and Release Guide, GC28-1725*
- *OS/390 JES2 Initialization and Tuning Guide, SC28-1791*
- *OS/390 JES3 Initialization and Tuning Guide, SC28-1802*
- *OS/390 Language Environment for OS/390 & VM Concepts Guide, GC28-1945*
- *OS/390 Language Environment for OS/390 & VM Customization, SC28-1941*
- *OS/390 Language Environment for OS/390 & VM Debugging Guide, SC28-1942*
- *OS/390 Language Environment for OS/390 & VM Programming Guide, SC28-1939*
- *OS/390 Language Environment for OS/390 & VM Programming Reference, SC28-1940*
- *OS/390 MVS Diagnosis: Procedures, LY28-1082*
- *OS/390 MVS Diagnosis: Reference, SY28-1084*
- *OS/390 MVS Diagnosis: Tools and Service Aids, LY28-1085*
- *OS/390 MVS Initialization and Tuning Guide, SC28-1751*
- *OS/390 MVS Initialization and Tuning Reference, SC28-1752*
- *OS/390 MVS Installation Exits, SC28-1753*
- *OS/390 MVS JCL Reference, GC28-1757*
- *OS/390 MVS JCL User's Guide, GC28-1758*
- *OS/390 MVS Planning: Global Resource Serialization, GC28-1759*
- *OS/390 MVS Planning: Operations, GC28-1760*
- *OS/390 MVS Planning: Workload Management, GC28-1761*
- *OS/390 MVS Programming: Assembler Services Guide, GC28-1762*
- *OS/390 MVS Programming: Assembler Services Reference, GC28-1910*
- *OS/390 MVS Programming: Authorized Assembler Services Guide, GC28-1763*
- *OS/390 MVS Programming: Authorized Assembler Services Reference, Volumes 1-4, GC28-1764, GC28-1765, GC28-1766, GC28-1767*
- *OS/390 MVS Programming: Callable Services for High-Level Languages, GC28-1768*
- *OS/390 MVS Programming: Extended Addressability Guide, GC28-1769*

- *OS/390 MVS Programming: Sysplex Services Guide, GC28-1771*
- *OS/390 MVS Programming: Sysplex Services Reference, GC28-1772*
- *OS/390 MVS Programming: Workload Management Services, GC28-1773*
- *OS/390 MVS Routing and Descriptor Codes, GC28-1778*
- *OS/390 MVS Setting Up a Sysplex, GC28-1779*
- *OS/390 MVS System Codes, GC28-1780*
- *OS/390 MVS System Commands, GC28-1781*
- *OS/390 MVS System Messages Volume 1, GC28-1784*
- *OS/390 MVS System Messages Volume 2, GC28-1785*
- *OS/390 MVS System Messages Volume 3, GC28-1786*
- *OS/390 MVS System Messages Volume 4, GC28-1787*
- *OS/390 MVS System Messages Volume 5, GC28-1788*
- *OS/390 MVS Using the Subsystem Interface, SC28-1789*
- *OS/390 Security Server (RACF) Auditor's Guide, SC28-1916*
- *OS/390 Security Server (RACF) Command Language Reference, SC28-1919*
- *OS/390 Security Server (RACF) General User's Guide, SC28-1917*
- *OS/390 Security Server (RACF) Introduction, GC28-1912*
- *OS/390 Security Server (RACF) Macros and Interfaces, SK2T-6700 (OS/390 Collection Kit), SK27-2180 (OS/390 Security Server Information Package)*
- *OS/390 Security Server (RACF) Security Administrator's Guide, SC28-1915*
- *OS/390 Security Server (RACF) System Programmer's Guide, SC28-1913*
- *OS/390 SMP/E Reference, SC28-1806*
- *OS/390 SMP/E User's Guide, SC28-1740*
- *OS/390 RMF User's Guide, SC28-1949*
- *OS/390 TSO/E CLISTS, SC28-1973*
- *OS/390 TSO/E Command Reference, SC28-1969*
- *OS/390 TSO/E Customization, SC28-1965*
- *OS/390 TSO/E Messages, GC28-1978*
- *OS/390 TSO/E Programming Guide, SC28-1970*
- *OS/390 TSO/E Programming Services, SC28-1971*
- *OS/390 TSO/E User's Guide, SC28-1968*
- *OS/390 DCE Administration Guide, SC28-1584*
- *OS/390 DCE Introduction, GC28-1581*
- *OS/390 DCE Messages and Codes, SC28-1591*
- *OS/390 UNIX System Services Command Reference, SC28-1892*
- *OS/390 UNIX System Services Planning, SC28-1890*
- *OS/390 UNIX System Services User's Guide, SC28-1891*
- *OS/390 UNIX System Services Programming: Assembler Callable Services Reference, SC28-1899*

