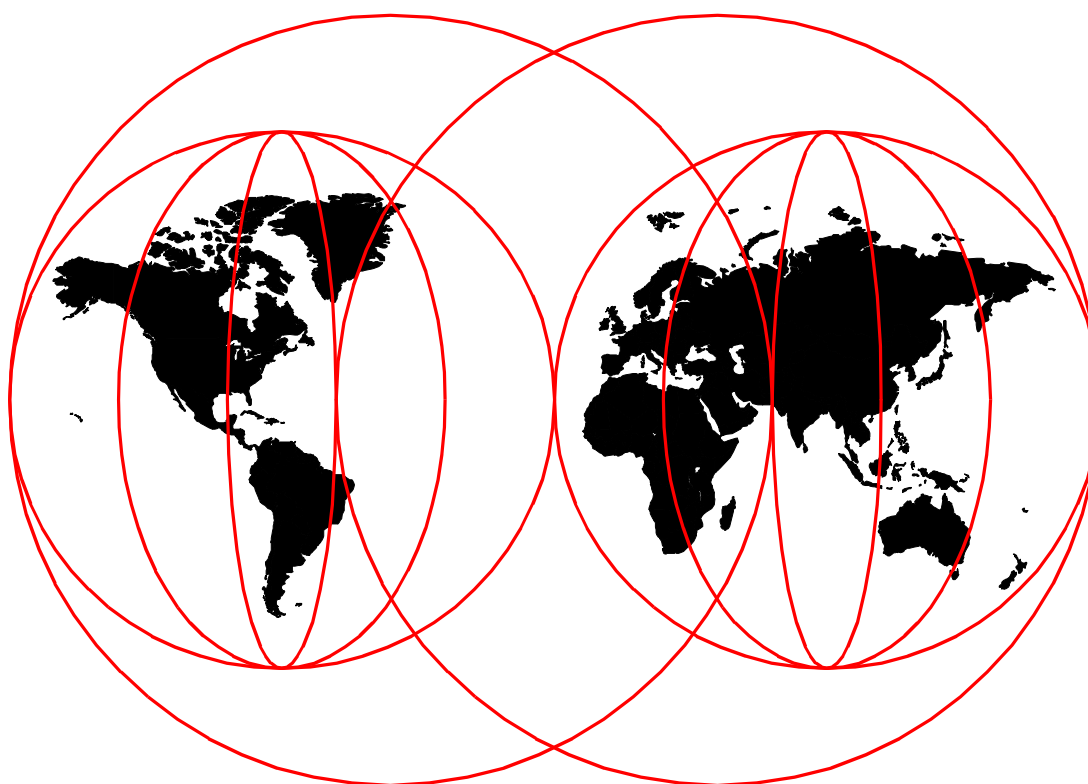




DB2 UDB Server for OS/390 Version 6 Technical Update

Paolo Bruni, Sarah Ellis, Rod Gibson, Vusumzi Kopo, Neil Toussaint



International Technical Support Organization

www.redbooks.ibm.com



International Technical Support Organization

SG24-6108-00

**DB2 UDB Server for OS/390 Version 6
Technical Update**

May 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, "Special notices" on page 281.

First Edition (May 2000)

This edition applies to Version 6 of IBM DATABASE 2 Universal Database Server for OS/390 (DB2 UDB for OS/390 Version 6), Program Number 5645-DB2.

The document was created or updated on May 11, 2000.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000. All rights reserved.

Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Preface	vii
The team that wrote this redbook	vii
Comments welcome	ix
 Chapter 1. Introduction	1
1.1 DB2 UDB for OS/390 Version 6 refresh	2
1.2 Functional enhancement areas	4
1.3 Performance measurements	7
 Chapter 2. Version 6 general news	9
2.1 Migration considerations	10
2.1.1 Release incompatibilities	11
2.1.2 Host variables must be preceded by a colon ":"	12
2.2 UDF performance considerations	14
2.2.1 Built-in or application program	16
2.2.2 External UDF or built-in function	17
2.2.3 Maximizing UDF efficiency	19
2.2.4 Use built-in functions	21
2.2.5 Consider sourced functions	23
2.2.6 UDF summary	25
2.3 Trigger performance considerations	26
2.3.1 Trigger overview	28
2.3.2 Transition variables	29
2.3.3 Transition tables	30
2.3.4 Transition variable/table usage	31
2.3.5 Row trigger or statement trigger	33
2.3.6 Trigger coding considerations	35
2.3.7 Understanding trigger performance	37
2.4 LOBs performance considerations	39
2.4.1 LOBs overview	40
2.4.2 LOBs processing	41
2.4.3 LOBs read performance	43
2.4.4 LOBs write performance	45
2.4.5 LOBs recommendations	47
2.5 ESS performance	49
2.5.1 ESS read performance	50
2.5.2 DB2 logging rates by disk type	52
2.6 Distributed functions performance	53
 Chapter 3. Application enhancements	55
3.1 Identity columns	56
3.1.1 Existing techniques to create new keys	59
3.1.2 Definition of identity columns	62
3.1.3 GENERATED options	64
3.1.4 Identity columns in DB2 catalog	66
3.1.5 Identity columns performance	67
3.1.6 Impact of caching	68
3.1.7 Applications and identity columns	70
3.1.8 IDENTITY_VAL_LOCAL function	72
3.1.9 Managing tables with identity columns	74
3.1.10 Advantages of identity columns	77

3.2	Savepoints	79
3.2.1	Connecting to other DB2 systems	80
3.2.2	Restrictions on using savepoints	80
3.2.3	Savepoint performance	81
3.3	Declared temporary tables	82
3.3.1	Main differences between table types	84
3.3.2	Considerations when converting from CTTs	85
3.3.3	Defining a declared temporary table	86
3.3.4	Authorization	87
3.3.5	Referencing declared temporary tables	88
3.3.6	Creating indexes	89
3.3.7	Usage considerations	90
3.3.8	Database and table space issues	93
3.3.9	Restrictions	95
3.4	Update with subselect	97
3.4.1	Conditions for usage	97
3.4.2	Self referencing considerations	98
3.5	Columns in order by not in select	99
3.6	Global transactions	100
3.6.1	Funds transfer example	101
3.6.2	Existing designs: 1 or 2 units of work	102
3.6.3	Re-engineering design B	103
3.6.4	Step 1 — Updates are performed under DB2 thread 1	104
3.6.5	Step 2 — DB2 thread 2 update times out	105
3.6.6	Thread 2 in same global transaction — problem solved	106
3.6.7	Where the XID is passed — example 1	107
3.6.8	Where the XID is passed — example 2	108
3.6.9	Where the XID is passed — example 3	109
3.6.10	Flow between participants	110
3.6.11	Flow with global transaction support	111
3.6.12	Considerations	112
Chapter 4.	Language support	115
4.1	DB2 REXX support	116
4.1.1	Host environment and handling errors	118
4.1.2	Isolation level and coding conventions	120
4.1.3	REXX and stored procedures	122
4.1.4	Set up WLM environment	124
4.2	SQL Procedure language	126
4.2.1	Reasons for using SQL stored procedures	128
4.2.2	Creating SQL stored procedures	130
4.2.3	Compound statements	131
4.2.4	Example of a compound statement	132
4.2.5	Handling errors	134
4.2.6	Debugging SQL stored procedures	136
4.2.7	Preparing SQL stored procedures	137
4.2.8	Preparation using DSNTPSMP	139
4.2.9	Preparation without DSNTPSMP	141
4.3	SQLJ/JDBC driver support	142
4.3.1	Using the new driver	144
4.3.2	Combining SQLJ and JDBC	145
4.4	Java stored procedures	147
4.4.1	Defining a Java stored procedure	148

4.4.2	Java SP coding considerations	149
4.4.3	Returning results set	150
4.4.4	Preparing Java stored procedures.	151
4.4.5	Preparing Java SPs using SQLJ	152
4.4.6	Running Java stored procedures.	153
Chapter 5.	Operational enhancements	155
5.1	Suspend update activity	156
5.1.1	Deciding whether to use this method for disaster recovery	157
5.1.2	Use of SET LOG SUSPEND command	158
5.1.3	Effects of commands.	160
5.1.4	Suspend updates recommendations	162
5.1.5	Offsite recovery considerations	164
5.1.6	Re-establish recoverability offsite	165
5.1.7	Operational considerations	166
5.2	Defer defining data sets	170
5.2.1	Effect of deferring DEFINE of VSAM data sets	170
5.2.2	Impact on DDL performance	172
5.2.3	Where define no helps	173
5.2.4	Restrictions	174
5.2.5	Things to watch out for	175
5.3	DDF suspend	177
5.3.1	Applications may retain incompatible locks	177
5.3.2	STOP DDF MODE(SUSPEND)	179
5.3.3	STOP DDF MODE(SUSPEND) WAIT(n)	181
5.3.4	STOP DDF MODE(SUSPEND) CANCEL(n)	182
5.3.5	DDF command options	183
5.4	Faster cancel thread	184
5.4.1	Cancel thread example	185
5.4.2	Operational improvement	186
5.4.3	Restrictions	186
5.5	Data sharing enhancements	187
5.5.1	Faster data sharing member shutdown	188
5.5.2	New IMMEDIATE(PH1) bind option	190
5.5.3	IFI and commands with group scope	192
5.6	New EDM pool parameter.	194
5.7	New CHECKPAGE option for COPY	196
5.7.1	How to activate page checking	196
5.7.2	Exploitation of CHECKPAGE.	197
5.7.3	How to detect and resolve errors.	197
5.7.4	CHECKPAGE performance	198
5.7.5	Usage and recommendations	198
5.8	Runstats improvements	198
5.8.1	Non uniform statistics for SYSCOLDIST	199
5.8.2	Additional space statistics	199
Chapter 6.	Performance	201
6.1	Star join	202
6.1.1	Introduction to star schema design	203
6.1.2	Introduction to star join support in DB2 V6	207
6.1.3	More about DB2 V6 star join	210
6.1.4	Performance results	235
6.2	Volatile tables to use indexes	239

6.3 Query parallelism enhancements	241
6.3.1 New feature to limit degree of parallelism	241
6.3.2 Short running static SQL running with parallelism	241
6.4 Active log I/O performance	243
6.4.1 Reducing contention from log readers	244
6.4.2 DB2 log write improvements	245
6.5 Data sharing improvements	247
6.5.1 Insert performance	247
6.5.2 Remove CLOSE YES as requirement for data set physical close	248
6.5.3 Name class queue support	248
6.5.4 Improved trace for asynchronous requests	249
Chapter 7. Additional functional enhancements	251
7.1 Unicode client toleration support	252
7.2 IEEE float toleration	254
7.3 Controlling updates to partitioning key	256
7.4 Toleration of separator differences	258
7.5 New LANGUAGE bind option	258
7.6 New operator for NOT	259
7.7 DBPROTCL default change	259
7.8 Instrumentation enhancements	260
Appendix A. DB2 APARs cross references	261
A.1 Functional enhancements	261
A.2 Performance related maintenance	264
Appendix B. Sample external user defined function	265
Appendix C. DB2 and REXX	269
C.1 Sample main program REXX code	269
C.2 JCL to invoke REXX main program	273
C.3 REXX stored procedure	274
C.4 Create procedure statement for REXX SP	275
C.5 COBOL program calling REXX SP	275
C.6 WLM SP started task source JCL	277
C.7 WLM configuration	277
C.8 Commands to manipulate WLM and SP	278
Appendix D. Special notices	281
Appendix E. Related publications	285
E.1 IBM Redbooks	285
E.2 IBM Redbooks collections	285
E.3 Other resources	286
E.4 Referenced Web sites	286
How to get IBM Redbooks	289
IBM Redbooks fax order form	290
Abbreviations and acronyms	291
Index	293
IBM Redbooks review	297

Preface

This redbook, in the format of a presentation guide, describes the changes introduced to-date in DB2 UDB Server for OS/390 Version 6 (DB2 V6 throughout the book) since its general availability in June 1999. Several enhancements have been provided through service in order to meet the increasing demands of customers and vendors for new functions. These enhancements are planned to be consolidated and integrated in a code refresh level available in May 2000.

This redbook describes the major functional enhancements and also provides performance considerations based on the measurements performed at Santa Teresa Laboratory. Since general availability of DB2 Version 6, more information has been provided on DB2 functions with white papers or performance reports. This redbook also includes recent performance and usage considerations not strictly related to the refresh level enhancements, but generally applicable to DB2 V6.

The information provided in this book will help DB2 system programmers, database administrators, and application developers in understanding, assessing, and utilizing the new functions of DB2 for OS/390 Version 6.

The team that wrote this redbook

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

Paolo Bruni is a Certified Consultant I/T Architect currently on assignment as Data Management Specialist for DB2 for OS/390 at the International Technical Support Organization, San Jose Center, where he conducts projects on all areas of DB2 for OS/390. Before joining the ITSO in 1998, Paolo worked in IBM Italy as account SE at Banking, Finance and Securities ISU.

Sarah Ellis is a DB2 Specialist in Hursley, UK where she is responsible for the DB2 for OS/390 Introduction Programs in EMEA countries. She also provides technical support and consultancy on all areas of DB2. Before joining IBM in 1998 Sarah worked for 10 years for financial institutions specializing in database design and performance.

Rod Gibson is a Certified Consultant I/T Specialist based in London, UK. He works on an IBM team supporting one of the large UK financial institutions. Rod has 10 years of experience in DB2, having previously worked with IMS for several years. His areas of special interest within DB2 include performance and availability. His work with his customers ranges from high-level architecture and design for systems that use data (whether in DB2 or not) to database design and performance work that is specifically related to DB2, or related products, such as DB2 Data Propagator (DPropR).

Vusumzi Kopo is an I/T Specialist based in South Africa. He provides technical and marketing support to DB2 customers. His areas of interest include Net.Data, application performance and tuning, and accessing DB2 data from other relational databases. He has recently worked on projects converting applications from SQL/DS to DB2 for OS/390.

Neil Toussaint is a Systems Programmer with IBM Global Services and is based in the UK. He has 12 years of experience working with DB2 in a variety of market sectors and on different platforms. He holds a Ph.D. in Biological Sciences from The University of Edinburgh. His work with customers includes sub-system installation, maintenance and support, problem diagnosis and resolution, database design and consultancy services.

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

Vasilis Karras
Rich Conway

International Technical Support Organization, Poughkeepsie Center

Maria Sueli Almeida
Emma Jacobs
Yvonne Lyon
Elsa Martinez

International Technical Support Organization, San Jose Center

Terry Allen
Bill Bireley
Frank Bower
John Campbell
Roy Cornford
Karelle Cornwell
Curt Cotner
Ramani Croisettier
Dick Crus
Gene Fuh
James Guo
Akiko Hoshikawa
Eva Hu
Jeff Josten
John Kelly
Gopal Krishnam
Phil Lamb
Marsha Larson
Ching Lee
Debbie Matamoros
Claire McFeely
Roger Miller
Chris Munson
Todd Munk
Connie Nelin
Mai Nguyen
Dave Oberstadt
Mary Paquet
Mary Petras

James Pickel
Jim Pizor
Dave Raiman
Jim Ruddy
Jack Shedden
Frank Sherwin
Akira Shibamiya
Bryan Smith
Roy Smith
Jim Teng
Hong Tie
Annie Tsang
Jay Yothers
Jane Wang
Julie Watts
Maryela Weihrauch
Chung Wu
IBM Santa Teresa Laboratory

Namik Hrle
IBM SAP Competency Center, Waldorf

Norbert Jenninger
Georg Rohonyi
IBM Boeblingen Laboratory

Mike Bracey
IBM PISC, Hursley

Adrea Harris
Nin Lei
Dino Tonelli
IBM Teraplex Center, Poughkeepsie

Steve Bower
IBM UK

Bart Steegmans
IBM Belgium

Comments welcome

Your comments are important to us!

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

- Fax the evaluation form found in “IBM Redbooks review” on page 297 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Introduction

Introduction



Objectives of the redbook

- **Version 6 refresh specifics**
 - general description
 - functional enhancements areas
 - performance measurements
- **Version 6 general news**
 - migration
 - performance and usage information

The main objective of this redbook is to bring the reader up-to-date with what has been made available through standard maintenance to DB2 V6 in terms of functions, what measurements have taken place since general availability, and what major considerations can be drawn from these measurements.

The V6 general news will be discussed in detail in Chapter 2, “Version 6 general news” on page 9, where we will mention general migration considerations and provide information on triggers, UDFs, LOBs.

Version 6 refresh level



- Available May 2000
- Rolls in functional enhancements
- Integrated testing
- Refresh distribution
- Useful target point for production
- Major new functions
- Updated PDF version of the manuals

1.1 DB2 UDB for OS/390 Version 6 refresh

In 1998 IBM began refreshing the DB2 for OS/390 Version 5 product for its customers by delivering not only maintenance, but also significant new DB2 capabilities. With Version 6, DB2 UDB for OS/390 continues to meet customer needs in a rapidly changing business environment. DB2 V6 will receive maintenance changes and periodical refreshes in order to deliver some more functions that will increase the value of your investment.

The refresh provides a DB2 product with accumulated service that has undergone integration and performance testing. This allows system programmers and database administrators to plan for and perform an easier installation of a service level that includes several new functions. This refresh is automatically shipped to new orders of DB2 after May 26, 2000.

For existing users, the equivalent service level is obtainable by ordering the preventive service update PDO level 0014, which should include service up to PUT 0003, or ESO level 0003.

Before installing the refresh you must review the current Preventive Service Planning (PSP) information referring to UPGRADE DB2610.

When ordering the DB2 V6 refresh level product, two new options are available:

- The REXX language support, previously downloadable from the Web, is now integrated in the product as a no-charge feature.
- The DB2 Forms tool is now added to the growing family of separately chargeable productivity tools.

This refresh delivers new options for application programming and database design to improve your productivity. It also incorporates substantial improvements in performance and continuous availability. SQL can be used from REXX to simplify many tasks in programming and administration. The SQL Procedures Language provides a new option for stored procedures that meets international standards. Coding applications, especially if moving from other products to DB2, is easier now, with savepoints, identity columns, and declared temporary tables.

DB2 family compatibility is improved and many customer requests are satisfied with the ability to update from a subselect in the SET clause. Star join optimization can improve the elapsed time for certain queries by more than an order of magnitude. The ability to defer data set definition helps in both the time to install and management for some applications. Being able to suspend and resume distributed processing and all DB2 processing makes some processes much faster and less disruptive.

These functions are delivered with no additional charge. If you are running Version 5 today, you can migrate directly to this level, so your systems programmers have an easier task. There has been additional system and performance testing, so this is the recommended maintenance level if you are either migrating from V5 or already have DB2 Version 6.

The new DB2 capabilities are fully documented in the updated edition of the DB2 manuals available in PDF format at the following Web site:

<http://www.ibm.com/software/data/db2/os390/v6books.html>

DB2 Performance Monitor has been enhanced as well. It now provides an API to the Online Monitor Data Collector. This allows you to retrieve performance information about the subsystem being monitored. You can obtain raw data and derived performance information including snapshot information and recent history collected to a dataset. This also includes exception alerts based on DB2 events. DB2 PM also supports the enhanced functionality of DB2. The new and updated DB2 PM manuals are also available at the Web site listed above.

Functional enhancement areas



- **Application enhancements**
- **Language support enhancements**
- **Operational enhancements**
- **Performance enhancements**
- **Additional functional enhancements**

1.2 Functional enhancement areas

These are the areas in which we have arbitrarily grouped the new functional enhancements of DB2 V6 with a brief description of the key enhancements that are being delivered.

- *Application enhancements*

- Better optimization for complex queries

Using a new star join method, DB2 for OS/390 can provide better optimization and execution performance for queries that join tables together in a star schema. A star schema consists of a fact table and a number of dimensions, each of which consists of one or more tables. In addition to improving execution performance, this enhancement enables DB2 to handle more complex star schemas and more tables in a join.

- Savepoints

Your application can set a savepoint within a transaction. Application logic can undo the data and schema changes made, since the application sets the savepoint without affecting the overall outcome of the transaction. Using savepoints makes coding applications more efficient. You no longer need to include so much contingency logic in your applications.

- Support for defining identity columns

When a column of a table is an identity column, the column has an attribute that enables DB2 to automatically generate a unique numeric value for each row that is inserted into the table. Identity columns are ideally suited to the task of generating unique primary key values. Applications that use

identity columns may be able to avoid concurrency and performance problems that sometimes occur when applications implement their own unique counters.

- Ability to declare temporary tables

Declared temporary tables complement the existing global temporary tables available in Version 5 of DB2 for OS/390. Declared temporary tables do not have descriptions in the catalog tables. These new tables also support indexes, UPDATE statements, and positioned DELETE statements. You can implicitly define the columns and use the result table from a SELECT.

- Global transaction support

Applications can take advantage of global transaction support: different DB2 units of recovery can share locks and access the same data when they are part of the same global unit of work. A syncpoint manager, such as Component Broker or IMS, must coordinate commit operations using a two-phase commit protocol.

- Update with subselect

You can use a subselect to determine the value that is to be used in the SET clause of an UPDATE statement. This enhancement improves DB2 family compatibility.

- *Language support enhancements*

- SQL support for REXX

With this enhancement, application programmers can issue SQL statements from REXX programs. The SQL statements can be anywhere a REXX command can be. Programmers can also write DB2 stored procedures in REXX. The SQL interface to REXX supports almost all SQL statements that DB2 for OS/390 supports.

- Stored procedures that are written entirely in SQL

With SQL procedures, you can now write stored procedures consisting entirely of SQL statements. An SQL procedure can include declarations (of variables, conditions, cursors, and handlers), flow control, assignment statements, and traditional SQL for defining and manipulating relational data. These extensions provide a procedural language for writing stored procedures, and they are consistent with the PSM (Persistent Stored Modules) portion of the SQL standard. You can use the Stored Procedure Builder to build SQL stored procedures. The Stored Procedure Builder is an element of the DB2 Management Tools Package, which is a no-charge feature of DB2 for OS/390 Version 6.

- Support for Java stored procedures

With this enhancement, DB2 for OS/390 handles stored procedures that are written in the Java programming language. You can write Java stored procedures that contain either static SQL (by using SQLJ) or dynamic SQL (by using JDBC). Alternatively, you can use the Stored Procedure Builder (an element of the DB2 Management Tools Package) to build Java stored procedures. Java stored procedures that run on DB2 for OS/390 can also run on other operating systems and platforms, including those of other database vendors, without being recompiled or modified.

- *Operational enhancements*

- Deferred definitions of data sets

You can defer defining data sets when you have many tables that do not have data in them. This enhancement will provide faster installation of applications and make data set management easier for SAP R/3 and PeopleSoft customers.

- DDF suspend and resume

You can use DDF suspend and resume commands at a server to temporarily suspend activity from requesters without terminating connections. Suspending requester activity enables data definition operations at the server to complete.

- Suspend update activity

The LOG SUSPEND command suspends update activity and logging while you take an external copy of your production system. The LOG RESUME command causes update activity and logging to resume. During the brief suspension, you can take a copy using a fast-disk copy facility, such as Enterprise Storage Server FlashCopy or RAMAC Virtual Array SnapShot.

- Faster cancel thread

Threads inactive or suspended within DB2 can now be cancelled and proceed to termination processing without waiting for thread control being resumed by DB2. Thread termination becomes independent of application activity.

- New CHECKPAGE option during Image Copy

With this option you can ask the Copy utility to perform the extra validity checks that previously were only done by a separately executed DSN1COPY with CHECK option.

- *Performance enhancements*

- Star Join

This provides a new way of processing multiple dimension tables to a single fact table that can enhance the performance in star schema database designs, which are typical of business intelligence systems.

- Log I/O

This provides improvement in the concurrency of read I/Os by accessing the active log second copy and the higher concatenation of write I/Os.

- Volatile tables to use indexes are now available.

- Improved query parallelism has been implemented.

- Data sharing improvements have been provided.

- *Additional functional enhancements*

Several other miscellaneous enhancements for better compatibility, portability and instrumentation have been provided.

Performance measurements



- Refresh level related functions
- Version 6 post GA measurements
- Santa Teresa Lab Performance Department
- S/390 Teraplex Center Poughkeepsie

1.3 Performance measurements

The performance measurements mentioned in this redbook come mostly from the Performance Department in Santa Teresa Laboratory.

Measurements are part of the development plan and are meant to verify that either no regression or the expected improvements take place. Most of the DB2 V6 GA code level measurements were referenced in the redbook *DB2 UDB for OS/390 Version 6 Performance Topics*, SG24-5351 and they still apply. Other measurements were performed after the cut-off date for that redbook and are mentioned in Chapter 2, “Version 6 general news” on page 9. New specific measurements were performed for the enhancements included in the code refresh: they are referenced in the other chapters of this redbook.

The S/390 Teraplex Center in Poughkeepsie is often involved in proof-of-concepts, installations and verifications, so interesting performance measurements are performed for leading edge application environments. We have referenced some measurements in 2.5.1, “ESS read performance” on page 50. For information on the Teraplex Center activities refer to the URL:

<http://www.ibm.com/solutions/businessintelligence/teraplex/index.htm>

Chapter 2. Version 6 general news

Version 6 general news



Considerations on:

- Migrating from DB2 V5 to V6, base level or refresh

Considerations based on measurements of V6 base level functions:

- User defined functions performance
- Triggers performance
- Large objects performance
- DB2 performance with the IBM Enterprise Storage Server
- Distributed functions performance

In this chapter we provide a summary reminder of considerations on migration from DB2 UDB for OS/390 Version 5 to Version 6, and then we provide usage considerations on triggers, user-defined functions (UDFs), and large objects (LOBs). These usage considerations are based on measurements implemented at Santa Teresa Laboratory on the V6 base code level, but are generally applicable. We also briefly describe and point to other V6 relevant documents on DB2 performance with the IBM Enterprise Storage Server (ESS) and on distributed functions.

Migration considerations



- **Refer to current standard documentation**
 - DB2 UDB for OS/390 Version 6 Release Guide, Installation Guide, Program Directory, PSP bucket, and info APARs II11442, II12343
- **Toleration APARs/PTFs**
 - PQ17740/UQ90001, PQ30684/UQ36939, PQ34199/UQ40803
- **Please note that PQ30684 applies to ALL customers**
 - Required to avoid further action after fallback
 - Must be applied to all members for data sharing coexistence

2.1 Migration considerations

This section is meant to be only a quick summary of topics to consider when migrating to DB2 V6 refresh level. You must consult the available updated official documentation available with the product or from the Web:

- *DB2 UDB for OS/390 Version 6 Release Guide, SC26-9008-02*
Chapter 8 helps in planning for migration and fall-back.
- *DB2 UDB for OS/390 Version 6 Installation Guide, SC26-9008-01*
Section 2.7 on Migrating the DB2 subsystem is the prime source for migration planning and guidance.
- Program Directories

These documents are available with product or on the Web site:

<http://www.software.ibm.com/data/db2/os390/books.html>

Also consult the Preventive Service Planning and the information APARs II11442 and II12343, available through the IBM Information/Access or ServiceLink facility. II1142 details migration related maintenance, while II12343 is specific for the V6 May refresh level, often mentioned as RML (Recommended Maintenance Level).

Migration to DB2 UDB for OS/390 Version 6 is only permitted from DB2 Version 5.

Notes on toleration and fallback

Verify that you have installed all the required maintenance on your DB2 Version 5 before you start to migrate. The following toleration APARs/PTFs for fallback must be applied to your DB2 Version 5 system:

- PQ17740/UQ90001 for toleration of fallback from DB2 UDB Version 6
- PQ30684/UQ36939 and PQ34199/UQ40803 for toleration/fallback of plans/packages bound in DB2 Version 6 with PQ30652/UQ38405 applied (the APAR for identity column functionality).

UQ36939 and UQ40803 must be installed on the DB2 subsystem or on all DB2 data sharing members of the group, in order for customers to fallback to V5 without further action. These PTFs are also required to be able to run a data sharing system in coexistence mode. Carefully check the supporting documentation for advice on staging these PTFs, and related preconditioning PTFs, through all the members of a data sharing group. If toleration PTF UQ36939 is not installed and a fallback is required, the following will occur:

- DB2 will be unable to run plans and packages that have been bound or rebound on the new level of DB2. It will try to auto-REBIND them, but this will fail, and they must be explicitly rebound before they can be executed.
- Programs that have been precompiled and bound on the new level of DB2 must be precompiled and bound after fallback before they can be executed.
- Programs that use the new functionality will be frozen on fallback.

This issue may affect the decision of whether to run REBIND(*) after migration. Please note that this APAR applies to all customers whether using data sharing or not, and is not limited to customers exploiting the new identity column functionality. APAR PQ36815 gives further details on this issue.

PQ36405/UQ41904 is important to fix problems with stored procedures in DB2 Version 6 where also the MEPL does not appear in the dump. This may significantly impact problem diagnosis.

2.1.1 Release incompatibilities

For your pre-migration activities, browse through the following checklist and refer to the *DB2 UDB for OS/390 Version 6 Installation Guide*, SC26-9008-01 for details:

- Identify unsupported objects
 - Convert indexes to type 2
 - Remove data set passwords
 - Eliminate shared read-only data
 - Remove views on two catalog tables
- Save critical access paths (optional)
- DRDA support for three-part names (optional)
- Examine all new and changed values for DB2I panels
- Make adjustments for release incompatibilities
 - Adjust application programs
 - Maintenance level requirements

- SQLCODE -101
- No colon on a host variable is an error (see 2.1.2, “Host variables must be preceded by a colon “:” on page 12)
- Changed format for DSN messages
- Changed format for message DSNU050I
- SQL reserved words
- Using the Euro symbol
- Using aliases
- QUIESCE return code
- DSNH message ID lengths
- Positive SQLCODE from PREPARE
- Changed SQLSTATES
- New meaning for SQLCODE
- New DBPROTOCOL default option
- Changed default for RELCURHL subsystem parameter
- Changed default for DYNRULS subsystem parameter
- Changed default for PTASKROL subsystem parameter
- Using new column called CLUSTERRATIOF
- Support for large objects
- A technique for reducing number of matching columns no longer works
- New reserved qualifier for tables, SESSION
- Changes to the RLST
- SYSIBM.SYSPROCEDURES no longer used
- An 'X' plan in the PLAN_TABLE
- Limit backouts with system restarts
- Changes to IFCID fields
- DISPLAY BUFFERPOOL changes
- Index changes
- ALTER INDEX syntax
- RECOVER INDEX becomes REBUILD INDEX
- Work space formulas changed for utilities
- Support for up to 150,000 connections
- Change to parameter in IRLMPROC startup procedure

2.1.2 Host variables must be preceded by a colon “:”

DB2 Version 6 enforces the standard that requires all host variables preceded by a colon “:”. All host variable references must have the leading colon. If the colon is missing the precompiler will issue a DSNH104I message or interpret the variable as a column name. Converting to the standard requires a REBIND. If the DBRM was produced by DB2 V2R2 or prior release it is in an old format, called Format 1, and the REBIND with DB2 V6 may fail. APAR II12100 contains information on this problem, and APARs PQ26922 and PQ30390 may be applicable in your case.

A sample REXX procedure, which analyzes all Format 1 DBRMs to check that all host variables are preceded with a colon, is available from the Web. The REXX/DB2 interface is used, and therefore DB2 for OS/390 V5 or V6 is required. The procedure creates temporary EXEC libraries, copies the REXX EXEC, executes DSNTIAUL using PARM('SQL') to extract data from the catalog, extracts DBRM listings from the catalog, executes the REXX to analyze the output looking for missing colons preceding host variables ":hv", and produces a report.

You need to examine the exceptions identified by the REXX program. It should be obvious where you need to amend the source SQL and re-precompile.

It is a sample only and it is provided without any implied warranty or support. We have not checked all eventualities, so we cannot guarantee that every invalid DBRM will be found, but it can assist you with the migration.

The procedure is called *F1 DBRM colon finder* and it is available from the URL:

<http://www.ibm.com/software/db2os390/downloads.html>

UDF performance considerations



Large number of built-in functions

- mathematical functions, string/date/time manipulation
- perform well
- can be used together
- can be used as source for user-defined functions

Can write external programs and define as functions

- extends power of SQL
- assists migration from other DBMS to DB2
- simplifies SQL syntax

Performance evaluation

- built-in function vs. coding functionality in application program
- external user-defined function vs. built-in function
- sourced user defined function vs. built-in function
- what are the important factors to obtain good performance
- main vs. sub-program

2.2 UDF performance considerations

The number of built-in functions increased considerably in version 6 over previous releases. There are now over 90 different functions that perform a wide range of string, date, time and timestamp manipulations, data type conversions, and arithmetic calculations. These built-in functions can be used in conjunction with each other and as the basis for sourced user-defined functions.

In addition, you can write your own user-defined functions (UDF) that call an external program. This extends the functionality of SQL to whatever you can code in an application program; essentially, there are no limits. For more information, refer to the DB2 for OS/390 Version 6 standard manuals.

Motives for wanting to use UDFs include:

- Simplifying SQL syntax
- Extending power of SQL
- Helping customers who are considering migration from other database management systems to DB2.

Just as there are techniques to ensure efficient access paths using SQL, there are ways you can maximize the efficiency and reduce the costs of UDFs. We have compared the performance of coding functions within your application program against DB2 built-in functions and UDFs. We have also compared the efficiency of coding external UDFs as a main program or sub-program.

For the performance measurements, we took a simple example of converting a string to lower case. This was done in three ways:

- Using the DB2 built-in function `LCASE(string)`
- Retrieving the string variable into the host program and using programming logic to convert the string to lower case
- Defining an external user-defined function to perform the character translation

The following section, 2.2.1, “Built-in or application program” on page 16, compares the performance of these techniques.

Built-in or application program



LCASE built-in function

```
EXEC SQL declare c1 cursor for
select LCASE(col1)
from table_name
while(SQLCODE==0)
{
EXEC SQL fetch c1 into :hv
}
```

invoke DB2 built-in function to perform case translation in cursor

Elapsed (sec)	CPU (sec)
26.2	12.2

Code equivalent of LCASE

```
EXEC SQL declare c1 cursor for
select col1
from table_name
while(SQLCODE==0)
{
EXEC SQL fetch c1 into :hv
for (count = 0; count < strlen(hv); count++)
hv[count] = tolower(hv[count]);
}
```

get data back from DB2 and use application logic to perform case translation

Elapsed (sec)	CPU (sec)
32.9	13.5

DB2 built-in function marginally more efficient

2.2.1 Built-in or application program

Here, we consider how you would convert a string from upper or mixed to lower case. On the left side, we have used the LCASE built-in function as part of the declare cursor statement and then retrieve all rows from the table. On the right side, we have coded the same functionality within an application program retrieving all the raw data from DB2 into a cursor.

The table definition is:

```
CREATE TABLE ACTIVLOG01 (ACTTABLN CHAR(8) NOT NULL,
ACTITEMN CHAR(14) NOT NULL,
ACTVDATE CHAR(2) NOT NULL,
ACTDESCR CHAR(7) NOT NULL,
ACTLTIME CHAR(4) NOT NULL)
IN DBITRK02.TSACTI01;
```

The table contains 510,000 rows that are retrieved sequentially. The measurements were run on a G6 processor, 9672-ZZ7, with OS/390 V2R7 using DB2 PM accounting trace.

Elapsed and CPU time in seconds for processing a large table are shown. For this simple function, the built-in function is marginally even more efficient. s compared with writing your own code, you may want to consider that using the LCASE function within the SQL call is easier to write, read, maintain, and it may perform better

External UDF or built-in function



LCASE built-in function

```
EXEC SQL declare c1 cursor for
select LCASE(col1)
from table_name
while(SQLCODE==0)
{
EXEC SQL fetch c1 into :hv
}
```

invoke DB2 built-in function to
perform case translation in cursor

Elapsed (sec)	CPU (sec)
26.2	12.2

UDF equivalent of LCASE

```
EXEC SQL declare c1 cursor for
select ibmtest1.udfx1(col1)
from table_name
while(SQLCODE==0)
{
EXEC SQL fetch c1 into :hv
}
```

call external user defined function
to perform case translation

Type	Elapsed (sec)	CPU (sec)
subroutine	93.6	93.2
main	108.9	108.3

DB2 built-in function much more efficient

2.2.2 External UDF or built-in function

Again, we are considering different ways in which you can perform string translation from upper to lower case. On the left side, as before, we have called the built-in DB2 function LCASE. On the right side, a call is made to an external user-defined function.

To create your user-defined function, you need to do the following things:

1. Write application code which carries out your function and place the load module into a library accessible to the WLM address space in which the function executes.

Since the UDF is performing the same task as the application program, which does the translation after receiving the raw data (see previous diagram), the piece of C code that does the case conversion is identical to the way it was written before. In both cases, the code is as follows.

```
void udfx1(char *parml, char *result,
short *F_indl, short *f_indr,
char *udf_sqlstate, char *udf_fname,
char *udf_specname, char *udf_msgtext,
struct sql_dbinf *udf_dbinf)
{
for (count = 0; count < strlen(hv); count++)
hv[count] = tolower(hv[count]);
}
```

2. Use the SQL create function statement to define the function and specify its characteristics to DB2. The SQL statement we used to create the function as a subroutine was as follows.

```
create function ibmtest1.udfx1 (char(14))
returns char(14)
external name udfx1
not null call
language c stay resident yes no sql
parameter style db2sql
deterministic fenced no scratchpad
no final call
wlm environment wlmenv1
program type sub
external security db2
no dbinfo;
```

In this case the program type was a subroutine; we also repeated the tests specifying a program type of main.

3. In the calling application program, you invoke your UDF exactly as if it were a built-in function.

The results show that the cost of using DB2's built-in function is significantly cheaper in terms of CPU time than invoking an external UDF. Also note that the cost of invoking an external UDF as a sub-routine is more efficient than calling it as a main program. This is because of differences in the way the language environment handles main programs and sub-routines. It is also worth noting that this comparison is a worst case scenario for the UDF because we have implemented a very simple function. As functions become more complex the percentage of overhead tends to decrease.

Maximizing UDF efficiency



Eliminate WLM address space creation

- choose currently existing WLM address space

Allow single copy of UDF code to be shared

- multiple invocations of a UDF can share code (if reentrant)
- STAY RESIDENT option of CREATE FUNCTION
 - ▶ default is NO
 - ▶ if load module is reentrant, specify STAY RESIDENT YES
 - ▶ load modules remain in storage after loading

Keep number of input parameters to the minimum

Tune SQL with UDFs: don't forget EXPLAIN

Ensure UDF code is efficient

- invoke as subroutine
- code pragmas correctly

Exploit DB2's functionality

2.2.3 Maximizing UDF efficiency

The difference between the cost of DB2's built-in functions and a user-defined function can be understood when it is remembered that the UDF is fenced, by definition. It does not execute within the DB2 address spaces. This protects the integrity of DB2 from application code. Your external UDF executes under LE in a WLM address space. Conversely, DB2 built-in functions are a component of the data base engine. Therefore, overhead is necessarily associated with external UDFs.

We have found, however, that there are several ways you can improve the efficiency of external UDFs:

- You can avoid the cost of WLM address space creation if you use an existing WLM address space. This may not always be possible, though, if you have a requirement to isolate different workloads and applications.
- If you can, code your load module as reentrant. This will allow you to override the default NO of the STAY RESIDENT option of the CREATE FUNCTION statement. If you specify YES:
 - The load module remains in storage after having been loaded.
 - This one copy can then be shared across multiple invocations of the UDF.

The impact of STAY RESIDENT YES is very important if multiple instances of a UDF are specified in the same SQL statement.

- There is overhead processing for each input parameter, so keep the number to the minimum required.

- Remember that, just as with built-in functions, or with any change to your application, the access path chosen by DB2 can be affected by an external UDF. A statement that is indexable without the function may become non-indexable adding a non properly coded function. There are two obvious cases in which the statement can become non-indexable:
 - the UDF is returning a CHAR value with length different from the one that is compared to.
 - the UDF is returning a nullable result and the compared value is not nullable.

We strongly recommend that you use EXPLAIN to determine whether the access path is what you expect, and whether it is as efficient as it can be. If you think the UDF is preventing DB2 from choosing an efficient access path, experiment by coding the statement with and without the UDF. This will help you understand the impact of the UDF on the access path.

UDFs have been fully integrated into the SQL language. This means that the UDF call can appear anywhere in the statement. Also, a single SQL statement can often be written in different ways and still achieve the same result. Use this to:

- Ensure that the access path is efficient.
- Code the SQL statement such that the UDF processes the fewest rows. This will reduce the cost of the statement.
- Exploit the fact that the architecture of LE makes processing subroutines more efficient than main programs by defining the program type as SUB.
- It is evident that you should make your UDF application code as efficient as possible. Two frequently overlooked opportunities to maximize efficiency are:
 - Ensure that all variable types match. This ensures that additional overhead is not incurred within LE performing unnecessary conversion.
 - Ensure that pragmas are coded correctly. An example of the pragma statement in a C program is shown in a full listing in Appendix B, “Sample external user defined function” on page 265.
- Since the cost of DB2 built-in functions is low, exploit them wherever possible. We will consider techniques to help you do this in the next section.

Use built-in functions



- existing built-in functions powerful
- can be used together

```
create table paolor8.addr
(address_line_1 char(40) not null)
in paolor8.addr;
insert into addr values('112, Malvern Wells Road');
insert into addr values('97 , Church Street');
insert into addr values('1878, Eaton Road');
insert into addr values('11 , 1st Street');
create view paolor8.addr_details
(house_number, address_line_1)
as select int(
    substr(address_line_1,1,
        posstr(address_line_1,',')-1
    ), address_line_1
from addr;
```

sample base table

house number found before comma
but format of data unpredictable

- use power of DB2 built-in functions
- view can mask complexity
 - get position of ,
 - substring from 1 to that point
 - convert to integer

Results

```
select * from addr_details;
-----+-----+-----
HOUSE_NUMBER ADDRESS_LINE_1
-----+-----+-----
112          112, Malvern Wells Road
97           97 , Church Street
1878         1878, Eaton Road
11           11 , 1st Street
DSNE610I NUMBER OF ROWS DISPLAYED IS 4
```

2.2.4 Use built-in functions

There are a large number of DB2 built-in functions which perform very well. These functions can be used together. Built-in functions are also the basis for sourced UDFs.

Casting can help perform translation between different DB2 and user-defined data types. Before you start coding your own functions, evaluate what is supplied with DB2 and understand how to use it. This allows you to:

- Maximize the efficiency of your application. Consider here not just the cost of executing your external function compared to DB2's built-in functions, but the best access path that can be achieved with a UDF as compared to a DB2 built-in function. For instance, a UDF can be stage 2 when compared to an equivalent stage 1 built-in function.
- Improve your productivity, as you do not need to develop and maintain your own code.

Just as you would consider how to code SQL to get the best access path, consider the best way to develop the UDFs you require.

The diagram above shows an example, albeit artificial, of how DB2 built-in functions can be nested. Assume that you want to know the house number (as an integer), but in your address table, you combined the house number with the first part of the address in the address_line_1 column. The functions posstr, substr, and int can be combined to isolate the house number.

The function posstr determines the location of string “,” within the address line. Subtracting 1 from the value returned gives you the end of the house number. You can then use substr to extract the house number starting at character 1 to that point. To convert this string to a number, you can use the int function.

If you needed to code this select statement many times, you could “hide” the complexity in a view. We have created a view called addr_details which includes the column house_number. This column is derived from address_line_1 using the posstr, substr, and int functions. On the right you can see the result of a select statement from this view which shows both the raw data and the derived value.

The power of views in masking physical data structures from applications should not be underestimated. Suppose you are converting from a non-DB2 data base management system and you have a table created with the following definition:

```
create table region
(region_id      integer      not null,
 region_name    char(40)     not null)
in region;
```

Also, suppose that the application calls a function CHARNSI that converts the region_id column from integer to a character (perhaps for display purposes). Your application will have code in it and would produce output as follows:

```
select charnsi(region_id),region_name from region;
REGION_ID      REGION_NAME
-----+-----+-----+-----+-----+-----+
1              North America
2              Europe
3              Australasia
4              Middle East
5              Africa
DSNE610I NUMBER OF ROWS DISPLAYED IS 5
```

If you were to rename the base table to tregion and create a view as follows:

```
create view region ("charnsi(region_id)",region_name)
as select char(region_id),region_name
from tregion;
```

Then a view would be created whose first column name was the string "charnsi(region_id)". You could then select from the view as follows.

```
select "charnsi(region_id)" as region_id,region_name from vregion;
REGION_ID      REGION_NAME
-----+-----+-----+
1              North America
2              Europe
3              Australasia
4              Middle East
5              Africa
```

This simulates the presence of a function CHARNSI operating on the region_id column of the base table. This technique illustrates what can be achieved, although we would not recommend its wide use because of these restrictions:

- The maximum column length is limited to 18 characters
- You would need to rename the base table
- And the application would need to put quotes around the column names

Consider sourced functions



Sourced functions are as efficient as built-in functions

- Convert smallint to string
 - ▶ External UDF
 - potentially attractive for conversions to DB2
 - C code to convert smallint to string `sprintf(pOut, "%-d", *p1In);`
 - ▶ Sourced UDF
 - superior performance compared to external UDFs
 - as attractive for conversions
 - ▶ Use the CAST function or built-in functions
 - excellent performance
 - less attractive for conversion - application changes required

CHARNSI as sourced UDF

```
create function charnsi (decimal(6,0)) returns varchar(32)
source sysibm.smallint(decimal(6,0));
```

```
select charnsi(4899) from sysibm.sysdummy1;
-----+-----+-----+-----+
4899
```

```
select substr(charnsi(4899),1,2) from sysibm.sysdummy1;
-----+-----+-----+-----+
48
```

CHARNSI using CAST

```
select substr(cast(4898 as char(6)),1,2) from
sysibm.sysdummy1;
```

```
-----+-----+-----+-----+
48
```

CHARNSI using built-in

```
select substr(char(4889),1,2)
from sysibm.sysdummy1;
```

2.2.5 Consider sourced functions

When you want to determine the most efficient way to code your function, consider whether you can source your function from the built-in DB2 functions.

Taking the example of a need to translate a smallint data type to a character for some subsequent string-based manipulation, you have several options, depending on your precise requirements:

- Write an external UDF.

This may appear a highly attractive option if, for example, you are converting from another data base management system to DB2. The application might extensively use a function that is called something different, or behaves slightly differently from DB2's version of the same function. Suppose, for example, the function used by the application to convert smallint data to a string is called CHARNSI. There is no function in DB2 with this name. To reduce the need to alter application code, you could code your own external UDF in a host language. The application will then run without change and invoke your UDF.

The diagram above shows the line of C code that performs this functions. The full source along with the CREATE FUNCTION SQL necessary is given in Appendix B, "Sample external user defined function" on page 265.

- Create a sourced UDF.

Since a sourced UDF is based on an internal DB2 built-in function, you can expect comparable performance. There is no call to LE, and the UDF does not need to execute under a WLM environment. In the diagram above, we show how you might code the sourced UDF. It can be called CHARNSI, which would satisfy your requirement that the application could be readily converted to DB2.

- Use the CAST function or use DB2 built-in functions.

Both of these options are illustrated in the diagram above. You can expect good and comparable performance from both. The disadvantage, if you are converting from another data base management system, is that application code will need to be changed.

If you need to change application code anyway or choose to do it for other reasons, then we recommend switching to DB2 built-in functions.

UDF summary



Considerably enhance power of SQL language

Important tool to aid migration to DB2

Use them if they work for you

Understand and manage overheads

- DB2 built-in functions cheaper than external UDFs
- Sourced functions as cheap as built-in functions
- Reduce WLM overheads if practicable in your environment
- Code efficient UDFs
- Always use EXPLAIN
- UDFs as sub-routines are more efficient than as main programs
- Make UDFs reentrant and specify STAY RESIDENT YES

Evaluate in your environment pre-production

2.2.6 UDF summary

UDFs are a very powerful extension to the SQL language, and they ease considerably the task of migrating to DB2 from other data base management systems.

We recommend reviewing all the techniques we have suggested, as this will help you exploit this enhancement as efficiently as possible.

Common with the exploitation of any feature, if resources are constrained and/or high performance is a critical success factor, we recommend that you carry out your own benchmark tests before you move to production.

Trigger performance considerations



Define a set of actions that are executed when a delete, insert or update operation occurs on a specified table

Encapsulate logic into the database

- To enforce data integrity rules
- Assist migrations from other DBMS

Can be defined as BEFORE or AFTER triggers

Simple example:

```
CREATE TRIGGER ROW_UPT
AFTER UPDATE ON EMPLOYEE
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    INSERT INTO CHANGE_LOG
    VALUES ('ROW_UPT INVOKED');
END
```

2.3 Trigger performance considerations

Trigger functionality was introduced in the DB2 V6 base code. Refer to the standard DB2 for OS/390 documentation for details. Triggers bring processing logic into the database by automatically executing a set of SQL statements whenever a specified SQL event occurs.

You can define triggers to validate and edit database changes, read and modify a database, or invoke functions that perform operations both inside and outside the database.

Their main benefits are:

- Enforce data integrity rules. No matter what application performs inserts, updates, or deletes on a table, you will then be certain that the associated business rules will be carried out. This is especially important with highly distributed applications.
- Enables migration from other DBMSs which have trigger support.
- Faster application development. Because triggers are stored in the database, the functionality coded in the trigger does not need to be coded in every application.
- Code reusability. A trigger can be defined once on a table, and is used by every application that accesses the table.
- Easier maintenance. If the business rules change, the applications do not need to be changed.

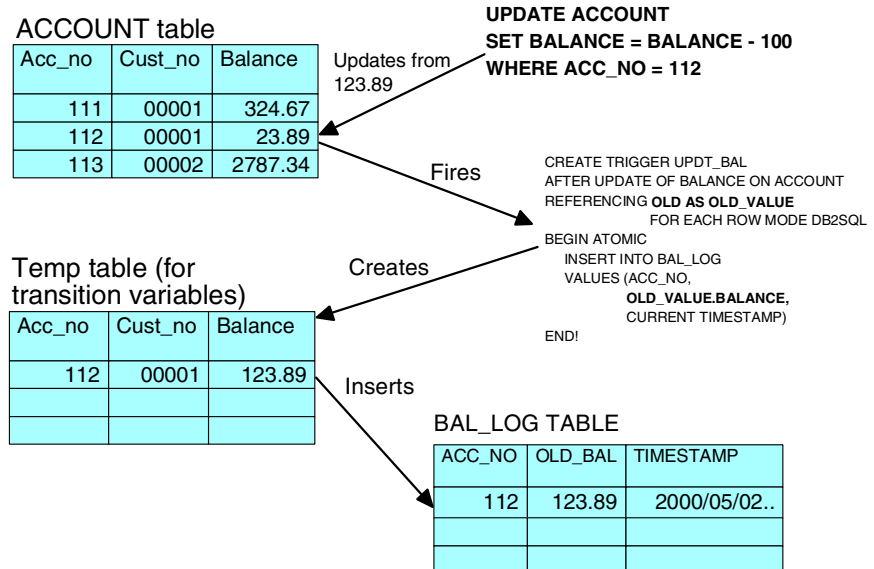
A trigger is defined on a table using the CREATE TRIGGER statement (a simple example is given above). It specifies the condition on which the trigger is to be activated, and whether the condition should be checked for each row modified by the triggering statement, or just once for each statement execution.

You can specify BEFORE or AFTER to determine when the trigger is activated. BEFORE triggers are activated prior to any updates being made to the triggering table. BEFORE triggers cannot activate any other triggers. AFTER triggers are activated after DB2 has made changes to the triggering table and can activate other triggers.

The trigger body consists of the set of statements that should be executed when the condition is met. It is delimited by BEGIN and END.

This section includes performance related advice on triggers. For detailed information on how to create a trigger please, refer to the *DB2 UDB for OS/390 Version 6 SQL Reference*, SC26-9014.

Trigger overview



2.3.1 Trigger overview

This section gives an overview of how triggers work before discussing the performance issues.

In the diagram above, the table ACCOUNT has an AFTER trigger defined on it. This is designed to insert a row into the BAL_LOG table when the BALANCE column is updated. The information logged will include the old balance (the pre-updated) value.

In our example, the UPDATE statement updates a balance value from 123.89 to 23.89, which causes the trigger to be executed. The INSERT statement within the trigger requires the pre-updated balance in order to insert it into the BAL_LOG table. It is able to reference this, as the trigger includes the REFERENCING OLD AS OLD_VALUE clause. This causes DB2 to store the pre-updated data in a temporary table. It will populate it with only those rows that have been processed by the update. DB2 uses this table as a work area and is able to use the values in the SQL statements in the trigger body. The temporary table used for trigger processing is created implicitly by DB2 using a workfile.

Transition variables



To refer to the value of columns of the row in the set of affected rows for which the trigger is currently executing

OLD refers to original value, **NEW** refers to the value it will be updated to

```
CREATE TRIGGER BUDG_ADJ
AFTER UPDATE OF EMPSALARY ON EMPLOYEE
REFERENCING OLD AS OD_ROW
NEW AS NU_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
  UPDATE DEPT
    SET DEPTBUDG = DEPTBUDG + (NU_ROW.EMPSALARY
                              - OD_ROW.EMPSALARY)
  WHERE DEPTID = NU_ROW.EMPDPID
END!
```

AFTER triggers use workfiles to process them

- image of entire row captured in workfile transition table
- one workfile for NEW another for OLD

2.3.2 Transition variables

If you specify FOR EACH ROW, you can use transition variables to refer to the values of columns in each updated row in the triggering table. You can do this by specifying the REFERENCING clause in the CREATE TRIGGER statement.

OLD transition variables capture the value of the columns before the triggering SQL statement updates them. NEW transition variables capture the values of the columns after the triggering statement updates them.

For AFTER triggers, transition variables are processed using workfiles. A temporary table is created to hold the transition variables. If you have NEW and OLD transition variables, two temporary tables are created. These temporary tables are similar to created temporary tables in their use of workfile space and lack of logging. However, these tables are not defined in the DB2 catalog.

The layout of the temporary table is the same as that of the triggered table — the image captured in the temporary table is the entire row, not just the transition variables. This means that the longer the row length, the greater the overhead of a trigger, regardless of the number and size of the transition variable(s) used.

Consequently, before implementing triggers on a large scale, you should review the size allocations of your workfile data sets. You should note that the temporary table is defined for the full row length, so it could be quite large if varchar columns are defined on the triggering table.

Please note that workfiles are not required to process transition variables in BEFORE triggers.

Transition table



A trigger may refer to the set of affected rows by using a transition table - OLD_TABLE & NEW_TABLE

Valid for row and statement triggers

```
CREATE TRIGGER BIG_SHOT
AFTER UPDATE ON EMPLOYEE
REFERENCING NEW_TABLE AS NU_TABLE
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
  INSERT INTO BIG_SHOTS
  SELECT NU_TABLE.EMPID FROM NU_TABLE
  WHERE (NU_TABLE.EMPSALARY > 100000.00);
END!
```

Overhead is similar to that for a transition variable

Additional workfile will be used if there is sort processing required in the expression of the where clause

2.3.3 Transition tables

Transition tables can be used to refer to the entire set of rows that the triggering statement modifies. In the same way as transition variables, you can define them in the REFERENCING clause of the CREATE TRIGGER statement.

The performance overhead of invoking the trigger, and allocating/deallocating the temporary table is similar between using transition variable and transition table, but the cost of referencing the transition table depends on the amount of data stored in the transition table. The new transition table contains the full set of the updated or inserted rows, even it is referenced in a row trigger. For each row triggered, DB2 scans the full set of the affected rows in the temporary transition table: referencing a large temporary transition table can be costly.

Transition table is useful when it is necessary to refer to the whole set of affected rows. For example, applying aggregations (MAX, MIN or AVG) of some column values.

Like transition variables, the overhead will be doubled if you reference OLD and NEW tables. Therefore, you should consider their use carefully and not reference them unless really necessary.

Since transition tables are stored within temporary tables additional workfile will be used if sort processing is required to satisfy the expression of a WHERE clause.

Transition variable/table usage



Consider their use carefully as they significantly impact the trigger overhead

- Overhead includes workfile allocation, use and deletion
- Overhead doubles approximately if OLD and NEW variables are referenced
- Overhead is much less in BEFORE triggers

Ensure that you have enough workfile space

- Transition variables and tables are processed using workfiles
- Make sure you have enough space to accommodate all concurrent trigger processing in addition to other created temporary table usage.
- Make sure you have enough workfiles spread across DASD to avoid contention problems

2.3.4 Transition variable/table usage

The cost of a trigger is increased if transition tables or variables are used. Acquiring locks on the workfile database, allocating space, performing any required I/O, and deallocation of these resources represents a significant part of the cost of a trigger. If you do not use transition variables/tables, the cost of your trigger will be much less. Similarly, if the logic allows you to use BEFORE triggers, most of the overhead will be avoided, as they do not require workfiles.

In addition to the processing overhead, the use of transition variables increases the workfile requirements. DB2 allocates 24 pages of workfile space for each temporary table and then allocates more if required. You will need to reassess your workfile data set allocations to accommodate the maximum concurrent trigger activity in addition to existing workfile usage. Note that the workfile space used to manipulate transition variables and tables can span workfile data sets.

A general recommendation we have found to be successful is to have at least 5 workfile data sets of sufficient size to accommodate all workfile usage. This includes space required for sorting, creating temporary tables, sysplex query parallelism, and trigger processing.

Unlike other table spaces, secondary extents do not help the space management of the data sets, as the data is temporary. A large sort could cause all secondary extents to be allocated, and these extents will not be released unless the data sets are redefined. The fact that there are many extents does not necessarily indicate a space shortage. Consequently, we recommend that you set the primary quantity large enough to accommodate the workload, and set the secondary quantity to 0.

You should also make sure that the workfile data sets are spread across disks to avoid I/O contention, and if possible, do not place them on the same devices as other critical system data sets, such as the active logs.

Row trigger or statement trigger



Row trigger:

```
CREATE TRIGGER ROW_UPT
AFTER UPDATE ON EMPLOYEE
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    INSERT INTO CHANGE_LOG
    VALUES ('ROW_UPT INVOKED');
END!
```

Statement trigger:

```
CREATE TRIGGER STM_UPT
AFTER UPDATE ON EMPLOYEE
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
    INSERT INTO CHANGE_LOG
    VALUES ('STM_UPT INVOKED');
```

Generally statement triggers perform better

- consider SQL statement execution and performance
- EXPLAIN SQL statements executed

2.3.5 Row trigger or statement trigger

You can define the scope of a trigger as either a row trigger or statement trigger.

FOR EACH ROW: The trigger is activated once for each row that is modified in the triggering table. If DB2 modifies no rows, the trigger is not activated.

FOR EACH STATEMENT: Applies only to AFTER triggers. The trigger is activated once when the statement is executed, even if no rows are modified.

The primary factor that influences your decision as to which to use should be based on your processing logic requirements. Where you have a choice between implementing a row or statement trigger, consider the performance and resource utilization of the two options. Three factors to be considered are:

- The access path of the SQL statements that will be executed.

It is likely that the SQL you need to code for a row and an equivalent statement trigger will be different. Use EXPLAIN and/or the DB2 Estimator to help you evaluate the relative costs of the SQL statements.

- The number of times the SQL in the trigger body is executed.

To execute a statement trigger, the trigger manager invokes SQL statements in the trigger body once. Conversely, to execute a row trigger, the SQL statements will be executed once for each row that satisfies the trigger condition. Therefore, the cost of a statement trigger tends to be cheaper than an equivalent row trigger.

- Whether transition variables or tables are required.

Processing transition variables and transition tables is expensive. If you can code triggers to avoid them, you will improve performance and reduce the resource required to execute your triggers considerably.

Trigger coding considerations



Nesting triggers

- up to 16 levels allowed
- coding restrictions
 - if a table is being modified it cannot be referenced in a lower level nested trigger
 - if any table is being accessed by a select statement, that table cannot be modified in any lower level nesting SQL statement

Explain

- create trigger creates a trigger package
- COST_CATEGORY 'B' in DSN_STATEMNT_TABLE
- predictive governor cost category B - cost indeterminate
- REBIND trigger package with EXPLAIN(YES) to check access path

Rebind trigger packages when necessary

- keep access path history
- helps diagnose performance problems (access path change)

2.3.6 Trigger coding considerations

When an SQL statement in a trigger body is executed, it may cause another (or even the same) trigger to be fired for AFTER triggers. This in turn could cause another trigger to be activated. When we examined the performance implications of the recursive calling of triggers, we found there were no costs above the overhead of executing each of the triggers. DB2 will allow up to 16 levels of nesting.

Please note the following rules to avoid an SQLCODE -746

- If a table is being modified by insert, update, or delete, it cannot be accessed by a nested trigger below it.
- If any table is being accessed by a SELECT statement, no table can be modified in any lower level nesting SQL statement.

Like referential integrity, triggered SQL statements do not show up when the triggering SQL statement is explained. EXPLAIN will record the statement as being in COST_CATEGORY 'B' with a REASON of 'TRIGGERS' in the DSN_STATEMNT_TABLE. For the DB2 governor, the RLF_CATEGORY_B predictive governor rules apply. This indicates that the predictive cost of the statement is indeterminate.

When you create a trigger, DB2 automatically creates a trigger package with the same name as the trigger name. The collection name of the trigger package is the schema name of the trigger. Multiple versions of the trigger package are not allowed. You can REBIND this trigger package with the EXPLAIN(YES) option to obtain information about the access path.

Since the cost of the statement is indeterminant, it is easy to miss the cost of the execution of a trigger unless you make a point of checking it when the trigger is created and/or after a REBIND of the trigger package with EXPLAIN(YES).

We recommend that you treat trigger packages in the same way as standard packages in that you REBIND them when you REBIND other types of package, for example, when there are significant changes to the statistics. This will ensure that access paths are based on accurate information.

We also recommend that you keep access path history information. That way, you can see if any performance degradation can be correlated with a change in access path.

Understanding trigger performance



Triggers overheads

- Base cost to process a trigger is about same as a FETCH
- Must load trigger package into the EDM pool
- Creation, use and deletion of workfiles for transition variables/tables
- So...
 - ☐ Monitor your EDM pool and DASD requests
 - ☐ Review release(deallocate) option - but consider implications
 - ☐ Avoid workfiles if possible - minimize use of transition variables/tables
 - ☐ Row length influences size of workfiles
 - ☐ Consider design
 - if trigger does not fire cost negligible
 - update trigger has negligible cost if most activity is insert

Compare trigger cost with application implementation

Consider stored procedure rather than multiple triggers

Use triggers for what they are designed - not a replacement for DPROPR or constraints

2.3.7 Understanding trigger performance

Triggers enable you to encapsulate business logic into the database, and this has many advantages. They are also helpful for customers considering migration from other data base management systems that have triggers to DB2. Here we review factors that influence the cost of triggers. Understanding these factors will help you evaluate the likely cost of triggers and estimate their cost relative to an application implementation of the logic.

- The base cost of a trigger (that is, excluding the execution of the triggered SQL) is about equivalent to the cost of a fetch. Where a trigger is defined but not fired, for example, if the trigger is defined as ON UPDATE OF column_1 and an update updates column_2, the trigger is not activated, and so the overhead is normally negligible. However, very complicated and badly coded WHEN clause in the trigger can still impact performance whether or not the trigger is fired.
- The trigger package has to be loaded into the EDM pool. I/O will have to occur if it is not already in the EDM pool. Options to alleviate problems in this area include:
 - Monitor and increase the size of the EDM pool.
 - Consider REBIND of the trigger package with the `RELEASE (DEALLOCATE)` option. Be aware, though, that `RELEASE (DEALLOCATE)` will result in more resources being held — you face the same issue as binding application packages with `RELEASE (DEALLOCATE)`.
- The object descriptor (OBD) of the trigger package is a part of the table. Therefore, be aware of the impact to the DBD size and its potential to impact the EDM pool if you already have large DBDs.

- As we have indicated, transition variables and transition tables in AFTER triggers represent a significant proportion of the cost of such a trigger. Where a trigger has to manipulate workfiles, contributions to the cost are:

- Creation, use, and deletion of workfiles
- Any I/O and/or GETPAGE requests necessary to process the data
- Base cost of trigger processing

The cost of a trigger that processes workfiles depends critically on the amount of workfile processing required (including row length and whether you have OLD and NEW transition variables). Transition tables need to be processed with a table space scan for each row trigger because there are no indexes. Trigger performance will also depend on whether there is contention for the workfiles. You should anticipate the cost to be several times greater than the base cost of the trigger, because significantly more work has to be carried out.

- There is no overhead for an SQL statement that is not the triggering action. For example, if a INSERT trigger was defined on a table, it would have no overhead on update or delete statements.

We recommend that you prototype your physical design first if you are considering using triggers for tables that are heavily updated and/or fire SQL statements that process significant quantities of data. You can then evaluate their cost relative to the cost of equivalent functionality embedded in your applications.

When you begin physical design, you may find that you need several triggers defined on a single table. When there are multiple triggers on the same table with the same event and same execution time, they will be processed in the order that they were created. You can determine what the order is by looking at the creation timestamp in the catalog.

To avoid the overhead of multiple triggers, you can write a stored procedure to do all the triggered processing logic. The body of the trigger could then simply consist of a CALL stored-procedure-name.

Please note that triggers are not the best solution for all situations. If the triggered logic is just for validation purposes, you can achieve better performance using constraints or views with the CHECK option. In addition, although triggers can be used for simple propagation (for example, to create an audit trail), they are not intended to be used as an alternative to or a replacement for Data Propagator.

Note: APAR PQ34506 provides an important performance improvement for triggers with a WHERE clause and a subselect. A where clause in the subselect can now be evaluated as a Stage 1 predicate.

LOBs performance considerations



- Overview
- Processing
- Read performance
- Update performance
- Summary

2.4 LOBs performance considerations

Large objects (LOBs) are string data elements used to store binary data such as documents and videos. They differ from CHAR and VARCHAR in that a single object (that is, a single column of a table containing a LOB) can be up to 2 GB in size. In this section we briefly introduce the topic, then we look at read and update performance, and provide recommendations.

LOBs overview



Base table space

Base table			
Key	ROWID	Column_2	LOB indicator
Key A	ptr to LOB 1	user data A	LOB indicator 1
Key B	ptr to LOB 2	user data B	LOB indicator 2

- Rows represent LOBs
- LOBs stored outside base table in auxiliary table
- Base table space may be partitioned
- If so separate LOB table space for each part

Auxiliary index:
based on ROWID
used to navigate to LOB data

LOB table space

Auxiliary table	
ROWID	LOB data
LOB 1 ROWID	LOB data for row user data A
LOB 2 ROWID	LOB data for row user data B

2.4.1 LOBs overview

A LOB column is conceptually part of the base table, but it is physically stored in a separate table. Because it is not part of the base table, it is termed an auxiliary table. The auxiliary table resides in a separate LOB table space.

A base table may point to many LOB columns of different types and lengths. Each auxiliary column is stored in its own auxiliary (LOB) table in its own LOB table space. An auxiliary index must be created on every auxiliary table before it can be used.

To create a base table that contains a LOB column, you must define a ROWID column. The ROWID acts as a pointer to the LOB data associated with the particular row. The auxiliary index, whose key is based on the ROWID, is used to navigate to LOB data associated with the row.

If a base table that contains LOB data is partitioned, you create a separate LOB table space and auxiliary table for each partition.

A LOB table space can have a page size of 4, 8, 16 or 32 KB. Since the length of a LOB can exceed 32 KB it is clear that a LOB can span physical pages. To reduce the volume of logging, you can specify `LOG NO` in your `CREATE LOB TABLESPACE` statement. This suppresses redo records.

There are no UNDO records for LOB updates (except for system pages, spacemap) even with `LOG YES`. LOBs always insert the new value at a different place and delete the old one at commit marking the old space as free.

LOBs processing



V6 introduced BLOBs, CLOBs and DBCLOBs

To store data objects up to 2 GB in size

- audio, documents, pictures
- to store data that will not fit into 32 KB maximum page size

Programming with LOBs is challenging

- Buffer requirements may exceed capacity
- LOB locators introduced to process LOBs efficiently

Data manipulation

- Process as though (very long) strings
- Delete is a logical delete
- Update is logical delete and insert

2.4.2 LOBs processing

There are three data types that are used to store different large objects:

- Binary large objects (BLOBs) store binary data such as image, video and sound data. They have no CCSID associated with them.
- Character large objects (CLOBs) store character data which is larger than 32 KB such as documents. CLOBs have the normal single byte and mixed CCSID associated with them.
- Double-byte character string (DBCLOBs) store data that consists only of DBCS data and have the graphic CCSID associated with them.

The maximum size of a LOB column is 2 GB. You can also use LOBs to store data, for example, long characters, which does not fit entirely within DB2's largest page size of 32 KB.

Although you can manipulate LOB columns like any other data type, there are a number of issues:

- They are subject to the same restrictions as long VARCHARs. Substrings of LOBs which are less than 255 bytes can be CAST as CHAR to avoid these restrictions.
- Acquiring buffers in a program to accommodate a large LOB can be difficult.

Since LOBs are essentially long strings, you can use string functions to manipulate them and parts of them. For example, SUBSTR(LOB,1,200) will retrieve the first 200 bytes of the LOB. This can more easily be managed in an application program.

LOBs are inserted, selected, deleted, and updated using standard SQL, although there are some special considerations:

- A LOB locator, which is a 4-byte value stored in a host variable, can be used to reference the LOB, and can be used wherever you would use a LOB. The locator essentially acts as a pointer to the LOB. For full details, refer to *DB2 UDB for OS/390 Version 6 Application Programming and SQL Guide*, SC26-9004-01.
- The LOB update operation does not update in place, it performs a LOB delete followed by an insert.
- The LOB delete is a logical delete: it flags the space as available.

LOBs read performance



	NON-LOB 20 KB	20 KB	200 KB	2 MB
Select into :HV	0.025 (0.0006)	0.027 (0.0008)	0.117 (0.0023)	0.867 (0.0175)
KB/sec	800.0	740.7	1,709.4	2,306.8
Select into Locator		0.020 (0.0007)	0.029 (0.0007)	0.046 (0.0007)

All measurements in seconds

Elapsed time is on the top line, CPU time shown in brackets

Processor is LPAR of RX5

- **Large LOBs can be processed (KB/sec) more efficiently than small**
- **LOB locators offer superior performance and lower CPU overhead**
 - nearly 19 times faster elapsed
 - 25 time less CPU required
 - we recommend you use LOB locators

2.4.3 LOBs read performance

We have measured the efficiency of reading data from tables that contain LOB data. We created 4 different tables. The first was our control and consisted of a non-LOB 20 KB character column. The table space page size had to be 32 KB. Operations performed on this table provided the baseline against which we could compare the efficiency of LOB processing.

The second table was conceptually identical to the first except that the 20 KB character column was defined as a CLOB. We created a LOB table space and auxiliary table and index to support this. For our LOB table spaces, we used a 32 KB page size, so that comparisons with the non-LOB table were reasonable.

The third and fourth tables had LOB columns of 200 KB and 2 MB, respectively.

When we retrieved LOB data into the host application, we repeatedly called SUBSTR to “walk” down the entire length of the column. We used the same technique to process LOB data with and without the use of LOB locators. Any differences in the performance of processing LOBs can, therefore, be attributed solely to the use of locators or the processing of LOBs into host variables.

The diagram above shows the elapsed and CPU time taken to process non-LOB and LOB data of different lengths and shows the following results:

- LOB processing is more efficient for larger LOB columns.

The time taken to process a select of 20 KB worth of LOB data into a host variable is 0.027s. In other words DB2 can process 740.7 KB LOB data per second for a 20 KB LOB column. To select a 2 MB LOB into a host variable (that is, 100 times more data) does not take 100 times longer; rather, DB2 can process 2306.8 KB LOB data per second for a 2 MB LOB column.

- The efficiency by which large LOBs can be processed is increased enormously by using LOB locators.

With 20 KB of LOB data, there is only a small improvement in performance using LOB locators. At 2 MB we saw an it took nearly 19 times longer and used 25 times more CPU to select a LOB into a host variable rather than into a LOB locator.

- You get very consistent responses and CPU times reading LOB data using locator variables regardless of the size of the LOB column.

These measurements lead us to recommend that you use LOB locators when selecting LOB data.

LOBs write performance



	NON-LOB 20 KB	20 KB	200 KB	2 MB
Insert with :hv	0.011 (0.0007)	0.025 (0.0012)	0.219 (0.0033)	2.029 (0.0194)
Insert with locator		0.028 (0.0012)	0.187 (0.0028)	2.08 (0.0199)
Update (see notes)	0.0036 (0.0008)	0.027 (0.0013)	0.223 (0.0033)	2.229 (0.021)
Delete	0.03 (0.001)	0.024 (0.001)	0.035 (0.001)	0.045 (0.001)

All measurements in seconds

Elapsed time is on the top line, CPU time shown in brackets

2 MB insert takes about 2 seconds

- insert more efficient as length of LOB data increases

Delete is very quick - logical delete

Update is roughly the same as an insert

- internally consists of insert+delete so this makes sense

2.4.4 LOBs write performance

We also measured the performance of update, delete, and insert operations against LOB data. The scenario we used was the same as for LOB reads: one non-LOB table with a 20 KB CHAR as a control for the 20 KB LOB table, and tables with LOB sizes of 200 KB and 2 MB. The processor in a RX5 LPAR and LOG NO was specified for the LOB runs.

We examined the cost of an insert operation using a host variable compared with using a LOB locator with different sized LOBs. We also measured the cost of a delete and an update.

2.4.4.1 LOB insert

It is more expensive to process 20 KB of LOB data, even if a LOB locator is used, than processing an equivalent volume of non-LOB data. As with select processing, the greater the LOB length, the greater the efficiency of insert processing — to insert 100 times the data does not require 100 times the resource. The relative improvement of insert efficiency with respect to LOB size results, however, much less dramatic than for select processing. The reason is that the cost of preformatting disk space to accommodate the newly inserted LOB increases with the size of the LOB and it accounts for most of the elapsed time.

There is relatively little difference in insert performance if a LOB locator is used.

2.4.4.2 LOB update

Internally a LOB update is equivalent to a LOB insert and a LOB delete. Most of the cost of a LOB update can, therefore, be attributed to the cost of the LOB insert.

2.4.4.3 LOB delete

A LOB delete is a logical delete, and so is relatively cheap and gives consistently good performance. The CPU cost was independent of the size of the LOB column (to 2MB). The larger the LOB column, the greater the number of space map pages that have to be updated to reflect the logically deleted row. There is, therefore, a relationship between elapsed time and LOB size for deletes. The larger the LOB column, the greater the efficiency of the delete operation. DB2 can delete 833 KB per second if the LOB is 20 KB in length. With a 2 MB LOB, DB2 can delete 45,511 KB per second — more than 50 times faster.

LOBs recommendations



**Use them for what they are intended for - over 32 KB objects
(there is only one row per page with LOBs)**

Use locators to process them

Do not use as a normalization technique

Do not use them as a technique to avoid logging

Expect Load utility overhead

Tune GRS for data sharing

2.4.5 LOBs recommendations

- Use LOBS for what they are intended — large (>32 KB) objects.

On balance, we observed that the larger the LOB, the greater the efficiency. LOB insert (and therefore update) processing is expensive for small length LOB columns. We recommend, therefore, that you use LOBs only for the purpose for which they were intended — processing large objects over 32 KB in size. You must take into account that there is only 1 LOB per page; if you have specified a pagesize of 32 KB, and a row of 100 bytes, the overhead in dealing with each row in terms of I/O and CPU cannot be negligible.

- Use locators to manipulate LOBs.

LOB locators process LOB data consistently and more efficiently than using host variables and avoid application buffering problems.

- Do not use LOBS as a means of normalizing data.

When running a complex workload, we noticed a puzzling improvement in overall throughput when we replaced the 20 KB VARCHAR column by a 20 KB LOB.

To illustrate what was happening, consider an employee table with base data of name, department number, employee identifier and so on. At the end of the row is a VARCHAR(20 KB) containing the employees curriculum vitae. Most applications did not read the curriculum vitae, but this long column contributes to the row length and thus reduces the number of rows per page. Therefore, the number of GETPAGE requests required to process the employee data was quite large.

When the 20 KB curriculum vitae column was defined as a LOB, it was located in a different table space that was rarely read. The hit rate for the frequently accessed base data, therefore, increased improving overall performance.

We do not recommend, however, that you use LOBs as a technique to obtain performance improvements that normalization and proper physical design can achieve. Note that:

- Any improvements are entirely application dependent.
- At the physical design stage, you should have been able to identify that the curriculum vitae column was infrequently accessed. There is a 1:1 relationship with the employee. Depending on the application, there may be a net benefit in placing the curriculum vitae into its own base table, keyed on employee number and holding the 20 KB VARCHAR and possibly other infrequently accessed columns.
- Do not use LOBs as a means to deactivate REDO logging.

The additional cost, particularly when processing small LOBs, means you are likely to create a different bottleneck from the perceived one of logging that you were trying to solve. Use LOBs only for their intended purpose.

- Expect overhead when running LOAD

You should be aware that the LOAD utility processes LOBs using insert mode processing. To estimate very roughly the elapsed time for LOAD, assume that the cost to load each row is equivalent to the cost of randomly inserting every row.

- Tune VSAM and GRS with data sharing

In data sharing environments, we observed some contention during drop LOB table space and unusually high Coupling Facility (CF) CPU activity. We could attribute this to VSAM making use of GRS. When we specified GRS=STAR, these problems were eliminated. We recommend that you investigate the GRS setting in data sharing environments if you are observing high CF CPU activity.

ESS performance



- New disk with excellent performance for DB2 data
- No special DB2 parameter to obtain improvements
- Two sets of measurements with DB2 are referenced to show:
 - read performance (random and prefetch)
 - logging write rates
- Larger cache and faster processors have been announced

See white paper DB2 for OS/390 Performance on IBM Enterprise Storage Server at <http://www.software.ibm.com/data/db2/os390/pdf/db2ess.pdf>

2.5 ESS performance

The IBM 2501 Enterprise Storage Server (ESS, but often called by its code name Shark) is a recent disk drive which can provide excellent performance in your DB2 environments. No special setting is needed from the DB2 side to exploit the new disk's capabilities. Refer to the ESS official documentations for its set up and management. For ESS performance management, you can refer to the redbook *IBM Enterprise Storage Server Performance Monitoring and Tuning Guide*, SG24-5656.

The measurements shown in this section show that the ESS disk offers a basic raw power in reads and writes that allows the I/O time to be approximately cut in half when compared with other IBM disk drives. For the DB2 applications and utilities that are I/O bound (like Copy and Recover) you can expect meaningful proportional improvements.

Other advantages can be expected from the Parallel Access Volume optional ESS feature, if software enabled, for those non-finely tuned environments where concurrent I/O processing is present. Common candidates for PAV can be the volumes where the DB2 workfiles of DSNDB07 are allocated.

ESS is a recent device, the measurements mentioned here were taken at Santa Teresa Lab in 4Q1999 and at the S/390 Terplex Center during the 1Q2000 on the first release of the product. Larger cache, faster processors, and advanced copy functions have been announced, more measurements are being implemented.

ESS read performance



Performance measurements of table space scan
against a 200 partition 100 GB table
Comparison of RVA and ESS

Processing	Disk	Rate in MB/sec
Parallel sequential prefetch DEGREE 50	1 ESS 2 RVAs	148.50 43.80
Sequential prefetch DEGREE 1	1 ESS 2 RVAs	12.40 4.50
Synchronous read DEGREE 1 (VPSEQT=0)	1 ESS 2 RVAs	2.80 1.85

2.5.1 ESS read performance

Performance measurements have been carried out at the S/390 Teraplex Center comparing table space scan times on ESS and RVA. The table involved was associated to a 10 GB table space with 200 partitions, and it is representative of a data warehousing type of application. These measurements give a clear indication of the 'raw' power of the new disk drives during sequential prefetch and synchronous reads. The hardware configuration for the measurements consisted of two RVA model 9393-T82 and one ESS model 2105-E20 attached to a 9672-YX6 system. The table space partitions were allocated as follows:

- Partitions distributed across 4 Logical Control Units (LCUs) of an ESS device. Each LCU has 8 channel path identifiers (CHPIDs)
- Partitions distributed across 2 RVAs. Each RVA has 8 CHPIDs.

When the query was run with DEGREE 1, the I/O rate in MB per second improved from 4.5 for the RVAs to 12 for the ESS.

When the query was run with DEGREE ANY the I/O rate improved from 12.4 MB/sec for the RVAs to 43.8 MB/sec for the ESS, showing a better improvement with higher parallelism. The actual maximum degree of parallelism was set to 50 by using the PARAMDEG installation parameter to control the contention from the possible 200 concurrent I/O streams.

In these two tests VPSEQT was set to the default value of 80, which means that sequentially accessed pages can take up to 80% of the virtual buffer pool. In the third test, this parameter was set to 0; this disables prefetch. Prefetch disabled means that instead of DB2 reading data into the buffer pool ahead of the program requiring the data, DB2 had to do synchronous reads for each page. Using the ESS device, the I/O rate was 2.8 Mb/sec, doubling the rate of the RVAs for a test indicative of random read accesses.

More detailed results of performance measurements of a table space scan against a 20 partition 10 GB table are also reported below.

Processing DEGREE 1	Disk Drive	Elapsed time (min)	CPU time (min:sec)	Other read I/O (min:sec)	I/O rate (MB/sec)
Prefetch VSPQT=80	1 ESS	5.1	4:13	0:54	12.40
	2 RVAs	14.5	4:12	10:18	4.50
Synchronous read VSPQT=0	1 ESS	22.7	5:26	15:50	2.80
	2 RVAs	34.0	5:24	29:06	1.85

When the query was run with sequential prefetch, without parallelism, the elapsed time was reduced from 14.5, for the RVAs, to 5.1 minutes, for the ESS, with the CPU time remaining constant.

In the case of single synchronous read (with VPSEQT set to 0) using the ESS device, the read time was approximately 22.7 minutes, as opposed to 34 minutes for the RVAs, showing the improvement indicative of random read accesses.

For information on the mission and activities of the Teraplex Center see the URL:

<http://www.ibm.com/businessintelligence/teraplex/index.htm>

DB2 logging rates by DASD type



DASD Subsystem Type	Logging rate (MB/sec)
3390, RAMAC 1, RAMAC 2	2.5
RVA2 Turbo	3.2
RAMAC 3	4.2
RVA X82	4.5
ESS	8.2

This table shows the maximum observed rate of writing to an active log data set, when using dual logging, with different storage disks and DB2 V6 under OS/390 V2R6.

The ESS offers the highest logging rate that can be sustained by a single DB2.

2.5.2 DB2 logging rates by disk type

For some particularly large throughput, heavy update profile, DB2 installations logging can become a bottleneck. As can be seen from these measurements, extracted from the *DB2 for OS/390 Performance on IBM Enterprise Storage Server* white paper, the ESS supports logging rates nearly double those possible under the previous IBM disks.

The white paper is available from the Web site:

<http://www.software.ibm.com/data/db2/os390/pdf/db2ess.pdf>

or it can be linked to, as a redpaper, from the ITSO Web site:

• <http://www.redbooks.ibm.com>

The referenced documents also covers performance benefits from using ESS for DB2 tables and indexes under the following headings:

- Query performance
- Utilities performance
- Transaction performance
- Distributed performance

Distributed functions performance



Recent measurements:

- TCP/IP with Netfinity Escon adapter
- Connection pooling
- Extra block query support

Referenced documentation at

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

2.6 Distributed functions performance

The following referenced white papers are all available from the Web site:

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

- TCP/IP with Netfinity Escon Adapter

In a client/server environment there is currently no faster, more scalable way to connect to DB2 for OS/390 other than with DB2 Connect, IBM's Netfinity adapter, and TCP/IP with MPC+ driver. TCP/IP performs better than SNA and uses less CPU on OS/390 in a client/server environment. This topic is covered in the white paper *TCP/IP and SNA Performance Using IBM's Netfinity Escon Adapter*.

- Connection pooling performance

Connection pooling, which was introduced with DB2 UDB for OS/390 V6, makes inactive thread processing more efficient. Threads executing packages bound with RELEASE(DEALLOCATE) can now become type 2 inactive threads. This leads to CPU savings since there is no reason to acquire and release intent locks after every commit. There is also less storage usage with inactive threads as compared to active threads. Although the benefit of connection pooling is less for long running inactive threads, there is better performance than using active threads. This is documented in the white paper *Performance Analysis of Connection Pooling in DB2 UDB for OS/390 Version 6*.

- Extra query block support performance

You can now retrieve large result sets of data during query processing due to extra query block support in DB2 UDB for OS/390 V6. This is done by specifying the number of rows to be returned on the optimize for n rows of the declare cursor SQL statement. The response time and rows returned would be affected by network limits (that is bandwidth, packet thresholds), processor speed (both client and server), memory, and other unrelated work performed.

Although there are no costs to DB2 for using extra block query, network bandwidth might be stressed. The anticipated performance benefits of extra query block support may not be realized if the network bandwidth is the limiting factor. This is documented in the white paper *Performance Analysis of Extra Query Block Support in DB2 UDB for OS/390 Version 6*.

Chapter 3. Application enhancements

Application enhancements



- Identity columns
- External savepoints
- Declared temporary tables
- Update with subselect
- Columns in order by not in select
- Global transaction

In this chapter we describe enhancements that have been made available through the maintenance stream of DB2 V6, and have been rolled in the May refresh. We look at functions that we have grouped under the heading application enhancements since they can be beneficial when developing applications:

- Identity columns
- External savepoints
- Declared temporary tables
- Update with subselect
- Columns in order by not in select
- Global transaction

Identity Columns



- New method of generating sequential primary key values
 - traditional methods include next number table, CICS TS queues
- Identity columns
 - Ideally suited for generating unique primary key values such as employee numbers, order numbers, line item numbers

```
CREATE TABLE CUSTOMER  
(CUST_NO INTEGER GENERATED BY DEFAULT AS IDENTITY  
          (START WITH 1000),  
FIRST_NAME CHAR(18) NOT NULL,  
LASTNAME CHAR(18) NOT NULL)...
```

Cust_no	First_name	Last_name
1000	Patrick	Vabre
1001	Charles	Ludeman
1002	Gordon	Bell

```
INSERT INTO CUSTOMER  
(CUST_NO, FIRST_NAME, LAST_NAME)  
VALUES (DEFAULT, :hv_fname, :hv_lname)
```

3.1 Identity columns

It is a common physical database design requirement to allocate a sequential number as the primary key of a table. Common examples include customer number, stock number, and order number. The primary key must be unique. If the table participates in referential relationships with other tables, this primary key is propagated to the dependent “child” tables as a foreign key.

When it comes to the implementation of your design, you have to decide how you are going to generate the next sequential number in the series. Traditional methods include having a single row DB2 table that holds the highest allocated number, or using a CICS Temporary Storage (TS) queue.

This enhancement adds an additional method with several advantages over existing techniques. It also assists with the migration of applications based on other data base management systems to DB2.

You can now specify a new column attribute **AS IDENTITY**. Then DB2 will automatically generate unique, sequential, and recoverable values for the column for each row in the table. A column defined in this way is referred to as an identity column. Identity columns are ideally suited for the task of generating unique primary key values.

In this section we will cover the following topics:

- How to define identity columns and design considerations
- Performance improvements you can obtain
- How to code for identity columns within applications

- ## Syntax

```

--column-name--data-type--'-column-options-'
COLUMN-OPTIONS:
<----->
|-----|
| -NOT NULL-----|
| -.UNIQUE-----|
| '-PRIMARY KEY-'|
| -FIELDPROC--program-name--|
|                                     | <-,-----< |
|                                     |'-(--constant--)-'|
|-----|
| -references-clause-----|
| -check-constraint-----|
| -generated-column-spec-----|
GENERATED-COLUMN-SPEC:
|-----|
| +-WITH+|
| |-----|'-DEFAULT-----|
| |                                     |'default-clause-'|
|-----|
| +-GENERATED--.ALWAYS-----|
| |                                     |'BY DEFAULT-'| |'-as-identity-spec-'|
AS-IDENTITY-SPEC:
|-----|
| -AS IDENTITY+|
| |                                     | <-,-----< | |
| |                                     | +---1---+ |
| |                                     | | |
| | +- (---+START WITH--+nconst+--+--+)-+ |
| | | | |
| | | | +---1---+ |
| | | | |
| | | -INCREMENT BY--+nconst--+ |
| | | | |
| | | +CACHE 20-----+ |
| | | | |
| | | +-NO CACHE-----+ |
| | | | |
| | | +CACHE-integer+ |
LIKE-clause (CREATE TABLE)
|-----LIKE+---table-name+----->>
| | |
| | +---view-name---+
>>+-----+
| | | |
| | | +COLUMN ATTRIBUTES+ |
| | | |
| | +---INCLUDING IDENTITY+-----+

```

- Chapter 3. Application enhancements
- 57**

- **START WITH numeric-constant:** Specifies the first value for the identity column. The value can be a positive or negative value that could be assigned to the column, as long as there are no non-zero digits to the right of the decimal point. The default is 1.
- **INCREMENT BY numeric-constant:** Specifies the interval between consecutive values of the identity column. This value can be any positive or negative value that is not 0, does not exceed the value of a large integer constant, and could be assigned to this column, as long as there are no non-zero digits to the right of the decimal point. The default is 1. If the value is positive, the sequence of values for the identity column ascends. If the value is negative, the sequence of values for the identity column descends.
- **CACHE or NO CACHE:** Specifies whether to keep some preallocated values in memory. Preallocating and storing values in the cache improves performance for inserting rows into a table that has an identity column.
- **CACHE integer:** Specifies the number of values of the identity column sequence that DB2 preallocates and keeps in memory. The minimum value that can be specified is 2, and the maximum is the largest value that can be represented as an integer. The default is 20. During a system failure, all cached identity column values that are yet to be assigned are lost and, thus, will never be used. Therefore, the value specified for CACHE also represents the maximum number of values for the identity column that could be lost during a system failure. In a data sharing environment, each member gets its own range of <integer> consecutive values to assign. For example, if CACHE 20 is specified, DB2A might get values 1-20 for a particular sequence, and DB2B might get values 21-40. Therefore, if transactions from different members generate values for the same identity column, the values that are assigned might not be in the order in which they are requested.
- **NO CACHE:** Specifies that caching is not to be used. In a data sharing environment, use NO CACHE if you need to guarantee that the identity values are generated in the order in which they are requested.
- **LIKE-clause (for CREATE TABLE):** Specifies that the columns of the table have exactly the same name and description as the columns of the identified table or view. However, for an identity column, the new table inherits only the data type of the identity column; none of the other column attributes are inherited unless the new INCLUDING IDENTITY clause is specified.
- **INCLUDING IDENTITY COLUMN ATTRIBUTES:** Specifies that the new table inherits all of the column attributes of the identity column. If the table identified by LIKE does not have an identity column, the INCLUDING IDENTITY clause is ignored. If the identified object of LIKE is a view, INCLUDING IDENTITY COLUMN ATTRIBUTES cannot be specified.