j

PL/I for MVS & VM

- *IBM PL/I MVS & VM Language Reference, SC26-3114*
- *IBM PL/I MVS & VM Programming Guide, SC26-3113*

OS PL/I

- *OS PL/I Programming Language Reference, SC26-4308*
- *OS PL/I Programming Guide, SC26-4307*

Prolog

- *IBM SAA AD/Cycle Prolog/MVS & VM Programmer's Guide, SH19-6892*

Remote Recovery Data Facility

- *Remote Recovery Data Facility Program Description and Operations, LY37-3710*

Storage Management

- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *MVS/ESA Storage Management Library: Leading a Storage Administration Group, SC26-3126*
- *MVS/ESA Storage Management Library: Managing Data, SC26-3124*
- *MVS/ESA Storage Management Library: Managing Storage Groups, SC26-3125*
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide, SC26-4659*

System/370 and System/390

- *ESA/370 Principles of Operation, SA22-7200*
- *ESA/390 Principles of Operation, SA22-7201*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*

System Network Architecture (SNA)

- *SNA Formats, GA27-3136*
- *SNA LU 6.2 Peer Protocols Reference, SC31-6808*
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2, GC30-3084*
- *SNA/Management Services Alert Implementation Guide, GC31-6809*

TCP/IP

- *IBM TCP/IP for MVS: Customization & Administration Guide, SC31-7134*
- *IBM TCP/IP for MVS: Diagnosis Guide, LY43-0105*
- *IBM TCP/IP for MVS: Messages and Codes, SC31-7132*

- *IBM TCP/IP for MVS: Planning and Migration Guide, SC31-7189*

VS COBOL II

- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, GC26-4047*
- *VS COBOL II Installation and Customization for MVS, SC26-4048*

VS FORTRAN

- *VS FORTRAN Version 2: Language and Library Reference, SC26-4221*
- *VS FORTRAN Version 2: Programming Guide for CMS and MVS, SC26-4222*

VTAM

- *Planning for NetView, NCP, and VTAM, SC31-8063*
- *VTAM for MVS/ESA Diagnosis, LY43-0069*
- *VTAM for MVS/ESA Messages and Codes, SC31-6546*
- *VTAM for MVS/ESA Network Implementation Guide, SC31-6548*
- *VTAM for MVS/ESA Operation, SC31-6549*
- *VTAM for MVS/ESA Programming, SC31-6550*
- *VTAM for MVS/ESA Programming for LU 6.2, SC31-6551*
- *VTAM for MVS/ESA Resource Definition Reference, SC31-6552*

Index

Numerics

16-TB table space
  IFCIDs 295
8-KB and 16-KB page size
  IFCIDs 295

A

access path
  direct row access 53
  hints 59
active log
  IFCID indicating impending shortage 85
advisory restart-pending status 27
ALTER COLUMN clause
  ALTER TABLE statement 24
ALTER TABLE statement 24
APAR, Version 5
  PQ01040 86
  PQ04053 64
  PQ06465 89
  PQ07327 88
  PQ08342 90
  PQ09901 118
  PQ09947 85
  PQ10633 230
  PQ12126 88
  PQ12390 87, 88
  PQ14255 91
  PQ15670 41
  PQ15854 64
  PQ17797 43
  PQ18543 15
  PQ18710 36
  PQ18941 34
  PQ19077 34
  PQ19897 34
  PQ20032 34
  PQ21014 42
  PQ25091 97
application program
  object extensions 123
APPLNAME column
  DSN_STATEMNT_TABLE 80
archive log
  data set
    support for multi-volume 98
ARCHLOG
  option of REPORT utility 268
ASUTIME column
  resource limit specification table (RLST) 76
automatic
  restart function of MVS
    IRLM 89
AUXERROR
  option of CHECK DATA utility 262
auxiliary storage 301
auxiliary table
  LOCK TABLE statement 140
availability
  improvements in Version 6 19