Existing techniques to create new keys



Next number in single table

- + application controlled (very flexible but have to code it)
- serializes processing
- additional processing overhead
- data sharing concurrency risks
- must ensure everyone follows the rules

Timestamp

- + a reversed form is good for random keys
- not a sequential number and long field

CICS temporary storage queue

- + good performance
- all insert activity must be via CICS
- data sharing and recovery issues

3.1.1 Existing techniques to create new keys

There are a number of implementation options open to you when considering how to generate unique key values.

The first decision is whether the number should be sequential. The main advantage of a random number (which can be generated from a reverse timestamp) is that the insert activity is spread across the index (and table space if the index is defined as clustering). This avoids 'hot spots' and contention when throughput is high, which is exacerbated in a data sharing environment.

However, in many cases it is a requirement that the values are allocated sequentially for business or processing reasons. A full discussion on all the design issues of sequential versus random keys is beyond the scope of this book. However, you should note that if the sequential key is the first column in an index, all insert activity is at the end of the index (and the end of the table space if it is the clustering index). Consequently you should take into account your maximum insert rate when designing indexes for the table to avoid hitting contention problems.

The redbook *DB2 for OS390 Application Design for High Performance*, GG24-2233 provides further information and will assist you in your decision.

We will assume here that a sequential number is required and discuss the traditional implementation methods.

3.1.1.1 Next number in single row table

As part of the database design for the application that is required to generate the next number, a table is created with a single row. This table holds the next number to be used. When it comes to insert the next row, the application reads for an update to obtain the value, creates the next row, and updates the counter within the same unit of work.

The advantages of this technique are that you have control of the type of number that is generated, and the processing can add check digits if necessary. In addition, it is in a format that is acceptable for business and customer use. Provided that every application process that performs inserts follows the rules to generate the number, uniqueness is guaranteed.

The major problem with this technique is seen in high transaction rate environments. Since every transaction acquires an exclusive lock on the counter in order to increment it, serialization inhibits the transaction throughput of the system. We have performance figures comparing the use of DB2's identity column with this technique. They demonstrate a significant improvement in the transaction rates that can be achieved, as reported in 3.1.5, "Identity columns performance" on page 67.

In data sharing environments, a significant consideration is the availability of the table in the event of member failure. It is possible for locks held by the failing member to be retained, thus preventing access to the shared counter by the others members.

Another disadvantage with the next number table technique is that you have to code and maintain the application logic to manage the generation of the next number. You must also ensure that all applications performing insert processing will follow the same procedure. Although you can use functions, triggers, and procedures to minimize the overhead of application development and the risks that an application will not adhere to the rules, it is still your responsibility to develop this code.

3.1.1.2 Using a timestamp

In most cases, a timestamp format is not acceptable to the business, as the number will be quoted and will appear on documents. Timestamps are most useful for key generation when a random number is required. Then a reversed form can be a simple way of generating a key.

Timestamps are also useful when you require rows to be inserted randomly across partitions, but sequentially within each partition. A typical use would be tables which are cleared out each night and start the day empty. If a completely random key was used, DB2 would have the overhead of trying to maintain the clustering sequence as the data built up. This could have a significant effect on the performance of the system. Conversely, the insert arrival rate could be too high for a sequential key, as there would be a bottleneck at the end of the table space. An alternative would be to define a table space with 100 partitions and generate the key from the last part of a timestamp, but with the last 2 digits at the front. The table space would be partitioned on those 2 digits.

An example follows:

Set :hv = CURRENT TIMESTAMP.

2000-01-01 16:32:01 000001.

Use from the minutes onwards, that is, '3201000001'.

Move the last 2 digits to the front, that is, '0132010000'.

This key will be inserted at the end of partition 2 (00 going into partition 1).

This may avoid contention problems by spreading the insert activity across 100 'hot spots' rather than just one.

3.1.1.3 CICS temporary storage queue

Some CICS customers use a temporary storage queue to store the highest number allocated for a key in memory. Although this can be a good solution, it relies on all insert activity for the table being performed by CICS transactions.

Other considerations are how the number will be allocated in a parallel sysplex environment, and how the value will be recovered in the event of a CICS failure.

Definition of identity columns



Typical identity column definition:

CUSTNO INTEGER	(Alternatives are SMALLINT and DEC(n,0))
NOT NULL	
GENERATED ALWAYS	(Unless propagating or loading data)
AS IDENTITY	
(START WITH 1,	(Use another value if converting)
INCREMENT BY 1,	(Can be any specified value (+ or -))
CACHE 20)	(For performance but be prepared to lose those preallocated numbers)

CREATE LIKE tname INCLUDING IDENTITY COLUMN ATTRIBUTES
- otherwise the identity attributes of the column will not be inherited

3.1.2 Definition of identity columns

Below are some of the main points to consider when defining identity columns. Refer to *DB2 UDB for OS/390 SQL Reference*, SG26-9014-01 for details.

3.1.2.1 Usage and restrictions

You can specify the AS IDENTITY clause for SMALLINT, INTEGER and DECIMAL columns with a scale of zero. It is also valid for distinct types based on any of these data types.

Please note the following restrictions when considering using identity columns:

- Identity columns are not updateable if defined with GENERATED ALWAYS.
This restriction has profound implications that you must consider when loading or propagating data. Please see 3.1.9, “Managing tables with identity columns” on page 74.
- There is a limit of one identity column per table.
- Identity columns do not allow nulls.
- An identity column cannot have a fieldproc.
- A table with an identity column cannot have an editproc.
- WITH DEFAULT is not allowed with an identity column specification.
- Created global temporary tables cannot have identity columns.
- The identity column attributes are not inherited unless the INCLUDING IDENTITY COLUMN ATTRIBUTES is valid for and included in any statement that clones a table using the LIKE clause.

3.1.2.2 INCREMENT BY and the potential for gaps

The INCREMENT BY clause specifies the interval between consecutive values of the identity column. If you specify a positive value, the sequence of values ascends; and if the value is negative, the sequence descends.

The counter for an identity column is incremented (or decremented) independently of the transaction. You can see gaps in the sequence for the following reasons:

- A particular transaction might see a gap between two numbers that it generated because other transactions have been concurrently incrementing the same identity counter by inserting rows into the table. If a single transaction must have a consecutive range of numbers, it should take an exclusive lock on the table or table space that has the identity column. Obviously, this must be weighed against concurrency requirements.
- If a transaction that generated a value for the identity column is rolled back, the number allocated will be unused. DB2 will not reallocate it.
- If you recovered the table space to a previous point-in-time, allocation would carry on from the highest number that had been allocated (unless you had also recovered the catalog table SYSIBM.SYSSEQUENCES to the same point-in-time.)
- A DB2 subsystem that cached a range of values crashed before all the cached values were assigned. For considerations on the impact of caching, see 3.1.6, “Impact of caching” on page 68.

3.1.2.3 Creating views on identity columns

Views can be created on tables with identity columns. A column of a view is considered an identity column (in that insert follows the same rules as for the base table column) if it maps directly or indirectly to an identity column of a base table, except:

- If the select-list of the view definition includes multiple instances of the name of the identity column.
- If the FROM clause of the view definition directly or indirectly includes a join.
- If a column in the view definition includes an expression that refers to an identity column.

GENERATED options



GENERATED ALWAYS

- + Guaranteed uniqueness
- + More index design options
 - for example extra columns to enable index only access for some queries
- Would need to change definition to GENERATED BY DEFAULT to drop table and reload data
- Cannot be used if propagating data
- BE CAREFUL!
 - When coding an INSERT INTO... SELECT FROM
 - When loading data

GENERATED BY DEFAULT

- + Can drop table and reload data
- + Allows propagation and loading from other sources
- Index required to ensure uniqueness and retry logic in applications
- Transactions will fail if range DB2 is using has been inserted by another application

3.1.3 GENERATED options

You need to give careful consideration when deciding whether to use the GENERATED ALWAYS or GENERATED BY DEFAULT option.

When GENERATED ALWAYS is specified, DB2 will always generate the value so uniqueness is guaranteed. A unique index on the column is not required. This would be a particular advantage if you would prefer the index to include other columns, perhaps to enable index-only access for some common queries.

The GENERATED BY DEFAULT option allows you to supply values for the identity column. DB2 will only generate a value when it has not been provided with one. No checking is performed on the supplied value, so duplicates could occur. To ensure uniqueness, you would need to define a unique index on the identity column. Applications would then need to check all inserts for SQLCODE -803 and retry if necessary. If the range of values DB2 is currently allocating has already been inserted into the table by another application, transaction failures will occur.

You must specify GENERATED BY DEFAULT if you are propagating data or loading data from other sources.

Otherwise, you could specify GENERATED ALWAYS to avoid having an index to ensure uniqueness and to avoid applications needing retry logic on the insert. However, as the values cannot be updated, you would be unable to use the load utility to reload the values. If you needed to drop and recreate the table, you would need to change the definition to GENERATED BY DEFAULT and add a unique index. You should also bear in mind the coding considerations documented in 3.1.7, “Applications and identity columns” on page 70.

Consequently, you may decide to use GENERATED BY DEFAULT from the beginning. However, please note that you can minimize the need to recreate tables by using design standards for high availability. For example, rather than changing a column definition, adding a new one and transferring the data across. These considerations are documented in the redbook *DB2 for OS/390 and Continuous Availability*, SG24-5486.

Refer to 3.1.7, “Applications and identity columns” on page 70 and 3.1.9, “Managing tables with identity columns” on page 74 for more information. This documentation warns that when GENERATED ALWAYS is defined, the following actions will generate the identity values rather than populating them from the source data:

- When you load a table and do not include the identity column in the field specification
- When you INSERT INTO SELECT FROM.... omitting the identity column from the VALUES clause

Identity columns in DB2 catalog



New table space SYSIBM.SYSSEQ with table SYSIBM.SYSEQUENCES

- one row for each identity column
- identity column definition fields and last assigned value

New table space SYSIBM.SYSSEQ2 with table SYSIBM.SYSEQUENCEDEP

- one row for each identity column
- release dependency and dependent table indicators

New values for DEFAULT column of SYSIBM.SYSCOLUMNS table

- I for AS IDENTITY and GENERATED ALWAYS
- J for AS IDENTITY and GENERATED BY DEFAULT

3.1.4 Identity columns in DB2 catalog

For the identity column support new DB2 catalog objects have been defined:

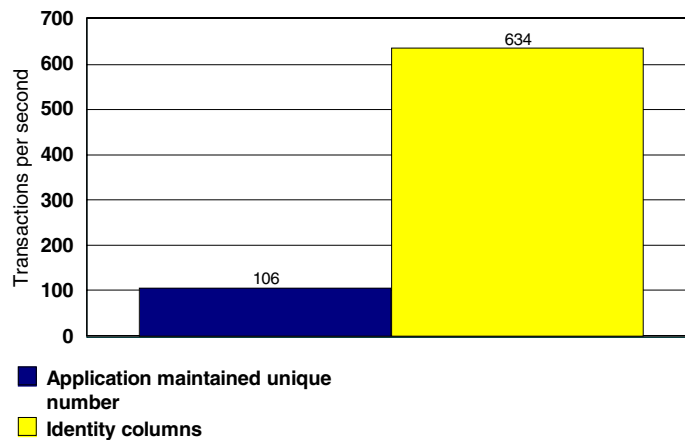
- table space SYSIBM.SYSSEQ
 - table SYSIBM.SYSEQUENCES
 - indexes DSNSQX01 and DSNSQX02
- table space SYSIBM.SYSSEQ2
 - table SYSIBM.SYSEQUENCEDEP
 - index DSNSRX01

Also new values have been added to the DEFAULT column of the SYSIBM.SYSCOLUMNS catalog tables to reflect the identity column definition options:

- I for GENERATED ALWAYS
- J for GENERATED BY DEFAULT

For details on the contents of the catalog tables refer to *DB2 UDB for OS/390 Version 6 SQL Reference*, SC26-9014-01.

Identity columns performance



3.1.5 Identity columns performance

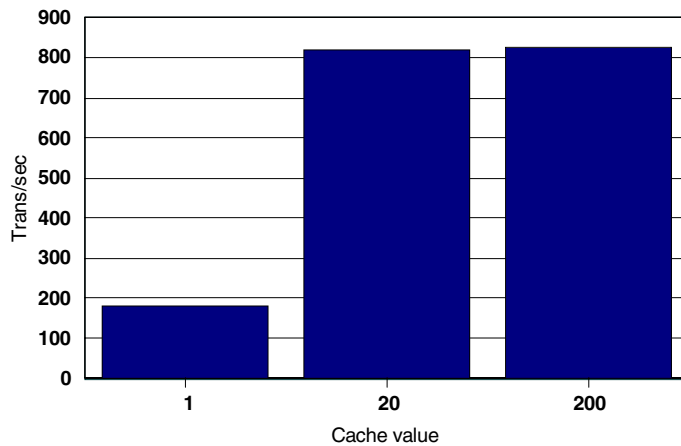
Our performance measurements compare the throughput of an application exploiting identity columns with one generating a unique number by updating a next-number table. The application program to generate a unique number selected from the one-row table, inserted the next number into the table, and updated the counter before committing.

The table into which sequential unique numbers were being inserted was the same in all tests apart from the primary key being defined as an identity column. The primary key (identity) column was indexed in every test. The default value of CACHE 20 was used.

The tests were carried out on a 9672-ZZ7 with 12 CPUs in a non-data-sharing environment. The LPAR was capped at 25% of the total MIPS. We ran a TPNS workload on OS/390 V2.6 and IMS V6 DCCTL.

The throughput achieved was 106 transactions per second using the next number table compared to 634 when identity columns were used.

Impact of caching — data sharing



3.1.6 Impact of caching

You can tell DB2 whether to keep some preallocated values in memory using the `CACHE n / NO CACHE` clauses. Storing values in memory improves performance when inserting rows. The improvements are more significant in data sharing environments.

The chart above, which is based on our IRWW workload in a data sharing environment, shows a large increase in throughput when using `CACHE 20` rather than `CACHE 1`. In a data sharing environment there is a *synchronous* forced log write each time the counter is updated. If you do this once every 20 times rather than for every insert, the overhead is significantly reduced. Increasing this value above the default of 20 gave negligible benefit.

For non-data-sharing systems, log writes for the updated counter are *asynchronous*. We still recommend `CACHE 20`, but improvements compared to `CACHE 1` are less significant.

In the case of a system failure, any unassigned values in cache are lost and will never be used. Therefore, when deciding on the number of values to cache, consider how many values you would be prepared to lose.

Notes for data sharing users:

If you use the CACHE *n* option in data sharing, each member gets its own range of *n* consecutive numbers to assign. For example, DB2A might get the values 1-20 for a particular column and DB2B gets values 21-40. Therefore, when caching is used and transactions generate values for the same identity column from different members, the values are not assigned in order of request. A transaction can generate a value of '22' from DB2B and then later in time, another transaction can generate a value of '5' from DB2A. It should be noted that this is only important when the numbers must be in order of processing.

There is a high probability that you will see 50/50 splits of index pages in a data sharing environment, particularly if you use a value of CACHE greater than 1, because successive inserts from the data sharing members are unlikely to be in numerical sequence.

Applications and identity columns



SQL Statement	GENERATED ALWAYS	GENERATED BY DEFAULT
CUST_NO defined as identity column		
1. INSERT INTO CUSTOMER (CUST_NO, FIRST_NAME, LAST_NAME) VALUES (default, :hv_fname, :hv_lname)	VALID	VALID
2. INSERT INTO CUSTOMER (CUST_NO, FIRST_NAME, LAST_NAME) VALUES (:hv_custno, :hv_fname, :hv_lname)	INVALID	VALID
3. INSERT INTO CUSTOMER (CUST_NO, FIRST_NAME, LAST_NAME) OVERRIDING USER VALUE VALUES (:hvempno, :hv_fname, :hv_lname)	VALID BUT NOT RECOMMENDED	INVALID
4. INSERT INTO CUSTOMER SELECT * FROM ALT_CUSTOMER	INVALID	VALID
5. INSERT INTO CUSTOMER (FIRST_NAME, LAST_NAME) SELECT FIRST_NAME, LAST_NAME FROM ALT_CUSTOMER	VALID	VALID

3.1.7 Applications and identity columns

When you design your table to include an identity column, you can specify either **GENERATED ALWAYS** or **GENERATED BY DEFAULT**. The diagram above shows that your choice limits the application constructs you can use.

- If you tell DB2 to provide the value (SQL statement 1), either table definition is valid.
- To specify your own host variable value (SQL statement 2), you cannot have specified **GENERATED ALWAYS**.
- You can force DB2 to accept your value by coding **OVERRIDING USER VALUE** in insert statement (SQL statement 3).
- The rules for an insert with a subselect are similar, in that a value may only be specified if the identity column is specified as generated by default. Insert (SQL statement 4) is logically equivalent to having specified all columns in the values clause.
- If you want to propagate all non-identity columns from a table with the same definition, you code SQL statement 5.

Important notes:

- Because CUST_NO is not specified in the column list in SQL statement 5, it will be populated with identity columns generated value. If you need to be able to propagate the identity column, you need to define it with GENERATED BY DEFAULT.
- If you ever have to change the table definition from GENERATED ALWAYS to GENERATED BY DEFAULT and you have coded OVERRIDING USER VALUE (SQL statement 3), the BIND to pick up the new table would fail. Application changes would be required. The circumstances in which you may want to change the GENERATED options are described in 3.1.9, “Managing tables with identity columns” on page 74. We recommend against using the OVERRIDING USER VALUE clause.

IDENTITY_VAL_LOCAL function



If you need to know the value DB2 inserted
code the following after a successful insert
`SET :HV = IDENTITY_VAL_LOCAL()`

Null will be returned if
Qualifying insert has not been performed at this level
Commit or Rollback has since been performed

Note: check SQLCODE from insert first

Think carefully before using the function in triggers

3.1.8 IDENTITY_VAL_LOCAL function

In many applications, you will need to know the value DB2 has generated for the identity column. Most commonly this will be when tables in a referential set (either application or DB2 maintained) are populated in one unit of work. An example would be to create an order number and order lines for that order. The order number would be required to create the order lines. This function is introduced by APAR PQ36328.

3.1.8.1 Usage and restrictions

The `IDENTITY_VAL_LOCAL()` function returns the most recently assigned value for an identity column that was populated by a single row insert. The value could have been generated by DB2 or supplied by the user if the identity column is defined as `GENERATED BY DEFAULT`. The result is a `DECIMAL(31,0)` regardless of the actual data type of the identity column.

Null will be returned when a commit or rollback occurred since the most recent insert that assigned a value. Rollback to savepoint does not affect the result.

The function will return the correct value only if the insert and the function call occur while DB2 is at the same processing level. Therefore, in order to obtain the value that DB2 inserted, we recommend that immediately after a successful insert, you store the value that was inserted into the identity column in a host variable with a statement such as `SET :HV = IDENTITY_VAL_LOCAL()` and check for the returned SQL code.

All other techniques, including coding the function in a predicate, invoking the function within the select statement of a cursor, using the function in the VALUES clause of an insert, inclusion in any SQL statement which is processed in parallel or as part of a result set from view materialization, will give unpredictable results.

Note: If the insert failed, the result of the function will be unpredictable. Consequently, before using the function, you should check the SQLCODE to ensure that the insert completed successfully.

3.1.8.2 Considerations when using triggers

You should be aware of the following issues when considering using the IDENTITY_VAL_LOCAL function within the body of a trigger:

- We do not recommend that you use this function within the body of a BEFORE trigger, as unpredictable results or a null value will be returned.
- When the function is used within an AFTER insert trigger, the value returned will be the value for the most recent single row insert within the body of the trigger. If an insert with an identity column in the VALUES clause has not been performed within the trigger, a null value will be returned. Consequently, you cannot use the function to determine the value assigned in the insert that caused the trigger to be actioned.
- You should not use the IDENTITY_VAL_LOCAL function across triggers. A table may have a number of triggers defined on it, and the function cannot be used to determine the value inserted by an insert statement in another trigger.

Managing tables with identity columns



Load utility

- Be careful loading data into identity columns defined as GENERATED ALWAYS

Adding an identity column to a table

- ALTER TABLE AND REORG

Converting a table to use an identity column

- Recreate table with GENERATED BY DEFAULT identity column and use START WITH clause

Copying tables across subsystems

- Stop source table and run catalog query to determine start value and increment for target table

Data propagation considerations

- Use GENERATED BY DEFAULT on target
- For 2 way replication - INCREMENT BY 2 to avoid duplicates

3.1.9 Managing tables with identity columns

In this section we describe how to go about loading, adding, converting to, and copying tables with identity columns. We also include data propagation considerations.

3.1.9.1 Load utility

An identity column that is defined as GENERATED ALWAYS cannot be included in the field specification list, or be implied by a LOAD FORMAT UNLOAD or LOAD with no field list.

Important note: Be careful when loading tables with identity columns that have been defined as GENERATED ALWAYS. If you leave the identity column out of the field specification, DB2 will regenerate the identity column values. It is likely that each row would have a different key value from before, and relationships with other tables will be lost. If loading of the data is required, GENERATED BY DEFAULT should be used. There is no way to go back to GENERATED ALWAYS.

Identity columns defined as GENERATED BY DEFAULT can either be set by the load utility from input data, or generated by DB2 if the DEFAULT attribute is used. The input values are of the numeric data type that is specified in the identity column. No additional validation of the values is performed.

If, at design time, you think that you will have to unload and reload your table, choose GENERATED BY DEFAULT for the identity column.

3.1.9.2 Adding an identity column to an existing table

You can use ALTER TABLE ADD COLUMN to add an identity column to an existing table. If the table is not empty, DB2 puts the table space into REORG state. When the reorg is subsequently run, DB2 generates unique values for the identity column and remove the REORG state.

3.1.9.3 Converting a table to use an identity column

To alter the attributes of an existing column to be an identity column, you must drop and recreate the table (using your normal procedures for conserving the table data, indexes, authorities and other dependent objects).

When recreating the table, you should ensure that the START WITH value in the CREATE TABLE is set to the next value that you want DB2 to start generating after the table is reloaded with the previous data. In addition, as you will be using the LOAD utility to reload the table after it is recreated, you must specify GENERATED BY DEFAULT. This allows the load utility to reuse the previously allocated identity column values. There is no way to go back to GENERATED ALWAYS.

3.1.9.4 Copying tables between subsystems

Special consideration has to be given to avoid overlaps when copying tables with identity columns between subsystems.

Consider the following procedure:

1. Stop the table space containing the table on the source system, to prevent any new rows from being inserted.
2. Run the following catalog query to determine the START WITH and INCREMENT BY values for the create table statement on the target system:

```
SELECT B.DCREATOR, B.DNAME,B.DCOLNAME, INCREMENT,  
A.INCREMENT+A.MAXASSIGNEDVAL AS NEW_START_VALUE  
FROM SYSIBM.SYSSEQUENCES A INNER JOIN  
SYSIBM.SYSSEQUENCESDEP B  
ON B.DCREATOR = table creator  
AND B.DNAME = table name
```

3. Create the table on the target subsystem, using the start and increment values determined from the previous step.
4. Stop the table space on the target system, to prevent rows from being inserted until it is populated.
5. Copy the data from the source to the target (using DSN1COPY or similar).
6. Start the table on the target subsystem (and also the source if required).‘

3.1.9.5 Data definition changes

If a drop and recreate of the table is required in the future, perhaps for a schema change, you would need to be recreated as GENERATED BY DEFAULT in order to load the existing data back. Please refer to 3.1.3, “GENERATED options” on page 64. You will also need to save the value in SYSIBM.SYSSEQUENCES before the drop and recreate the table with the START WITH parameter.

3.1.9.6 Data propagation considerations

If a source table in a one-way data propagation environment contains an identity column with the GENERATED ALWAYS attribute, you should define the corresponding identity column on the target table with GENERATED BY DEFAULT. This will allow the propagation process to explicitly supply a value for the column.

For two-way replication, both source tables must use GENERATED BY DEFAULT for the same reason. Also, if unique values are required, you should take care when specifying the START WITH and INCREMENT BY values for the two tables. To prevent the same value being generated at each site, we suggest you have one table generating odd values (using START WITH 1, INCREMENT BY 2) and one generating even values (using START WITH 2 INCREMENT BY 2.)

Advantages of identity columns



Better performance and concurrency than application generated counters

Guaranteed uniqueness both within an individual subsystem and a data sharing group

Recoverability in the event of a DB2 system failure

The failure of one data sharing member will not impact the other members from generating key values

Simple implementation which is internal to DB2

3.1.10 Advantages of identity columns

Having considered other techniques by which you can generate values for primary key columns, we now can review the advantages of using identity columns in context.

3.1.10.1 Improved performance and concurrency

Serialization on the next number table and the costs of retry are eliminated. Because of this, significant (6-fold) increases in transaction throughput are possible.

3.1.10.2 Can guarantee uniqueness

If you define an identity column as GENERATED ALWAYS AS IDENTITY, then DB2 always generates the column value and guarantees that each value is unique across the system. In this case a unique index is not required to ensure uniqueness, although we do recommend that you create one (see 3.1.3, “GENERATED options” on page 64 for details).

3.1.10.3 Recoverability in the event of failure

In the case of a system failure, recoverability will be provided by reconstructing from the log the last value before the outage. This ensures that uniqueness is maintained. No action on your part is necessary.

3.1.10.4 Enhanced data sharing availability

Should a member fail within a data sharing group, the other members will continue to generate unique key values. No retained locks are held by the failing member.

3.1.10.5 Simple and flexible implementation

Identity columns are simple to implement and require no extra application logic. Whenever an insert is processed, regardless of which application performs the insert, a unique value will be generated. You choose what format the column takes so you have more flexibility to choose something acceptable to the business and customers.

Since the value is allocated by DB2 on insert, identity columns cannot be used where you need to concatenate a check digit to the value.

You can override the value that DB2 would have generated and provide your own. There are special circumstances when you need to do this, and important considerations, which are discussed in more detail in 3.1.7, “Applications and identity columns” on page 70, and 3.1.9, “Managing tables with identity columns” on page 74.

Note: Identity columns are similar to ROWID in that DB2 automatically generates a unique value for the column whenever a row is inserted into the table. ROWID is a new SQL data type which returns a 40 byte varchar value which is not regularly ascending or descending. It is the basis for direct row access and accessing LOB data. Therefore, it is not usually suitable for columns such as order number. In contrast, identity columns use existing numeric data types whose range you control and are sequentially allocated.

APAR identifier

The APAR identifier is as follows:

PQ30652, PQ30684, PQ36328, PQ36452.

Savepoints



DB2A

```

PROGA
SAVEPOINT A UNIQUE ON
ROLLBACK RETAIN CURSORS;

SQL statements....

IF...
ROLLBACK TO SAVEPOINT
  Data changes ✓
  Schema changes ✓
  Declared temporary tables ✓
  Created temporary tables ✗
  Other DBMS, CICS etc ✗
  Cursor activity ✗
  Lock activity ✗
  Caching ✗

ELSE...
RELEASE SAVEPOINT
    
```



DRDA CONNECT



Private protocol or
DRDA using
aliases or 3 part
names

3.2 Savepoints

A savepoint represents the state of data and schema at a particular point-in-time. An application may set named savepoints within a transaction, then as dictated by application logic, rollback subsequent data and schema changes without affecting the overall outcome of the transaction. The scope of a savepoint is the DBMS on which it is set.

Savepoints enable the coding of contingency or what-if logic and could be useful in the following scenarios:

- For programs with sophisticated error recovery.
- To undo stored procedure updates when an error is detected. A rollback after a stored procedures will rollback the entire unit of recovery, including all work done by the caller, and this could be beyond the wanted scope. Savepoints could be used to make the logic surrounding them transparent to the caller.

You can set them by using the SAVEPOINT syntax documented in the *DB2 UDB for OS/390 Version 6 SQL Reference*, SC26-9014-01. The savepoint name can be up to 128 characters and we suggest that you use a meaningful name. You can use the UNIQUE OPTION to assert that the savepoint name will not be reused within the transaction. If you omit the UNIQUE option and reuse the savepoint name, the old savepoint will be destroyed. Please note that this is different from using the RELEASE SAVEPOINT statement, which will release all savepoints set after up to the named savepoint.

When you issue the `SAVEPOINT` statement, DB2 writes an external savepoint log record. Savepoints that you create are often termed external as opposed to internal, because DB2 already uses internal savepoints. Internal savepoints are used by DB2 only, are a mechanism to back out elements smaller than a unit of recovery, cannot be accessed through an application, and may be liable to change in future releases or as a result of applying maintenance. In contrast, you have full control over external savepoints.

To rollback to an external savepoint you have set, use the `ROLLBACK TO SAVEPOINT svptname` statement. This will backout all data and schema changes that were made after the savepoint. The following will not be backed out:

- Any updates outside of the local DBMS, such as remote DB2s, VSAM, CICS, IMS
- Changes to created temporary tables (changes to declared temporary tables are backed out)
- The opening and closing of cursors
- Changes in cursor positioning
- The acquisition and release of locks
- The caching of the rolled back statements

The use of `ON ROLLBACK RETAIN CURSORS` and `ON ROLLBACK RETAIN LOCKS` clause will allow for cursors and locks not to be released upon rollback to the savepoint.

3.2.1 Connecting to other DB2 systems

While there are outstanding savepoints, you cannot access a remote DBMS using DB2 private protocol access or DRDA using aliases or three-part names. For example, if there is a savepoint set at location A, location A cannot connect to location B using either of these two ways to access data. This is due to the consideration that the programmer using three-part names or aliases normally is unaware of the remote sites involved.

DRDA access using a `CONNECT` statement is allowed; however, the savepoints are local to their site and do not cross an explicit Connect. For example, location A can connect to location B, but the savepoint set at A is unknown to B and does not cover any operations performed at location B. You should note that a savepoint set prior to a `CONNECT` is known only at the local site, and a savepoint set after a connect is known only at the remote site. Consequently application-directed connections are expected to manage the processing at the alternate sites.

We recommend that you code the `RELEASE SAVEPOINT savepoint name` statement to release savepoints that are no longer required for clarity and to reenable the operation of SQL that resolves to remote locations.

3.2.2 Restrictions on using savepoints

You cannot use savepoints in global transactions, triggers, user-defined functions, or in stored procedures, user-defined functions or triggers that are nested within user-defined functions.

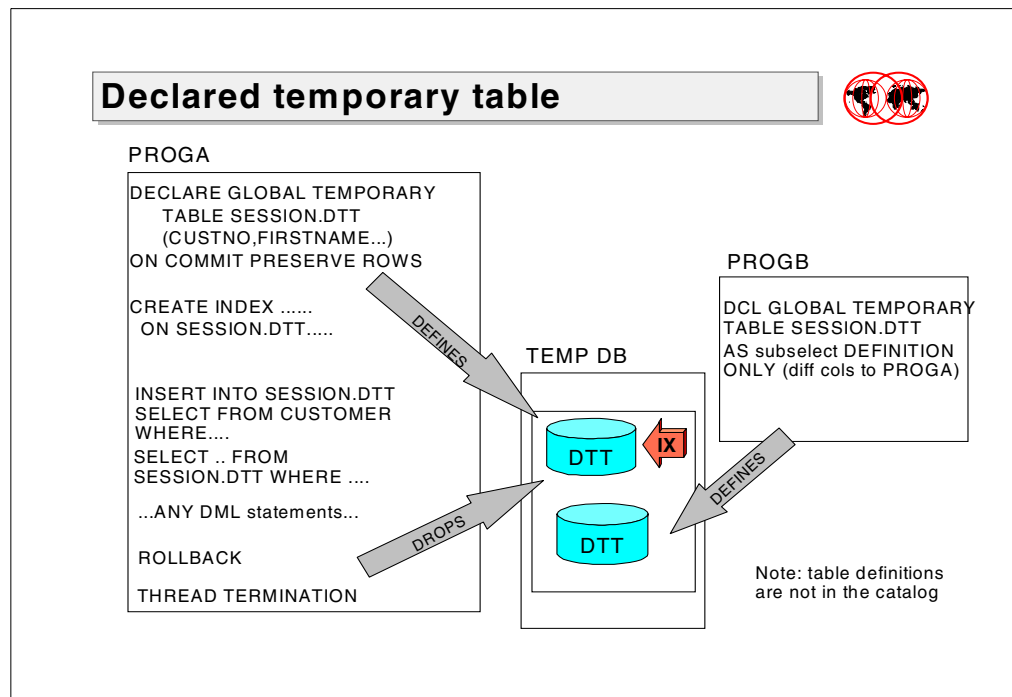
3.2.3 Savepoint performance

The overhead of maintaining a savepoint is minimal. The measurements performed show that the cost of taking a savepoint is equivalent to a simple fetch.

APAR identifier

The APAR identifier is as follows:

PQ30439.



3.3 Declared temporary tables

Declared temporary tables (DTTs) are a way to temporarily hold or sort data within a program. They could be used in the following scenarios:

- For Business Intelligence applications, when you want to process extracts of the data and perform further result set processing on them, rather than the original data including further joins.
- As a staging area for making IMS data accessible to ODBC. For example, a client application program can call a stored procedure to access IMS data. The IMS data can be inserted into a DTT. The data in the DTT may then be processed by the client using standard SQL.
- To hold result sets in stored procedures as implemented in other RDBMSs. Typically it is more efficient to exploit standard result set processing using locator variables rather than using declared temporary tables, but you need to weigh up ease of use and simplicity of conversion against your performance and functional requirements.

A temporary table is defined by an application using the `DECLARE GLOBAL TEMPORARY TABLE` statement. They are not held in the catalog and so cannot be shared across application processes. In fact many processes running concurrently can declare the same table name and each have their own table with different structures.

The application must reference the table using the `SESSION` qualifier and may create indexes on the table in addition to performing the full range of DML.

You can specify `SESSION` explicitly in the table name reference or you can use the `QUALIFIER BIND` option to specify `SESSION` as the qualifier for all SQL statement in your plan or package.

Rollback is supported whether it be to a savepoint or the last commit point. The table exists until thread termination or thread reuse when it is implicitly dropped.

No row, page, or table locks are acquired on the table, although locks on the table space and DBD may be taken. The table does not require an associated cursor declared with `hold` to keep rows across commits.

Main differences between table types



	Base table	Created temporary table	Declared temporary table
Table description in catalog so shareable and persistent	YES	YES	NO
Data persistence and shareability	YES	NO	NO
Indexable	YES	NO	YES
Full DML	YES	NO	YES
Locking	YES	NO	Data - NO TS & DBD - Share
Logging	YES	NO	Undo records only
Recovery	YES	NO	Rollback only
Table space management	3 types	Work file DB	Segmented table space in TEMP DB
ROLLBACK to SAVEPOINT	YES	NO	YES

3.3.1 Main differences between table types

DB2 now recognizes three different types of tables. These are base tables (BT), created temporary tables (CTT), and declared temporary tables (DTT). This section compares the main attributes of the three types.

3.3.1.1 Table creation

When you create a base table or a CTT, the description is stored in the DB2 catalog. Consequently, the table definition is persistent and shareable across application processes.

The description of a DTT is not stored in the catalog and is not shareable. It is possible for concurrent processes to declare temporary tables of the same name and with different structures. The table exists only until thread termination or thread reuse, unless the program explicitly drops it earlier.

3.3.1.2 Data persistency and shareability

For base tables, one instance of the table is created by the CREATE TABLE, and it is used by all applications.

CTTs and DTTs differ from base tables in that an application has its own instance of the table, which is not persistent beyond thread termination.

3.3.1.3 Index and DML support

Base tables and declared temporary tables have index and full DML support. Indexes, UPDATE (searched or positioned) and DELETE (positioned only) are not supported for CTTs.

3.3.1.4 Locking, logging and recovery

Full logging, locking, and recovery apply to base tables.

Locking, logging, and recovery do not apply for CTTs.

For DTTs, no row or table locks are acquired, although share level locks are acquired on the table space and DBD. A segmented table lock is also acquired when all the rows are deleted from a table or a table is dropped. Locks acquired when declaring or processing DTTs do not affect concurrency.

DTTs log UNDO but not REDO records. Consequently, rolling back to a savepoint or commit point is supported, although forward log recovery is not.

3.3.1.5 Table space requirements

Base tables can be stored in simple, segmented, or partitioned table spaces that have been created in user-defined databases. They are limited in size by the PRIQTY and SECQTY and possibly other tables if in a simple or segmented table space.

CTTs are stored in table spaces in the workfile database. A CTT can span work files and so do not reach size limitations easily. This may be undesirable if your workfile database is constrained because other processes running sorts may run out of space.

DTTs are stored in segmented table spaces in a database defined AS TEMP. As with any segmented table space, the table cannot span table spaces and so is limited by the PRIQTY and SECQTY and the shared usage of the table space among multiple users (see 3.3.8, “Database and table space issues” on page 93 for details).

3.3.1.6 Support for ROLLBACK to SAVEPOINT

Base tables and global declared temporary tables allow to rollback the the specified SAVEPOINT. Created temporary tables do not support this function.

3.3.2 Considerations when converting from CTTs

If you are using created temporary tables as they were introduced in DB2 Version 5 and you wish to create indexes on the tables and do positioned updates and deletes, you may want to convert them to declared temporary tables. Here are some considerations:

- You will see an increase in logging. UNDO records are written to support rollback when using DTTs.
- There is a difference in the way in which space is managed. A single DTT is limited to the space available within the TEMP table space in which DB2 has placed it. CTTs are stored in the workfiles (typically DSNDB07) and can span multiple workfiles.
- Since declared temporary tables are stored in their own database, you have the option of preventing impact to other work using the workfiles. This might be isolation of buffer pools, I/O or prevention of out-of-space conditions with concurrently sorting processes.

Defining a declared temporary table



- How to define a declared temporary table

```
DECLARE GLOBAL TEMPORARY TABLE SPRESULT  
(CUST_NO INTEGER GENERATED ALWAYS AS IDENTITY,  
LAST_NAME CHAR(18) NOT NULL WITH DEFAULT...)  
CCSID ASCII  
ON COMMIT DELETE ROWS;
```

```
DECLARE GLOBAL TEMPORARY TABLE SPRESULT  
LIKE CUSTOMER  
INCLUDING IDENTITY COLUMN ATTRIBUTES
```

```
DECLARE GLOBAL TEMPORARY TABLE SPRESULT  
AS (SELECT CUST_NO, SUM(ACC_BALANCE),...  
FROM CUSTOMER GROUP BY BRANCH) DEFINITION ONLY  
USING TYPE DEFAULTS
```

- PUBLIC has implicit authority to create and access

3.3.3 Defining a declared temporary table

You can define the columns in any of the following ways:

- A column list. Normal rules for creating tables should be followed.
- LIKE a table, view, alias or synonym that exists at the current server. The implicit definition includes all attributes of the columns as they are described in SYSIBM.SYSCOLUMNS.
- AS subselect DEFINITION ONLY. This specifies that the columns are to have the same name and attributes that would appear in the derived result table if the subselect were executed. For result columns that are derived from expressions, constants and functions, the subselect must include the AS column name clause in the subselect. If you want to transfer the default and identity attributes you need to explicitly code the INCLUDING IDENTITY COLUMN ATTRIBUTES, INCLUDING COLUMN DEFAULTS or the USING TYPE DEFAULTS clauses. Please note that a declare with subselect will not populate the declared temporary table with the data. To do this, follow the CREATE with an INSERT INTO using the same subselect statement.

The ON COMMIT DELETE/PRESERVE ROWS clause can be used to tell DB2 whether the rows will be deleted (if there is no WITH HOLD cursor open) or preserved past commit.

Please note that if you use the ON COMMIT PRESERVE ROWS option, the thread cannot be inactivated or reused unless the program explicitly drops the table before the commit.

3.3.4 Authorization

No authority is required to declare a temporary table unless the LIKE clause is used when SELECT access is required on the base table or view specified.

PUBLIC implicitly has authority to create tables in the TEMP database and USE authority on the table spaces. PUBLIC also has all table privileges on declared temporary tables implicitly.

The PUBLIC privileges are not recorded in the catalog nor are they revokable.

Despite PUBLIC authority, there is no security exposure, as the table can only be referenced by the application process that declared it.

Referencing declared temporary tables



Must be qualified with SESSION

Full DML syntax allowed (but note no locking)

Static SQL referencing a table qualified with SESSION will be incrementally bound at run-time

- run-time cost equivalent to dynamic SQL statement
- performance consideration if high transaction rate

DBPROTOCOL(DRDA) is required to access a remote declared temporary table using a three part name

- backward references are not allowed

Consider creating an index on the table

- weigh up benefit vs cost of creating index
- Beware of high volume DDL - increase in logging and GRS activity

Explicitly drop the table when no longer required

3.3.5 Referencing declared temporary tables

To reference a declared temporary table you must qualify the table name with the owner of SESSION. That is, declared temporary tables are called `SESSION.table_name`. If in a program you refer to a table qualified by SESSION that has not been previously declared, DB2 will assume you are referring to a persistent table with a qualifier of SESSION and search the DB2 catalog.

If you already have base tables qualified by the owner SESSION, be aware of the following results if you should create a declared temporary table with the same name within a program:

- The DECLARE GLOBAL TEMPORARY TABLE statement will succeed. DB2 will not complain that the object already exists.
- The search sequence for objects owned by SESSION is:
 - a. Declared temporary table
 - b. Base or created temporary table defined in the catalog

You can use the complete SELECT, INSERT, UPDATE, DELETE SQL syntax against a declared temporary table, including unions and joins. However, no locks are acquired on temporary tables, so the WITH CS/RR/RS clause and FOR UPDATE OF clauses are ignored.

For static SQL at BIND time, declared temporary tables do not exist. Therefore, any SQL statements in a static program that reference a table name qualified by SESSION will be incrementally bound at run-time. Only at execution time DB2 can resolve whether the SQL is referring to a temporary or base table. If there are no other errors the BIND will succeed and the catalog tables SYSSTMT or SYSPACKSTMT will show the new value M for that SQL statement in the column STATUS.

Please note that dynamic SQL statements that resolve to a declared temporary table are not placed in the dynamic statement cache. This is because every transaction that defines a declared temporary table, defines its own specific and unique version of it — even if the layout is identical. Similarly, it is possible that every transaction could define a table with the same name, but with a different definition. If the SQL were cached, an incorrect version of the table could be used.

Please note that plans and packages are not invalidated when a declared temporary table or its table space is dropped.

You can reference a declared temporary table at a remote DB2 using a three part name if DBPROTOCL(DRDA) BIND option is in effect. However, you cannot make a backward reference — that is, if you declare a temporary table and connect to another site, you cannot refer back to the DTT at the local site from the remote site.

3.3.6 Creating indexes

You can use the CREATE INDEX statement to create an index on a declared temporary table. If you do not specify a qualifier for the index name, DB2 uses the keyword SESSION. The statement and indexing works in the normal way. However, only the following clauses are supported:

- ON tablename — you must specify the qualifier SESSION
- Column names ASC/DESC
- CLUSTER
- CLOSE — however, DB2 will decide whether the data sets can be closed
- PIECESIZE
- UNIQUE with or without WHERE NOT NULL
- USING STOGROUP

In order to assist DB2's access path selection for declared temporary tables, some basic statistics are collected by DB2 and are maintained and used dynamically for optimizing the access path as rows are processed, but not stored in the catalog. If you are concerned about access path selection, you should use EXPLAIN for SQL executed against declared temporary tables, analyze the output in your PLAN_TABLE as you would for any other SQL and review your indexing strategy. You cannot run RUNSTATS: since there is no catalog definition for the temporary objects there are no statistics available for you to modify.

When considering whether to create an index on a declared temporary table, bear in mind the overhead of creating it. In some cases it may be quicker to simply scan the table than for DB2 to create an index and using that to access the data.

You should also consider carefully the volume of create indexes that will be performed. Normally DDL is avoided or kept to a minimum in an OLTP environment. A large number of create index statements will have an impact on system performance. You will see much more logging occurring. In addition, large scale creation, opening and deletion of VSAM data sets for the indexes will significantly increase global resource serialization (GRS) activity.

You may want to consider using the star form of GRS to improve the performance. OS/390 R2 introduced the GRS STAR methodology of global resource serialization which replaces the traditional ring mode protocol and potential ring disruptions. The star configuration is built around a coupling facility, which is where the global resource serialization lock structure resides. By using the coupling facility, ENQ/DEQ service times will be measured in microseconds, not milliseconds. This provides significant performance improvements. Installations that currently use GRS ring-mode or a third party alternative and host a fair number of OS/390 images should consider migrating to GRS STAR for the improved performance, scale and availability it provides. For more information on GRS you can start with the redbook *Parallel Sysplex Configuration: Cookbook*, SG24-2076-00.

3.3.7 Usage considerations

Similar to following good practice of closing cursors when you no longer need them, we recommend that you explicitly drop the table when it is no longer required rather than leaving it to be dropped implicitly at thread termination. This improves the likelihood of thread reuse and enables better DBD management.

Since rollback and savepoints are supported for declared temporary tables, you may see an increase in the amount of logging activity compared to that expected for created temporary tables. The same number of log records will be written as for activity against a base table. However, the individual records will be shorter as only UNDO information is recorded rather than UNDO/REDO.

Undo records are written for updates to DTTs to support rollback and savepoint processing. It is therefore possible, just like normal table spaces, for the TEMP table spaces to be added to the logical page list (LPL). The procedure to resolve this is to issue the `-start database (TEMP database name) space (TEMP space name) access (RW)` as you would for normal table spaces. DB2 will issue a reset (either logical or a delete/define depending on circumstances) in response to this command. If for any reason there are underlying problems, such as media failure, catalog errors and so on, you should review messages in the DB2 MSTR address space to solve the problem.

To simulate the sorts of messages you might see we performed the following test:

- Declare temporary table.
- Perform a series of updates to the table.
- Cancel IRLM.
- With DB2 down, rename the VSAM data sets underlying the TEMP database. This simulates a failed DASD unit. The pagesets will be unavailable to DB2 on restart.
- Restart DB2.

DB2 added pages to the LPL. If we had not renamed the VSAM data sets, restart completed successfully. Normal restart is what you should expect to see. The screen captured below shows you the messages that DB2 issues — there is no difference between temporary table space pages being added to the LPL and normal data pages. The `-DISPLAY DATABASE(TEMP database name) SPACE(*) LPL` will indicate which pages are in the LPL.

```
DSNB250E =DB2Y DSNIIIMPD A PAGE RANGE WAS ADDED TO THE
LOGICAL PAGE LIST
      DATABASE NAME=TEMP
      SPACE NAME=TEMP4K01
      DATA SET NUMBER=1
      PAGE RANGE X'00000074' TO X'00000074'
      START LRSN=X'000009A48BEC'
      END LRSN=X'000009C50F99'
      START RBA=X'0000099A6733'
DSNI028I =DB2Y DSNIFLAF THE NUMBER OF QUALIFIED
LOG RECORDS READ DURING THE FAST LOG APPLY PROCESS IS 2063
AND THE NUMBER OF FAST LOG APPLY BUFFERS PROCESSED ARE 1
DSNR005I =DB2Y RESTART...COUNTS AFTER FORWARD
RECOVERY
IN COMMIT=0, INDOUBT=0
```

- Without fixing the underlying problem (the rename in our case), we issued the `-start database` command. This is to show you that in this particular case DB2 issued a VSAM delete and re-define to resolve the LPL errors for the temporary tables. In some circumstances the RESET will be logical (just like a REORG with the `REUSE` option — the physical data sets are not always delete/defined). In any case you need to examine the `DSNP` prefixed messages which identify the underlying problem. Sample error messages are shown below.

```
DSNP009I =DB2Y THE FOLLOWING ERROR MESSAGES WERE
RECEIVED FOR DELETE CLUSTER ON
DB2V610Y.DSNDBC.TEMP.TEMP4K01.I0001.A001
IDC3012I ENTRY DB2V610Y.DSNDBC.TEMP.TEMP4K01.I0001.A001 NOT FOUND
IDC3009I ** VSAM CATALOG RETURN CODE IS 8 - REASON CODE IS
IGG0CLA3-42
IDC0551I ** ENTRY DB2V610Y.DSNDBC.TEMP.TEMP4K01.I0001.A001 NOT
DELETED
DSNP010I =DB2Y END OF MESSAGES.
      CONNECTION-ID=DB2Y
      CORRELATION-ID=014.STARTCT
      LUW-ID=*
DSNP004I =DB2Y DSNPDLT1 - DELETE FAILED FOR
      DB2V610Y.DSNDBC.TEMP.TEMP4K01.I0001.A001.
      RC=00D70025
      CONNECTION-ID=DB2Y, CORRELATION-ID=014.STARTCT ,
      LUW-ID=*
```

- After fixing the underlying problem, the `-start database` command should resolve all pages in the LPL. Since you cannot use the `RECOVER` utility on table spaces in the temporary database, the other option you have at your disposal is to `DROP` and re-`CREATE` some or all of the temporary table spaces. If resolution of the DASD problem is expected to take a long time, `DROP/CREATE` may be quicker though we recommend that you try the `-start database` command first.

Database and table space issues



```
CREATE DATABASE DBTEMP  
AS TEMP FOR DB2A  
BUFFERPOOL BPx STOGROUP SGTEMP
```

```
CREATE TABLESPACE TEMP4K01  
IN DBTEMP BUFFERPOOL BP8 SEGSIZE n  
USING STOGROUP SGTEMP  
PRIQTY nnn SECQTY nnn
```

3.3.8 Database and table space issues

In order to be able to declare temporary tables, you must create a database AS TEMP. It is similar to DSNDB07 in that you must define a single TEMP database for each DB2 member in the data sharing group. The database is not shareable across members. CCSID is not a valid option, as the database can contain a mixture of encoding schemes.

Within the TEMP database, you should create a number of segmented table spaces. DB2 will determine which tables get put in which table spaces. We recommend that you use the same SEGSIZE and PRIQTY for each. As with the workfile table spaces, if you specify SECQTY you are likely to find the secondary extents are created at some point. Ensure that you have sufficient space to accommodate growth if you specify it and that you allocate enough TEMP space to meet your requirements for all concurrently executing threads. The normal rules for segmented table space apply in that a table cannot span table spaces. When considering the size and number of table spaces to allocate, you should consider the following:

- The maximum size required for a DTT, as they cannot span table spaces.
- The maximum amount of space required for all the DTTs that could exist at a particular point-in-time.
- You may want to have several smaller table spaces rather than a few large ones to limit the amount of space any one DTT can use.
- Whether you need to spread the DTT I/O across a number of volumes

- You will need to have one or more table spaces for each pagesize that the DTTs will need to use. That is, you need to create table spaces of 4 KB, 8 KB, 16 KB and 32 KB page sizes. When you declare a temporary table, DB2 will select one of the table spaces with the appropriate page size. If one is not available the declare will fail.

The standard CREATE TABLESPACE syntax applies, except that only the following clauses are allowed:

- IN database name — must be the TEMP database
- BUFFERPOOL
- CLOSE — however, DB2 will ignore this and decide for itself
- LOCKMAX — DB2 will ignore this value
- MAXROWS
- SEGSIZE
- USING STOGROUP/VCAT

PUBLIC will implicitly have authority to declare temporary tables in these table spaces.

You should bear in mind that the TEMP database could become quite large if the usage of declared temporary tables is high. You may need to increase the size of the EDM pool to cater for this extra database. The size of the DBD will be limited to 25% of the EDM pool.

START, STOP and DISPLAY DB are the only supported commands against this database. The standard command syntax should be used but please note the following:

- You cannot start a TEMP database as RO.
- You cannot use the AT COMMIT option of the STOP DB command.
- You cannot stop and start any index spaces that the applications have created.

Note: The only DB2 utility that is allowed against the TEMP database and table spaces is REPAIR DBD.

Restrictions



- LOBS, ROWID and user defined types not allowed
- Cannot be specified in referential constraints
- Cannot be specified in a TABLE LIKE parameter
- Cannot be referenced using private protocol
- Multi-CEC parallelism disabled
- No dynamic statement caching
- ODBC and JDBC functions that rely on catalog definitions not supported
- Thread reuse not supported for DDF pool threads
- Cannot be used within triggers (the trigger body SQL)
- Some DDL restrictions (create view, alter table and so on)

3.3.9 Restrictions

Please note the following restrictions when using declared temporary tables:

- LOBS, ROWID, and user-defined data types (UDT) columns are not allowed.
- They cannot be specified in referential constraints.
- They cannot be specified in a TABLE LIKE parameter to a user defined function (UDF) or stored procedure.
- They cannot be referenced using private protocol when BIND option DBPROTOCOL(PRIVATE) is in effect. For further information see 3.3.5, “Referencing declared temporary tables” on page 88.
- Multi-CEC parallelism is disabled for any query containing a declared temporary table.
- Dynamic statement caching is not supported for any statement containing a declared temporary table.
- ODBC and JDBC functions such as SQLTables and SQLColumns cannot be used, as the information required does not exist in the catalog.
- Thread reuse with DTTs is allowed for CICS but not for DDF pool threads.
- Triggers cannot be defined on declared temporary tables.
- Currently, DTTs cannot be used within the body of a trigger. However, a trigger can call a stored procedure or UDF that refers to a declared temporary table.

The following statements are not allowed against a declared temporary table:

- CREATE VIEW
- ALTER TABLE
- ALTER INDEX
- RENAME TABLE
- LOCK TABLE
- GRANT/REVOKE table privileges
- CREATE ALIAS
- CREATE SYNONYM
- CREATE TRIGGER
- LABEL ON/COMMENT ON

APAR identifier

The APAR identifier is as follows:

PQ32670, PQ35416.

Update with subselect



You can now use a subselect in the UPDATE SET CLAUSE

```
EXEC UPDATE DEPT
SET (DEPTSIZE) = (SELECT COUNT(*)
                  FROM EMPLOYEE
                  WHERE WORKDEPT = 'P222')
WHERE WORKDEPT = 'P222';
```

Allowed in both searched and positioned update

Number of columns selected must equal number updated

Subselect must return a single row

3.4 Update with subselect

This enhancement allows you to use a subselect which returns a single row in the UPDATE SET clause.

3.4.1 Conditions for usage

The subselect can reference a table, view, synonym or alias or a join of any of these. You must make sure the number of columns selected is equal to the number of columns to be updated and the data types must be compatible.

You need to ensure that the subselect will not return more than one value, as the statement would then fail with SQLCODE -811. You should also consider whether it is possible for the statement to return no rows. In this case, the null value will be assigned to the column to be updated. If the column does not accept null values an SQLCODE -407 will be returned. Consequently, this function is ideally suited when you want to update a column to the result of functions such as COUNT, MAX, SUM or where the subselect is accessing data by its primary key.

As only one row must be returned from the subselect, you cannot use the GROUP BY and HAVING clause in this situation.

Self referencing considerations



Self referencing is NOT allowed:

```
UPDATE EMPLOYEE
SET (DEPTSIZE) = (SELECT COUNT(*)
                  FROM EMPLOYEE
                  WHERE WORKDEPT = 'P222')
WHERE WORKDEPT = 'P222';
```



You CAN use correlation names in searched updates:

```
UPDATE EMPLOYEE T1
SET (DEPTSIZE) = (SELECT COUNT(*)
                  FROM DEPT T2
                  WHERE T2.DEPTNO = T1.WORKDEPT)
WHERE WORKDEPT = 'P222'
```



3.4.2 Self referencing considerations

When using a subselect in the SET clause of an UPDATE the object of the update and the subselect must not be the same table.

However, for a searched update, you may reference a column in the table to be updated within the subselect. You can do this using correlation names, as can be seen in the example above.

APAR identifier

The APAR identifier is as follows:

PQ30383, PQ31272, PQ33028. Please note that a toleration APAR is also required for this function: PQ31272.

Columns in order by not in select



**SELECT NAME
FROM SYSIBM.SYSTABLES
ORDER BY CREATOR**

The query should not contain:



UNION or UNION ALL



GROUP BY clause



SET FUNCTION in the select list



DISTINCT in the select list

3.5 Columns in order by not in select

DB2 no longer requires columns which are referenced in the order by clause to also be included in the select list. Prior to this enhancement the query above would have returned an SQLCODE -208.

The following restrictions apply:

- There is no UNION or UNION ALL (SQLCODE -208).
- There is no GROUP BY clause (SQLCODE -122).
- There are no SET FUNCTION in the select list (SQLCODE -122).
- There is no DISTINCT the select list (new SQLCODE -214).

Please note that the column should be included in the sort data length, as well as the sort key length, when calculating the amount of sort pool required for the query.

APAR identifier

The APAR identifier is as follows:

PQ23778.

For QMF users, PQ34118 is the APAR for the associated design change for QMF QBE

Global transaction support



Allows multiple DB2 units of recovery to share locks if they are all part of one large unit of work

- The large unit of work is called a global transaction

DB2 already supports distributed units of work

- What is new is only the ability to share locks

We will use an example to show what it means

Supported for transactions from IMS, RRS and DDF

3.6 Global transactions

The new global transaction support in DB2 V6 includes special lock processing for the situation where there are multiple DB2 units of recovery that are actually all part of one large, local or distributed, unit of work (called a global transaction). In particular, locks can now be shared between multiple DB2 units of recovery in a single DB2 subsystem when they all belong to the same global transaction.

There are a number of situations in which you may want to combine two or more existing transactions into a single larger transaction. A frequent example occurs in the course of putting an object-oriented interface in front of existing transactions.

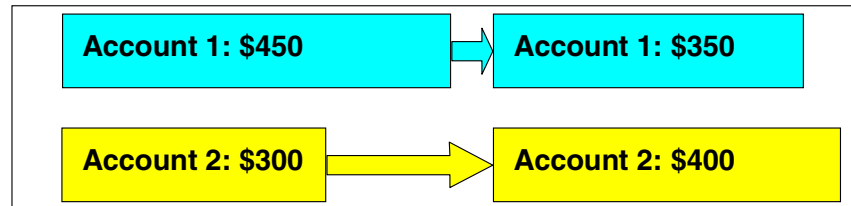
Global transaction support in DB2 provides a way to avoid deadlocks on DB2 resources that would otherwise occur when certain kinds of transactions are combined in this way.

This support is not always in effect every time DB2 performs work that is part of a distributed unit of work. It only comes into play when an optional new token (called a global transaction ID, or XID) is supplied to DB2, either through DDF from another DB2 for OS/390 subsystem, or through one of the attach facilities that includes the ability to specify the token.

This support is currently provided for DB2 units of recovery that come into DB2 using the IMS attach facility, or the Recoverable Resource manager Services Attachment Facility (RRSAF), or using DRDA through DDF.

The RRSAF support is documented in the DB2 manuals.

Funds transfer example



A transfer of funds between two bank accounts belonging to the same customer is logically broken down into 2 steps, such as:

- withdraw \$100 from account 1
- deposit \$100 in account 2

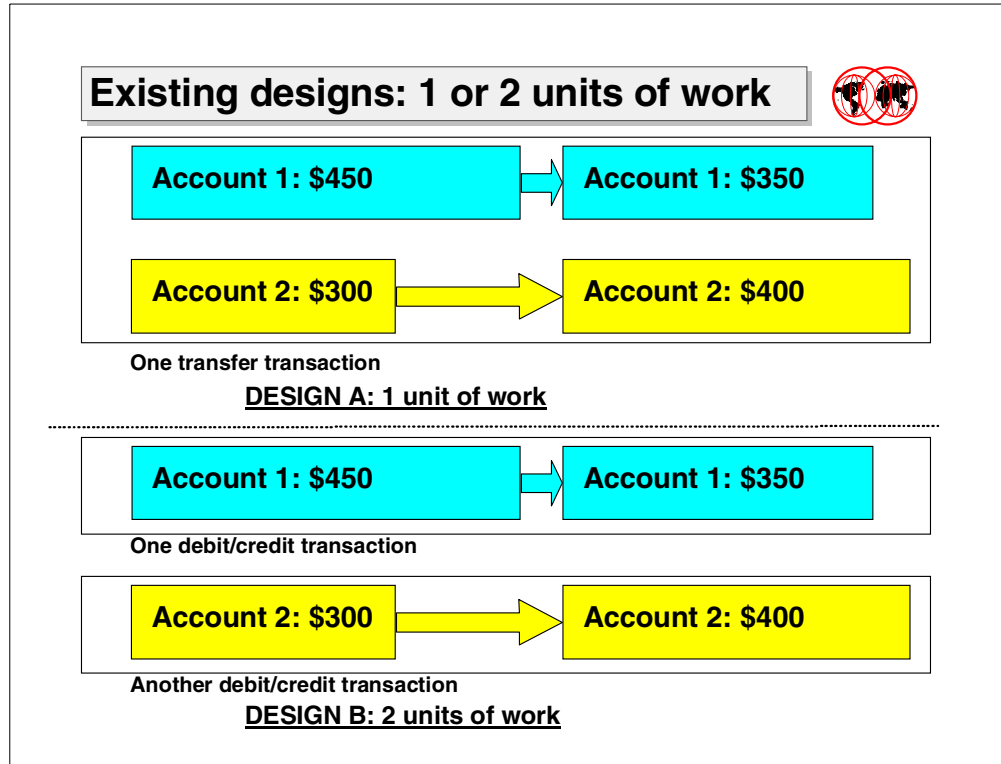
3.6.1 Funds transfer example

We now show a sample scenario.

Let us suppose that we have an existing business transaction to transfer money from one bank account held by a customer to another bank account held by the same customer.

There are two logical steps to this business transaction:

- Taking the funds from the first account, and
- Putting the funds into the second account.

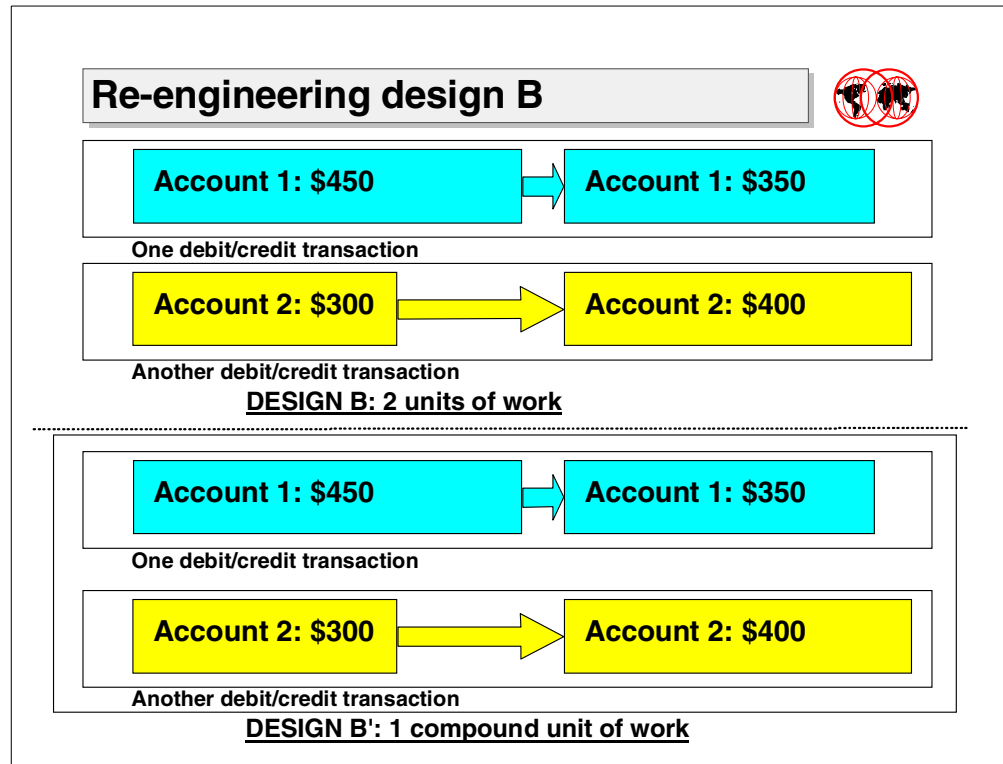


3.6.2 Existing designs: 1 or 2 units of work

We may have implemented the business transaction for the funds transfer in one of two ways:

- Design A: as one system transaction, which performs both logical steps, or
- Design B: as two system transactions, executed one after the other, each of which performs one of the two steps.

The new global transaction support in DB2 is intended to help the scenario where we have used the second of these designs, and now want to be able to combine the two transactions into one bigger transaction, while still allowing them to be executed separately in other circumstances.



3.6.3 Re-engineering design B

One of the issues we have with the original design B is that the first unit of work has been committed before we start executing the second unit of work. If there is a problem with the second unit of work, and it gets rolled back, we probably want to roll back the first unit of work as well. We have to handle this at the application level, by essentially executing another unit of work which undoes what our first unit of work did.

If we combine the two transactions into one bigger transaction, we can benefit from the two-phase commit protocol between the commit coordinator (outside of DB2) and the agents (including the DB2 threads involved). This protocol will cause the first transaction to be automatically aborted if the second transaction is aborted, avoiding the need for some application logic that was previously required.

This benefit is one of the motivating factors for re-engineering design B.

The sequence here is:

- A set of inserts / updates for debiting account 1 (DB2 thread 1),
- A set of inserts / updates for crediting account 2 (DB2 thread 2),
- A commit of the global transaction, which includes DB2 thread 1 and thread 2.

A key point is that the work done under thread 1 is not committed until after thread 2 has successfully performed its inserts and updates.

In 3.6.4, "Step 1 — Updates are performed under DB2 thread 1" on page 104 and 3.6.5, "Step 2 — DB2 thread 2 update times out" on page 105 we illustrate a problem that you can get implementing the re-engineered design.

Insert / updates OK from DB2 thread 1



DB2 thread 1

INSERT INTO ENTRIES
VALUES(:acct,...);
(sqlcode=0, X-lock on row)

ENTRIES table

ACCOUNT	DATE	TIME	DOLLAR_AMOUNT	ENTRY_TYPE	...
ACCT1	2000:03:31	19:30:05	-100.00	atm withdrawal	...
ACCT1	2000:04:01	13:00:56	-100.00	transfer out	...
ACCT2	2000:03:30	00:10:11	+1.12	credit interest	...

UPDATE ACCTBAL SET
AMOUNT=AMOUNT-:amt WHERE
ACCOUNT=:acct;
(sqlcode=0, X-lock on row)

ACCTBAL table

ACCOUNT	DATE_LAST_ENTRY	TIME_LAST_ENTRY	DOLLAR_BALANCE	...
ACCT1	2000:03:31	19:30:05	450.00	...
	2000:04:01	13:00:56	350.00	...
ACCT2	2000:03:30	00:10:11	300.00	...

UPDATE CUSTBAL SET
AMOUNT=AMOUNT-:amt WHERE
CUSTOMER=:cust;
(sqlcode=0, X-lock on row)

CUSTBAL table

CUSTOMER	DATE_LAST_ENTRY	TIME_LAST_ENTRY	DOLLAR_BALANCE	...
CUST1	2000:03:31	19:30:05	750.00	...
	2000:04:01	13:00:56	650.00	...

3.6.4 Step 1 — Updates are performed under DB2 thread 1

DB2 thread 1 is processing the first transaction (debiting account 1).

We have assumed here that each transaction performs three SQL update operations, in addition to some SQL retrieval operations (not shown):

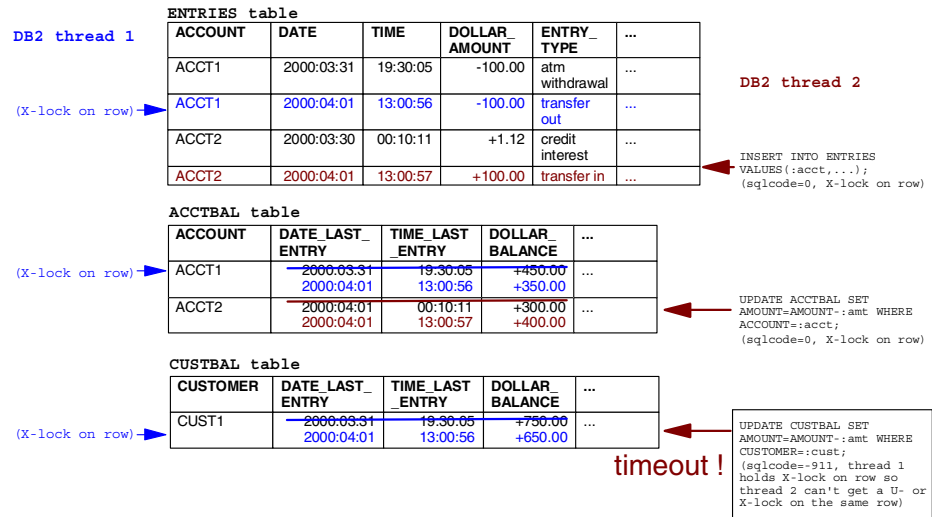
- An INSERT of a row into an ENTRIES table for this debit or credit,
- An UPDATE of the row in an ACCTBAL (account balance) table for this account, and
- An UPDATE of the row in a CUSTBAL (customer balance) table for this customer. This row indicates the net balance across all the customer's accounts.

We have also assumed that row-level locking is being used on these tables.

X-locks are taken on the 3 rows affected by the INSERT or UPDATE statements, in the usual way.

Remember that these statements are not committed until after transaction 2 has successfully finished its SQL update statements, so these X-locks remain in place for the time being.

DB2 thread 2 update times out



3.6.5 Step 2 — DB2 thread 2 update times out

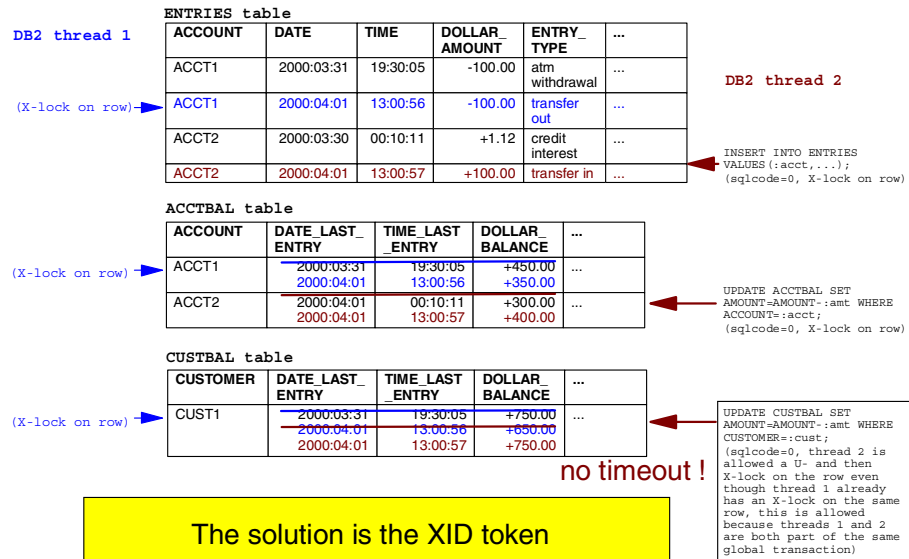
The same sequence is now executed for the second step, to credit account 2:

- INSERT the row into the ENTRIES table for the credit to account 2,
- UPDATE the row in the ACCTBAL table for account 2, and
- UPDATE the row in the CUSTBAL table for this customer. Here we hit a problem. Thread 1 is still holding an X-lock on this row. We time out. We roll back. The whole global transaction aborts, including the work done under thread 1.

This is a solid problem which will occur every single time we try to run this global transaction.

The next slide shows what happens with the solution in place.

Thread 2 in same global transaction



3.6.6 Thread 2 in same global transaction — problem solved

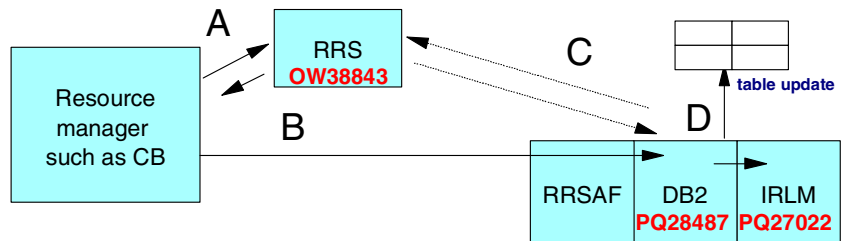
The new DB2 V6 global transaction support provides a way for DB2 to be told that the work done under thread 1 (call it unit of recovery 1) is part of the same global transaction as the work done under thread 2 (call it unit of recovery 2).

In the case where DB2 has been told this, it will allow UR2 on thread 2 to share locks taken by UR1 on thread 1.

There is support in both DB2 itself and the IRLM that enables this to happen.

It is achieved by use of a special token that represents the global transaction to which a particular DB2 unit of recovery belongs (if any). The name of this token is an XID (for X/Open Identifier). It is a standardized way of naming global transactions. Currently, the only use of it in DB2 is as a mechanism for sharing locks. It is not DB2's main identifier for a distributed unit of work, for example.

Where the XID is passed - example 1



3.6.7 Where the XID is passed — example 1

Here is a simple example of the flow of the XID token in practice.

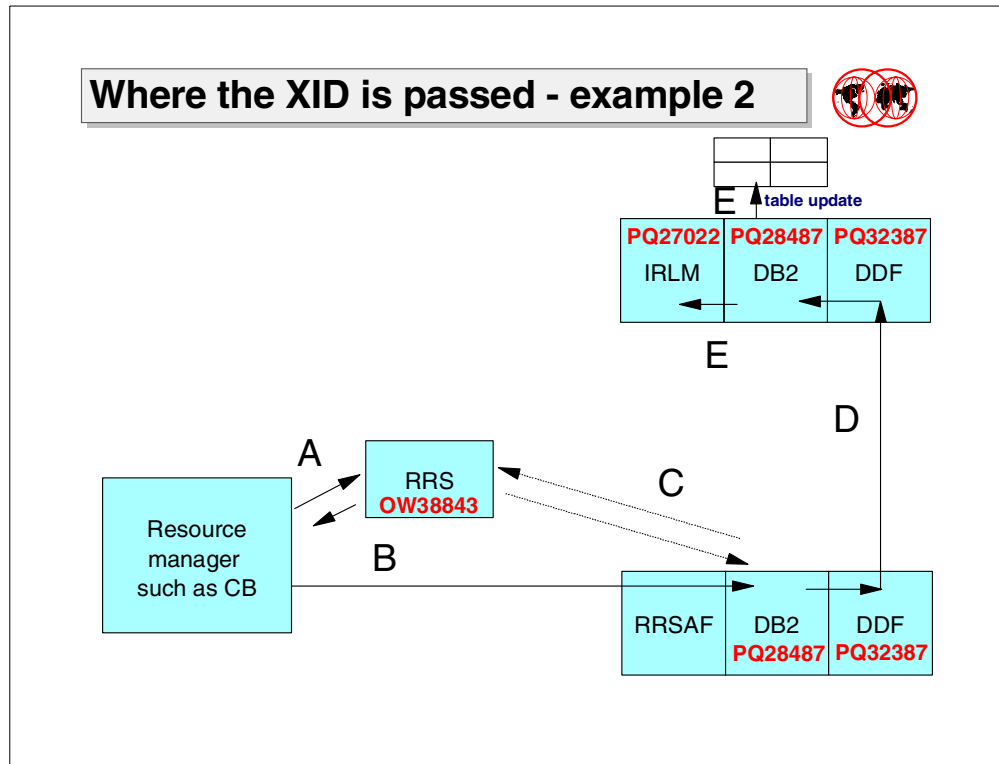
A). A resource manager such as component broker registers with RRS, and begins a global transaction. It can either assign an XID itself for the global transaction, or ask RRS to assign one on its behalf.

B). The XID is passed to DB2 on an RRSAF signon request.

C). Optionally, DB2 can check an XID value directly with RRS.

D.) When an SQL update is issued from this transaction, DB2 passes the IRLM a token based on the XID to permit sharing of locks with other DB2 URs that have the same associated token.

The reason APAR numbers are shown on this set of figures is to emphasize where new support has been added. There are 3 elements of added support in this example.



3.6.8 Where the XID is passed — example 2

This is a slightly more complicated example where the difference is that the first DB2 routes an update to a second DB2 using DDF.

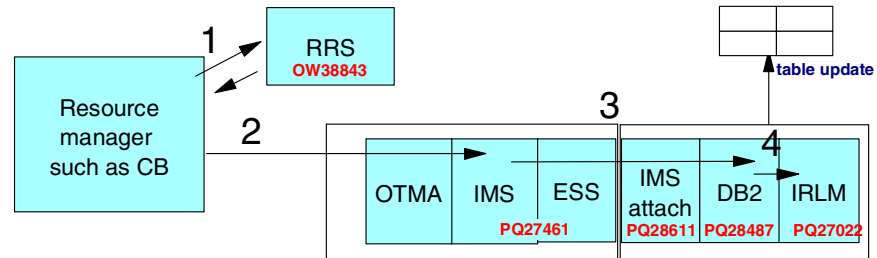
Steps A-C are as before.

D). The application requester DDF routes the update to the application server DDF.

E). The server DB2 passes the IRLM a token based on the XID to permit sharing of locks with other DB2 URs that have the same associated token eliminating locking problems.

There are four elements of added support in this example.

Where the XID is passed - example 3



3.6.9 Where the XID is passed — example 3

This is a more complicated example where the difference from the first example is that the work comes to DB2 from IMS.

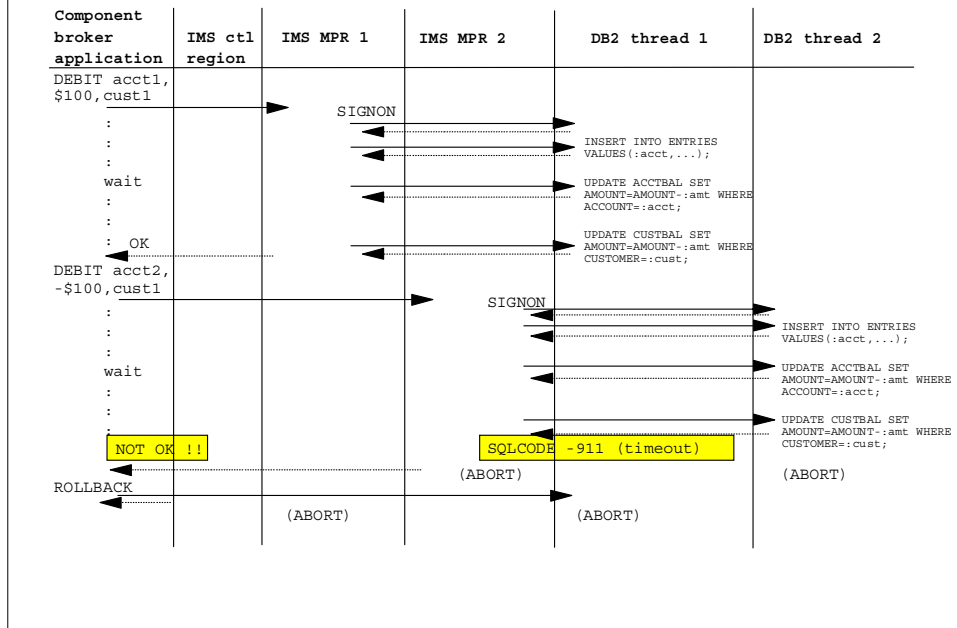
- 1). A resource manager such as component broker registers with RRS, and begins a global transaction. It can either assign an XID itself for the global transaction, or ask RRS to assign one on its behalf.
- 2). The XID is passed to IMS via the OTMA interface (Open Transaction Manager Access).
- 3). IMS passes the XID to DB2 via ESS (IMS's External Subsystem Attach Facility) and DB2's IMS attach facility.
- 4.) When an SQL update is issued from this transaction, DB2 passes the IRLM a token based on the XID to permit sharing of locks with other DB2 URs that have the same associated token.

There are 5 elements of added support in this example.

The next two slides show a simplified version of the flow of control between the participants in the global transaction where it is coordinated from a component broker application that runs 2 IMS transactions which update DB2 tables:

- First, without the DB2 global transaction support
- Second, with the DB2 global transaction support.

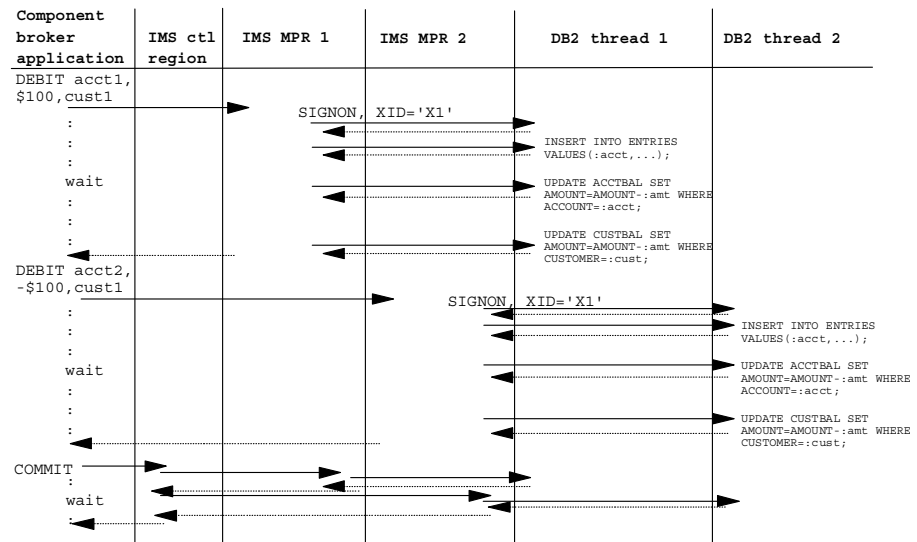
Flow between participants



3.6.10 Flow between participants

In this case, DB2 does not recognize that the work under thread 2 is part of the same global transaction as the work under thread 1. The timeout occurs, and the global transaction cannot run to successful completion.

Flow with global transaction support



3.6.11 Flow with global transaction support

In this case, DB2 is told, by a new parameter on the SIGNON request (that is part of the External Subsystem interface used to get from IMS applications to DB2 via the IMS attach facility), that:

- Transaction 1 is part of global transaction X1.
- Transaction 2 is part of the same global transaction, X1.

Considerations



Each leg can see uncommitted updates from other legs

All the DB2 URs within a single global transaction must run on the same DB2 subsystem

- not anywhere in a data sharing group
- the originator of the DB2 work must route all URs to the same DB2 subsystem

The sharing of locks is limited to normal transaction locks

- claims & drains cannot be shared in the same way
- CREATE, DROP, ALTER, GRANT, REVOKE in one leg could cause timeouts or deadlocks across other legs
- LOCK TABLE in one leg could cause timeouts or deadlocks across other legs
- Update of partitioning key in one leg could cause timeouts or deadlocks across other legs (because of drain request)

3.6.12 Considerations

Each leg can see uncommitted updates from other legs. This is a natural consequence of sharing locks. It may be an issue for re-engineering existing applications whose design is dependent on an assumption that any data they can see must have been committed.

All the DB2 URs within a single global transaction must run on the same DB2 subsystem. Component broker will send all the IMS/DB2 transactions belonging to a single global transaction to a single IMS, from which the particular IMS applications will always access the same DB2. For work coming in through DDF, the originator must use the available facilities to ensure that all requests for a given global transaction go to the same DB2 member.

You should consider the possible impact of any actions, that might be taken by one leg of the global transaction, which take locks other than normal transaction locks. Drain locks, or LOCK TABLE locks, for example cannot be shared across different legs, and so, if taken, these could result in timeouts or deadlocks.

Therefore, before using global transaction support, you should evaluate whether any of the SQL programs involved are exposed to these risks.

PTFs are required on several component products in order to make global transaction support work properly end to end. There is some documentation on global transaction support in the GA-level hardcopy manuals for DB2 V6, even though these PTFs came out later.

APAR Identifier

The following table lists the APARs, not only related to DB2, that address the various elements of support for global transactions. Check with the standard IBM support channels for the latest list of fixes you need if you want to use this new function and their applicability to your environment.

Component	APARs
IRLM	PQ27022
DB2 V6 (base support)	PQ28487
DB2 V6 (IMS attach)	PQ28611, PQ34849
DB2 V6 (DDF)	PQ32387
OS/390 V2R5+ RRS	OW38843
OS/390 V2R5+ APPC/MVS	OW39220
IMS	PQ27461

Chapter 4. Language support

Language support



DB2 REXX support

SQL stored procedures

SQLJ, JDBC driver

Java stored procedures

These enhancements extend the number of languages that can access data on DB2. Support has been provided for REXX programs to issue SQL statements; and, in addition, stored procedures can be written in Java and SQL.

DB2 REXX support



- Interface added to REXX to access DB2
- REXX support for V6 was available as download now it is included in the code refresh
- Concentrate here on hints and tips
- Can write stored procedures (SP) in REXX
 - invoked just like SP written in any other language
 - CALL procedure_name(parameters)
- Environmentals needed to get REXX SPs working

4.1 DB2 REXX support

The REXX language is widely used to automate system administrative tasks. It is powerful and relatively easy to learn. There are numerous interfaces to OS/390 environments. Added to DB2 V6 is support for the REXX to DB2 interface, DSNREXX.

Although REXX support for DB2 may be familiar to DB2 V5 customers, and it has been available for testing with V6 from the Web for some time, it has been included in DB2 V6 with this code refresh.

Rather than repeat what is already in documentation elsewhere, we will refer you to it. We concentrate here on providing some supplementary practical information based on our experiences of installing REXX support and using REXX stored procedures.

We document a sample REXX program and a REXX stored procedure called from a COBOL stub program. We describe the Workload Manager requirements and the method by which you can get diagnostic information from the called procedure.

With the refresh availability, the only method of acquiring DB2 REXX support is to specify the feature and the media when ordering DB2. Documentation is likely to remain accessible from the Web.

To order DB2/REXX support on the media of your choice, you can choose from:

Feature	Media
5108	3480 Cart
5543	4mm DAT
5216	9/6250 Mag Tape

When received as a feature, SMP support is included. Since the necessary load modules are located in SDSNLOAD, we suggest you use SMP to manage this software. When ordered as a feature, support for all the new DB2 V6 syntax is provided and all the DB2 V5 DB2/REXX functionality is supported.

In our installation we downloaded the REXX support from the Web site:

<http://www.ibm.com/software/data/db2/os390>

Host environment and handling errors



Establish the host environment (DSNREXX)

- Since it is established by DSNREXX module, it requires DSNLOAD library in concatenation

Error handling and diagnostics

- Know where you are when you get an error
- Check SQLCODE
- Display all SQLCA variables
- Check SQLCODE after every SQL call
- Rollback to return to point of consistency

4.1.1 Host environment and handling errors

In this section we describe how to interface to DB2 and handle errors from the REXX program.

4.1.1.1 DSNREXX interface

The interface to DB2 from REXX is DSNREXX. This is a load module in the SDSNLOAD library. Therefore, the SDSNLOAD library must be included in the concatenation of your starting libraries. In your REXX program you need to establish that environment. To do this we coded

```
SUBCOM DSNREXX                                /* set up host environment */
IF RC THEN                                    /* is host command there */
x = RXSUBCOM('ADD', 'DSNREXX', 'DSNREXX') /* no: so create it */
```

4.1.1.2 Error handling and diagnostics

We recommend that whenever you issue an SQL call, you do the following:

- Set a diagnostic variable indicating what SQL call you are about to issue. This means that you can also identify which call gave you the problem.
- Check the SQLCODE returned from the call to DB2 immediately after the SQL statement.
- Code the check of the SQLCODE as a REXX function.
- Display the entire SQLCA and diagnostic variable identifying the SQL call that you were attempting.
- Consider coding a ROLLBACK in your error handling routine if you receive a bad SQL code.

Our SQL code checking function was invoked as follows:

```
ADDRESS DSNREXX 'CONNECT' ssid          /* connect to DB2          */
sqlcall="Called from MAIN: Connect to DB2" /* what are re trying to do*/
rc=check_sqlcode(SQLCODE);              /* check SQL return code  */
```

This is the check we made after attempting the connection to DB2:

- `ssid` is a REXX variable (a `PARSED` parameter) and contains the DB2 subsystem name.
- `sqlcall` is the diagnostic variable that identifies the call being made, and from which point in the REXX program.
- `check_sqlcode` is the REXX function name which takes the `SQLCODE` returned by DB2 as a parameter.

We made this call after setting the `sqlcall` variables following every SQL call, although we do not show it in the remaining examples.

Our `check_sqlcode` function is as follows:

```
/* check value of SQLCODE.  If 0, fine just leave.  Otherwise display
the error messages, issue a ROLLBACK and exit with the bad code  */
check_sqlcode:
IF SQLCODE = 0 THEN RETURN 0;
SAY "Error detected at " sqlcall
SAY "SQLCODE = "SQLCODE
SAY "RETCODE = "RETCODE
SAY "SQLSTATE = "SQLSTATE
SAY "SQLERRMC = "SQLERRMC
SAY "SQLERRP = "SQLERRP
SAY "SQLERRD ="SQLERRD.1',' ,
|SQLERRD.2',' ,
|SQLERRD.3',' ,
|SQLERRD.4',' ,
|SQLERRD.5',' ,
|SQLERRD.6

SAY "SQLWARN ="SQLWARN.0',' ,
|SQLWARN.1',' ,
|SQLWARN.2',' ,
|SQLWARN.3',' ,
|SQLWARN.4',' ,
|SQLWARN.5',' ,
|SQLWARN.6',' ,
|SQLWARN.7',' ,
|SQLWARN.8',' ,
|SQLWARN.9',' ,
|SQLWARN.10

ADDRESS DSNREXX "EXECSQL ROLLBACK"
exit SQLCODE
```

Note: You do not need to code an `INCLUDE SQLCA` statement. The `CONNECT` statement assigns values to the `SQLCA` variables we display in the example above. If the `SQLCODE` is non-0, our return exits after issuing a `ROLLBACK`.

Isolation level and coding conventions



Set isolation level as appropriate

- Improves concurrency and reduces overhead
- Can be set and changed dynamically
- SET CURRENT PACKAGESET

Coding conventions

- Cursor and statement names are not arbitrary
- Host variables must be in capitals and preceded by :
- Do not declare host variables before use
- Parameter markers (?) to substitute values at execution

4.1.2 Isolation level and coding conventions

In this section we suggest the appropriate isolation level and coding conventions.