B

backout processing
  postponing 26
backup and recovery enhancements 30
BLOB (binary large object)
  option of LOAD utility 264
buffer pool
  defaults 83
  tuning 61

C

CALL statement
  SQL procedure 113
CASE statement
  SQL procedure 113
casting
  in user-defined function invocation 199
  ROWID data type 70
catalog tables
  columns
    summary of changes 286
  indexes
    new 288
    revised 290
  new tables 285
catalog, shadow 119
CD-ROM, books on 6
CHANGELIMIT
  option of COPY utility 262
character conversion 95
CHECK DATA utility
  description 262
CHECK LOB utility
  description 261
CHECKPAGE
  option of COPY utility 262
checkpoint
  frequency
    changing dynamically 24

checkpoint *(continued)*
group buffer pool
enhancement in Version 6 50
CLISCHEMA keyword 118
CLOB (character large object)
option of LOAD utility 264
cluster ratio
enhancement 42
coexistence
considerations for specific capabilities 245
data sharing environment 244
distributed data environment 244
cold start
postponed abort URs 28
COLLID column
DSN_STATEMNT_TABLE 80
column
altering length 23
commands
changes in Version 6 253
compound statement
SQL procedure 114
conditional restart
postponed abort URs 28
connection
displaying
group buffer pool 49
COPY utility
description 262
parallelism 31
specifying a list 31
cost category 82
COST_CATEGORY column
of DSN_STATEMNT_TABLE 80
COST_CATEGORY_B column of RLST 77
CREATE DISTINCT TYPE, description 202
CURRENTFUNCTIONPATH keyword 118

D

DASD requirement 301
DATA
option of REPAIR utility 267
data communications devices 302
data set
statistics 84
data sets and device types 301
data sharing
enhancements 43
environment, coexistence 244
IFCIDs 295
data space
EDM pool 17
LOB materialization 132
data type
ROWID 67
DB2 Administration Tool 231
DB2 books online 6
DB2 Buffer Pool Tool 231
DB2 DataPropagator Relational 230
DB2 Estimator 229
DB2 Extenders 225
DB2 Installer 228
DB2 ODBC
enhancements in Version 6 118
DB2 Performance Monitor (PM) 230
DB2 Stored Procedures Builder 228
DB2 UDB Control Center 227
DB2 Visual Explain 228
DB2I defaults 91
DBCLOB (double-byte large character)
option of LOAD utility 264
DBINFO structure
user-defined function 164
DDF inactive connection
IFCIDs 295
DEFAULT clause
INSERT statement 70
deferred write threshold (DWQT)
recommendation for LOBs 136
DESCRIBE INPUT statement 119
determining when to reorganize data
REORG utility 96
direct row access 53
DISCARD
option of REORG TABLESPACE utility 266
DISCARD option
REORG utility 34
DISCARDDN
option of REORG TABLESPACE utility 266
option of REORG utility 34
discarding records
REORG utility 33
DISCONNECT IRLM
field of panel DSNTIPJ 87
DISPLAY DATABASE command
enhancements in Version 6 97
DISPLAY GROUPBUFFERPOOL command
summary report example 49
distinct type
assignments 206
comparing 205
creating 202
description 202
IFCIDs 295
with UNION 208
distributed data
environment, coexistence 244
moving from DB2 private protocol access to DRDA access 106
DRAIN
option of REORG INDEX utility 265

DRAIN *(continued)*
option of REORG TABLESPACE utility 266
DRDA support
three-part names 103
IFCIDs 295
DSMAX
maximum number of open data sets 15
DSN_FUNCTION_TABLE 197
DSN_STATEMNT_TABLE table
column descriptions 80
DSN1COMP utility
description 262
DSN1COPY utility
description 262
DSN1PRNT utility
description 263
DSNACCAV stored procedure 251
DSNACCQC stored procedure 251
DSNTEJ1L sample program 91
DSNTEP2 sample program
available in object form 91
DSNTIJSG job
installation 74
DSNTPSMP stored procedure 251
DSSIZE
option of DSN1COMP utility 262
option of DSN1COPY utility 263
option of DSN1PRNT utility 263
DUMP
statement of REPAIR utility
description 267
DUPLEX option of CFRM policy 45
duplexed group buffer pools 43
See also 'group buffer pool, duplexing'
dynamic SQL enhancements
IFCIDs 295

E

EDM pool
in a data space 17
EDMPOOL DATA SPACE SIZE field of panel DSNTIPC 17
enhancements
user productivity 91
escalation, lock
message 84
estimating
statement's cost 79
EXCLUSIVE
lock mode
LOB 138
EXPLAIN
description of PLAN_TABLE 293
estimating a statement's cost 79
EXPLAIN_TIME column
DSN_STATEMNT_TABLE 80

F

failure scenario
duplexed group buffer pool 49
fallback
automatic rebind 247
description 247
frozen objects 247
preparation 247
release incompatibilities 249
fast log apply
description 29
features, DB2 for OS/390 227
FLOAT
option of LOAD utility 264
frozen objects 247
FULLCOPY
option of DSN1COPY utility 263
option of DSN1PRNT utility 263
function level of IRLM 85
function resolution 193
user-defined function 193
functions
built-in, description 67
with distinct types 204

G

GENERATED clause
of CREATE and ALTER TABLE 68
GET DIAGNOSTICS statement
SQL procedure 113
GOTO statement
SQL procedure 113
governor (resource limit facility)
combining reactive and predictive modes 78
predictive
description 72
example 74
group buffer pool
caching options in Version 6 51
checkpoint
enhancement in Version 6 50
duplexing
description 43
IFCIDs 295
performance 46
starting 45
stopping 45
summary of failure scenarios 49
GROUP_MEMBER column
DSN_STATEMNT_TABLE 80

H

hardware requirements
  auxiliary storage 301
  data communications devices 302
  function-dependent 302
hints, optimization 59
host variable
  ROWID data type 71

I

ICOPY (informational copy) pending status 30
identity column
  ROWID column, contrasted with 93
identity columns 91
IF statement
  SQL procedure 113
IFCID (instrumentation facility component identifier)
  0199 84
  0330 85
  16-TB table space 295
  8-KB and 16-KB page size 295
  active log shortage 295
  buffer pool data set statistics 295
  data sharing 295
  DDF inactive connection 295
  distinct type 295
  DRDA support for three-part names 295
  dynamic SQL enhancements 295
  group buffer pool duplexing 295
  inline statistics 295
  LOB (large object) 295
  optimization hints 295
  parallel index build and REORG 295
  parallelism for COPY and RECOVER 295
  predictive governor 295
  ROWID 295
  stored procedure 295
  trigger 295
  user-defined function 295
IN
  predicate 94
INCLUDE statement
  COBOL program 94
INDEX
  option of COPY utility 262
  option of REPORT utility 268
INDEXSPACE
  option of COPY utility 262
  option of RECOVER utility 265
  option of REPORT utility 268
INDREFLIMIT
  option of REORG TABLESPACE utility 266
inline statistics collection
  IFCIDs 295
inline statistics collection *(continued)*
  LOAD utility 33
  REBUILD INDEX utility 33
  REORG utility 33
INSERT statement 94
inserting
  into a ROWID column 70
installation panel changes 250
INTENT EXCLUSIVE lock mode 139
INTENT SHARE lock mode 139
  LOB table space 139
IRLM (internal resource lock manager)
  automatic restart 89
  automatic restart manager 89
  coexistence 85
  command to display storage use 88
  display storage use 88
  function level 85
  NODISCON value of SCOPE option 86
  stopping and deregistering from ARM 89
  storage enhancements 87
  summary of enhancements 85