4.1.2.1 Choose appropriate isolation level

During the installation the DSNREXX package is bound five times using different isolation levels. Packages DSNREXX and DSNREXCS are both bound with isolation level CS. You can change the isolation level within your REXX program as follows:

```
ADDRESS DSNREXX "EXECSQL SET CURRENT PACKAGESET='DSNREXUR'"
sqlcall="Called from MAIN: Set packageset" /* select appropriate ISOLATION*/
rc=check_sqlcode(SQLCODE);                /* check SQL return code */
```

4.1.2.2 Coding conventions

- Cursor and prepared statement names are not arbitrary, you must use a predefined set.
- If you prepare an SQL statement from a REXX host variable, the host variable name is preceded by a colon ":" and must be in capitals.
- You can use parameter markers in SQL where you want to substitute the value of the host variable.

The following extract of code shows examples of these three principles. We have removed error checking calls for clarity, but check `SQLCODE` after every DB2 call. The full listing of our sample program can be found in Appendix C, "DB2 and REXX" on page 269.

1. The SQL statement is put into a REXX variable called `SQL_STMT`. This statement retrieves data for table space scans where the number of pages is greater

than a specified number. NPAGES is a parameter marker. The owner of the plan table is held in a REXX variable called creator.

```
SQL_STMT =,
"SELECT '  LARGE SCAN'  ,      ",
"      PLN.QUERYNO      ,      ",
"      PLN.BIND_TIME    ,      ",
"      PLN.APPLNAME     ,      ",
"      PLN.PROGNAME     ,      ",
"      TB.CREATOR       ,      ",
"      TB.NAME          ,      ",
"      TB.CARD          ,      ",
"      TB.NPAGES        ,      ",
"      TB.DBNAME        ,      ",
"      TB.TSNAME        ,      ",
"FROM SYSIBM.SYSTABLES TB,      ",
creator".PLAN_TABLE PLN      ", /* this is the owner of the */
"WHERE PLN.CREATOR = TB.CREATOR  ", /* plan_table we set as a */
" AND PLN.TNAME   = TB.NAME      ", /* REXX variable */
" AND PLN.ACCESTYPE='R'          ",
" AND TB.NPAGES > ?              " /* NOTE: parameter marker */
```

2. The next step is to declare a cursor and prepare the statement from the SQL_STMT host variable. Statement and cursor names belong to the predefined set of names. Note that :SQL_STMT is in capitals.

```
ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :SQL_STMT"
```

3. Set the value of the parameter marker and open the cursor using the host variable. Note that :NPAGES is in capitals.

```
NPAGES=20
ADDRESS DSNREXX "EXECSQL OPEN C1 USING :NPAGES"
```

4. You do not declare host variables, so in the FETCH just use upper case host variables which are then available to your REXX program. In this case we fetch rows back from the cursor, perform some processing on them within the code (not shown) and insert rows into an exception table.

```
ADDRESS DSNREXX "EXECSQL FETCH C1 INTO ",
":REASON, :QUERYNO, :BIND_TIME, :APPLICATION_NAME, ",
":PROGRAM_NAME, :TB_CREATOR, :TB_NAME, :CARD, :ACTUAL_PAGES,",
":DBNAME, :TSNAME"
IF SQLCODE < 0 THEN                                /* any SQL error - handle */
rc=check_sqlcode(SQLCODE);                          /* check SQL return code */
...
/* set up the insert statement into a REXX variable */
ISRT_STMT="INSERT INTO PAOL08.EXCEPTION ",
"(QUERYNO, BIND_TIME, APPLNAME, PROGNAME, TB_CREATOR,",
"TB_NAME, TB_CARDINALITY, TB_NPAGES)",
"VALUES (?, ?, ?, ?, ?, ?, ?)"
/* prepare the insert statement */
ADDRESS DSNREXX "EXECSQL DECLARE C3 CURSOR FOR S3"
ADDRESS DSNREXX "EXECSQL PREPARE S3 FROM :ISRT_STMT"
/* execute the insert */
ADDRESS DSNREXX "EXECSQL EXECUTE S3 USING ",
":QUERYNO, :BIND_TIME, :APPLICATION_NAME, ",
":PROGRAM_NAME, :TB_CREATOR, :TB_NAME, :CARD, :ACTUAL_PAGES"
```

REXX and stored procedures



Code and test your REXX program

- code program with arguments as necessary
 - code `PARSE ARG var1` in REXX
 - test by executing `%DB2REXX1 parm1`
- with `CONNECT` statement
 - `ADDRESS DSNREXX CONNECT ssid`
 - Valid in REXX program
 - Invalid in stored procedure

Convert to stored procedure

- remove the `CONNECT`
- create procedure (as REXX)
 - create procedure...
 - NOTE: WLM environment name,....
- issue `CALL` from program passing parameter(s)
 - `CALL SP_name(:hv)`

4.1.3 REXX and stored procedures

If you want to use REXX to code your stored procedure (SP), since there is little difference between a REXX program and REXX SP, we recommend that you first code what you plan to be your SP as a REXX main program. You should include the `DSNREXX CONNECT` statement to allow you to issue DB2 calls. This enables you to test the logic of the program easily and quickly. Any problems executing the SP can then be isolated to environmental problems rather than programming logic.

The arguments you plan to pass to your SP are defined in the `PARSE ARG` REXX statement as for a main REXX program. You can, therefore, code the arguments you intend to use exactly as you would in a main program and pass values in when you invoke the REXX. For example, you could have a REXX subroutine calls `TESTREXX` which has a `PARSE ARG p1 p2` statement. It is expecting two parameters, the values of which are assigned to REXX variables `p1` and `p2`. You could test your program as follows:

```
%TESTREXX parameter1 parameter2.
```

To convert you REXX program to a stored procedure, all you need to do is remove the `DSNREXX CONNECT`, which is not valid in an SP.

You then need to define the procedure to DB2 using the `create procedure` statement. The full `create procedure` statement we used is reported in Appendix C, “DB2 and REXX” on page 269.

The important parameters to note are:

- The procedure name

You will refer to the procedure name in the calling program. Note the owner. You probably will not qualify the SP name, but use the BIND QUALIFIER option of the calling program. In our case, we would use `QUALIFIER(ADMF001)`.

- The parameters

These are the parameters and their types that the SP expects to be passed (IN) and return (OUT). Note you are limited to a single output parameter.

- The `EXTERNAL NAME`

This is the name of the REXX program (the PDS member name) in your REXX library.

- The WLM application environment name

The `WLM ENVIRONMENT` parameter in the create procedure statement should correspond to that in the WLM stored procedures address space. We describe next how to set the WLM stored procedures address space.

Your REXX SP is invoked from a calling program. We have used a stub COBOL program to illustrate that the language of the SP is transparent to the caller. A basic but full COBOL listing has been included in the Appendix C, “DB2 and REXX” on page 269. The essential components are the definition of the parameter and the call statement.

```
01 WS-INT1                PICTURE S9(9) COMP VALUE +0.
...
EXEC SQL
CALL ADMF001.SPA4 (:WS-INT1)
END-EXEC.
```

We linked it with the DSNELI language interface and executed it under batch TSO using the `RUN PROGRAM` sub-command of `DSN`.

The SP executes from within a WLM stored procedure address space. Therefore, you must set up the WLM environment.

Set up WLM environment



Create WLM stored procedures address space

- DDNAME SYSEXEC identifies REXX library
- DDNAME SYSTSPRT for diagnostics

Set up WLM application environment

- ▶ APPLENV parameter matches that in create procedure statement
- ▶ identifies DB2 subsystem
- ▶ can run other SPs too (for example Java, C, COBOL and so on)

Useful commands

- ▶ /d wlm,applenv=wlmenv2
- ▶ /d wlm
- ▶ /vary wlm,applenv=wlmenv2,quiesce
- ▶ /vary wlm,applenv=wlmenv2,resume
- ▶ /=db2y dis proc(*.*)

4.1.4 Set up WLM environment

There are two parts to setting up the WLM environment:

- Create the started task JCL which will execute your REXX (and possibly other) SPs.
- Define the workload manager application environment in which your WLM stored procedure address space will run.

We suggest the following when creating your JCL procedure:

- Set up the JCL to match the WLM procedures considering a sufficient number of TCBs to avoid creating an address space for each CALL, without exceeding the 2 GB address space size.
- If you are running data sharing, consider specifying the group name DB2SSN.
- Do not use one WLM SP address space to manage the workload from multiple DB2 subsystems, or if data sharing from multiple groups.
- Consider having multiple address spaces to process a large number of concurrent stored procedures or to separate the workload.
- Give it a name that identifies the DB2 subsystem name. We used DB2YWLM so that the started task had the same prefix as the other DB2 address spaces.
- The `APPLENV` parameter defines the application environment. This should correspond to the `WLM ENVIRONMENT` parameter in your create procedure statement.

If the number of TCBs is insufficient, another WLM address space with the same name will be started.

The DB2SSN is passed to the procedure by the invoker, if the start parameters specify &IWMSSNM (like in "DB2SSN=&IWMSSNM,NUMTCB=15,APPLENV=WLMENV2"), different DB2 subsystems always get different WLM address spaces.

The SYSEXEC DDNAME specifies the library in which your REXX SP is located.

The SYSTSPRT DDNAME is for diagnostics and the output from SAY statements.

A full description of setting up WLM environments is beyond the scope of this book. A step-by-step account is provided in the redbook *DB2 UDB for OS/390 Version 6 Management Tools Package*, SG24-5759.

From the WLM ISPF dialog we set up a service definition for db2yrexx, an application environment called WLMENV2, and we identified the JCL procedure name to use. See Appendix C.7, "WLM configuration" on page 277 for screen prints of our WLM configuration.

Output from some useful commands we used to manipulate SPs in a WLM environment is shown in Appendix C.8, "Commands to manipulate WLM and SP" on page 278. The commands are summarized below:

Command	Purpose
D WLM	Display WLM settings
D WLM,APPLENV=WLMENV2	Display status of WLM application environment
VARY WLM,APPLENV=WLMENV2,QUIESCE	Will stop the WLM SP address space. Use this if you wish to incorporate JCL changes, such as adding DDNAME SYSTSPRT. For goal mode only.
VARY WLM,APPLENV=WLMENV2,RESUME	Will restart the WLM SP address space. For goal mode only.
-DISPLAY PROC(*.*)	DB2 command to display status of all your stored procedures
P DB2YWLM S DB2YWLM	Stop and start WLM address space. Compatibility mode only.

APAR identifier

The APAR identifier is as follows:

PQ30219, PQ33133.

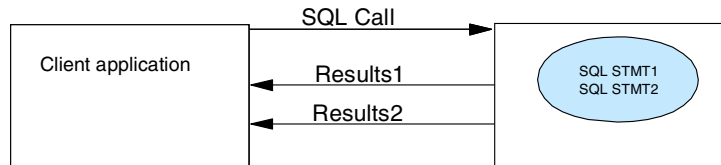
SQL Stored Procedure language



SQL Procedures are DB2 stored procedures written entirely in SQL/PSM

SQL/PSM is an ISO standard for procedural language extensions for SQL

- PSM - Persistent Stored Modules
- Similar to Oracle's PL/SQL and Sybase's Transact SQL



New statements: GOTO, GET DIAGNOSTICS and optional label on SET statement

4.2 SQL Procedure language

In this section we give an overview of SQL stored procedures (SPs), including the new language statements introduced with the code refresh.

On the basis of recent experience using SQL SPs we provide some usage recommendations and resolutions to commonly encountered problems.

DB2 offers a new stored procedure programming language which enables you to write stored procedures in a standard, portable language. Users need not know, as in regular languages, how to embed SQL statements and how to prepare the source code for use in conjunction with DB2. Instead, the SQL language has been extended to include procedural support, so that stored procedures can be written entirely in SQL statements.

The SQL Procedure language is based on SQL extensions as defined by the SQL/PSM (Persistent Stored Modules) standard. SQL/PSM, an ISO/ANSI standard for SQL3, is a high-level language, similar to other RDBMS languages such as Transact SQL from Sybase and PL/SQL from Oracle — that extends SQL to procedural support.

You write your SQL SP by using SQL. It is not executed in this form, however; you need to process the source code to generate a load module and a data base request module (DBRM). The load module must be located in a library accessible to the workload manager stored procedures address space from which it executes and the DBRM must be bound. This program preparation process is the same as that for the other supported languages except there is an additional first step. The SQL SP source is first translated into a C source code program. It is this code that is pre-compiled, compiled and link edited to form the load module and DBRM.

Note: Since SQL procedures are processed into C code, which is then pre-compiled and compiled, a C compiler is a prerequisite for SQL stored procedures.

IBM provides the Stored Procedure Builder tool. Not only does this tool perform the complete program preparation process, but will also assist with creation and testing of your stored procedure. This productivity aid is particularly useful in a development environment.

This chapter will give an overview of the language and information on how to prepare SQL stored procedures. For detailed information, please refer to *Developing Cross-Platform DB2 Stored procedures: SQL Procedures and the DB2 Stored Procedure Builder*, SG24-5485. Additional information on SQL/PSM can be found in the reference book, *Understanding SQL's Stored Procedures: A Complete Guide to SQL/PSM*, Jim Melton, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-461-8.

SQL stored procedures may already be familiar to you, as they were introduced to DB2 V5 prior to V6. Changes to the V6 implementation is that the following statements are also supported:

- GOTO statement
- GET DIAGNOSTICS statement
- Optional label on SET statement

More details in 4.2.4, “Example of a compound statement” on page 132.

Reasons for using SQL SPs



May be ported across the DB2 family

Easy and fast to code

- still need to follow good programming practice
 - ▶ performance considerations
 - ▶ error handling

Useful in highly distributed environment

General SP advantages including

- sensitive business logic on server
- business logic can be coded once and called many times
- faster execution
- security/shield mechanism
- network traffic reduction

No use of specific system services or resources

4.2.1 Reasons for using SQL stored procedures

A full discussion on the reasons for using them can be found in the redbook *Developing Cross-Platform DB2 Stored procedures: SQL Procedures and the DB2 Stored Procedure Builder*, SG24-5485. However, a brief outline of why and when you should use SQL stored procedures is reported here.

One of the main advantages of stored procedures is that they help portability across the DB2 family. Practically all of the procedural language is valid on all platforms of the DB2 family.

SQL stored procedures are based on industry standards, and the language is similar to the proprietary procedural languages of Oracle, Sybase, and Microsoft.

SQL SPs are easy and fast to code, particularly if you use the Stored Procedure Builder. The SQL Procedure language is an extension to SQL and straightforward. Programmers do not need to understand another language such as C or COBOL, nor do they need access to any resources outside of DB2. Although the SQL/PSM language is relatively easy to understand, one difficulty you are likely to face is that errors encountered while building the procedure are sometimes not reported very clearly. Consequently we recommend good programming practices such as:

- Consider how you will handle errors encountered during the execution of the stored procedure.
- Ensure that sufficient diagnostic information is passed back to the caller in the event of an error.

SQL SPs give the benefits common to all stored procedures, including improved performance over the client executing individual SQL statements, reduction in network traffic, ability for logic to be coded once and called by many applications (particularly for client applications that may be written in many different languages) and sensitive business logic that can be held on the server unknown to the clients that call it.

Considerations when using SQL stored procedures are

- You are unable to access system services or external resources. If your procedures need to do this, you will need to write them in another programming language (C, PL/1, Assembler, COBOL, REXX, and so on).
- Be aware of any special performance requirements. SQL procedures are not interpreted, but are translated into C, which is then compiled and the DBRM bound. While this is efficient, generated code is unlikely to be as efficient as a hand-crafted, optimized code written by an experienced programmer. The difference is likely to be insignificant in most cases, but may be a factor for procedures with a high performance requirement.

Creating SQL procedures



An SQL Procedure consists of

- a CREATE statement to define the procedure
- a procedure body which is made up of
 - ▶ a single simple statement
 - ▶ or a single compound statement

A simple example:

```
CREATE PROCEDURE UPDATE_SALARY_1
  (IN EMPLOYEE_NUMBER CHAR(10),
   IN RATE DECIMAL(6,2))
LANGUAGE SQL
MODIFIES SQL DATA
UPDATE EMP
  SET SALARY = SALARY * RATE
  WHERE EMPNO = EMPLOYEE_NUMBER
```

input parameters

SQL statement executed

4.2.2 Creating SQL stored procedures

An SQL procedure consists of the CREATE statement to define the procedure, together with a simple statement (as shown above) or a compound statement which is a group of SQL statements.

The statement types are:

- Local variable and condition declarations
- Local cursor declarations
- Handler declarations of the type CONTINUE and EXIT
- Procedural statements
 - Traditional DML and DDL
 - Flow control, for example, IF, CASE, LOOP, WHILE, CALL, RETURN
 - Assignment statements, for example, SET
 - Signal and resignal conditions

Although the procedure name can be up to 18 characters, we recommend that you limit it to 8 characters, so that it ties up with the loadlib member name of your generated C code and the DBRM.

Compound statements



A compound statement is delimited by a BEGIN and an END statement

Compound statements can contain

- SQL variables and condition declarations
- local cursor declarations
- condition handler declarations
- procedural statements
 - ▶ assignment statements
 - ▶ flow control
 - ▶ signal and resignal conditions
 - ▶ DDL (CREATE, ALTER, GRANT, REVOKE...)
 - ▶ DML (SELECT, INSERT, UPDATE, DELETE)

4.2.3 Compound statements

A compound statement is a group of SQL statements to be executed sequentially. All statements between the BEGIN and END clause belong to the same block or compound statement. In the example on the next foil, the compound statement does not have a label, but DB2 will define an implicit label.

The order of statements within a compound statement is as follows:

1. SQL variable and condition declarations
2. Cursor declarations
3. Handler declarations
4. Assignment statements, control-flow statements, SQL statements

Variables, cursors, conditions, and condition handlers declared inside a compound statement have the scope and lifetime of the containing compound statement. The name of a variable cannot be the same as another SQL statement within the same compound statement and cannot be the same as a parameter name or SQL reserved word.

Please note that the ATOMIC keyword is not yet supported on DB2 UDB for OS/390.

Compound statements can be nested. Hence, normal scope rules apply. That is, declarations in an inner compound statement occlude the declarations with the same name in an outer compound statement.

Example of a compound statement



```
CREATE PROCEDURE mysp (IN v_empno CHAR(6), IN rating INT) ...
BEGIN
  DECLARE new_salary DECIMAL(9,2);
  DECLARE max_sal CONDITION;
  DECLARE CONTINUE HANDLER FOR max_sal ..... ;
  SET new_salary =
    (SELECT salary FROM emp WHERE empno = v_empno);
  IF rating = 1 THEN
    SET new_salary = new_salary * 1.10;
  ELSEIF rating = 2 THEN
    SET new_salary = new_salary * 1.05;
  ENDIF;
  IF new_salary > 500000 THEN
    SIGNAL max_sal;
  ELSE
    UPDATE emp SET salary = new_salary
      WHERE empno = v_empno;
  ENDIF;
END
```

4.2.4 Example of a compound statement

An example of a stored procedure using a compound statement is shown above. It gives examples of how to declare variables, assign values to them, and declare condition handlers and signal logic.

Note: This example of a stored procedure is for illustration only. It is vital that you consider in your design how you will handle errors and debug problems. We consider this in detail below.

Below is a list of the control statements that can be used within a stored procedure:

- SET — for assignment
- LEAVE — terminates execution of labelled statements
- IF, THEN, ELSE — controls conditional execution of statements
- CASE — determines which execution path to follow
- LOOP — repeats statements until LEAVE
- WHILE — repeats statements until termination test fails
- REPEAT — repeats statements until termination test is true
- CALL — calls a stored procedure
- RETURN — stops procedure
- SIGNAL and RESIGNAL — can be used to explicitly raise conditions

The new language elements introduced in this code refresh are:

- Optional label on the SET statement. For example:

```
emplbl1: set empdtl='Investigate timekeeping';
```

This allows set statements to be the target of a GOTO statement.

- GOTO label

This new statement gives you greater flexibility coding SQL SPs. The target of the GOTO is a `label` which identifies a labelled statement where processing is to continue. There are some rules that you should follow using GOTO statements:

- If the GOTO statement is defined in a compound statement, the label must be defined inside the same compound statement, excluding a nested compound statement.
 - If the GOTO statement is defined in a handler, the label must be defined in the same handler, following the other scope rules.
 - If the GOTO statement is defined outside of a handler, the label must not be defined within a handler.
- GET DIAGNOSTICS meets a requirement to obtain information about the number of rows associated with the previous SQL statement.

An example of the use of this new feature is as follows

```
update emp set salary = salary * 1.25 where empid > '028290';  
get diagnostics numupdt = ROW_COUNT;
```

`ROW_COUNT` will be set to the number of rows updated by `update emp...`

The syntax of the new statement is `GET DIAGNOSTICS SQL-variable-name = ROW_COUNT;`

`SQL-variable-name` identifies the variable that is the assignment target. The variable must be an integer variable. SQL variables can be defined in a compound statement.

`ROW_COUNT` identifies the number of rows associated with the previous SQL statement. If the previous SQL statement is a DELETE, INSERT, or UPDATE, `ROW_COUNT` identifies the number of rows deleted, inserted, or updated, excluding rows affected by either triggers or referential integrity constraints. If the previous statement is a PREPARE, `ROW_COUNT` identifies the estimated number of result rows in the prepared statement.

Handling errors



Condition handlers

- detect error has occurred
- perform some action
- determine what happens next (CONTINUE or EXIT)

Specify set of conditions it will handle

- SQLSTATE, SQLEXCEPTION, SQLWARNING or NOT FOUND
- condition name that is user defined
- DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN...

Action to perform

- any SQL statement including a compound statement (BEGIN... END)

Passing error information back to the caller

4.2.5 Handling errors

There are three things you need to consider when handling errors encountered within any stored procedure:

- Detect that an error condition has occurred.
- Perform error recovery.
- Pass information back to the caller that the error occurred and where the problem was encountered. The caller then needs to make a decision as to the appropriate response.

4.2.5.1 Condition handlers

The `WHENEVER` statement is not valid in an SQL stored procedure; instead, condition handlers are used to trap errors. The main function of condition handlers is error processing, such as an SQL statement resulting in a negative SQL code, or the processing of a user condition. A condition handler gets executed automatically when the condition it is prepared to handle is detected during the execution of the containing compound statement. Condition handlers satisfy the first two requirements: detecting errors and error recovery.

The general form of a condition handler is:

```
DECLARE handler type HANDLER FOR condition SQL-procedure-statement;
```

Handler type can be one of these:

- **CONTINUE** — to resume the execution with the statement following the one that raised the condition

- EXIT — to resume the execution with the statement following the compound statement

Conditions you can specify are:

- A particular SQLSTATE value
- A condition name, which is user-defined
- SQLEXCEPTION, that is, all SQLSTATE values with class other than 00, 01, or 02
- SQLWARNING, that is, all SQLSTATE values with class 01
- NOT FOUND, that is, all SQLSTATE values with class 02

The SQL-procedure-statement can be any SQL statement, including a compound statement. You probably will want to use either `BEGIN` or use the new `GOTO` statement. This allows a series of actions to be carried out in response to the error.

A compound statement may contain any number of condition handlers.

4.2.5.2 Passing diagnostics back to the caller

It is essential that you consider how you will pass back to the caller sufficient diagnostic information to enable the point of failure and nature of the error.

In the redbook *Developing Cross-Platform DB2 Stored Procedures: SQL Procedures and the DB2 Stored Procedures Builder*, SG24-5485 a technique is introduced in some sample code. You create a DB2 table with, at least, two string columns. You may also wish to add `SQLID` and `CURRENT TIMESTAMP` for example. The first column stores an eye catcher set in the procedure to identify the SQL statement being attempted. The second describes the nature of the problem. For example:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
insert into error_log values ('Procedure your_proc_name processing SQL call
003','EXIT HANDLER FIRED: SQLSTATE='||SQLSTATE||' SQLCODE='||CHAR(SQLCODE));
```

The handler will ensure control returns to the caller. You can pass back `SQLCODE` or `SQLSTATE` anyway (see 4.2.6, “Debugging SQL stored procedures” on page 136). In this case, the first thing the caller should do is to check the contents of the `error_log` table. The exact location of the failure can then be determined.

We recommend that you exploit declared temporary table or created temporary table support rather than a persistent table, because you will not encounter locking problems, and in any case, the data is required only for the caller to read and display.

It is possible to imagine some circumstances in which the insert to the `error_log` table will fail, such as problems with the work or temporary databases, at the same time that the SP was encountering an application error. In this case, although you will see console messages from DB2's MSTR address space, the underlying problem and point of failure will not get reported back to the SQL SPs calling program.

Debugging SQL stored procedures



When debugging your SQL stored procedure:

- define a return variable for `SQLCODE` or `SQLSTATE`
- if you specify `EXIT` you can only do one thing - make sure it's setting up the return parameter list with the `SQLCODE`
- if you specify `CONTINUE` you can do more but it is not as clean
- in debugging mode have something tell you where you are in the program

Can use Debugger to debug the generated C code

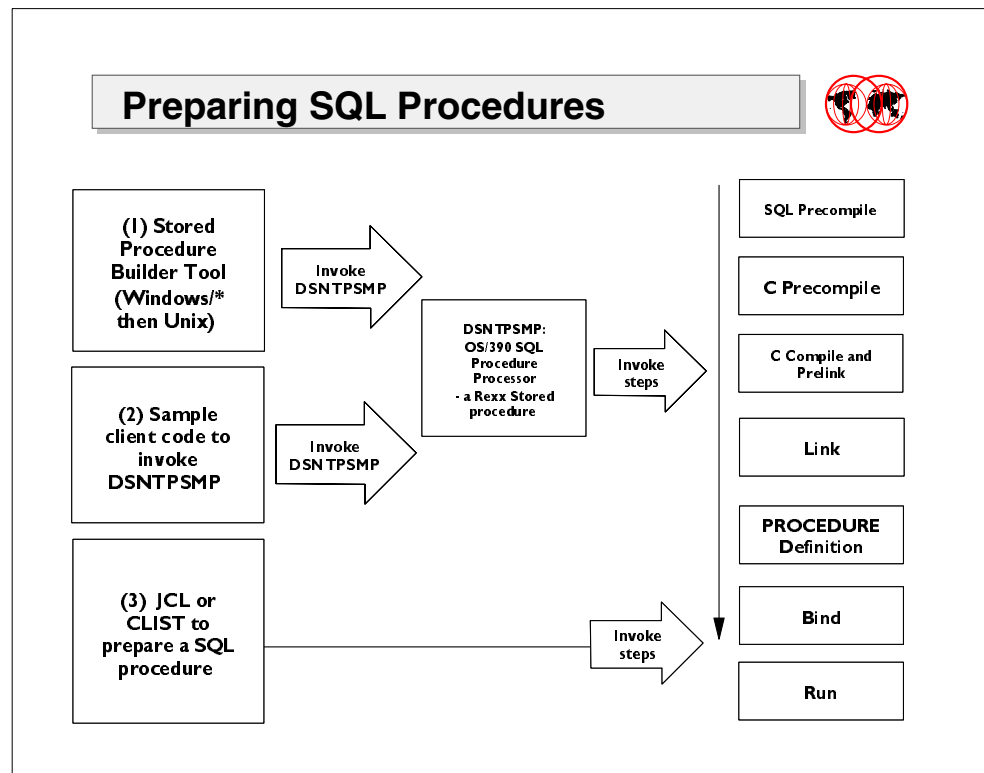
4.2.6 Debugging SQL stored procedures

In order to assist you with debugging your SQL stored procedure we recommend the following:

Always define a return variable for either the `SQLCODE CHAR(6)` or `SQLSTATE CHAR(8)`. Make sure that the error handler passes back the `SQLCODE` or `SQLSTATE` to the program. If you specify `EXIT` for the error handler, you are only allowed to do one thing. You could specify `CONTINUE`, which allows you to do more, but it is not as straightforward.

If you are familiar with C, in a development environment you can add `printf` statements to assist with debugging.

You can also use the IBM Distributed Debugger tool to debug the generated C code.



4.2.7 Preparing SQL stored procedures

There are three main methods for developing SQL stored procedures on DB2 for OS/390:

- Using the Stored Procedure Builder (SPB) tool, which runs on all current Windows platforms. It provides an easy-to-use development environment for creating, installing, and testing stored procedures. With the DB2 SPB you can focus on creating the stored procedure logic rather than on the details of registering, building, and installing stored procedures on a DB2 server. Once you have created or modified the stored procedure it invokes the OS/390 Procedure Processor (DSNTPSMP) to perform all the steps on the right hand side of the diagram. For information on how to use the SPB, please refer to the redbook *DB2 UDB for OS/390 Version 6 Management Tools Package*, SG24-5759.
- Using the OS/390 SQL Procedure Processor directly. You can code a client program to invoke DSNTPSMP to generate and build your stored procedure. You may choose this option over the SPB if you want to standardize the bind, link and compile options within a project. You could limit which parameters the developers can specify. Information on how to invoke DSNTPSMP can be found in the SQL procedures redbook.
- Using JCL or CLIST to manually perform all the steps on the right-hand side of the diagram. Unlike the other two options, this does not require you install DB2 REXX support or create a Workload Manager environment and stored procedures address space in which to run DSNTPSMP.

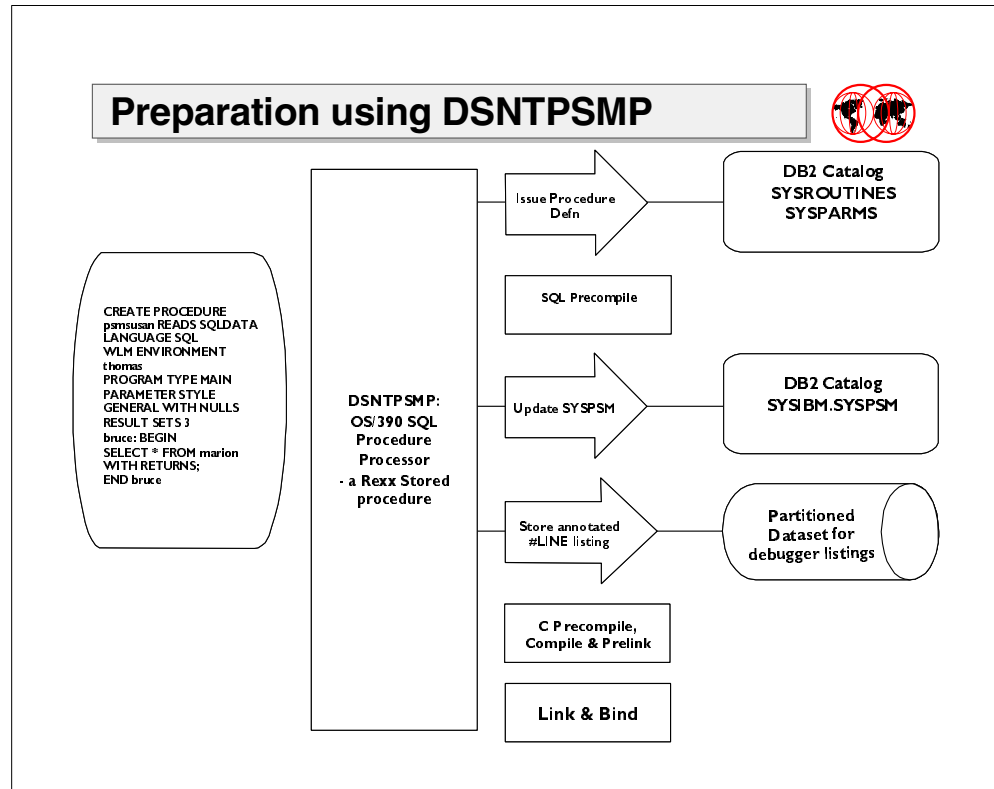
In a development environment, we recommend that you use the SPB tool, as it helps you in coding, testing and debugging, thereby improving the development process.

To install stored procedures into production you will probably want to use the batch option, invoking the JCL procedure DSNHSQL (or a modified version thereof) as part of your change control process.

The steps involved in the creation of the SQL stored procedure are as follows:

1. The user creates the SQL procedure logic source (manually or using SPB).
2. The SQL procedure source is precompiled by DSNHPSM resulting in a C language program complete with SQL and logic.
3. The SQLC source is precompiled by the normal DB2 precompiler like any other C program.
4. The modified C source is compiled and link edited.
5. The DBRM is bound into the chosen collection.
6. The procedure must be defined to DB2 using the CREATE PROCEDURE statement.

Regardless of the method you use to build your stored procedure, you can invoke the Debugger tool to debug it if you have the Distributed Debugger client code on your workstation. This is shipped with the DB2 Software Developers Kit (SDK).



4.2.8 Preparation using DSNTPSMP

The OS/390 Procedure Processor (DSNTPSMP) automates and performs all the steps required to generate and build your SQL stored procedure. You can use it either from the SPB or by writing a client program to invoke it.

DSNTPSMP is a DB2 supplied stored procedure written in REXX that performs all the steps required to prepare an SQL procedure. As well as building and defining the procedure, it populates and makes use of a new catalog table called SYSIBM.SYSPSM. This table contains information such as the CREATE PROCEDURE statement.

DSNTPSMP can be invoked to perform three major functions:

- Build/rebuild a procedure
- Destroy a procedure
- Rebind

Output from the OS/390 SQL procedure processor includes the PSM state indicating how far the process got, the SQLCODEs for PSM CREATE and Bind.

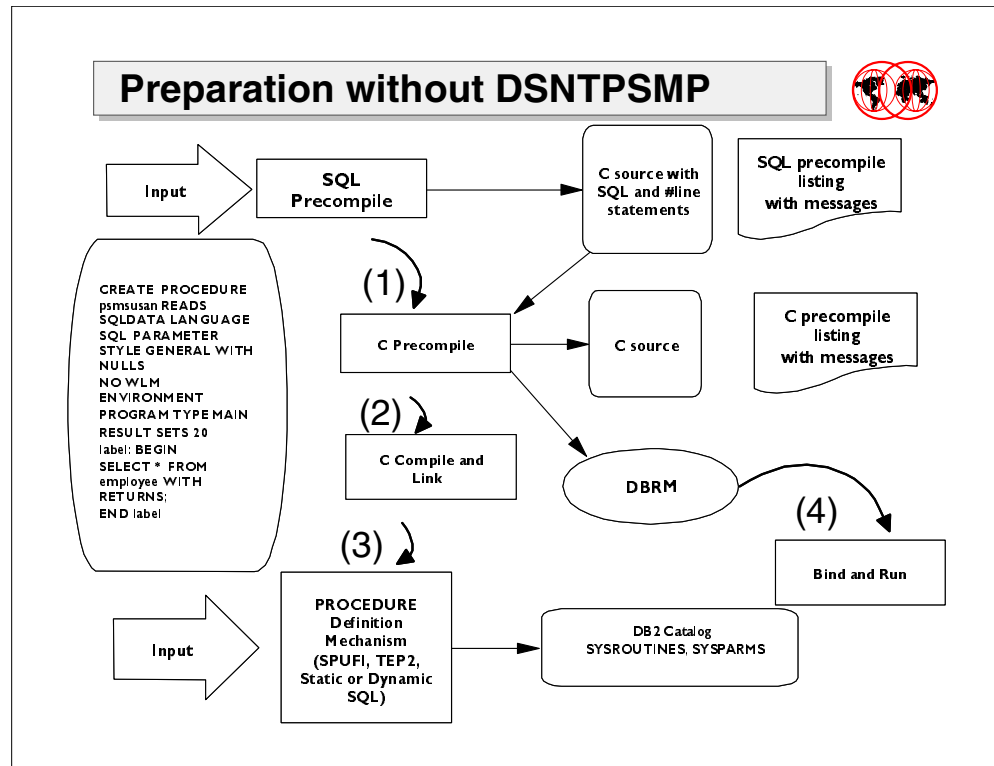
4.2.8.1 WLM requirements

DSNTSMP should be set up in a WLM environment of its own. It is recommended that no other stored procedure should be defined in this environment. The OS/390 Procedure Processor requires that the NUMTCB in the WLM region be 1. The main consequence is that only one SQL stored procedure can be built in each WLM environment at one time. However, it is possible to have more than one WLM environment, each running its own version of DSNTSMP. You may want to set up multiple WLM environments for different projects or testing levels, each referencing different sets of libraries.

Sample JCL can be found in DSN8WLMP.

4.2.8.2 REXX requirements

Because the DSNTSMP stored procedure builder is written in REXX, you must install REXX support for DB2.



4.2.9 Preparation without DSNTPSMP

This method uses JCL which invokes the procedure DSNHSQL to prepare the SQL Procedure. DSNHSQL is installed by member DSNTIJSQ in your SDSNSAMP library. As part of the Installation Verification Process (IVP) you will run job DSNTEJ63 which prepares an SQL stored procedure by invoking DSNHSQL in batch. The IVP job performs the following steps:

1. Uses DSNTPE2 or DSNTIAD to drop the procedure in case it already exists
2. Calls DSNHSQL which performs the following:
 - DSNHPC is called with PARM='HOST(SQL)'. This converts the SQL procedures source into an equivalent C language program.
 - DSNHPC is called for a second time with PARM='HOST(C)' to precompile the C source to process the embedded SQL calls.
 - C compile and link edit
3. Uses DSNTIAD to run the CREATE PROCEDURE statement. This will insert rows into SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS
4. Bind the DBRM

Please note that Workload Manager and REXX support are not required for preparation of stored procedures using JCL.

APAR identifier

The APAR identifier is as follows:

PQ29782, PQ30467, PQ30492, PQ33026, PQ33560, PQ24199.

SQLJ/JDBC driver support



Optimized JDBC type 2 driver (replaces type 1 driver)

- Translates calls into native programming requests
- Performance improvement

One common runtime environment for both SQLJ and JDBC

- Establish interoperability to combine SQLJ and JDBC
- Can process JDBC result sets using SQLJ locators

Support for SQLJ/JDBC JVM applications under CICS

4.3 SQLJ/JDBC driver support

DB2 introduced Java support initially in V5 with JDBC type 1 driver support. JDBC is an API that Java applications use to access any relational database. It implements a series of methods (like ODBC) that permit dynamic SQL statements to DB2. Since the JDBC methods are generic, there is a translation process carried out by a JDBC driver to interface to DB2. The type 1 driver acts as a JDBC ODBC bridge and translates all JDBC calls to ODBC.

DB2 V6 provided support for SQLJ Part 0, which is an ANSI standard set of extensions that enable Java applications to include embedded static SQL.

The enhancement in this code refresh is that support is now provided for the SQL/JDBC driver. It was also added to DB2 V5, so it may be familiar to you. The new driver enables you to write a Java program that consists of both static and dynamic SQL. The same application can execute SQLJ clauses and invoke JDBC methods.

It has been implemented as a type 2 driver, one of four types of JDBC drivers defined by Javasoft. The type 2 driver translates JDBC calls into native programming requests. Consequently, this will perform better than the JDBC type 1 driver that it replaces.

Support is also provided for Java Virtual Machine (JVM) applications to now run under CICS. Currently, Java applications running on the OS/390 platform must be compiled by the High Performance Java (HPJ) compiler (part of the Enterprise Toolkit for OS/390). However, this new driver support together with CICS Transaction Server for OS/390 1.3 and CICS APAR PQ34321, gives you the option of running CICS/DB2 applications using JVM. The JVM is a complete software microprocessor with its own instruction set and operation codes. It provides automatic memory management, garbage collection, and other functions for the programmer.

Please note that this section includes an overview of Java enhancements since V6 GA. For detailed information on Java in general, please refer to the redbook *How to Build Java Stored Procedures: DB2 UDB Gets Wired With SQLJ and JDBC*, SG24-5945, due to be published soon, and the manual *Application Programming Guide and Reference for Java*, SC26-9018-01.

Using the new driver



Identifying target data source - URLs

- jdbc:db2os390:<location name>
- jdbc:db2os390sqlj:<location name>

New special URLs - application does not need to know DB2 location name

- "jdbc:db2os390:" for JDBC
- jdbc:default:connection for SQLJ

Driver class name

- COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver

4.3.1 Using the new driver

A Java application identifies the target data source it wants to connect to by passing a database Uniform Resource Locator (URL) to the DriverManager. For a DB2 for OS/390 data source, you can specify either:

- jdbc:db2os390:<location name>
- jdbc:db2os390sqlj:<location name>

There are two new special URL values for the SQLJ/JDBC driver:

- "jdbc:db2os390:" — The JDBC application does not need to know the location name of the local DB2 subsystem that the driver is using
- jdbc:default:connection — SQLJ specification to connect the application to the local site without knowing the location name.

Several packages are included with the DB2 for OS/390 SQLJ/JDBC driver. These packages represent the DB2 for OS/390 implementation of the java.sql JDBC API. The driver packages include all of the JDBC classes, interfaces, and exceptions that comply with the JDBC 1.2 specification.

The DB2 for OS/390 SQLJ/JDBC driver is available under two different Java class names. The preferred driver name is:

```
COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver.
```

The `ibm.sql.DB2Driver` is also supported, but just for compatibility reasons.

Combining SQLJ and JDBC



Can combine SQL and JDBC in a single program

Set environmental variables and properties in SQLJ/JDBC run-time

Establish interoperability

- By binding JDBC packages in SQLJ plan
- Make sure JDBC profile is accessible

Particularly useful processing JDBC results sets using SQLJ iterators

- Columns and data types must match
- Use `getResultSet` to generate JDBC result set from SQLJ iterator

4.3.2 Combining SQLJ and JDBC

In order to enable programmers to write Java programs that use both JDBC and SQLJ, you must carry out the following setup tasks:

- Set environmental variables
- Set parameter in the `sqlj/jdbc` run-time properties file

Information on setting the parameters can be found in the *Java Application Programming Guide and Reference for Java*, SC26-9018-01.

You must also establish interoperability by doing the following:

- When you bind a plan for SQLJ, include the JDBC packages in the PKLIST of that SQLJ plan. The default names of the JDBC packages are:

```
DSNJDBC.DSNJDBC1
DSNJDBC.DSNJDBC2
DSNJDBC.DSNJDBC3
DSNJDBC.DSNJDBC4
DSNJDBC.DSNJDBC5
```

- Make sure that the JDBC profile is accessible in a directory specified in the CLASSPATH environmental variable.

This new support means that you can use JDBC result sets in SQLJ applications. This enables you to take advantage of the flexibility of JDBC and the type checking of SQLJ.

APAR identifier

The APAR identifier is as follows:

PQ36011.

Java stored procedures



You can call a Java stored procedure in the same way as for other languages

JVM is not required. Use HPJ (part of Visualage for Java) to compile the SP

Can include JDBC and SQLJ

Must run as a subprogram

Note: Available June 30, 2000 as a follow on to the refresh level

4.4 Java stored procedures

Stored procedures can be written in the following languages

- C
- COBOL
- Assembler
- PL/1
- REXX
- Persistent Stored Modules (an ISO standard procedural SQL language)
- Java

Java is a new member of this list. You can call a Java stored procedure exactly as you would call a procedure written in any other language.

JVM is not required for Java stored procedures. They are compiled using the High Performance Java (HPJ) compiler, which is part of VisualAge for Java Enterprise Edition for OS/390. This gives better performance than if they were interpreted.

This section will provide an overview of the issues involved when writing Java stored procedures and an outline of how to prepare them. For detailed instructions and advice refer to the redbook *How to Build Java Stored Procedures: DB2 UDB Gets Wired With SQLJ and JDBC*, SG24-5945.

Please note that this functionality is not included as part of the May Refresh (or PUT03). However it will soon be available through additional PTFs. Check the Web site <http://www.ibm.com/software/data/db2/os390/> for availability.

Defining a Java stored procedure



```
CREATE PROCEDURE JSPTEST  
(CHAR(10) IN, DECIMAL(31,2) OUT)  
FENCED  
READS SQL DATA  
LANGUAGE COMPJAVA  
EXTERNAL NAME(classname.methodname)  
PARAMETER STYLE JAVA  
WLM ENVIRONMENT WLMCJAV  
RESULT SETS 1  
PROGRAM TYPE SUB;
```

4.4.1 Defining a Java stored procedure

You define a Java stored procedure to DB2 using the CREATE PROCEDURE statement in the same way as for stored procedures written in other languages. However, the following parameters have different meanings for Java stored procedures:

EXTERNAL specifies the program that runs when the procedure name is specified in a CALL statement. For Java stored procedures, the form is EXTERNAL NAME 'class-name.method-name' which is the name of the Java executable code that is created by the HPJ compiler. If the class is defined in a package, it is prefixed with the package name.

The following parameters must be specified:

- LANGUAGE COMPJAVA
- PARAMETER STYLE JAVA — required so that DB2 uses a parameter passing convention that conforms to the Java language and SQLJ specifications.
- WLM ENVIRONMENT — Java has to run in a workload managed environment.
- PROGRAM TYPE SUB — Java SPs cannot run as MAIN routines

RUN OPTIONS will be ignored if you specify any. Because the Java Virtual Machine (JVM) is not destroyed between executions, language environment options cannot be specified for an individual stored procedure.

Java SP coding considerations



Must have PACKAGE as first statement

- bear in mind performance when grouping the procedures

Must be declared as static and public

- pass return codes back as parameters

Parameters must be mapped to base SQL types

Note that nulls require special handling

Cannot include the following statements

- Call, Commit, Set current sqlid
- Connect, Release, Set connection

Cannot make IFI calls

LOBS not supported

4.4.2 Java SP coding considerations

The Java stored procedure must have PACKAGE as the first statement in the source. This refers to a Java package, rather than a DB2 one, and is subsequently mapped to the load module produced by the HPJ.

You can group stored procedures into Java packages. A Java package equates to a single HPJ executable module. When deciding on the groupings, bear in mind that the entire module will be loaded in at run-time; therefore, it is worthwhile clustering small often-used packages together and keeping large less-used procedures separate.

Java stored procedures must be defined as static and public. Consequently, they provide no return codes. You should define output parameters to pass back information, including return codes, to the caller.

The following should also be noted:

- Parameters must be mappable to base SQL data types.
- Nulls require special handling
- Results set issues

Please note the following restrictions:

- As in other stored procedures, you cannot include COMMIT, SET CURRENT SQLID.
- In addition, like any Java program, you cannot include connect, release, or set connection.
- A Java SP cannot make IFI calls
- LOBS are not supported

Returning results set



Other languages simply use WITH RETURN clause

In Java you must do the following:

- use the "with returnability" clause
- include an object type Resultset in the parameter for the SP method
- SELECT to obtain contents of result set
- retrieve any rows you do not want to return to client
- assign contents to the ResultSet object

4.4.3 Returning results set

Stored procedures written in other languages would use the WITH RETURN clause on the DECLARE CURSOR clause to return the result set from a stored procedure.

An equivalent in Java would be to specify the "with returnability" clause in an SQLJ iterator declaration clause. However, this is not enough to return the results set.

For each result set, your stored procedure must:

- Include an object type Resultset in the parameter list for the stored procedure method, but not in the parameter list of the stored procedure definition.
- Execute a SELECT statement to obtain the contents of the result set.
- Retrieve any rows that you do not want to return to the client.
- Assign the contents of the result set to the ResultSet object that is in the parameter list.

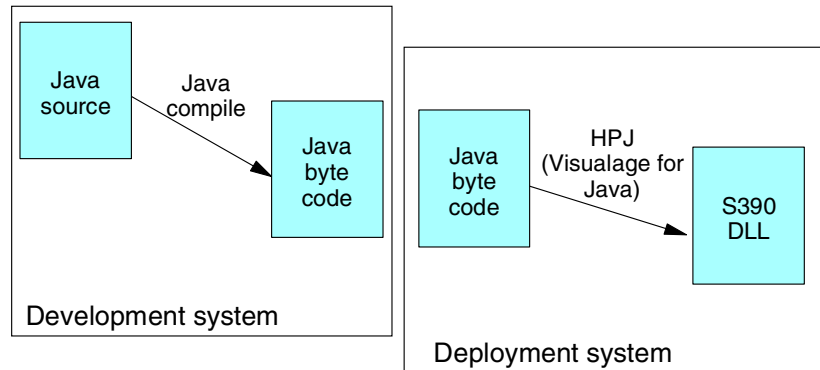
Your stored procedure can return multiple query result sets to a DRDA client if:

- The client supports the DRDA code used to return query result sets.
- The value of RESULT_SETS in the stored procedure definition is greater than one.

Preparing Java stored procedures



Similar to preparing other Java programs - but one extra step
Need to compile SP using ET/390 binder



4.4.4 Preparing Java stored procedures

Preparing a Java stored procedure is similar to preparing any other Java DB2 program. However, there is one extra step. Java programs must run as compiled Java programs so you must use the VisualAge for Java Enterprise Edition for OS/390 (ET/390) binder.

To use Java stored procedures in an OS/390 environment you need

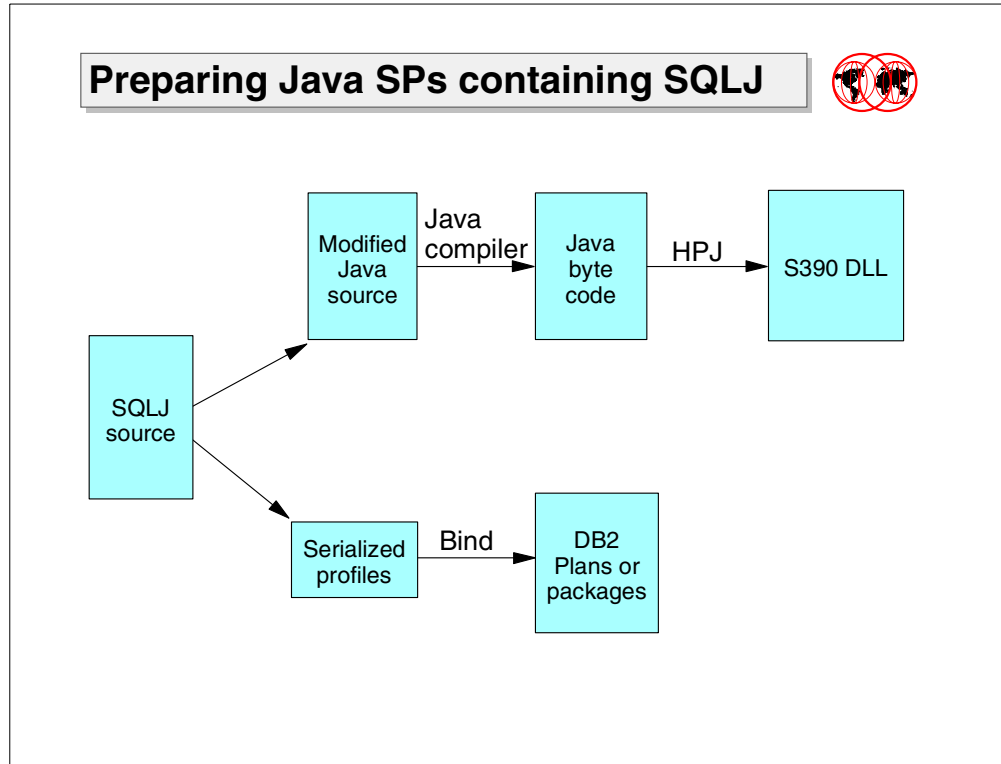
- OS/390 V2R6 or above
- UNIX System Services must be enabled
- DB2 UDB for OS/390 Version 6 with APARs PQ36011 and PQ31846
- Enterprise Toolkit for OS/390 which provides the High Performance Java (HPJ) compiler. This is part of VisualAge for Java, Enterprise Edition.
- OS/390 Recoverable Resource Services (RRS) and Workload Manager (WLM) are mandatory for running the Java stored procedures.

For more information refer to the redbook *Java Programming Guide for OS/390*, SG24-5619.

If the program contains only JDBC calls and no SQLJ statements:

1. Issue the `javac` command to compile the Java program
2. Bind the Java bytecode file produced into a Java DLL using the ET/390 HPJ.

If the program contains SQLJ clauses, there are some additional steps, as shown in the next section.



4.4.5 Preparing Java SPs using SQLJ

The diagram above shows the process of preparing a Java stored procedure which contains SQLJ clauses:

1. Issue the sqlj command to translate the source code into modified Java source code and serialized profiles.
2. Issue the db2profc command on USS to customize the serialized profiles to produce DBRMs.
3. Issue the javac command to compile the Java program.
4. Bind the Java bytecode files for the stored procedure and any packages it uses into Java DLLs in a PDSE using the ET/390 binder. Create an external link to the PDSE member.
5. Bind the DBRMs into packages and plans.

Running Java stored procedures



Uses default RRS connection to the local data source

Extra DD cards statements required for Java SPAS

Must set environmental variables using JAVAENV DD card

- Must set CLASSPATH and LIBPATH

Include JSPDEBUG DD card

4.4.6 Running Java stored procedures

A Java stored procedure does not establish its own connection to the local data source like a standard Java program would. Instead, the stored procedure uses the default RRS connection to the data source that processes the CALL statement.

A Java stored procedure must run in a WLM-established stored procedure address space. The startup procedure requires extra DD statements not required by other stored procedures. These are:

- **JAVAENV** — This holds the Java environmental variables that specify system properties for the ET/390 Java execution environment. The CLASSPATH variable must be set to include the directories for user external links, compiled SQLJ/JDBC external links and the ET/390 links. LIBPATH must include the path for the ET/390 code and the path for the SQLJ/JDBC driver code.
- **JSPDEBUG** — This is an optional DD statement that will help you debug your stored procedure.

The steplib concatenation should include the PDSE that contains the Java program objects for the stored procedures and Java classes used by those procedures. It should also include the PDSE that contain the VisualAge for Java compiler and run-time library.

APAR identifier

The APAR identifier is as follows:

PQ36011 and PQ31846 (open at the time of writing).

Chapter 5. Operational enhancements

Operational enhancements



- Suspend update activity
- Defer defining data sets
- DDF suspend
- Faster cancel thread
- Data sharing enhancements
- New EDM pool parameter
- Copy enhanced integrity checking option
- Runstats enhancements

In this chapter we describe enhancements that will help you with DB2's operations:

- Suspend update activity
New commands allow to temporary freeze logging activity so that a consistent, almost instantaneous copy of your data can be taken.
- Defer defining data sets
Option to delay the VSAM definition of your DB2 objects to when they are really utilized for the first time.
- DDF suspend
Option to SUSPEND DDF in case high priority DDL needs executing.
- Faster cancel thread
Faster termination of a DB2 local thread even when it is not within DB2.
- Data sharing enhancements and a new EDM pool parameter
These provide performance and virtual storage constraint relief.
- The new CHECKPAGE option for the Copy utility
This performs enhanced integrity checking of your data without the need for running a separate DSN1COPY CHECK utility.
- Better statistics provided by Runstats

Suspend update activity



Temporarily "freezes" all updates to a DB2 subsystem

Intended for use with RVA SnapShot or ESS FlashCopy

Minimal disruption to take backup for disaster recovery

- Suspend updates for a brief period while the system is 'snapped'

Straightforward and rapid restart at a secondary site

- When DB2 is started forward and backward recovery completes as normal
- As fast as DB2 restart following a crash at the local site

Can be useful as testing tool

- development only
- freezes activity so you can simulate problems (how applications respond to deadlock and time-out and so on)

5.1 Suspend update activity

This new feature enables you to suspend all updates to a DB2 subsystem. This allows you to take a snapshot of the system for local or remote site recovery with minimal impact on the availability of the system and restart with a consistent copy of your DB2 data..

It is designed to be used in conjunction with external copy of DB2 data such as provided by RVA SnapShot or Enterprise Storage Server (ESS) FlashCopy technology. Prior to this enhancement, these techniques alone could not be used to take a copy suitable for recovery while DB2 was active. The reasons for this are discussed in 5.1.2, "Use of SET LOG SUSPEND command" on page 158.

Should a disaster occur, the snapshot (here used to mean either of the two techniques mentioned) can be used to recover the system to a point of consistency simply by starting DB2. Offsite recovery is as fast as a normal DB2 restart following a crash at the local site.

The snapshot can also be used to provide a fast, consistent copy of your data for reasons different from recovery; one example is to periodically snap an entire system to enable point in time query with minimal operational impact.

5.1.1 Deciding whether to use this method for disaster recovery

There are numerous techniques whereby offsite recovery can be achieved in the event of a disaster. The option you choose will depend on various factors, including:

- The permitted outage to the production system and data
- How quickly you need to reestablish the service at the recovery site
- How current you require the data to be
- Cost considerations

This new feature means that you can now take advantage of the speed of RVA Snapshot or ESS Flashcopy to create an offsite recovery backup. Suspending updates for just a minute or two should be enough to obtain the backup.

To explain why these facilities are so fast at backups, here is a brief outline of how RVA Snapshot works. For further details, please refer to *Using RVA and SnapShot for Business Intelligence Applications with OS/390 and DB2*, SG24-5333.

Within the RVA, the Functional Track Directory table (FTD) maps tracks as understood by DB2 to their physical location on disk. When a SnapShot copy is made, the RVA copies the pointers so that there are two sets: the original pointers and the copies. This is why the SnapShot process is so fast — no data is physically copied or moved, only the pointers are copied. To begin with, both the copy pointers and the original pointers address the same physical location. After an update, the copy pointers continue to address the old location, but the original points to the new location of the updated data.

ESS FlashCopy works by establishing a relationship between the source and target volumes. This means that disk space capacity for the target must exist at the time the copy is made. ESS initiates a background task to copy tracks from the source to the target volume. DB2 read requests will be satisfied from data sets on the source volume even if the background copy operation has not yet completed. If DB2 needs to update a track that has not yet been copied, ESS copies the tracks from source to target before allowing the update to the source. The integrity of the copy is therefore maintained. If you begin dumping to tape data from the target volume that has not yet been copied, ESS satisfies the read (dump) requests from the source volume.

Whichever of these is used, recovery from these backups is very fast and simple, as it just requires a start of the system which will resolve inflight units of recovery.

For more information on using ESS FlashCopy in a disaster recovery solution refer to the white paper *SAP R/3 Storage Management Split Mirror Backup and Recovery on IBM's Enterprise Storage Server on DB2 OS/390* available from the Web site <http://www.storage.ibm.com/hardsoft/diskdr1s/technology.htm>.

Use of SET LOG SUSPEND command



***Essential* to issue a log suspension before taking a SnapShot suitable for offsite recovery**

Recommended backup procedure

- Issue -SET LOG SUSPEND command
- Copy all data, logs and BSDS data sets using RVA SnapShot or ESS FlashCopy
- Resume update activity
- Send backup copies offsite
 - Peer-Peer Remote Copy (PPRC) - synchronous mirroring
 - Extended Remote Copy (XRC) - asynchronous mirroring
 - copy target volumes to tape

5.1.2 Use of SET LOG SUSPEND command

Prior to this enhancement, it was essential to stop DB2 before taking a SnapShot copy suitable for offsite recovery. The following sequence of steps illustrates why without the log suspend feature, we could not recommend use of SnapShot or other copy techniques while DB2 is running.

1. The logs and BSDS data sets are copied. Remember that the copy process, although fast, is asynchronous with respect to DB2 activity which continues as normal.
2. DB2 updates continue. Log records are externalized and log buffers written.
3. Pending writes are externalized by the buffer manager. The RBA at which these writes complete is greater than the maximum RBA on the copy of the logs obtained in step 1.
4. Updated pagesets are copied from disk.
5. Copies are taken offsite for disaster recovery purposes.

The copies of the pagesets contain RBA ranges not represented in the copy of the log. You have introduced a hole in the offsite copies of your log.

DB2 always writes the log record before making an update. Consequently by flushing the log buffers and preventing further writes to the logs, DB2 will not be able to make any updates to the data until logging is resumed.

5.1.2.1 Recommended backup procedure

We recommend that you use the following procedure:

1. Issue the `-set log suspend` command and wait for confirmation message `DSNJ372I` which indicates update activity has been suspended. This command is synchronous, so there should be no delay between command and suspension. For a data sharing group, the command should be issued for each member.
2. Invoke RVA Snapshot or ESS FlashCopy to backup the entire DB2 subsystem. This should include the following and be virtually instantaneous:
 - Active logs
 - Bootstrap data sets
 - Catalog, directory
 - All DB2 application data
 - DB2 system libraries
 - Archive logs
3. As soon as this has completed, re-instate DB2 update activity with the `-set log resume` command.
4. Copy the snapped data offsite (for example, using Peer-to-Peer Remote Copy, or a manual procedure such as dumping the copied volumes to tape, which may then be taken offsite).

We recommend that you take copies of the archive logs in your SnapShot. However, you could decide against this if you can guarantee that there will be no very long-running units of recovery inflight at the time of the SnapShot. On restart, DB2 has to back out inflight URs, and this involves reading all the UNDO records. If some are on the archive logs and DB2 cannot access them, restart will terminate. You would then be forced to conditionally restart, and this would introduce data inconsistencies. Another option is to specify LBACKOUT as mentioned in 5.1.5, “Offsite recovery considerations” on page 164.

Please note that we do not recommend using log suspension to take backups using facilities other than RVA Snapshot and ESS Flashcopy. Although you could use a backup method such as DFDSS DUMP specifying the `TOLERATE(ENQFAILURE)` option, this method is significantly slower than snapshot technology. The prolonged suspension is likely to severely impact your production service. Please see 5.1.4, “Suspend updates recommendations” on page 162 for further information on this.

Effects of commands



SET LOG SUSPEND command

- Log buffers are externalized
- A system checkpoint is taken
- BSDS is updated with highest written RBA
- A log write latch is taken which prevents updates
- Note : Pending writes from the bufferpool are NOT externalised

SET LOG RESUME command

- The write latch is released to enable logging and update activity
- Held message DSNJ372I is cleared

5.1.3 Effects of commands

We provide a description of the DB2 actions and messages when issuing the log suspension commands.

Effects of SET LOG SUSPEND

The `-set log suspend` command performs the following actions:

- All unwritten log buffers are externalized to the log.
The `-set log suspend` command flushes only the log buffers. Pending writes from the bufferpools are not externalized.
- A system checkpoint is taken.
- The BSDS is updated with highest written RBA. This guarantees that PGLOGRBA (which records the RBA or LRSN of the last page update) in all the pagesets is no higher than the highest written RBA on the log when copied.
- A log write latch is taken which prevents further log records from being written until either the `-stop db2` or `-set log resume` commands are issued. If DB2 terminates abnormally, the latch is lost and update activity is permitted on restart.

- The following message will be issued to identify the subsystem name and the RBA at which log activity has been suspended. The message is held until update activity is resumed.

```
=DB2Y SET LOG SUSPEND
*DSNJ372I =DB2Y DSNJC09A UPDATE ACTIVITY HAS BEEN 030
SUSPENDED FOR DB2Y AT RBA 000007BC6D18
DSN9022I =DB2Y DSNJC001 '-SET LOG' NORMAL COMPLETION
```

Effect of SET LOG RESUME

The following actions are performed when the `-set log resume` command is issued:

- The write latch is released to enable logging and update activity
- Held message DSNJ372I is cleared
- The following output is issued:

```
=DB2Y SET LOG RESUME
DSN9022I =DB2Y DSNJC001 '-SET LOG' NORMAL COMPLETION
DSNJ373I =DB2Y DSNJC09A UPDATE ACTIVITY HAS BEEN RESUMED FOR DB2Y
```

Suspend updates recommendations



Suspend updates for the minimum time

- Locks and claims are retained while all updaters are frozen
- Increased chance of timeouts, deadlocks and abends
- May see IRLM and DB2 diagnostic dumps
- Ideally suspend for less than lock time-out interval
- Read-only work may also be impacted

Avoid using during heavy update activity

- It will take longer for you to get access to all your data
 - ▶ pending writes are not externalized
 - ▶ restart is equivalent to crash recovery
 - ▶ in-flight URs must be rolled back
- RVA SnapShot will require additional capacity
 - ▶ while copy and target identical, no space required
 - ▶ create offsite copy before significant updates

Use extra care with 32 KB pages

5.1.4 Suspend updates recommendations

We recommend the following guidelines are observed when using the `-set log suspend` to take a snapshot of the data for offsite recovery.

5.1.4.1 Suspend updates for the minimum time

While the log suspension is in effect, all applications which perform updates, all DB2 utilities and many commands will freeze. This includes utilities running with LOG NO which will halt because they update the DSNDB01.SYSUTILX table space.

All locks and claims held by hanging updating threads will continue to be held. If the period of suspension is greater than the lock timeout interval, you will see timeouts and deadlocks. The longer you suspend update activity and the more work inflight, the greater the likelihood and number of timeouts and deadlocks.

In addition, if there is a prolonged suspension, you may see DB2 and IRLM diagnostic dumps. This is more likely in a data sharing environment, where non-suspended members cannot get a response from a suspended member.

In general, read-only processing, both static and dynamic, will continue. However, there are some circumstances which mean that a system update is required to satisfy a read request. One possible cause is during lock avoidance, when the possibly uncommitted (PUNC) bit is set, but the page (or row) lock is successfully acquired. DB2 would then attempt to reset the PUNC bit. Another example is auto-rebinds, which cause updates to the catalog. Please bear in mind that, although updates during read only processing are rare, when they do occur, the suspension may cause other locks to be held longer than normal, causing contention within the system.

After issuing the `-set log resume` command, you will have to restart all abnormally terminated jobs and redo failed transactions. Since this impacts availability and requires intervention, we recommend suspending updates at a quiet time for the minimum period.

5.1.4.2 Do not use during heavy update activity

In addition to affecting more users and therefore increasing the likelihood of timeouts and contention, there are two adverse consequences of taking a snapshot at a time of heavy update activity:

It will take you longer to get access to all your data

Since pending writes from the bufferpools are not externalized, the table and index spaces on the offsite copy are not guaranteed to be in a consistent state. DB2 restart processing at the recovery site resolves all inconsistencies during the normal phases of forward and backward recovery. You can envisage the restart process following the restore at the remote site as being precisely equivalent to the restart processing at the local site after a system crash.

RVA SnapShot will require additional capacity

An RVA SnapShot copy takes no physical space within the RVA provided there is no difference between the original and the copy of the data. The more updates made before the copy can be backed up to tape or sent offsite with Peer-to-Peer Remote Copy, the more spare physical storage is needed onsite.

5.1.4.3 Extra care with 32 KB pages

There is a small exposure when dealing with 32 KB pages if the write is suspended when writing the extents of a 32 KB page, or when crossing the volume boundaries. If the snapshot happens between the two I/Os the page will be inconsistent. Issuing a `SET LOG LOGLOAD(0)` to force a checkpoint, say 10 minutes before suspending the updates, will further reduce the exposure because it will externalize all updates. However, it would be safer to run a `DSN1COPY CHECK` to exclude inconsistencies in the volumes backup, or recover the page from a DB2 data set based backup.

Off-site recovery considerations



Backout processing considerations

- LBACKOUT and BACKODUR parameters

Off-site copy is log-consistent (only)

- DB2 restart equivalent to crash recovery
- Application data is not necessarily at a logical point of consistency
- Need procedures to
 - ▶ resume from point of failure
 - ▶ recover to application point of consistency
- Consider this when choosing your moment to copy

Reestablish recoverability off-site

- EITHER include image copies in your SnapShot
- OR take image copies after restart
- Watch for PCLOSEN & PCLOSET if using RECOVER with LOGONLY

5.1.5 Offsite recovery considerations

There are a number of things to consider when you use your snapshot copy to recover the system at the remote site.

5.1.5.1 DB2 start-up considerations

If there were long units of recovery inflight at the point you suspended activity and took the snapshot, they must be rolled back at DB2 restart. You can reduce the time taken for DB2 to restart by setting the DSNZPARM LBACKOUT parameter (limit backout processing) to YES or AUTO. We recommend that you choose AUTO. If URs are not backed out within the specified backout duration (DSNZPARM parameter BACKODUR) page sets will be placed in the restart pending (RESTOP) status. If any pageset remains in restart pending you then need to manually complete backout processing after DB2 restart with the `-recover postponed` command to enable read/write access to that data.

5.1.5.2 Offsite copy is log consistent (only)

Following restore of your offsite copies at the disaster recovery site, DB2 will perform normal restart processing. After completion of forward and backward recovery, the data will be consistent in that there are no outstanding units of recovery.

It could be that batch processing was occurring at the time of the snapshot and DB2 restart rolls the data back to the last commit point. If the batch program commits regularly, the data will reflect the position midway through the batch job or suite. To take your applications to a point of logical consistency, you may need to:

- Recover to a previous point in time at which you know the application data to be consistent.
- Design your applications and JCL to support restart processing. Ensure that you back up any non-DB2 data required to restart incomplete jobs (sequential data sets, VSAM files, and so on).
- Ensure that there is no work inflight at the point you suspend update activity that you cannot resolve at the disaster recovery site after DB2 restart.

Online transactional update activity typically presents less of a challenge to resolve than batch processing because it will be rolled back during backward log recovery. The updates made by these transactions are “lost” and need to be re-done. Although this requires re-work, obtaining a point of application consistency is more straightforward. Although this is also true of batch applications that do not commit, we recommend strongly against this approach which can compromise the operability and availability of your production system.

5.1.6 Re-establish recoverability offsite

After DB2 has restarted at the disaster recovery site, you need to protect yourself against media or application failure. There are three ways you can achieve this:

- Take full image copies of the DB2 catalog, directory, and application data when DB2 has been restarted.
- When you take the snapshot of your DB2 system, include snapshot copies of your image copy data sets.
- Take no extra backups and if a recovery is required, re-restore the table spaces to the snapshot copy and use the RECOVER utility with the LOGONLY option.

We recommend the first option so that your production image copies do not need to be on disk during the Snapshot and to minimize the volume of data to be sent offsite. However between the time of the start-up and image copy completion the data is unrecoverable.

If you intend to rely on recovery with LOGONLY we recommend that you review the DSNZPARM parameters PCLOSEN and PCLOSET and consider how they may influence offsite recovery time. Values of these parameters affect when the header page HPGRBRBA (recover base RBA or LRSN) field is updated which is the starting point for LOGONLY recovery. Since pending writes in the bufferpools are not externalized by `-set log suspend`, when the copies of pagesets are taken HPGRBRBA could be very old. This may increase substantially the time taken to recover, as well as increase the likelihood of requiring archive logs.

Operational considerations



Scope of the command is member, not group

- Recommend automation

Suspend will fail if `-archive log mode(quiesce)` or `-stop DB2 in progress`

Suspend not reported in BSDS or DB2 log

Understand how to diagnose effects

- frozen CPU and I/O counts
- no response to ATTN from TSO users
- highlighted message DSNJ372I
- STOP DB2 needs to log

5.1.7 Operational considerations

You should be aware that `-set log suspend` will fail if an `-archive log mode(quiesce)` or `-stop DB2` is in progress.

The `-set log suspend` command is **not** logged or recorded in the BSDS. The console and messages in the MSTR address space indicate that update activity was suspended.

5.1.7.1 Data sharing considerations

The scope of the `-set log suspend` command is the member of the data sharing group to whom the command is addressed.

To snapshot-copy an entire data sharing group, you need to:

- Issue `-set log suspend` on each member of the data sharing group
- Snapshot copy all members logs, BSDS and system data sets as well as the data.
- Resume update activity on all members

We recommend that you use automation to achieve this.

5.1.7.2 Understand how to diagnose problems

You need to be aware of the impact of the `-set log suspend` command on executing jobs, transactions, commands and utilities. You may be called upon to diagnose an apparent system hang caused by a prolonged suspension of log updates. The following are all symptoms:

- DB2 utilities and batch jobs freeze. CPU and I/O counters stop increasing, but the jobs will not appear to be swapped out.
- Highlighted message DSNJ372I is displayed on the console, indicating that logging is suspended. You should check the console if you suspect a problem due to a suspension of logging activity.
- If you try to cancel a thread that has performed updates, it will need to roll back. However, this cannot occur while log activity is suspended, so the job will appear to be swapped out until either you issue the `-set log resume` command or `-stop db2`. After a `-stop db2` you will receive DSNY002I by way of confirmation that the subsystem is stopping, then immediately afterwards, you will receive DSNJ373I, indicating that update activity is resumed. Resumption of logging activity must occur, as log updates are required to achieve the shutdown checkpoint. Only then will the cancel command take effect.
- Jobs may abend with S322 because the TIME parameter on the job card or for their execution class is exceeded. You should also anticipate timeouts, deadlocks, and utility abends, as shown below:

```

QUIESCE TABLESPACE PAOLR8.CHCKPAGE
DSNU000I  DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = PAOLR8P.S0080
DSNU099I  =DB2Y DSNUGASU - IRLM LOCK REQUEST FAILED,
          IRLM RETURN CODE = X'00000008'
          IRLM REASON CODE = X'2000'

```

- TSO users performing DB2 updates see the system clock waiting for a response from DB2. ATTN (which initiates abort processing) will not give control back to the user. Attempts to cancel the TSO user with the `/c u=<userid>` command will fail; the cancel cannot take effect until logging is resumed.
- If you use RMF Monitor III to diagnose delays, you will see that the DB2 subsystem started tasks and frozen jobs are delayed as shown below:

Name	C	Class	WFL %	USG %	DLY %	IDL %	UKN %	PROC	DEV	STOR	SUBS	OPER	ENQ	Primary Reason
DB2YDBM1	S	SYSSTC		0	0	0	100	0	0	0	0	0	0	
DB2YMSTR	S	SYSSTC		0	0	0	100	0	0	0	0	0	0	
DB2YIRLM	S	SYSSTC		0	0	0	100	0	0	0	0	0	0	
DB2YDIST	S	SYSSTC		0	0	0	100	0	0	0	0	0	0	
DB2YSPAS	S	SYSSTC		0	0	0	100	0	0	0	0	0	0	

Sample image copy job...

Job: PAOLR8I Primary delay: Job is in an unknown state.

Job: DB2YMSTR Primary delay: Job is in an unknown state.

- If you use an online monitor such as DB2 PM to diagnose performance and system problems, you need to remember to activate required traces before you suspend log activity, because the `-start trace` command will be suspended. Threads will appear to be in-DB2. The output below shows thread activity after a `-set log suspend` has frozen their update activity at two times approximately 25 seconds apart. Accounting time for classes 1 and 2 (also class 3 — not shown) continues to accumulate, as you can see below:

Program	Connection	----- Elapsed -----				
Primauth	Planname	name	ID	Status	Class 1	Class 2
PAOLOR8	DSNUTIL	N/P	UTILITY	LOCK	2:04.24755	2:04.18673
PAOLOR8	DSNUTIL	N/P	UTILITY	LOCK	1:35.30055	1:35.25117
PAOLOR8		N/P	BATCH	DB2	2:01.96169	2:01.96157
25 seconds later...						
Program	Connection	----- Elapsed -----				
Primauth	Planname	name	ID	Status	Class 1	Class 2
PAOLOR8	DSNUTIL	N/P	UTILITY	LOCK	2:30.20922	2:30.14840
PAOLOR8	DSNUTIL	N/P	UTILITY	LOCK	2:01.26222	2:01.21284
PAOLOR8		N/P	BATCH	DB2	2:27.92336	2:27.92324

- Usually those DB2 commands which do not update will function normally but this will depend on circumstances. This includes -display thread, -display database, -display util and so on. Output from the -display thread(*) command is shown below and indicates that the threads are active in DB2:

```
-DIS THD(*)
DSNV401I =DB2Y DISPLAY THREAD REPORT FOLLOWS -
DSNV402I =DB2Y ACTIVE THREADS -
NAME      ST A  REQ ID          AUTHID  PLAN      ASID TOKEN
SERVER    RA *   206 spbzeta2.exe PAOLOR7  DISTSERV 003F   80
V437-WORKSTATION=N00E97F8, USERID=paolor7,
APPLICATION NAME=spbzeta2.exe
V445-G90196D5.0704.01C427170305=80 ACCESSING DATA FOR 9.1.150.213
BATCH     T *    3 PAOLOR8T     PAOLOR8           0033   106
BATCH     T *    3 PAOLOR8C     PAOLOR8           0030   102
UTILITY   T *    4 PAOLOR8I     PAOLOR8  DSNUTIL  0031   104
UTILITY   T *    4 PAOLOR8Q     PAOLOR8  DSNUTIL  0032   105
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I =DB2Y DSNVDT '-DIS THD' NORMAL COMPLETION
```

The -dis log command should function normally, and indicates whether logging has been suspended (see sample output below). Commands such as -start database, -stop database, -cancel thread, -term util cannot be processed. Commands, including for example -dis util(*), may hang where there are locking conflicts on the directory:

```
=DB2Y DIS LOG
DSNJ370I =DB2Y DSNJC00A LOG DISPLAY 055
CURRENT COPY1 LOG = DB2V610Y.LOGCOPY1.DS03 IS 4% FULL
CURRENT COPY2 LOG = DB2V610Y.LOGCOPY2.DS03 IS 4% FULL
H/W RBA = 000007BC6D18, LOGLOAD = 50000
FULL LOGS TO OFFLOAD = 0 OF 6, OFFLOAD TASK IS (AVAILABLE)
DSNJ371I =DB2Y DB2 RESTARTED 13:56:51 MAR 16, 2000 056
RESTART RBA 0000055FD000
DSNJ372I =DB2Y DSNJC00A UPDATE ACTIVITY HAS BEEN SUSPENDED FOR DB2Y
057
AT RBA 000007BC6D18
DSN9022I =DB2Y DSNJC001 '-DIS LOG' NORMAL COMPLETION
```

APAR identifier

The APAR identifier is as follows:

PQ31492.

Defer defining data sets



Ability to defer definition of underlying VSAM data sets

- For table spaces and index spaces (DEFINE NO)
- From CREATE TABLESPACE or INDEX to INSERT

Good for large applications which create empty tables

Simpler space management

Saves DASD

Faster install of ERP products

5.2 Defer defining data sets

VSAM data sets underly table spaces and indexes. For STOGROUP implicitly defined table and index spaces, the VSAM define is performed automatically on behalf of DB2 when the DB2 object is created. It takes approximately 0.5 - 0.6 seconds for each define to be processed. If your application has a large number of objects, the overhead of VSAM define processing can be significant. In addition, if you use only a subset of the functions of a packaged application, you may find you have to create large numbers of objects that you do not use.

The enhancement introduced with this refresh allows you to defer the physical definition of the underlying VSAM data set until first use.

5.2.1 Effect of deferring DEFINE of VSAM data sets

The CREATE TABLESPACE and CREATE INDEX statement syntax has been changed. The option DEFINE YES is the default and works as today. DEFINE NO has the following effects.

- The VSAM data set are not created. There are no Integrated Catalog Facility (ICF) entries made. The data sets are not defined in the VVDS or VTOC.
- DB2 catalog entries are added for the new objects. The catalog tables, SYSIBM.SYSTABLEPART, SYSIBM.SYSTABLESPACE and so on are populated to record the definition of the objects. The actual status of the object, undefined if you specify DEFINE NO, is recorded as a value of -1 in the SPACE column of SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART.

- You can find out all the objects for which define is deferred by searching for the value -1 in the SPACE column of SYSIBM.SYSTABLEPART for table spaces or SYSIBM.SYSINDEXPART for indexes.
- You can issue ALTER TABLESPACE (or INDEX) after creation and before the VSAM data set is defined, in which case the definitions as specified in the catalog are updated by the ALTER as normal. This does not apply to the DEFINE attribute that cannot be changed with ALTER.
- Applications which attempt to access data in undefined objects receive return codes as for an empty table, no special error is returned. A qualified SELECT will receive SQLCODE 100, a SELECT COUNT(*) will return SQLCODE 0 and a value of 0.
- The STOSPACE utility tolerates undefined objects. The value of SPACE -1 results in the object being skipped by STOSPACE.
- At the first write (insert or LOAD specifying either RESUME or REPLACE), DB2 resets the undefined status in the catalog by updating the SPACE column and creates the underlying VSAM data sets.

Impact on DDL performance

DDL operation	DEFINE NO Elapsed Time	DEFINE NO VSAM time	DEFINE YES Elapsed time	DEFINE YES VSAM time
Create table space	0.60	0.01	53.74	51.8
Create table	0.55		6.06	
Create Index	0.61	0.01	60.12	57.8
Total elapsed (sec)	1.76		119.92	

- All time measurements are in seconds
- Shows time taken to create 100 table spaces, tables and indexes (Elapsed)
- For create table space and create index separate figure is shown for the VSAM component (VSAM time)
- VSAM element for DEFINE YES is large proportion of the total elapsed time
- Improvement in create - 68 times faster elapsed

NOTE: elapsed time for first insert increased

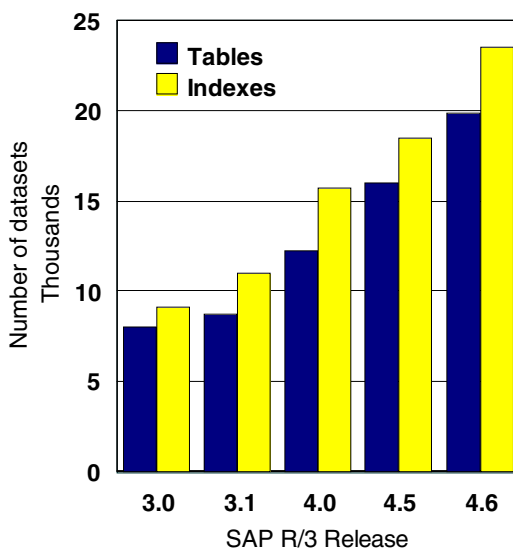
5.2.2 Impact on DDL performance

To assess the performance improvements you may expect with this enhancement, we measured the elapsed time to create 100 table spaces, tables and indexes comparing DEFINE YES and DEFINE NO. The time spent in VSAM is shown separately.

This analysis shows that:

- VSAM activity represents a significant part of the cost of the create table space and create index statements.
- DEFINE NO improves elapsed time by 68 times.
- The create table statement is significantly faster with DEFINE NO because there is no VSAM open and DB2 does not need to access the header page. Although the elapsed time improvement is small for our test of only 100 tables, when you need to create many thousands, the performance improvements might be significant. Reducing the elapsed time may also help improve concurrency, as the duration of locks will be reduced.

Where DEFINE NO helps



- typically half of pagesets empty
- DEFINE NO
 - + much faster installation
 - + simpler DBA operations
 - + better DASD utilization
 - utility considerations
 - housekeeping issues

5.2.3 Where define no helps

The most apparent benefit from this enhancement is the performance improvement to DDL processing, particularly if you are creating large numbers of objects. The diagram above shows the number of pagesets required for SAP R/3 for various releases. There are some additional benefits introduced by this enhancement that you may be able to exploit.

- Some generalized vendor products, including SAP R/3 and PeopleSoft, can create many more DB2 objects than you intend to use. This is most apparent when you are using a small subset of the features or modules the product provides. If you specify DEFINE NO:
 - Installation of the product is faster.
 - You will realize a saving on the amount of disk required, because only those data sets you use are physically defined.
- The size of your ICF catalog is reduced, because there are entries only for those data sets which contain data rather than every DB2 object in the DB2 catalog.
- Since the DEFINE NO does not create or open the VSAM data set until data is written, you will see some relief for DBM1 storage in circumstances where you are creating many thousands of data sets and you are close to hitting the maximum number of open data sets.
- Since the data sets are not created until first use, you can create all the DB2 objects you will need for an application ahead of your acquisition of the disk storage necessary to physically support the data. This enables you to carry out the tasks required to build your applications in parallel.

5.2.4 Restrictions

- If you create an index on a table which is already populated with data, DEFINE NO is ignored and the VSAM data sets are created, opened and DB2 populates the index.
- You may not use the DEFINE NO option on the work database. The create will fail with SQLCODE -620.
- The DEFINE NO is valid only for STOGROUP defined objects. If you specify both VCAT and DEFINE NO options together you will not see an error but the DEFINE NO will be ignored.
- DEFINE NO is not supported for LOBs or indexes on auxiliary tables.
- DEFINE NO is not supported for temporary databases.
- DB2 online utilities that access the underlying VSAM data set will tolerate the absence of the VSAM data set. The service aids such as DSN1PRNT and DSN1COPY that access the data set cannot process undefined objects. They will fail with an allocation error identifying the missing data set name.
- For partitioned table spaces the DEFINE option applies to the entire table space. You cannot specify a mixture of DEFINE YES and DEFINE NO for different partitions.
- When you perform an insert or LOAD into a single partition of a partitioned table space, VSAM data sets for all underlying partitions are created, not just the one affected by the write operation.

Things to watch out for



Impact and possible problems with create are deferred to first write time (insert or load)

- installation problem may become production failure

Storage administrators may interpret free disk as unused

- be aware of latent demand
- sudden unpredictable demand for disk
- they may deploy it elsewhere

First insert pays price of define and open

Downlevel DB2 or member cannot access undefined object

Automated housekeeping might need adjustments

ISV utilities

- must be aware of SPACE
- must not get DB2 and VSAM out of synchronization

5.2.5 Things to watch out for

You need to be aware that the use of DEFINE NO might cause some operational issues. They are caused by the change in the way your installation behaves when exploiting the feature:

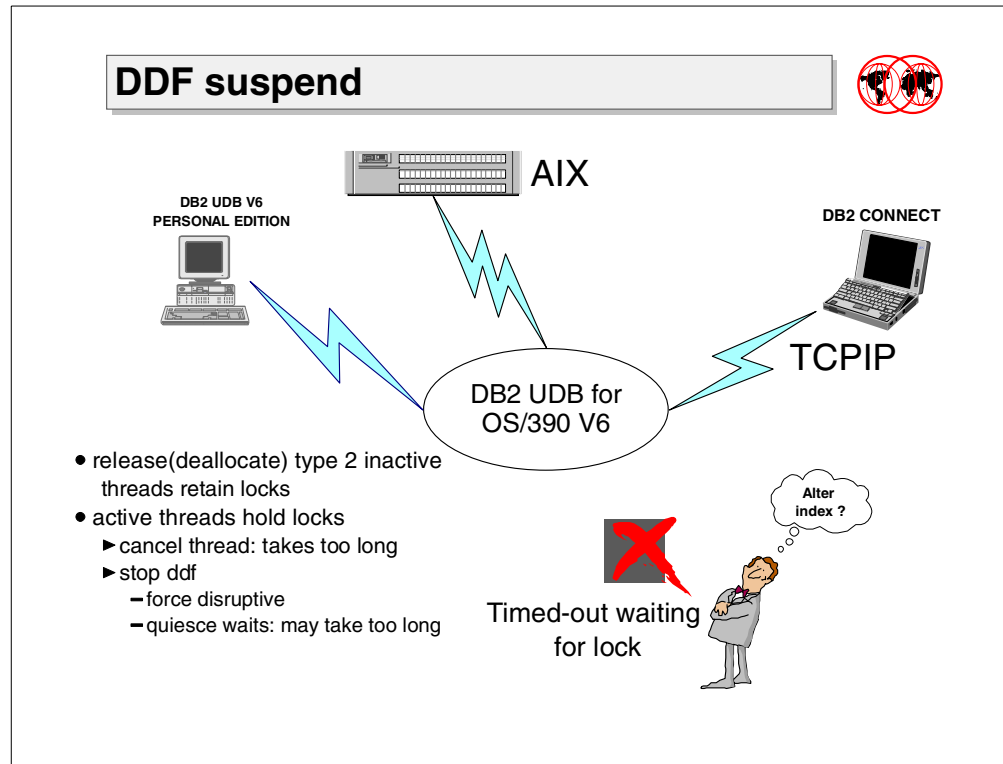
- Since you are deferring the physical definition of the data set until write, you will find that some errors you currently associate with create are now deferred until the first insert or LOAD.
- A consequence of deferring an error to first use means that problems you would have encountered at the time you were creating your database (ahead of production implementation) could now become run-time production failures. You will not see an error you would have not had before but it might be more urgent.
- We recommend that you inform your storage administrators that you are using this feature and ensure they are aware of any latent demand for disk from undefined DB2 objects. It is possible to imagine a scenario where, by post-installation of an application, a capacity planner observes the actual space requirement to be considerably less than expected. If capacity was required elsewhere, you might find that storage managers have moved what appeared to be free capacity to another SMS storage group. When the DB2 application using the undefined objects comes to use them, you might see:
 - Run-time failures.
 - Sudden and dramatic changes in disk utilization that are apparently unrelated to the implementation of an application.

- The first time your application performs inserts, especially if to a large number of undefined physical objects, you could see an initial but transitory performance problem. The work formerly done by DB2 at create is deferred until write. The application, rather than the installation process, must pay for at least the cost of physical define (and open).
- If any of the members of a datasharing group are downlevel or if you fallback to a prior release, an attempt to access an undefined object results in SQLCODE -904.
- If you have a highly automated housekeeping environment in which utility execution is determined dynamically based on catalog queries, you might need to interrogate the SPACE column and skip undefined objects. Most online utilities, however, will tolerate table spaces and indexes without underlying data sets and will not generate non-zero return codes. The restrictions apply to offline utilities and COPY with a list of objects.
- In some situations, the DEFINE NO option is not applicable. You need to be aware of these so that you have sufficient disk. APAR PQ34029 introduces the SQLCODE warning +20122 DEFINE NO OPTION IS NOT APPLICABLE IN THE CONTEXT to indicate that your request of DEFINE NO has been ignored and the physical define has occurred. DEFINE NO is ignored in the following circumstances
 - CREATE INDEX statement that included the VCAT clause
 - A CREATE TABLESPACE statement that included the VCAT clause
 - CREATE INDEX statement for a non-empty table
 - CREATE INDEX statement for an auxiliary table
 - CREATE LOB TABLESPACE statement
- If you use independent software vendors (ISV) products that directly access VSAM data sets, there are two important considerations:
 - Utilities may fail with missing VSAM data set errors if the product vendor does not check for undefined objects.
 - If an ISV utility such as load creates and loads data directly into an underlying VSAM data set, there will be a mismatch between what DB2 expects (no VSAM data set) and reality. If DB2 detects a VSAM data set that should not be there according to its catalog definition, reason code 00D70043 is returned to the calling application.

APAR identifier

The APAR identifier is as follows:

PQ30999, PQ34029, PQ34386, PQ34592, PQ34030.



5.3 DDF suspend

When performing data definition language (DDL) statements such as create, alter and drop, exclusive locks on catalog and directory objects must be acquired. Applications accessing data from tables in the same database as those against which you want to perform DDL may hold incompatible locks and prevent the execution of DDL.

5.3.1 Applications may retain incompatible locks

If you only have a small window to perform some database maintenance, you would need to cancel threads holding incompatible locks in some way. If you operate in a distributed environment, these problems can occur:

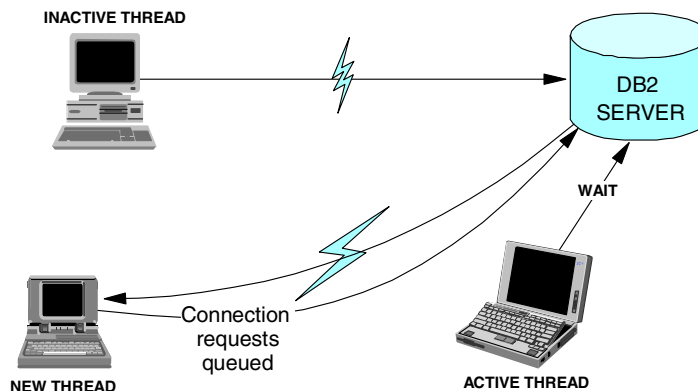
- Because DB2 is capable of supporting a very large number of remote connections (up to 150,000), you will probably find `-cancel thread` is a poor option. It could take longer than your maintenance slot to identify all the connections blocking your DDL operation, and it is possible for new threads to connect and continue to thwart progress.

- It is not unusual for client applications to be connected but inactive. It is possible that locks are retained while threads are connected. This is particularly true for packages bound with RELEASE(DEALLOCATE) and where thread pooling for inactive DDF connections (type 2 inactive threads) is enabled. You determine whether DDF threads can become inactive in the DDF THREADS field of the installation panel DSNTIPR. If you choose the recommended value INACTIVE, remote threads can become 'inactive' after successful commit or rollback (provided no cursors are held). The associated pool thread can then be reused by other database access threads. However, in the mean time, the pool thread still holds allocation duration locks that may prevent DDL operations.
- Although a `-stop ddf` command will successfully terminate inactive threads, DDF will not stop until all the active remote connections have terminated. This is one reason why we recommend that you choose INACTIVE for DDF THREADS. Stopping DDF can be quite disruptive — all threads, even those accessing databases you are not trying to apply maintenance to, are affected. Since you must wait until active DDF threads have disconnected, the outage can be considerable.
- The command `-stop ddf mode(force)` is highly undesirable, because you may be forced to recycle the requesting DDF address space(s) with `-stop ddf` and `-start ddf` to re-establish the conversation. If this DDF is a system which you do not administer, or which operates on a different time zone, it could be very difficult to negotiate a mutually agreeable maintenance slot. If there are many DDF connections, re-establishing access to your DB2 system could require significant effort.

STOP DDF MODE(SUSPEND)



- **STOP DDF MODE(SUSPEND)**
 - ▶ **DSNL069I** =DB2Y DSNLSSRS DDF IS SUSPENDING
 - ▶ **DSNL066I** =DB2Y DSNLSSRS STOP DDF MODE(SUSPEND) COMPLETE



5.3.2 STOP DDF MODE(SUSPEND)

The STOP DDF command has been extended by adding a new SUSPEND mode option. DDF server processing is suspended without terminating remote connections. The effect of this suspend option is to quiesce DDF activity as follows:

- Inactive DDF threads remain inactive; they cannot become active until a subsequent `-start ddf` command is issued.
- DDF pool threads are terminated.
- Requests for new server connections are queued. Inbound DDF work (other than re-synchronization activity to resolve in-doubt threads) cannot be initiated.
- Outbound DDF threads are unaffected.

Suspension of DDF activity will clear inactive threads, so resources held by packages bound with DEALLOCATE will be released. However, since active threads remain, it is possible that your DDL operations will still fail because active threads may hold locks you need. At this point, you will typically need to consider two options:

- Abandon the maintenance slot altogether and reschedule.
- Cancel all those active threads holding incompatible locks that continue to frustrate your DDL statements. At least with DDF suspended, no new connections can be made while you remove the active threads.

Two optional parameters to the `-stop ddf mode(suspend)` command help ease the manual intervention required to effect your decision.

- CANCEL(n) option cancels all active threads when suspend processing has not completed successfully in n seconds. Cancel has a range of 0 - 9999 seconds.
- WAIT(n) resumes DDF processing again if suspend processing has not completed successfully in n seconds. Wait has a range of 0 - 9999 seconds.

It is an operational decision as to whether you manually issue `-cancel ddf thread` commands (or use VTAM commands) or use the new options of the `-stop ddf mode(suspend)` command. Using `-cancel ddf thread` requires manual intervention and can be time consuming, particularly if there are many threads frustrating your maintenance work. It has the advantage, though, that if only a few of the active threads are holding incompatible locks, the majority of inflight distributed work can continue unaffected.

If it is unacceptable to cancel active threads, you could use the WAIT keyword and if threads did not terminate by the end of the time period specified, DDF is restarted automatically. You would then have to reschedule your maintenance work.

If there are many connections to be cancelled and it is acceptable to cancel active remote threads, it may be better to disconnect all the active threads using the CANCEL option.

STOP DDF MODE(SUSPEND) WAIT (120)



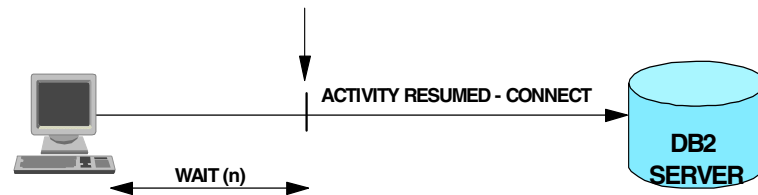
=DB2Y STOP DDF MODE(SUSPEND) WAIT(120)

DSNL069I =DB2Y DSNLSSRS DDF IS SUSPENDING

DSNL072I =DB2Y DSNLSSRS WAIT TIME EXPIRED, DDF RESUME
PROCESSING INITIATED

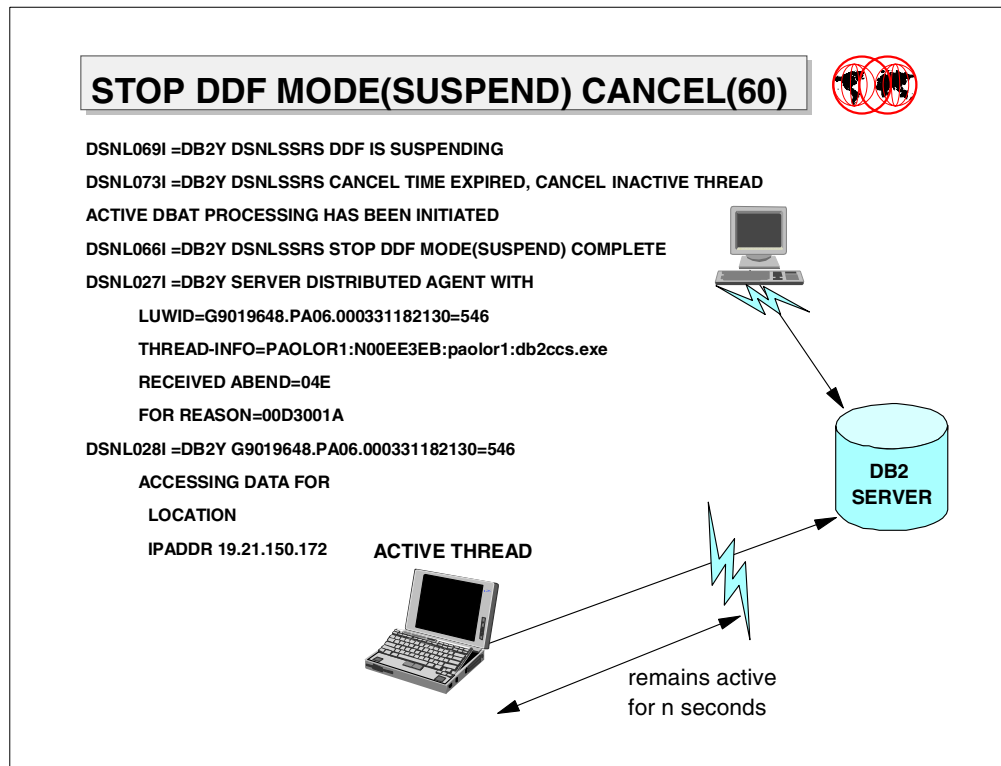
DSNL070I =DB2Y DSNLSSRS DDF IS RESUMING DSNL068I =DB2Y
DSNLSSRS START DDF (RESUME PROCESSING) COMPLETE

Note: as if you had issued START DDF



5.3.3 STOP DDF MODE(SUSPEND) WAIT(n)

The diagram shows DB2's response to use of MODE(SUSPEND) with the WAIT keyword. A distributed thread attempts to connect to DB2 after the stop command has been issued. It waits until DB2 determines the suspend time has been exceeded. Message DSNL072I indicates that this has occurred. The situation then is just as if you had issued a `-start ddf`, and the connection can now be made.



5.3.4 STOP DDF MODE(SUSPEND) CANCEL(n)

The diagram shows DB2's response to use of MODE(SUSPEND) with the CANCEL keyword. Inactive threads are disconnected immediately. Active threads remain connected for the specified time and are then cancelled. Applications receive `ABEND S04E` with reason code `00D3001A`. Subsequent SQL calls result in an error with message `SQL30081N` (communication error) returned.

Console messages identify DDF threads abnormally terminated as a result of the CANCEL option.

DDF command options



Command - DDF STATUS	START (DB2 or DDF)	STOP (DB2 or DDF)	STOP FORCE (DB2 or DDF)	STOP DDF MODE(SUSPEND)
Starting	Error	Error	Error	Error
Started	Error	OK(stop)	OK(stop)	OK(suspend)
Stopping	Error	Error	Error	Error
Stopped	OK(start)	Error	Error	Error
Suspending	OK(resume)	OK(stop)	OK(stop)	Error

Recommendations

- avoid cancelling many long running units of work
- avoid suspending activity for long periods of time

5.3.5 DDF command options

The table above shows what combinations of START DDF and STOP DDF are valid depending on the state of the DDF. You can issue the STOP DDF MODE(SUSPEND) command only when DDF is started. All other combinations result in an error.

We recommend that you avoid use of the `-stop ddf mode(suspend)` command with the cancel option if it would result in the cancellation of a large number of active threads, particularly if this will result in prolonged rollback activity. You will have to wait for rollback to complete before the locks are released and it will result in unnecessary logging activity. The work lost will then have to be re-done.

We also recommend that you do not suspend activity for too long, particularly if this results in extensive queues. This could cause problems for DDF requesters and when activity is resumed you may see a spike of abnormally high activity on your server.

APAR identifier

The APAR identifier is as follows:

PQ27123.

Faster cancel thread



Previously the effect depended on status of thread:

- no effect until inactive or suspend thread resumed in DB2
 - end users
 - persistent threads awaiting work
- locks would be retained, a -stop DB2 would wait

Thread termination will now occur even if thread is inactive in DB2

Application will find out about disconnection at next DB2 request

Applies only to local threads

Still no effect during "must complete" functions like commit, rollback

5.4 Faster cancel thread

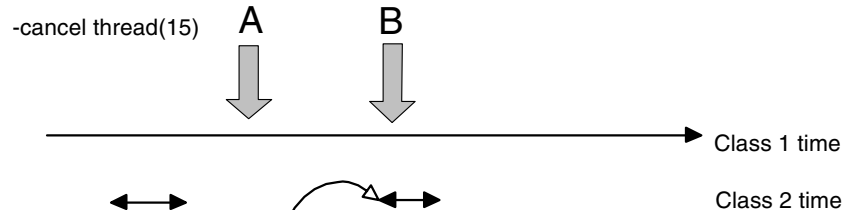
Prior to this enhancement, the point at which the cancel thread command was effective depended on the threads status. If the thread was active in DB2, it was terminated at the next awaited internal event such as a wait for a synchronous I/O or a lock.

If the thread was not in DB2, because inactive or suspended, the cancel was not effective until the next time the thread was resumed in DB2. This caused a variety of problems:

- It could take a long time for the cancel command to be honored. This is especially true for inactive or suspended threads where the user may no longer be at the workstation.
- The resources acquired by the thread were retained which could cause locking problems for other applications or utilities.
- A scheduled `-stop db2` would wait until the thread terminated. This extends a planned outage of DB2 for maintenance and reduces the DB2 down time available for support personnel.
- In extreme cases, it might even be necessary to cancel the allied task or shut down DB2 with the force option to disconnect the thread.

This enhancement improves operability because now, even if the thread is not active in DB2, it undergoes termination processing. Thread termination is now independent of application activity.

Cancel thread example - QMF user