J

Java support 101
JDBC support 102
join operation
  star join 38
  star schema 38

L

LARGE
  option of DSN1COMP utility 262
  option of DSN1COPY utility 263
  option of DSN1PRNT utility 263
LEAFDISTLIMIT
  option of REORG INDEX utility 265
LEAVE statement
  SQL procedure 113
library
  online 6
LIMITKEY value
  changing 24
LOAD utility
  description 263
LOB (large object)
  data space 132
  declaring host variables 127
  declaring LOB locators 127
  description 124
  IFCIDs 295
  locator 132
  lock duration 139
  LOCK TABLE statement 140

LOB (large object) *(continued)*
lock, description 136
locking 136
LOCKSIZE clause of CREATE or ALTER TABLESPACE 141
materialization 132
modes of LOB locks 138
modes of table space locks 139
option of DSN1COPY utility 263
option of DSN1PRNT utility 263
recommendations for buffer pool DWQT threshold 136
with indicator variables 135
lock
duration
LOBs 139
escalation
message 84
LOB locks 136
LOB table space
LOCKSIZE clause 141
LOCK TABLE statement
effect on auxiliary tables 140
LOCKSIZE clause
effect of options
LOB table spaces 141
log
apply processing 29
IFCID indicating impending shortage 85
option of CHECK DATA utility 262
LOGLOAD subsystem parameter
changing via SET LOG command 24
LOOP statement
SQL procedure 113

M

maintenance level for selected enhancements 237
MAP
option of REPAIR utility 267
materialization
LOBs 132
MAXCSA option of IRLMPROC
changing dynamically 88
maximum
number of tables in a view 94
message by identifier
DSNB508I 17
DSNI031I 85
DSNT736I 97
DSNV435I 28
DXR100I 88
DXR175E 88
migration considerations 233
private protocol not enhanced 234
remove data set passwords 233
migration considerations *(continued)*
remove shared read-only data 233
remove views on two catalog tables 234
type 2 indexes required 233
MODIFY irlmproc,ABEND command of MVS
NODUMP option for stopping IRLM 89
MODIFY irlmproc,DIAG,DELAY command of MVS
description 90
MODIFY irlmproc,SET,TRACE command of MVS
description 89
MODIFY irlmproc,STATUS command of MVS
example showing IRLM storage use 88

N

Net.Data for OS/390 229
NOAUXCHKP
option of REPAIR utility 267
NOAUXWARN
option of REPAIR utility 267
NOCHECKPEND
option of REPAIR utility 267
NOCOPYPEND
option of LOAD utility 264
option of REPAIR utility 267
NODISCON value of IRLM SCOPE option 86
NOPAD
option of REORG TABLESPACE utility 266
notices, legal 313

O

ODBC
enhancements in Version 6 118
shadow catalog 119
OFFPOSLIMIT
option of REORG TABLESPACE utility 266
ON clause
joining tables
enhancements in Version 6 94
online books 6
optical storage 301
optimization hints 59
IFCIDs 295
optional features, DB2 for OS/390 227

P

package
RLFPKG column of RLST 77
PAGE
option of REPAIR utility 267
page size
choosing 58
PAGESIZE
option of DSN1COMP utility 262

PAGESIZE *(continued)*
option of DSN1COPY utility 263
option of DSN1PRNT utility 263
PARALLEL
option of COPY utility 262, 265
parallel index build
IFCIDs 295
LOAD utility 32
REBUILD INDEX utility 32
REORG TABLESPACE utility 32
parallelism
COPY and RECOVER utilities 31
IFCIDs, for COPY and RECOVER 295
parameter marker
casting 199
descriptions 118
PART
option of REORG TABLESPACE utility 266
option of REPAIR utility 267
partitioning index
using ROWID as partitioning key 69
pending status
check pending 30
informational COPY pending (ICOPY) 30
performance
improvements in Version 6 19
query enhancements 35
PLAN_TABLE
format 293
pool
type 2 inactive threads 120
predictive governing
combined with reactive governing 78
description 72
example 74
IFCIDs 295
in a distributed environment 79
with DEFER(PREPARE) 79
writing an application for 79
priced features, DB2 for OS/390 227
PRIMARY_ACCESSTYPE column of PLAN_TABLE 53
PROGNAME column
DSN_STATEMNT_TABLE 80
program requirements
function-dependent 303
operating systems 303
optional 305
PUNCHDDN
option of REORG TABLESPACE utility 266