```
=DB2Y DIS THD(*)
DSNV401I =DB2Y DISPLAY THREAD REPORT FOLLOWS -
DSNV402I =DB2Y ACTIVE THREADS -
NAME  ST  A  REQ ID    AUTHID  PLAN          ASID TOKEN
TSO   T   171 PAOLOR8  PAOLOR8  DSNESPCS    0042  15
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I =DB2Y DSNVDT '-DIS THD' NORMAL COMPLETION

=DB2Y DIS DATABASE(DSNDB06) SPACE(*) LOCKS ONLY
NAME    TYPE STATUS    CONNID  CORRID  LOCKINFO
SYSDBASE TS      RW      TSO    PAOLOR8  H-IS,S,C
SYSDBAUT TS      RW      TSO    PAOLOR8  H-IS,S,C
SYSUSER  TS      RW      TSO    PAOLOR8  H-IS,S,C
***** DISPLAY OF DATABASE DSNDB06 ENDED *****
```

5.4.1 Cancel thread example

The improvement introduced by this enhancement is illustrated in the diagram above. A thread is connected to a DB2 subsystem from a TSO session (such as QMF or SPUFI) and is running a query against the catalog. Output from `-display thread(*)` and `-display database(dsndb06) space(*) locks only` is shown. The thread's token is 15. The output indicates that the thread is connected, but is processing within the application outside DB2 (shown by the active indicator being set to blank) in the display thread report rather than an asterisk. See the *DB2 UDB for OS/390 Version 6 Messages and Codes*, SC26-9011-01 for details about how to interpret information from these commands.

The accounting time line shows the time spent in-DB2 (class 2) and the total time since first connection to DB2 and thread termination (class 1). As is often typical of TSO attached threads, the class 1 accounting time is significantly larger than class 2 time. The points A and B are different points where the cancel thread command is issued. At point A, the thread is active, but not in DB2; whereas at point B, the thread is in DB2 accumulating class 2 accounting time.

Prior to the enhancement `-cancel thread(15)` at point A results in message DSNV426I confirming cancellation but the resources remain allocated. Subsequent `-display thread(*)` commands show the thread is still connected to DB2 and all its locks retained. Only at the next DB2 request (point C) is the thread actually cancelled and resources freed. The application receives abend S04E with reason code 00E50013. If the thread is cancelled at point B, the cancel occurs at the next internal awaited event at point D. Although there is a wait, this will appear more or less instantaneous to the operator who issued the cancel command, and DB2 resources will have been freed.

5.4.2 Operational improvement

So, in the diagram above, the cancel issued at point A occurs at A and DB2 resources are freed immediately.

The application finds out about its disconnection at the next DB2 request. The error returned to the application depends on the precise timing of the cancel command relative to the point at which the thread attempts to reconnect to DB2. If the reconnect occurs at the point at which the cancel is issued you may see S04E with reason code 00E50013. Normally though, termination processing will occur before the next DB2 call and so you will see SQLCODE -924 at the next attempted SQL call to indicate a connection failure.

You can still request a dump by specifying the DUMP keyword on the cancel thread command.

5.4.3 Restrictions

The enhancements to the cancel thread command apply only to local threads. If you need to cancel a distributed thread that is active in DB2 you should issue the `-cancel ddf thread` command. If the DDF thread is inactive hanging in VTAM then you can use VTAM commands to cause VTAM to return processing to DB2 which will result in thread termination. Refer to *DB2 UDB for OS/390 Version 6 Command Reference*, SC26-9006-01.

To protect integrity, it is still true that the cancel has no effect during “must complete” functions such as commit and rollback.

APAR identifier

The APAR identifier is as follows:

PQ34465, PQ34466, PQ36702.

Data sharing enhancements



- Improved shutdown performance
CASTOUT(NO) option
- New IMMEDIATEWRITE(PH1) BIND option
- IFI and commands with group scope
data sharing group view

5.5 Data sharing enhancements

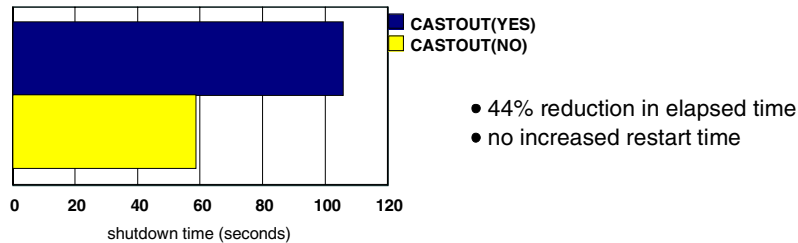
Three enhancements in this code refresh bring operational improvements in a data sharing environment. These are improved shutdown performance, a new IMMEDIATEWRITE BIND option, PH1, to help order dependent processing, and enhancements to the Instrumentation Facility Interface (IFI) which extend the scope of some commands to the group rather than member.

Faster data sharing member shutdown



- -STOP DB2 CASTOUT(NO)
 - ▶ castout of member pages from GBP to DASD not performed
 - ▶ members P-locks are retained
 - ▶ GBP connection enters 'failed-persistent' state for member
 - ▶ display group shows member in QC status
- performance improvements
- do not shut down all members using CASTOUT(NO) - group recovery required at restart

Shutdown time to stop one member after running in a 2 way data sharing system at 57 commits/second for at least 10 minutes with CASTOUT(YES) against CASTOUT(NO)



5.5.1 Faster data sharing member shutdown

The normal shutdown process of a data sharing member includes castout processing. If the member is the castout owner of updated pages in the group buffer pool (GBP), it will determine whether it can transfer castout ownership to another member. However, if it is the only member in the group updating the pageset/partition, it will write the updated pages out to disk during shutdown. Since there is no physical connection between the group buffer pool and disk, these writes are performed by the closing member's DBM1 address space. This process can be time consuming.

In most situations, DB2 shutdown is required simply to recycle DB2 to apply maintenance, pick up a changed DSNZPARM or free up any unused storage. Even with Name Class Queue support which was introduced with Coupling Facility Level 7 and is discussed in 6.5, "Data sharing improvements" on page 247, shutdown takes longer than desired.

To speed up the process, the CASTOUT(NO) option has been added to the `-stop db2` command. In a non-data sharing environment, the castout option is ignored. In a data sharing environment, the member shutting down bypasses castout and delete name processing.

The consequences are that:

- The `-display group` command will show member(s) of the group that were shut down with CASTOUT(NO) in 'QC' status - quiesced but with some castout work incomplete.

- For those member(s) shut down with CASTOUT(NO), P-locks will be retained in IX mode for those objects for which the member was the last updater. These IX locks will enable other members to update the object, but will prevent utilities from running against the object until the member is restarted.
- Since castout does not occur, group buffer pool connections from member(s) shut down with CASTOUT(NO) enter the 'failed-persistent' state.

DB2 restart converts retained P-locks to active and drives a special open to perform pseudo close for those objects for which the member was the last updater. This closes off the log range for the pageset.

Performance improvements with CASTOUT(NO)

The shutdown time for a member of a data sharing group is significantly faster. The average reduction in elapsed time was 44% (ranging from a 13-85% improvement) when CASTOUT(NO) is specified. Similar improvements are observed if multiple members of a group are shut down simultaneously with the CASTOUT(NO) option. We observed no increase in member restart time.

Consideration

If you are shutting down members to obtain a consistent copy of the databases, you should avoid using CASTOUT(NO) for all members on shutdown since the latest version of the data might still be in the GBP. Even though all members are inactive, the data may not be consistent until the latest copy of the data is retrieved from the structure during restart.

APAR identifier

The APAR identifier is as follows:

PQ29907 and PQ35845.

IMMEDWRITE(PH1) BIND option



- Added to current IMMEDIATE BIND options
 - YES, page written to GBP at buffer update
 - NO, page written asynchronously
- IMMEDIATE(PH1) new option to allow sequence of updates with minimal impact

5.5.2 New IMMEDIATE(PH1) bind option

The IMMEDIATE bind option determines when DB2 writes an updated page to the GBP. If you specify IMMEDIATE(YES), the page is written to the GBP as soon as the buffer update completes. This option is useful for order dependent transactions being processed in a data sharing environment.

This can be illustrated by the following scenario. Imagine you have a data sharing group with two members. An IMS transaction T1 makes a change to a data page on the first member of the group. A second transaction, T2 bound with anything other than repeatable read (RR) and therefore capable of exploiting lock avoidance, is spawned before phase 2 commit processing. Both T1 and T2 are bound with IMMEDIATE(NO). Assume T2 depends on the data base changes made by T1. Since phase 2 commit has not completed it is possible that cross-invalidation and the update to the group buffer pool will not have been completed. Therefore, T2 will not see T1's update. This only happens if T2 runs on a different member from T1.

Today, ways of ensuring T1's update is visible to T2 include

- Execute the two transactions on the same member
- Bind with IMMEDIATE(YES) which has an effect on performance
- Bind with RR which impacts concurrency

The new IMMEDIATE(PH1) option writes GBP dependent pages at phase one commit so allows T2 to see T1's updates. There is almost no overhead with this option compared to IMMEDIATE(NO) unless transactions abort after commit of phase 1. This is because each updated GBP-dependent page would be written twice — once during the phase 1 commit and again at the end of the abort.

There is a new DSNZPARM parameter IMMEDIATEWRITE in macro DSN6GRP which takes values of NO, PH1 or YES.

Here are the new IMMEDIATEWRITE bind options for plans/packages:

IMMEDIATEWRITE	Explanation
NO	Normal write activity is done. Group buffer pool (GBP) dependent buffers are written to the coupling facility at or before the end of the commit or rollback.
YES	IMMEDIATEWRITE writes are done for the buffers that are updated that contain pages that belong to GBP dependent page sets or partitions. This may have some performance impact because a page may be written out to the CF multiple times, once for each row update. Updated pages are also immediately written for buffer updates of a rollback.
PH1	The updated GBP-dependent pages are written to the coupling facility at or before phase 1 of commit.

Here are the new IMMEDIATEWRITE system parameter bind option:

IMMEDIATEWRITE	Explanation
NO	Writes of GBP-dependent pages are done according to the IMMEDIATEWRITE option that is specified a plan or package level.
PH1	All plans or packages running on this DB2 member use IMMEDIATEWRITE(PH1), except those that are bound with IMMEDIATEWRITE(YES).
YES	All plans and packages that are run on this member use IMMEDIATEWRITE(YES) to write updated GBP-dependent pages.

APAR identifier

The APAR identifier is as follows:

PQ25337.

IFI and commands with group scope



Data sharing instrumentation enhancements

- IFI data has group scope
- single interface to gather instrumentation from all members
- DB2PM will support this

Data sharing - some commands have group scope

- scope(local) is the default - works as now
- trace, procedure, function and display commands
- operator can enter single command that affects all members

5.5.3 IFI and commands with group scope

Operational enhancements to DB2 data sharing introduce the extension of group scope to:

- Instrumentation Facility Interface (IFI)
- Selected DB2 commands

5.5.3.1 Data sharing IFI consolidation

The IFI enhancement allows you to collect group-wide instrumentation data from a single member of a data sharing group. This means that rather than running monitoring programs (for example, DB2 PM) against every member of a data sharing group and manually collating the data to get a group perspective, group-wide reporting will be possible by running DB2 PM against a single member. DB2 PM will provide the corresponding enhancement through maintenance.

5.5.3.2 Data sharing commands with group scope

The following DB2 commands now take an optional scope parameter. The valid values are `scope(group)` and `scope(local)`. The default is `local`.

- Start trace scope(group)
- Display trace scope(group)
- Stop trace scope(group)
- Display thread scope(group)
- Start procedure scope(group)
- Display procedure scope(group)
- Stop procedure scope(group)
- Start function specific scope(group)
- Display function specific scope(group)
- Stop function specific scope(group)

An operator can enter a single command from one DB2 terminal and it will take effect on all members of a data sharing group.

APAR identifier

The APAR identifier is as follows:

PQ29031 and PQ25094.

New EDM pool parameter



EDMBFIT - New DSNZPARM

- helps if virtual storage constrained
- affects EDM pools > 40 Mb
- adjusts free chain search algorithm
- provides relief when loading large objects

EDMBFIT= YES

- uses better fit algorithm
- better storage management
- increase in latches

EDMBFIT= NO (default)

- uses a first fit algorithm
- better EDM pool latch management

5.6 New EDM pool parameter

A new system parameter, EDMBFIT, has been introduced to assist customers who are constrained on virtual storage. In a busy EDM pool, resource failures may occur as large objects cannot be loaded into the pool. Some customers are unable to increase the size of the pool as the DB2 address space is reaching its 2 GB limit.

The EDMBFIT parameter gives you the option of a new free chain search algorithm for EDM pools greater than 40 MB.

When EDMBFIT is set to NO, DB2 will search for free space using a first fit algorithm as before. It places objects in the first available free space where they will fit.

When YES is specified, DB2 will use a better fit algorithm. To make optimum use of the storage, it will search the EDM pool free space chain looking for the best place for the object. This improved storage management will provide virtual storage relief when large objects need to be loaded. However, it may cause an increase in latch suspension times (class 24).

Unless you have virtual storage constraints, we recommend using EDMBFIT=NO, as it avoids long free-chain searches which can cause latch contention.

APAR identifier

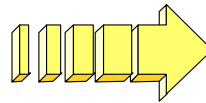
The APAR identifier is as follows:

PQ31969.

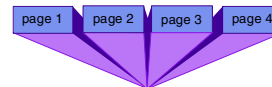
New CHECKPAGE option for COPY



```
COPY
TABLESPACE PAOLOR8.CHECKPAGE
FULL YES
COPYDDN (COPYL1)
INDEXSPACE PAOLOR8.CHECK1V$I
COPYDDN (IXCPL1)
CHECKPAGE
PARALLEL(2)
SHRLEVEL REFERENCE
```



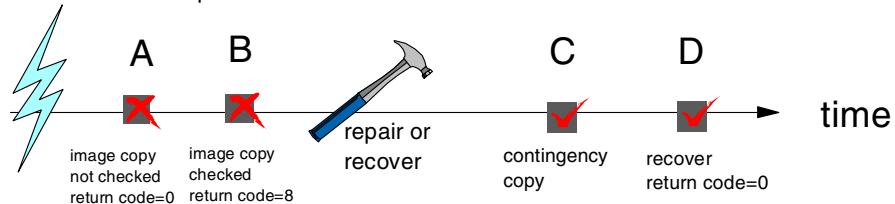
Copy checks every index and table space data and space map page



- Table space and index space pages
- support for 4 KB, 8 KB, 16 KB, 32 KB page sizes
- no impact on elapsed time, negligible increase in CPU time

Hardware error

- undetected corruption



5.7 New CHECKPAGE option for COPY

The COPY utility already performs integrity checking on DB2 pages as the image copy is taken. If you wish to perform additional validity checking, you run the DSN1COPY utility using the CHECK option, either against the underlying DB2 VSAM datasets or the image copies.

The image copy utility has been enhanced to enable you to perform extra validity checking when image copying table spaces and indexes, rather than having to invoke DSN1COPY in a separate step.

5.7.1 How to activate page checking

To activate page checking you invoke the COPY utility with the CHECKPAGE option. The CHECKPAGE option is not valid if the CONCURRENT keyword is specified. This is because page checking is performed by DB2 and concurrent copy calls upon DFDSS to perform physical copy of the data sets. The default is that additional page checking is not performed.

As each index or data page is processed, it is validated. If multiple objects are specified in the COPY statement (including a mix of index and table spaces), each object is checked. The scope of the CHECKPAGE applies to all objects in the list. All table space page sizes are supported.

5.7.2 Exploitation of CHECKPAGE

Although the risk is low when taking image copies containing some sort of corruption that prevents recovery is low, the impact of finding out later that you cannot recover is high. To avoid this scenario, some users run DSN1COPY with the CHECK option on some or all of their image copies. The new CHECKPAGE option of COPY means you can now do this in one step. Any error reported gives you the opportunity to immediately resolve the underlying problem.

The time line in the diagram above shows how you can use CHECKPAGE to validate your copies. Suppose a defect is introduced into a page by some hardware failure which is undetected by applications and routine image copies. Although undetected hardware errors and logical corruptions are rare, it is possible to create a defective copy (for example, at point A). If this went undetected for longer than your backup cycle, you could end up in the position where none of your image copies were valid for recovery.

A periodic COPY with the CHECKPAGE option reports the error (at point B) and is identified by the abnormal completion code. This gives you an opportunity to rectify the error either by using REPAIR or by falling back to a previous good image copy. After recovery or repair, you can then take a good contingency copy (C). This allows a subsequent recover (D) to complete normally.

The validation performed when you use the CHECKPAGE option is equivalent to that performed by DSN1COPY with CHECK.

Consider how frequently within your backup cycle you wish to use the CHECKPAGE option, as there is a trade-off between a potentially increased recovery time against extending the elapsed times of your image copy jobs.

5.7.3 How to detect and resolve errors

In the case where invalid pages are detected:

- The COPY utility identifies the page(s) as broken.
- If the page is a data page message DSNU518I is issued.
- If the page is a space map page message DSNU441I is issued.
- COPY completes with return code 8 and an error code is issued as shown below:

```
DSNU518I =DB2Y DSNUBASA - TABLESPACE PAOLOR8.CKCPAGE DSNUM 1 CONTAINS  
BROKEN PAGE X'00000008', ERROR CODE X'0C13'  
DSNU381I =DB2Y DSNUGSRX - TABLESPACE PAOLOR8.CKCPAGE IS IN COPY PENDING  
DSNU012I DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

- The pageset is marked copy pending (COPY or ICOPY)
- If there are many errors on a single page only the first is reported but the COPY utility continues to check remaining pages in the pageset

You should resolve any errors immediately by using recover page (using the page number identified in the DSNU518I or DSNU441I message), recover or repair utilities.

5.7.4 CHECKPAGE performance

In order to verify the impact of the extra checking that takes place during the Image Copy execution some measurements have been taken with and without activating the option.

DB2 V6 with PQ25084 and PQ34972	Number of pages	without CHECKPAGE		with CHECKPAGE	
		CPU time (sec)	Elapsed time (sec)	CPU time (sec)	Elapsed time (sec)
TABLESPACE	147090	2.80	238	3.20	236
INDEX	32440	0.65	58	0.68	57

The results show that the elapsed time is not impacted, and the CPU time increases between 5% for the index, and 15% for the table space. A negligible percentage for an I/O bound utility.

5.7.5 Usage and recommendations

Running the COPY utility with CHECKPAGE option increases the level of consistency checking performed on your image copies. If you intend to exploit this feature, we recommend that you include a copy with CHECKPAGE at least once per backup cycle. So if you keep 7 generations of image copies, perform at least 1 copy with CHECKPAGE every 7 days.

This enhancement simplifies the manual process of checking image copies consistency for those customers who currently choose to run DSN1COPY with the CHECK option on some or all of their image copies.

One difference between running COPY with CHECKPAGE, as compared to running DSN1COPY with CHECK against your image copies, is that a failure of COPY results in a COPY PENDING status whereas DSN1COPY does not. The COPY PENDING flag protects the integrity of your data — you are forced to secure a good copy. The disadvantage is that you may experience some loss of service as you cannot perform any updates until the COPY PENDING status is resolved. DSN1COPY against the image copy has the advantage that you will be alerted to the problem without being denied access to the table space. However, you must be vigilant about detecting and responding to any errors found before all your copies in your backup cycle become invalid.

APAR identifier

The APAR identifier is as follows:

PQ25084 and PQ34972.

5.8 Runstats improvements

In this section we look at two minor improvements in Runstats.

5.8.1 Non uniform statistics for SYSCOLDIST

Runstats will now collect statistics for SYSIBM.SYSCOLDIST when the data is uniform. This extra information may provide better access paths for some queries.

You should bear in mind that this may increase the size of the catalog table.

APAR identifier

The APAR identifier is as follows:

PQ21014.

5.8.2 Additional space statistics

Runstats now collects data which allows you to estimate the number of extents. You can then run sample queries which you will find in SDSNSAMP(DSNTEPS) to determine the number of extents used.

APAR identifier


The APAR identifier is as follows:

PQ25091.

Chapter 6. Performance

Performance



- Star join 
- Volatile tables to use indexes
- Parallelism performance improvements
- Active log I/O performance improvement
- Data sharing improvements

Star join



Introduction to star schema design

Introduction to star join support in DB2 V6

More about DB2 V6 star join using sample scenario

Performance measurements

6.1 Star join

This section covers the following topics:

- What star schemas are
- A summary of how DB2 V6 can better support joins in star schemas
- Examples of star join processing with DB2 V6
- Performance measurements of scenarios where star join helps

Introduction to star schema design



Introduction to star schema design

- Data warehouses, data marts and OLAP
- Example of flat file design for sales data mart
- Example of star schema design for sales data mart
- Example of snowflake schema design for sales data mart

Introduction to star join support in DB2 V6

More about DB2 V6 star join using sample scenario

Performance measurements

6.1.1 Introduction to star schema design

Many customers recognize the distinction between a central data warehouse and one or more data marts oriented to a particular subject area, often dependent on the central data warehouse.

Typically, the central data warehouse needs to support several data marts, which each look at a subset of the data in their own way.

An insurance company, for instance, may have separate data marts for:

- Sales analysis
- Marketing
- Claims analysis
- Risk analysis

These are all fed from a single data warehouse.

Often, the data warehouse will use a classical entity-relationship data model, while the predominant model for the data marts will be a star schema, with the fact tables differing from one data mart to another.

Earlier data marts may have been implemented using a simple database design that mimics a flat file design, with the data heavily denormalized into one or two tables.

This section compares such a denormalized design for a data mart with a star schema design, and a snowflake design, which goes one step further.

Example 2: sales data in star schema



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
39	Mar	1	2000

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
60,000 rows

id	title	firstname	lastname
123	Mr	Fred	Smith
345	Mrs	Hilary	Clinton
246	Ms	Julia	Roberts
432	Mr	Ryan	Giggs
999	Ms	Liza	Minelli
348	Mr	Bill	Clinton

customer (dimension)
10 million rows

time	locn	prod	customer	seller	...
1	1	1	123	22	
2	5	2	345	56	
2	2	2	246	67	
2	2	1	432	12	
3	3	3	999	88	
3	6	2	348	60	

sales (fact)
150 billion rows of <50 bytes each

id	seller name
22	Joe
56	Lynn
67	Herb
12	Mary
88	Joseph
60	Alice

seller (dimension)
5,000 rows

For: flexible, redundant data reduced
Against: more complicated

6.1.1.2 Example 2: sales data in star schema

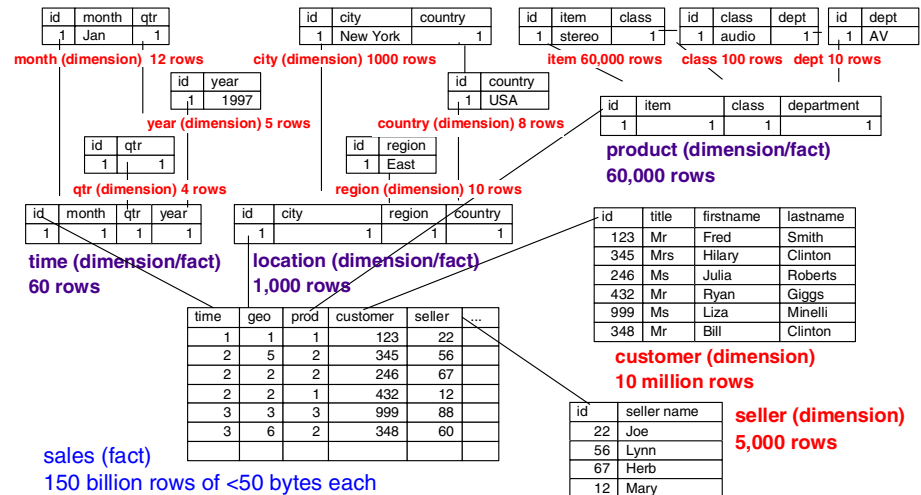
With a star schema design, both the data and any desirable indexes can be much more compact.

Note that this is one example of a star schema design. Do not assume that all star schema designs look like this. The thing that characterizes a design as a star schema is having a number of dimension tables around a fact table. That leaves a lot of room for differences between one star schema design and another, including:

- The degree of normalization
- Whether a dimension table contains rows from only one domain

From this you can already imagine that the applicability of the new star join support in DB2 V6 will vary between different design styles within the star schema design family.

Example 3: sales in snowflake schema



For: even more flexible, redundant data eliminated

Against: even more complicated (15 tables to join in this simple example)

6.1.1.3 Example 3: sales data in snowflake schema

The snowflake schema takes the star schema concept to the next level, with further normalization within the dimensions.

Probably the majority of star schema implementations include snowflakes in at least some of their dimensions.

Note that an alternative design might take different levels within what is shown here as a single dimension (for instance, time) and implement them as separate dimensions. Such a design would still qualify as a star schema, although its capabilities for performance and flexibility are likely to be different from the design shown here.

DB2 V6 star join support



Introduction to star schema designs

Introduction to star join support in DB2 V6

- Optional feature, disabled by default
- **Apply the fix to PQ36206 even if you don't intend to use star join**
- Fast recognition of conditions for star join access path
- Faster execution of certain query types with suitable data and index
- Good for queries that are highly selective on high order index columns
- Only enable it if you are sure you will get an overall benefit

More about DB2 V6 star join using sample scenario

Performance measurements

6.1.2 Introduction to star join support in DB2 V6

A new way of processing multiple-table joins has been added as an option to DB2 Version 6. This is known as star join because it is oriented to joins of several (dimension) tables to a single central (fact) table using the table design pattern known as star schema. It also covers joins of tables using the snowflake schema design which involves one or more extra levels of dimension tables around the first level of dimension tables.

DB2 V6 star join processing is disabled by default, and can be enabled by a new DB2 system parameter for the subsystems that specifically need this function. Star join support is delivered by the fixes to APARs PQ28813 and PQ36206. As always, check on the latest list of any other fixes that may be advisable. This support provides very good performance improvements for a specific class of query against a set of tables which have specific characteristics. The star join enhancement addresses the CPU time reduction at run time for very selective queries by applying the enhanced index repositioning technique, rather than the traditional cartesian join. Queries with low filtering or of I/O bound nature will not benefit from it.

In order to allow customers, whose queries and tables fully meet the required characteristics, to realize these benefits as soon as possible, this support has been made available to DB2 V6 customers through the service stream. An alternative approach would have been to delay delivering the new function until a more generalized solution had been developed, and this would almost certainly have meant that the function would have come in a future release of DB2 beyond V6.

The implications are that you should carefully evaluate whether DB2 star join support will benefit you, before turning it on, and that this is certainly a first stage delivery with more to come in the future. For queries outside the type that currently benefits from the new star join support, there is a possibility that performance will decrease when the support is enabled.

This new star join support is designed in a way that sidesteps potential issues at run time and bind time that have arisen with existing join support:

Run time: Existing join methods are generalized to execute queries well at run time in a wide variety of circumstances. This means there is a fair degree of CPU overhead at run time in testing the environment to make sure the most appropriate code path is taken, from the various options available. Adding further possible options using this technique has the potential to further add to the CPU overhead of executing the query.

Bind time: Existing join support is limited to a maximum of 15 tables in a join because an excessive amount of resource, including CPU and virtual storage, would have been needed to choose the best access path for a join of more tables using existing techniques.

The new star join support addresses these issues as follows.

If you enable star join, the DB2 optimizer uses a set of rules to assess whether the star join appears to be the appropriate technique to use for a join, and, if so, it will use a star join access path without further evaluating all the other possible access paths. These rules are documented later in this chapter.

The optimizer also has a set of rules to choose a fact table index which then drives the actual star join access path. These rules are also outlined later in this chapter.

The star join access path is driven by the order of columns in the chosen index, and is designed for optimal execution as stage 1 rather than by the more flexible, but more CPU intensive, stage 2 level.

This has the following implications:

- If a join qualifies as a star join, the resource used for access path selection is less, making this process more manageable for joins of more than 15 tables when a star join is chosen.
- While the processing costs are, in many cases, lower for a star join than for the prior alternatives, this is not always the case. This depends on how close the query and the tables are to the design point for the star join support.

Because the DB2 optimizer has made the tradeoff to cut the cost of access path selection by choosing star join if the rules are met (without actually estimating the cost of all possible alternatives for the access path) there is no internal check that the star join rules are likely to have the desired effect in a particular case.

The full benefits of the new star join processing can only be realized with both of the following conditions:

- The join columns are included in a multi-column index on the fact table.

- The order of the columns in the index of the fact table is chosen to exploit both relative densities of different column combinations and also the type of queries which are predominantly executed.
- Certain highly selected queries. Only certain types of query are able to benefit from the star join processing. One of the requirements for good exploitation is that the query should be highly selective, starting from the first column in the chosen index of the fact table.

Note that currently star join support can only be enabled, and disabled, at the level of a whole DB2 subsystem.

Currently, star join supports CPU query parallelism, but only within a single DB2 subsystem, not across multiple members of a data sharing group.

Given all these considerations:

- You should only enable star join processing if there are definite indications that it is reasonable to expect a significant net benefit from it, for your specific queries and tables.
- If you enable star join processing for a DB2 subsystem, try to closely monitor whether you are getting the sort of reductions in CPU and elapsed times for queries that meet those reasonable expectations.

Note that the fix to PQ36206 is even more important to you if your circumstances are such that you should disable star join.

Clearly, the more unpredictable your query workload is, the harder it will be for you to set your expectations of star join performance benefits for your workload, and to make representative measurements of any star join performance savings.

There may be circumstances where you could benefit by moving a self-contained group of tables to a new and separate DB2 subsystem, for which star join is enabled, if queries against these tables will perform better with the star join support but queries against other tables will perform worse. In this scenario, the existing DB2 subsystem, in which these other tables remain, should have star join disabled. As outlined in section 6.1.3.7, there is a new subsystem parameter that can be used to specify a minimum ratio of fact table cardinality to dimension table cardinality from which star join should take effect. In some cases, this could provide an alternative method of enabling star join only for some fact tables and not for others, without necessarily having to split them across 2 DB2 subsystems. This would only be possible if the fact tables that would benefit from star join happen to be the ones with the highest cardinalities relative to their dimension tables.

More about DB2 V6 star join



Introduction to star schema designs

Introduction to star join support in DB2 V6

More about DB2 V6 star join

- sample query
- what the access path looks like
- how it works
- the 9 conditions for star join
- fact table index design - key to performance
- how good performance can be achieved
- missing key predicate optimization
- using explain on star join queries

Performance measurements

6.1.3 More about DB2 V6 star join

This section will go into more detail about star join, using a sample scenario as an example. We will illustrate how the star join technique works, and discuss the circumstances in which it is most applicable.

Having a suitable index on the fact table is an absolute prerequisite for having a chance of getting good performance with star join, so we will take time to review how to proceed to get a good index.

Because star join is enabled or disabled at the DB2 subsystem level, you need to make sure you have good indexes for all fact tables in your system before you enable star join at all. You may have tables you don't even think of as being part of a star schema at all, but which the DB2 optimizer may well recognize as such.

With this in mind, we will review the 9 conditions the optimizer tests before choosing a star join access path; they may be met more often than you think!

This entire section will give you the required background to make more sense of the performance results which are presented in the following section.

Typical query with star schema



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
39	Mar	1	2000

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
60,000 rows

time	locn	prod	customer	seller	...
1	1	1	123	22	
2	5	2	345	56	
2	2	2	246	67	
2	2	1	432	12	
3	3	3	999	88	
3	6	2	348	60	
2	6	1	101	23	

sales (fact) 150 billion rows

```
SELECT * FROM SALES S, TIME T,
LOCATION L, PRODUCT P
WHERE S.TIME = T.ID
AND S.LOCATION = L.ID
AND S.PRODUCT = P.ID
AND T.YEAR = 1997
AND T.QTR = 1
AND L.CITY IN ('Boston','Seattle')
AND P.ITEM = 'stereo';
```

Suppose that this stereo is very expensive and is one of 20,000 products that are only ever sold in 600 of the locations, including Boston but not Seattle, and these 600 locations have a 1 in 3 month rota for a single sale. For 1Q97, suppose February was Boston's month.

6.1.3.1 Typical query with star schema

Here is a sample query against a sample set of tables.

```
SELECT * FROM SALES S, TIME T, LOCATION L, PRODUCT P
WHERE S.TIME = T.ID
AND S.LOCATION = L.ID
AND S.PRODUCT = P.ID
AND T.YEAR = 1997
AND T.QTR = 1
AND L.CITY IN ('Boston','Seattle')
AND P.ITEM = 'stereo';
```

Let us now see how DB2 services this query with a star join, assuming that star join processing has been enabled, and the qualifying conditions have been met.