Q

Query Management Facility (QMF) 229
QUERYNO column
DSN_STATEMNT_TABLE 80
QUIESCE utility
description 264

R

REBUILD INDEX utility
description 261
RECOVER POSTPONED command
description 27
RECOVER utility
description 264
parallelism 31
release coexistence 244
release incompatibilities 237
release level for selected enhancements 237
REORG INDEX utility
description 265
REORG utility
determining when to run 96
REPAIR utility
description 266
DUMP statement 267
SET INDEX statement 267
SET TABLESPACE statement 267
REPEAT statement
SQL procedure 113
REPORT utility
description 267
REPORTONLY
option of REORG INDEX utility 265
option of REORG TABLESPACE utility 266
reports
summary report
DISPLAY GROUPBUFFERPOOL command 49
requirements for DB2 for OS/390
hardware 301
program 303
virtual storage 307
resource limit facility (governor)
writing an application for predictive governing 79
restart
backward log recovery
postponing 26
restart-pending status 27
RETAINED LOCK TIMEOUT option of installation panel DSNTIPI 52
RETLWAIT subsystem parameter 52
REUSE
option of LOAD utility 264
option of RECOVER 265
option of REORG INDEX utility 265
option of REORG TABLESPACE utility 266
option, usage 34
RLFASUERR column of RLST 77
RLFASUWARN column of RLST 77

RLST (resource limit specification table)
columns 75
creating 74
routine authorization cache 64
ROWID
coding example 55
data type, description 67
IFCIDs 295
option of LOAD utility 264
option of REPAIR utility 267
ROWID column
identity column, contrasted with 93
RRSAF (Recoverable Resource Manager Services attachment facility)
transactions
using global transactions 64
RUNSTATS utility
description 268

S

savepoints 93
SCOPE
option of CHECK DATA utility 262
SCOPE option
IRLMPROC, NODISCON value 86
scratchpad
user-defined function 161
secondary structure
characteristics 45
SELECT INTO statement 94
serviceability enhancements for IRLM 89
SET Assignment statement 94
SET clause
UPDATE statement 94
SET INDEX statement of REPAIR utility 267
SET TABLESPACE statement of REPAIR utility 267
SETXCF STOP, REBUILD command of MVS
revert to simplex mode 46
shadow catalog 119
SHARE
INTENT EXCLUSIVE lock mode 139
lock mode
LOB 138
simplex mode of group buffer pool
reverting 46
SMF (System Management Facility)
type 72 records 84
softcopy publications 6
SORTKEYS
option of LOAD utility 264
option of REORG TABLESPACE utility 266
SQL (Structured Query Language)
coding
object extensions 123
extensions 94
SQL procedure
stored procedure 113
SQL procedure statement
CALL statement 113
CASE statement 113
compound statement 114
GET DIAGNOSTICS statement 113
GOTO statement 113
IF statement 113
LEAVE statement 113
LOOP statement 113
REPEAT statement 113
SQL statement 114
WHILE statement 113
SQL statement
SQL procedure 114
SQL statements
changes in Version 6 269
DESCRIBE INPUT 119
SQLCODE
-101 238
+495 79
SQLDA 94
SQLDescribeParam(), API 118
SQLGetLength(), API 118
SQLGetPosition(), API 118
SQLGetSubString(), API 118
SQLJ support 101
star schema 38
statement table
column descriptions 80
statistics
data set 84
option of LOAD utility 264
option of REORG INDEX utility 265
option of REORG TABLESPACE utility 266
temporary tables 42
STATISTICS option
LOAD utility 33
REBUILD INDEX utility 33
REORG utility 33
STMT_TYPE column
DSN_STATEMNT_TABLE 80
STOP DATABASE command
enhancements in Version 6 97
stopping
IRLM 89
storage
EDM pool
data space 17
IRLM (internal resource lock manager)
command to display 88
requirements
real 301
virtual 307

stored procedure
IFCIDs 295
invoking from a trigger 218
multiple versions 119
routine authorization cache 64
SQL procedure 113
summary report
example of DISPLAY GROUPBUFFERPOOL command 49
syntax diagrams, how to read 2
SYSIBM.SYSPROCEDURES catalog table
migration considerations 241

T

tables
maximum number in a view 94
TABLESPACESET
option of QUIESCE utility 264
temporary table
statistics 42
thread
distributed
pooling of inactive threads 120
TIMEOUT
option of REORG INDEX utility 265
option of REORG TABLESPACE utility 266
trace
controlling
IRLM trace buffer storage 89
diagnostic
IRLM 89
transaction
IMS
using global transactions 64
transition table
trigger 215
transition variable
trigger 215
trigger
activation order 220
cascading 219
coding 213
description 211
example 212
IFCIDs 295
interaction with constraints 221
overview 212
parts of 213
transition table 215
transition variable 215
Try and Buy program 227
type 2 inactive threads 120

U

unit of recovery
in-abort
postponing processing 26
UNLOAD
option of REORG TABLESPACE utility 266
UNLOAD EXTERNAL option
REORG utility 34
unloading data
REORG utility 33
UPDATE statement 94
UR (uncommitted read)
effect on reading LOBs 137
user-defined function
abnormal termination 200
accessing transition tables 182
assembler parameter conventions 167
assembler table locators 183
C or C++ table locators 185
C parameter conventions 168
casting arguments 199
COBOL parameter conventions 174
COBOL table locators 186
data type promotion 196
DBINFO structure 164
definer 144
defining 146
description 143
DSN_FUNCTION_TABLE 197
example 144
example of definition 148
function resolution 193
host data types 153
IFCIDs 295
implementer 144
implementing 150
invocation syntax 193
invoker 144
invoking 192
invoking from a predicate 200
invoking from a trigger 218
main program 151
overview 144
parallelism considerations 151
parameter conventions 153
PL/I parameter conventions 178
PL/I table locators 187
preparing 188
restrictions 150, 151
routine authorization cache 64
scratchpad 161
setting result values 159
simplifying function resolution 196
subprogram 151
testing 190

user-defined function *(continued)*
use of scratchpad 181
use of special registers 179
utilities
changes in Version 6 261
CHECK DATA 262
CHECK LOB 261
COPY 262
DSN1COMP 262
DSN1COPY 262
DSN1PRNT 263
LOAD 263
performance improvements 29
QUIESCE 264
REBUILD INDEX 261
RECOVER 264
REORG INDEX 265
REPAIR 266
REPORT 267
RUNSTATS 268

V

VALUES clause
INSERT statement 94
VALUES INTO statement 94
VDWQT option of ALTER BUFFERPOOL command 62
VERSION
option of REPAIR utility 267
vertical deferred write threshold (VDWQT) 62

W

wait time
accounting class 3 84
SMF type 72 records 84
WHILE statement
SQL procedure 113

How to send your comments

DB2 Universal Database for OS/390
Release Planning Guide
Version 6

Publication No. SC26-9013-02

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 for OS/390 documentation. You can use any of the following methods to provide comments.

- Send your comments by e-mail to db2pubs@vnet.ibm.com and include the name of the product, the version number of the product the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title, page number, or a help topic title).
- Send your comments from the Web. Visit the DB2 for OS/390 Web site at:

  http://www.ibm.com/software/db2os390

  The Web site has a feedback page that you can use to send comments.
- Complete the readers' comment form at the back of the book and return it by mail, by fax (800-426-7773 for the United States and Canada), or by giving it to an IBM representative.

Readers' Comments

DB2 Universal Database for OS/390
Release Planning Guide
Version 6

Publication No. SC26-9013-02

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Address

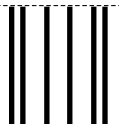
Company or Organization

Phone No.

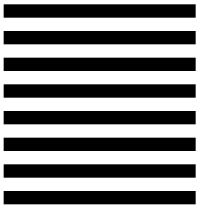
Readers' Comments
SC26-9013-02

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department BWE/H3
PO Box 49023
San Jose, CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

SC26-9013-02

Cut or Fold
Along Line

Program Number: 5645-DB2

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9013-02