DB2 star join for this query (full index)



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
39	Mar	1	2000

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
60,000 rows

prod	time	locn
1	1	1
1	2	2
1	2	6
2	2	2
2	2	5

index

sales (fact) 150 billion rows
(fact table indexed on all joined columns)

time	locn	prod	customer	seller	...
1	1	1	123	22	
2	5	2	345	56	
2	2	2	246	67	
2	2	1	432	12	
3	3	3	999	88	
3	6	2	348	60	
2	6	1	101	23	

```
SELECT * FROM SALES S, TIME T,
LOCATION L, PRODUCT P
WHERE S.TIME = T.ID
AND S.LOCATION = L.ID
AND S.PRODUCT = P.ID
AND T.YEAR = 1997
AND T.QTR = 1
AND L.CITY IN ('Boston','Seattle')
AND P.ITEM = 'stereo';
```

queryno	qblockno	method	tname	join_type	sortn_join	planno	matchcols
1	1	0	product	s	y	1	0
1	1	1	time	s	y	2	0
1	1	1	location	s	y	3	0
1	1	1	sales	s	n	4	3

method = 1 (nested loop join) plan_table

In this example all the join steps have been efficiently pushed down to data manager

6.1.3.2 DB2 star join for this query (full index)

If the fact table has an index containing all the join columns, then DB2 can perform an efficient star join processed entirely as stage 1.

The example above shows this case. The steps are performed in the order of the multi-column index that contains all the join columns. A nested loop join is used for each step.

We will later examine how you might choose the order of columns in the fact table index that you create to support this kind of efficient processing.

This figure includes an extract of some of the relevant columns from the plan table to give you an example of how a star join query looks when you explain it.

Note that an 'S' appears in the JOIN_TYPE column for the star join.

For reference, we have included below some aspects of how the access path varies between having star join enabled (queryno=1) and having star join disabled (queryno=10) for this same query.

Star join enabled:

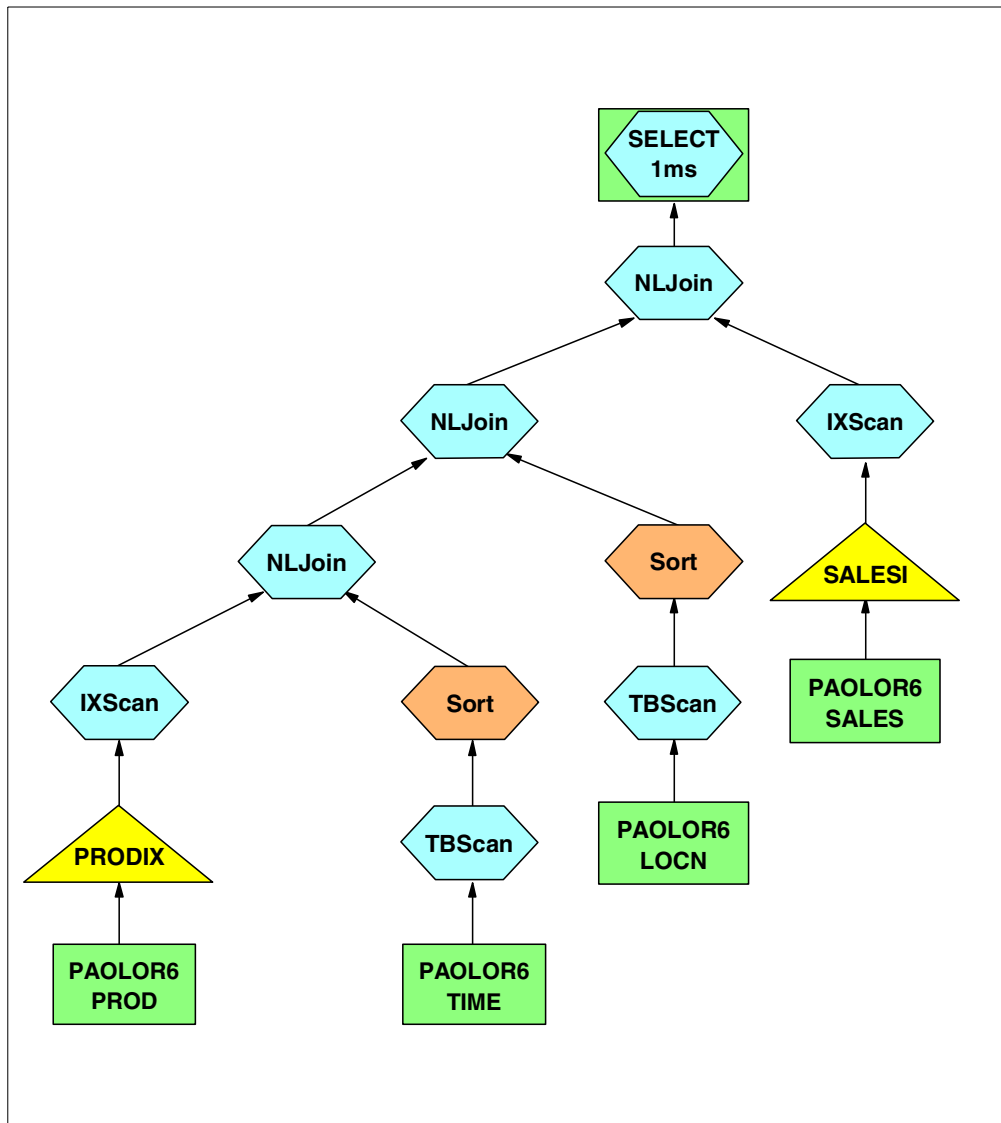
queryno	qblockno	method	tname	join_ type	sortn_ join	planno	matchcols	accesstype
1	1	0	product	s	y	1	0	I
1	1	1	time	s	y	2	0	R
1	1	1	location	s	y	3	0	R
1	1	1	sales	s	n	4	3	I

Star join disabled:

queryno	qblockno	method	tname	join_ type	sortn_ join	planno	matchcols	accesstype
10	1	0	time		n	1	0	R
10	1	1	product		n	2	0	R
10	1	1	location		n	3	0	R
10	1	1	sales		n	4	3	I

Visual Explain access path graph for this star join query

The following figure shows a Freelance approximation of the access path graph that Visual Explain produces for this query, with star join enabled.



Visual Explain access path report for this star join query

The report produced by Visual Explain for this query notes explicitly that star join is being used and that DB2 is executing the steps as stage 1 (**bold type**):

ACCESS PATH REPORT

IDENTIFICATION

Subsystem: DB2Y
Report generation time: (4/5/00 3:48:44 PM)

PLAN

Plan name:
Program name (DBRM): SQLLF000
Plan owner:
Last bind time: 2000-04-05-18.41.02.160000

STATEMENT INFORMATION

Statement number: 1

Statement Cost:

Cost Category: A
Cost in milliseconds: 1
Cost in service units: 5
Reason:

SQL Text:

```
SELECT *
FROM PAOLOR6.SALES S, PAOLOR6.TIME T, PAOLOR6.LOCN L, PAOLOR6.PROD P
WHERE S.TIME = T.ID AND S.LOCN = L.ID AND S.PROD = P.ID AND T.YEAR = 1997 AND T.QTR = 1 AND
L.CITY IN ('BOSTON','SEATTLE')AND P.ITEM = 'STEREO' WITH UR
```

Access Path Description:

Select Statement

```
PAOLOR6.PROD (13)
| PAOLOR6.PRODIX (12)
| Non-matching Index Scan (11)
PAOLOR6.TIME (17)
| Table space scan (16)
| Sort (15)
Nested Loop Join (14)
PAOLOR6.LOCN (111)
| Table space scan (110)
| Sort (19)
Nested Loop Join (18)
PAOLOR6.SALES (115)
| PAOLOR6.SALESIX (114)
| Matching Index Scan (113)
Nested Loop Join (112)
Return (116)
```

Access path step information:

Non-matching Index Scan (11)

Matching index keys used: 0
Index only access: No
Prefetch: Sequential
Access type: I
Page range screening: No
Column function evaluation: Not applicable or to be decided at execution

Table space scan (16)

Prefetch: Sequential
Page range screening: No
Column function evaluation: Not applicable or to be decided at execution

Sort (15)

Column function evaluation: Not applicable or to be decided at execution
Reason for sort: The new table is sorted during a join operation to make the join more efficient.

Nested Loop Join (14)

Join type: Star Join
Stage Stage 1

Table space scan (110)

Prefetch: Sequential
Page range screening: No
Column function evaluation: Not applicable or to be decided at execution

Sort (19)

Column function evaluation: Not applicable or to be decided at execution
Reason for sort: The new table is sorted during a join operation to make the join more efficient.

Nested Loop Join (18)

Join type: Star Join
Stage Stage 1

Matching Index Scan (113)

Matching index keys used: 3
Index only access: No
Prefetch: Unknown at bind time
Access type: I
Page range screening: No
Column function evaluation: Not applicable or to be decided at execution

Nested Loop Join (112)

Join type: Star Join
Stage Stage 1

Select Statement (116)

When optimize: Access path was determined at bind time using default filter factors for any host variables, parameter markers, or special registers.
Group member: Explain not executed in a data sharing environment
Remarks from plan table:
Primary accesstype:
Cost Category: A
Cost in milliseconds: 1
Cost in service units: 5
Reason:

Note: The nested loop joins (14), (18), and (112) are the three steps of the star join; the three steps are not really a simple nested loop join (NLJ); they now include the enhanced repositioning technique that is the characteristic of star join.

How it works



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
39	Mar	1	2000

time (dimension)
39 rows

A workfile is used to list the qualifying key values for each dimension.

prod	time	locn
1	1	2
	2	5
	3	

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension) 1,000 rows

prod	time	locn
1	1	1
1	2	2
1	2	6
2	2	2
2	2	5

index

(fact table indexed on all joined columns)

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
60,000 rows

time	locn	prod	customer	seller	...
1	1	1	123	22	
2	5	2	345	56	
2	2	2	246	67	
2	2	1	432	12	
3	3	3	999	88	
3	6	2	348	60	
2	6	1	101	23	

sales (fact) 150 billion rows

A series of index probes are made, and index manager gives next key feedback:

1. P=1 T=1 L=2; result: not found, next index key P=1 T=2 L=2 (so we can skip a probe for P=1 T=1 L=5)
2. P=1 T=2 L=2; result: 1 index entry found, 1 data row accessed, next index key P=1 T=2 L=6 (so we skip P=1 T=2 L=5)
3. P=1 T=3 L=2; result: not found, next index key P=2 T=2 L=2 (so we are done)

Without next key feedback, we would need 6 index probes to get all qualifying rows.

With next key feedback, we are able to retrieve all qualifying rows with just 3 index probes.

6.1.3.3 How it works

For each key column in the multicolumn index of the fact table, a workfile is generated to store the qualifying column values of the corresponding dimension table, after the local predicates have been applied.

In our example we get:

- One qualifying value for the first column, prod
- Three qualifying values for the second column, time
- Two qualifying values for the third and last column, locn

This is a good case for star join, because:

- We get good selectivity on the first column.
- The number of column values in our workfiles is small.

DB2 uses these column values in the workfiles to make a series of probes into the index, as shown in the diagram above. Extra support has been added to DB2 to provide feedback on the next highest column combination that exists in the index. This saves DB2 from having to make a number of fruitless requests for intervening column combinations which do not actually exist in the index.

Request	Response 1. found or not?	Response 2. next key found	Notes
P=1 T=1 L=2	not found	P=1 T=2 L=2	This tells us that P=1 T=1 L=5 is missing.
P=1 T=2 L=2	found	P=1 T=2 L=6	This tells us that P=1 T=2 L=5 is missing.
P=1 T=3 L=2	not found	P=2 T=2 L=2	This tells us that P=1 T=3 L=2 & P=1 T=3 L=5 are missing.

This table shows how the internal feedback allows DB2 to potentially avoid making several unproductive index probe requests.

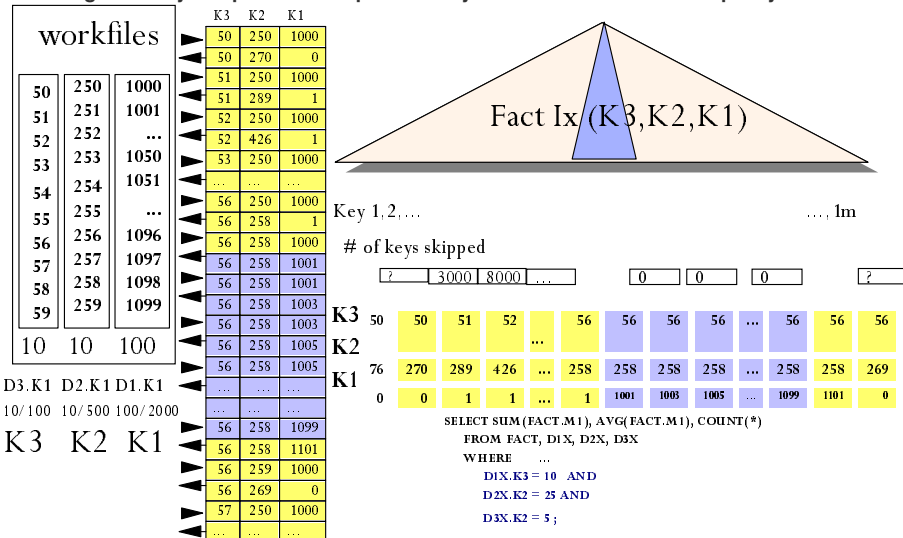
You can begin to see from this how the order of the columns in the index will affect the number of index probe requests that have to be made to find all the qualifying rows.

We will explore this aspect further in the next two figures.

Star join use of fact table index - 1



Example of exploiting correlated index. Efficient repositioning.
Taking relatively few probes to skip over many index entries that do not qualify.



6.1.3.4 Star join use of fact table index — 1

This example, and the one in the next figure, uses different sample fact tables and dimension tables from the previous example.

Here is an example of high efficiency in the way star join processing uses the workfiles to access the fact table via the chosen index.

The first request to DB2 asks for K3=50, K2=250, K1=1000.

DB2 returns "not found", next key is K3=50, K2=270, K1=0.

DB2 has been able to skip checking the other 999 possible qualifying column value combinations beginning with K3=50 (from K3=50, K2=250, K1=1001 to K3=50, K2=259, K1=1099 inclusive).

The second request to DB2 asks for K3=51, K2=250, K1=1000.

DB2 returns "not found", next key is K3=51, K2=289, K1=1.

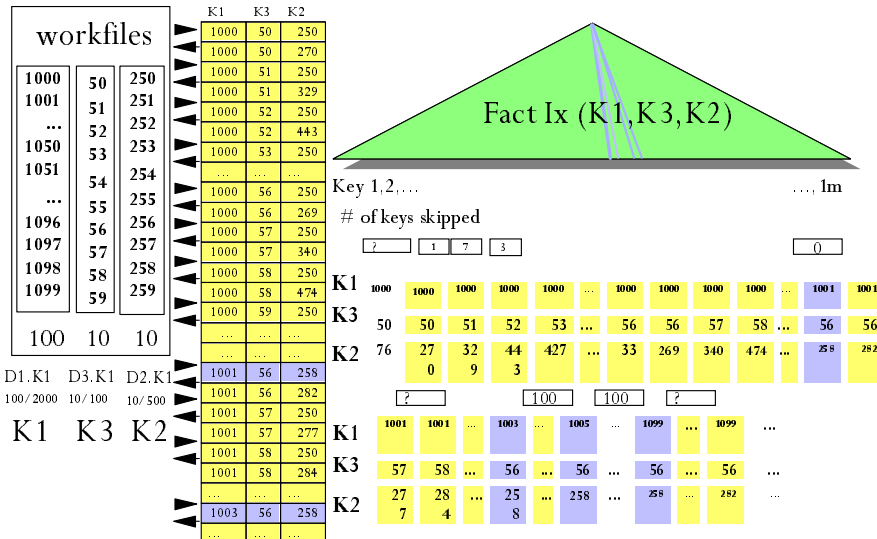
From our first to our second index probe, we have skipped 3000 index entries.

This type of efficiency is possible because of the next key internal feedback, in combination with the correlation between the first and second key columns in the index for the target set of rows.

Star join use of fact table index - 2



Example of less correlated index. Less efficient repositioning.
Taking more probes to skip over the index entries that don't qualify.



6.1.3.5 Star join use of fact table index — 2

Here is an example of lower efficiency in the way star join processing uses the workfiles to access the fact table through the chosen index, because there is less correlation of our qualifying column values in the index.

The first request to DB2 asks for K1=1000, K3=50, K2=250.

DB2 returns "not found", next key is K1=1000, K3=50, K2=270.

DB2 has been able to skip checking internally about the other 9 possible qualifying column value combinations starting K1=1000, K3=50 (from K1=1000, K3=50, K2=251 to K1=1000, K3=50, K2=259 inclusive).

The second request to DB2 asks for K1=1000, K3=51, K2=250.

DB2 returns "not found", next key is K1=1000, K3=51, K2=329.

From our first to our second index probe, we have skipped only 1 index entry in this example.

This case is not so efficient, because the key columns that are closely correlated for the target set of rows are now the second and third key columns in the index.

Good environment for star join



Good selectivity on the first column of the chosen index

- set of qualifying column value combinations needs to be highly selective
- AND most of the filtering has to happen on the first index column
- next most filtering on the second index column, and so on

Small number of column values in workfiles

- the number of qualifying values needs to be low in absolute terms, not just in proportion to the number of possible values
- each probe into the index incurs significant overhead
- want to get the most benefit from the least number of probes

No missing predicate for index columns

- to avoid the impact due to stage 2 predicate

If there are combinations of qualifying column values that do not actually occur in the fact table index and data, it is best if these combinations are clustered together

- again, reduces the number of probes needed

6.1.3.6 Good environment for star join

The considerations listed on this figure come from a combination of the theory behind the design of DB2 V6 star join support and practical experience gained through actual performance measurements in a variety of situations.

You will be able to see from these considerations that it is important to have all of the following conditions:

- Suitable queries that provide good filtering of the fact table, and ideally the dimension tables as well, unless they are extremely small.
- An index that matches these queries very closely, and no queries that don't match the index.
- If there are many combinations of column values which qualify from dimension tables but do not appear in the fact table, these need to be combinations that, when ordered in the sequence of the fact table index, all fit within a small number of gaps in the index.

Conditions to qualify for star join



- 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 must be equi-join predicates.
- All join predicates between the fact table and dimension tables must be Boolean term predicates.
- No correlated subqueries cross dimensions.
- A single fact table column cannot be 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 can exist.
- The data type and length of both sides of a join predicate are the same.
- The fact table contains at least N times the number of rows in the largest dimension table. **Integer N is 25 by default but can be changed.**

6.1.3.7 Conditions to qualify for star join

Star join support was introduced to DB2 V6 with the fix for APAR PQ28813 (PTF UQ33085). With this initial support, a query will be processed as a star join if and only if all the current **9** conditions listed above are met. The fix for APAR PQ36206 is needed to vary the value N in the last condition from the default of 25.

Subsequent to this initial support, two significant changes have been made:

- The increase in the maximum number of tables in a join (from 15 up to 225) has been changed to apply only to the star join case,
- More control has been provided over when star join is used.

These changes have been made available through the following APARs/PTFs:

- The fix for APAR PQ31326 (V6 PTF UQ39733 or V5 PTF UQ39732) re-instates the 15 table limit for all joins in V5 and V6 except those that qualify as star joins in V6. The reason for this is the excessive amount of storage needed for access path selection for joins of many tables that do not qualify as star joins. Star joins of up to 225 tables are still allowed.
- The fix for APAR PQ36206 (PTFs UQ42008, UQ42009) adds a new DB2 system parameter for star join. This provides the way to enable or disable star join, and introduces a new option for when star join is selected once it is enabled. This new option affects just the last of the 9 conditions for star join (ratio of fact table cardinality to largest dimension table cardinality). The qualifying ratio of fact table cardinality to largest dimension table cardinality can be adjusted from the default value (currently 25) to any positive integer.
- Note that access path hints cannot be used to force star join processing.

Fact table index design



Good indexing of the fact table is key to performance benefit

Assuming the query workload is not well understood, start with the index that gives best exploitation of the correlations in the data, for most of a random distribution of queries

- based on density of different column combinations in the fact table
- example follows

When/if query workload is better understood, factor in this guidance:

- Put columns that are accessed more selectively by typical queries at the beginning of the index
- If typical queries often do not reference a dimension, put the column for that dimension at the end of the index

6.1.3.8 Fact table index design

It will be clear to you by now that it is critical for star join to have a good index to use in order to achieve good performance. It is also critical that all the significant queries closely match this index.

In the next few figures, we will walk through a methodology for deriving a likely candidate to be the best choice for this index on the assumption that you know nothing about the queries likely to be run against the data.

The methodology use the concept of the density of a fact table column (or combination of columns) with respect to the matching dimension table (or combination of tables). The density is the cardinality of the combination of columns in the fact table divided by the product of the cardinalities of the matching dimension tables.

The example which follows will show how this works.

The reasoning behind this methodology is that the queries are likely to sample the fact table data in a way such that the numbers of values qualifying from each dimension are still in roughly the same proportion to one another.

Intuitively, we can see that a reasonable first estimate of the shape of the likely queries is that they will follow the shape of the data.

If you actually have better knowledge of the shape of the queries, it is critical that you apply that knowledge when deciding whether there is an index that will make star join viable.

How to index for efficient star joins - 1



Simple example with just 3 dimensions (n=3)

id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
...
39	Mar	1	2000

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA
...

location (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual
...

product (dimension)
60,000 rows

Step 1: Examine all combinations of 2 of the 3 columns, and calculate "density" as number of actual column value combinations in fact table divided by product of corresponding dimension table cardinalities.

	time	locn	prod	...
1	1	1	1	
2	5	2		
2	2	2		
2	2	1		
3	3	3		
3	6	2		
2	6	1		
...

index

sales (fact)
150 billion rows

	A=cardinality from dimension tables	B=cardinality from fact table	C=density =B/A
time,locn	39*1,000=39,000	39,000	1.0000
time,prod	39*60,000=2,340,000	1,799,772	0.7691
locn,prod	1,000*60,000=60,000,000	51,994,000	0.8666

Find the combination with the lowest density (here it is (time,prod)), and choose the column NOT in that combination (here it is locn) as the rightmost column in the index.

6.1.3.9 How to index for efficient star joins — 1

This figure shows the first step of choosing a likely index in the absence of any better information about the queries. This example has 3 dimensions, and starts by examining column combinations in pairs. The more general approach, for N dimensions, will start by examining column combinations (N-1) at a time.

The method determines which combination has the lowest density, and then chooses the column not included in that combination to be the low-order column in the index.

You can see that this method will come up with an N-column index on the fact table covering all N dimensions. The design point for star join is that it will use a single multi-column index on the fact table, and that it is best for this to cover all dimensions in a particular preferred order to match the queries.

This technique is documented in the updated DB2 V6 manuals available at:

<http://www.ibm.com/software/data/db2/os390/v6books.html>

here we provide a graphical example.

How to index for efficient star joins - 2



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
39	Mar	1	2000

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
60,000 rows

	time	locn

index

time	locn	prod	...
1	1	1	
2	5	2	
2	2	2	
2	2	1	
3	3	3	
3	6	2	
2	6	1	

sales (fact)
150 billion rows

Step 2: Examine the 2 remaining columns individually, and calculate "density".

	A=cardinality from dimension tables	B=cardinality from fact	C=density =B/A
time	39	39	1.0000
prod	60,000	59,994	0.9999

Find the column with the lowest density (here it is prod, just), and choose the OTHER column (time) as the next column from the right in the index.

6.1.3.10 How to index for efficient star joins — 2

The next step looks at combinations of N-2 columns, and selects the column not in the combination with the lowest density as the next column from the right in the index. For N=3, this is a pretty trivial step, as shown in this figure.

How to index for efficient star joins - 3



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
39	Mar	1	2000

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
60,000 rows

prod	time	locn

index

time	locn	prod	...
1	1	1	
2	5	2	
2	2	2	
2	2	1	
3	3	3	
3	6	2	
2	6	1	

sales (fact)
150 billion rows

Step 3: Take the remaining column as the high-order column for the index. Here it is prod.

6.1.3.11 How to index for efficient star joins — 3

For N=3, the next and final step is even more trivial. Since there is only one column left, this must be our high-order column for our index.

Characteristics for good performance



Use of index:

- Want to get the answer set with the smallest number of probes to the index, so want qualifying key combinations to be grouped in a small number of sets of contiguous index entries. This also increases chance of index sequential prefetch.

Data access:

- Want to be able to exploit features like data sequential prefetch.
Want good clustering of data with respect to index.

Parallelism:

- If much of the access can be satisfied well via one index, use this as a partitioning index to facilitate parallelism.

6.1.3.12 Characteristics for good performance

Up until now, we have concentrated most on the use of the workfiles to drive requests to the chosen fact table index, and the need to keep down the number of index probes. The other side of the coin is to get as much value as possible from each index probe. This is where features like index sequential prefetch and data sequential prefetch can help us get back very efficiently all the required columns of all rows that match the index probe.

You can see that the design point for star join is to cluster the data by the index that you have set up to support the star join. This increases the chances of getting value from sequential prefetch. It also increases the chances of getting value from query parallelism.

As we shall see in the performance results section later, parallelism is an area where star join may derive more benefit than the non star join alternative.

Missing key predicate optimization



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
39	Mar	1	2000

time (dimension)
39 rows

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension)
1,000 rows

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
60,000 rows

A workfile is used to list the qualifying key values for each dimension.

prod	time	locn
1	1	2
	2	5
	:	
	39	

prod	time	locn
1	1	1
1	2	2
1	2	6
2	2	2
2	2	5

index

time	locn	prod	customer	seller	...
1	1	1	123	22	
2	5	2	345	56	
2	2	2	246	67	
2	2	1	432	12	
3	3	3	999	88	
3	6	2	348	60	
2	6	1	101	23	

sales (fact) 150 billion rows
(fact table indexed on all joined columns)

```
SELECT * FROM SALES S, LOCATION L, PRODUCT P WHERE
S.LOCATION = L.ID AND S.PRODUCT = P.ID AND L.CITY IN
('Boston','Seattle') AND P.ITEM = 'stereo';
```

Example: there is no predicate on the 2nd of 3 cols in index

- DB2 simulates a predicate (that does no filtering) on the missing column
- allows MATCHCOLS=2 to be achieved rather than MATCHCOLS=1

6.1.3.13 Missing key predicate optimization

Consider the following SQL statement:

```
SELECT * FROM SALES S, LOCATION L, PRODUCT P
WHERE S.LOCATION = L.ID
AND S.PRODUCT = P.ID
AND L.CITY IN ('Boston','Seattle')
AND P.ITEM = 'stereo';
```

This query has no predicates on the time dimension table, which corresponds to the second of the three columns in the fact table index.

In this case, DB2 effectively simulates a predicate on the dimension that is missing from the query, and which does not provide any filtering.

You can see that the value of MATCHCOLS in the plan table for the explain of this query is 2, referring to the first column (prod) and the third column (locn).

This technique allows star joins in queries like this to still be serviced fully by the DB2 as stage 1.

For reference, we have included below the important aspects of how the access path varies between having star join enabled (queryno=2) and having star join disabled (queryno=20) for this same query.

Star join enabled:

queryno	qblockno	method	tname	join_ type	sortn_ join	planno	matchcols	accesstype
2	1	0	product	s	y	1	0	I
2	1	1	location	s	y	2	0	R
2	1	1	sales	s	n	3	2	I

Star join disabled:

queryno	qblockno	method	tname	join_ type	sortn_ join	planno	matchcols	accesstype
20	1	0	product		n	1	0	I
20	1	1	sales		n	2	1	I
20	1	1	location		n	3	1	R

Star join with missing index column - 1



Suppose time is not in the chosen fact table index

The standard stage 1 star join technique is used for the dimensions covered by the fact table index to produce an intermediate results table.

id	item	class	department
1	stereo	audio	audio-visual
2	cd player	audio	audio-visual
3	television	video	audio-visual

product (dimension)
50,000 rows

prod	locn
1	2
1	5

workfiles

prod	locn
1	1
1	2
1	6
2	2
2	5
2	6
3	3

sales index
(prod,locn)

time	locn	prod	customer	seller	...
1	1	1	123	22	
2	5	2	345	56	
2	2	2	246	67	
2	2	1	432	12	
3	3	3	999	88	
3	6	2	348	60	
1	1	1	101	23	

sales (fact) 150 billion rows

time	locn	prod	customer	seller	...
2	2	1	432	12	

temporary table dsnwfb(02)

id	city	region	country
1	New York	East	USA
2	Boston	East	USA
3	Chicago	East	USA
4	San Jose	West	USA
5	Seattle	West	USA
6	Los Angeles	West	USA

location (dimension) 1,000 rows

```
SELECT * FROM SALES S, TIME T,
LOCATION L, PRODUCT P
WHERE S.TIME = T.ID
AND S.LOCATION = L.ID
AND S.PRODUCT = P.ID
AND T.YEAR = 1997
AND T.QTR = 1
AND L.CITY IN ('Boston','Seattle')
AND P.ITEM = 'stereo';;
```

queryno	qblockno	method	tname	join_type	sortn_join	matchcols
1	1	0	time	s	y	0
1	1	2	dsnwfqb(02)	s	y	0
1	2	0	product	s	y	0
1	2	1	location	s	y	0
1	2	1	sales	s	n	2

plan_table
method = 1 (nested loop join)

6.1.3.14 Star join with missing index column — 1

Where there is a predicate in the query for a dimension that is missing from the index, the performance impact is more severe, since this means that the predicate is a type 2 predicate.

This figure and the next one show a specific example of this case.

The query has local predicates on the time, location and product dimension tables, but the chosen fact table index is a two-column index on (locn,prod).

In this example, the time dimension table has no index at all.

DB2 uses stage 1 star join processing for the dimensions which are included in the index, creating an intermediate work table.

Star join with missing index column - 2



id	month	qtr	year
1	Jan	1	1997
2	Feb	1	1997
3	Mar	1	1997
4	Apr	2	1997
5	May	2	1997
6	Jun	2	1997
42	Jun	2	2000

time (dimension) 60 rows

time	locn	prod	customer	seller	...
2	2	1	432	12	

temporary table dsnwfb(02)

merge scan join by RDS on sorted temporary table

time	locn	prod	customer	seller	...
2	2	1	432	12	

queryno	qblockno	method	tname	join_type	sortn_join	matchcols
1	1	0	time	s	y	0
1	1	2	dsnwfb(02)	s	y	0
1	2	0	product	s	y	0
1	2	1	location	s	y	0
1	2	1	sales	s	n	2

method = 2 (merge scan join)

plan_table

```
SELECT * FROM SALES S, TIME T,
LOCATION L, PRODUCT P
WHERE S.TIME = T.ID
AND S.LOCATION = L.ID
AND S.PRODUCT = P.ID
AND T.YEAR = 1997
AND T.QTR = 1
AND L.CITY IN ('Boston','Seattle')
AND P.ITEM = 'stereo';;
```

6.1.3.15 Star join with missing index column — 2

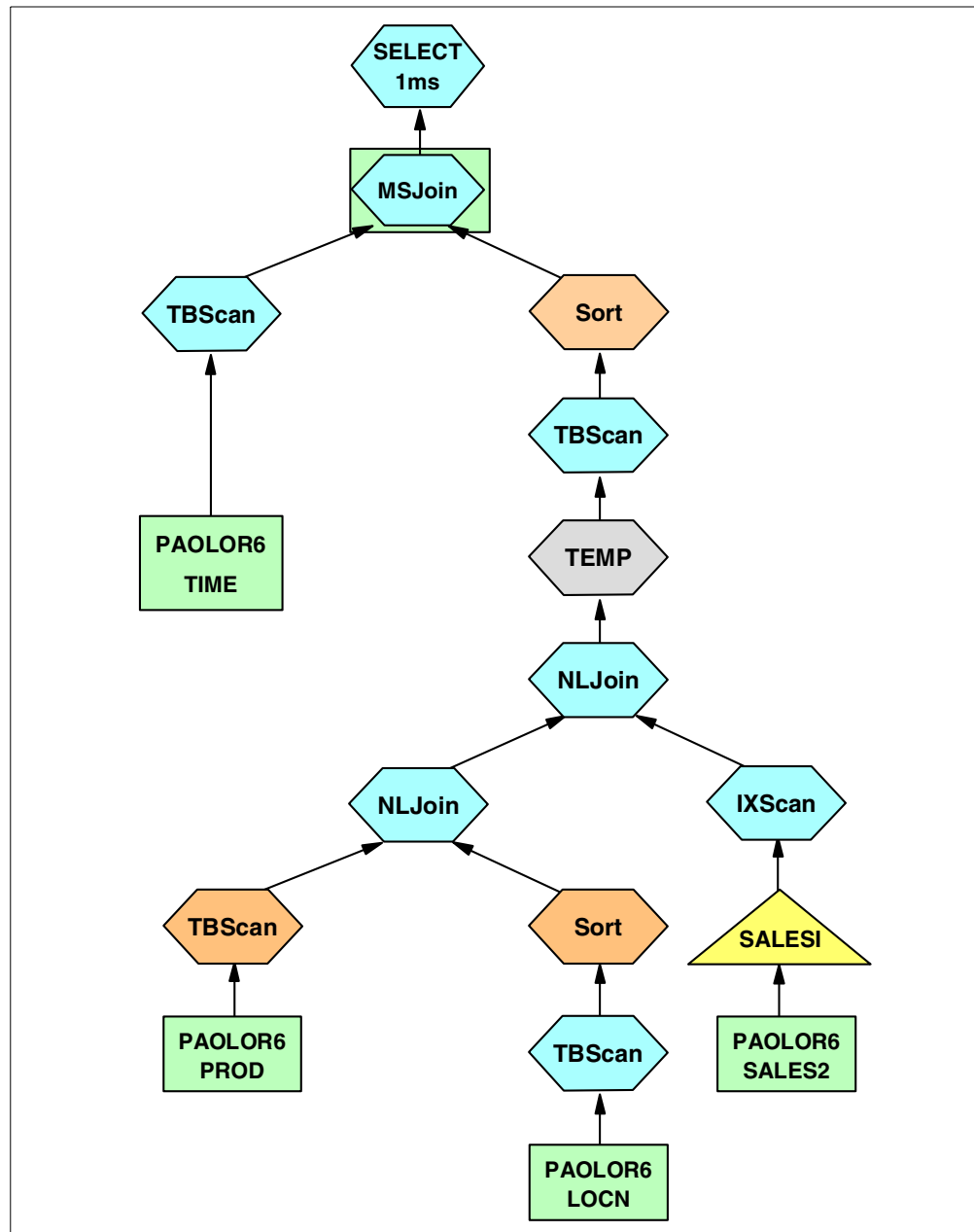
DB2 the uses a merge scan join to join the time dimension table to the intermediate work table. This is a stage 2 part of the star join process.

This processing for this kind of star join example will be significantly more costly than the cases we examined earlier, where all the dimensions with local predicates in the query were covered by the chosen fact table index.

The following three pages show the access path graph and report produced for this example by Visual Explain.

Visual Explain access path graph for deficient index example

The following figure shows a Freelance approximation of the access path graph that Visual Explain produces for this query, with star join enabled.



Visual Explain access path report for missing index example

ACCESS PATH REPORT

IDENTIFICATION

Subsystem: DB2Y
Report generation time: (4/6/00 11:04:53 AM)

PLAN

Plan name:
Program name (DBRM): SQLLF000
Plan owner:
Last bind time: 2000-04-06-13.56.57.950000

STATEMENT INFORMATION

Statement number: 1

Statement Cost:

Cost Category: A
Cost in milliseconds: 1
Cost in service units: 5
Reason:

SQL Text:

```
SELECT *
FROM PAOLOR6.SALES2 S, PAOLOR6.TIME T, PAOLOR6.LOCN L, PAOLOR6.PROD P
WHERE S.TIME = T.ID AND S.LOCN = L.ID AND S.PROD = P.ID AND T.YEAR = 1997 AND T.QTR = 1 AND
L.CITY IN ('BOSTON','SEATTLE')AND P.ITEM = 'STEREO' WITH UR
```

Access Path Description:

Select Statement

```
PAOLOR6.TIME (12)
| Table space scan (11)
PAOLOR6.PROD (22)
| Table space scan (21)
PAOLOR6.LOCN (26)
| Table space scan (25)
| Sort (24)
Nested Loop Join (23)
PAOLOR6.SALES2 (210)
| PAOLOR6.SALESIX2 (29)
| Matching Index Scan (28)
Nested Loop Join (27)
| Temporary Work File (16)
| Table space scan (15)
| Sort (14)
Merge Scan Join (13)
Return (17)
```

Access path step information:

Table space scan (11)

Prefetch: Sequential
Page range screening: No
Column function evaluation: Not applicable or to be decided at execution

Table space scan (21)

Prefetch: Sequential
Page range screening: No
Column function evaluation: Not applicable or to be decided at execution

Table space scan (25)

Prefetch: Sequential

Page range screening: No
 Column function evaluation: Not applicable or to be decided at execution

Sort (24)

Column function evaluation: Not applicable or to be decided at execution
 Reason for sort: The new table is sorted during a join operation to make the join more efficient.

Nested Loop Join (23)

Join type: Star Join
Stage Stage 1

Matching Index Scan (28)

Matching index keys used: 2
 Index only access: No
 Prefetch: Sequential
 Access type: I
 Page range screening: No
 Column function evaluation: Not applicable or to be decided at execution

Nested Loop Join (27)

Join type: Star Join
Stage Stage 1

Table space scan (15)

Prefetch: Unknown at bind time
 Page range screening: No
 Column function evaluation: Not applicable or to be decided at execution

Sort (14)

Column function evaluation: Not applicable or to be decided at execution
 Reason for sort: The new table is sorted to give the correct ordering for the join operation.

Merge Scan Join (13)

Join type: Star Join
Number of columns joined: 1
Stage Stage 2

Select Statement (17)

When optimize: Access path was determined at bind time using default filter factors for any host variables, parameter markers, or special registers.
 Group member: Explain not executed in a data sharing environment
 Remarks from plan table:
 Primary accesstype:
 Cost Category: A
 Cost in milliseconds: 1
 Cost in service units: 5
 Reason:

Performance results



In selected circumstances, star join performs much better than non-star join, especially with:

- multi-column fact table index with high clusterratio covering all dimensions
- queries that are selective in qualifying column values and fact table rows, especially for high-order columns of the index
- snowflakes in the table design, that are referenced in the query

Remember that star join performance is highly dependent on the circumstances, and the following examples suit star join well

Your mileage will vary !

6.1.4 Performance results

This section shows examples of some major performance improvements that can be achieved with star join.

However, it is very important for you to realize that there are other circumstances where non star join will perform better, whether with lower elapsed time or lower CPU time, or both.

For a given set of tables and indexes that support the kind of performance improvements for certain queries with star join that are shown here, it will always be possible to construct other queries for which non star join is faster.

This is why you should be very careful to ensure that you will get net performance benefits for the query workload against the tables and indexes in your DB2 subsystem, before you enable star join for that DB2.

Performance results - serial query



Characteristics of tables and sample query:

- 7 dimensions, 7-column index on fact table (26.4 million rows)
- local predicates on dimensions for 1st, 2nd and 5th index columns
- snowflake reference on 1 dimension
- 384 qualifying rows
- 12 groups returned

Non Star Join elapsed	Non Star Join CPU	Star Join elapsed	Star Join CPU	% elapsed saved by SJ	% CPU saved by SJ
3007.82 sec	126.41 sec	5.10 sec	4.34 sec	99.8%	96.6%

6.1.4.1 Performance results for serial query

The fact table in these performance measurements had a density of only 0.56% with respect to the dimension tables. There were only 26.4 million rows in the fact table but there were 4.7 billion possible combinations of the values that occurred in the 7 dimension tables.

Cardinalities of the dimension and fact tables:

dim1	dim2	dim3	dim4	dim5	dim6	dim7	product of the 7 dims	fact
1	36	1	5032	2180	12	1	4.7 billion	26.4 million

The set of fact table rows that qualified had an even lower density, at 0.02%. There were only 384 rows of the fact table that qualified, even though there were 1.69 million possible combinations of the column values that qualified from the dimension tables.

Cardinalities of the qualifying subsets of the dimension and fact tables:

dim1	dim2	dim3	dim4	dim5	dim6	dim7	product of the 7 dims	fact
1	1	1	5032	28	12	1	1.69 million	384

This is the sort of situation which is likely to suit the star join, and not suit the non-star join, and so it proved in this case.

Performance results - parallel query



Characteristics of tables and sample query:

- 7 dimensions, 7-column index on fact table (26.4 million rows)
- local predicates on dimensions for 1st column only
- no snowflake references
- 26.4 million qualifying rows (all rows)
- 15,000 groups returned

Parallel degree	Non Star Join elapsed	Non Star Join CPU	Star Join elapsed	Star Join CPU	% elapsed saved by SJ	% CPU saved by SJ
1	4079 sec	4047 sec	2343 sec	2277 sec	43%	44%
ANY	3782 sec	4259 sec	545 sec	2481 sec	86%	42%
% saved by parallelism	7%	-5%	77%	-9%		

Star join case exploited parallelism much better than non star join

- star join case drove parallelism from the fact table
- non star join case drove parallelism from a dimension table with only one qualifying value, so very little overlap was achieved

6.1.4.2 Performance results for parallel query

This example shows a different kind of performance improvement that may be achievable with star join. The tables and indexes are the same as in the previous example. This time, we do not have selective local predicates on the dimension tables. All rows from the fact table qualify. With serial execution, the savings in elapsed time and CPU time are much less dramatic than in the previous query, because we don't have any selectivity. Nevertheless, star join almost gets down to half of the elapsed and CPU time of non star join for the serial case.

The bigger benefit is the extra saving in elapsed time that can be seen when query parallelism is used with star join compared to when it is used with non star join. The problem with non star join in this case is that the parallelism was driven from a dimension table that had only 1 row, and so the benefit of DEGREE=ANY was very limited (although there was some benefit). With star join, parallelism is driven from the fact table, which in this case had 36 partitions, so there is much more chance of getting a big elapsed time reduction, as shown here. A 5-way processor was used for the measurements shown here.

For query parallelism with star join, the dimension tables are replicated across the individual parallel tasks as necessary. The degree of parallelism actually chosen depends on whether the chosen index of the fact table is partitioned.

- If the index is partitioned, the degree is taken to be the number of partitions.
- If not, the degree is the number of CPUs in the machine.

DB2 V6 star join summary



Apply the fix to PQ36206 even if you don't intend to use star join

Star join becomes an optional feature, disabled by default

Fast recognition of conditions for star join access path

Faster execution of certain query types with suitable data and index

Good for queries that are highly selective on high order index columns

Only enable it if you are sure you will get an overall benefit

Configuration tuning is important: watch your workfiles and buffer pools

6.1.4.3 DB2 V6 star join summary

This figure repeats a summary of some of the key points about star join support in DB2 V6.

It has the potential for major performance improvements in certain circumstances. However, it also has the potential for performance regression if used without great care. This section has covered the major considerations to take into account when deciding whether to use it.

The two most important points are:

- Apply the fix to PQ36206 even if you do not intend to use star join, since this is the fix that gives you the ability to enable, disable, and otherwise adjust it, and
- Only enable it if you are sure you will get an overall benefit.

Once you are using star join remember that normal tuning of your configuration still applies:

- star join tends to make heavy usage of workfile when there is low filtering on the dimension tables,
- physical separation of fact and dimension tables may improve the I/O
- association of data to different buffer pools helps when investigating

APAR identifier

The APAR identifier is as follows:

PQ28813, PQ33666, PQ31326, PQ36206.

Volatile tables to use indexes



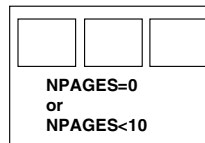
For tables that are small or empty at runstats time but large at execution time

Table space scan chosen when index access may be faster

New system parameter NPGTHRSH

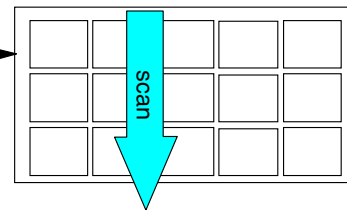
- 0 standard cost optimization (default)
- n if npages < n then prefer index access
- -1 always prefer index access (not recommended)

At runstats



SQL →

At execution



6.2 Volatile tables to use indexes

There can be performance problems for volatile tables which are small or empty when runstats is run but which have grown larger when accessed by applications. When the catalog statistics indicate an empty or very small table ($NPAGES < 10$), DB2 chooses a table space scan as this is more efficient than an index scan for this size of table. However, if the catalog statistics do not reflect the true table size and it has grown much larger this can lead to unacceptable performance.

A solution to this is to manually update the catalog statistics but this increases operational complexity and causes locks on the catalog tables. In addition, it may be difficult to identify which tables in the system are volatile.

This enhancement provides a new installation system parameter NPGTHRSH which will cause the DB2 optimizer to favor index access for tables whose statistics indicate less than a given number of pages and a matching index scan is possible.

For example, setting the NPGTHRSH value to 200 will cause all tables with $NPAGES < 200$ to favor index access over a table space scan.

For table where statistics have not been collected ($NPAGES = -1$) DB2 will assume 501 as the default value for NPAGES, so the NPGTHRSH should be set to a value less than 502.

The NPGTHRSH default is 0 and the DB2 optimizer will then select the access path based on cost as it normally does.

This solution is efficient and easy to implement. The disadvantage is that it could be less efficient than standard optimization in some cases. For example, if the small table was being accessed many times in a join, a table space scan may be more efficient. Consequently, this parameter should be used with care so that performance is not degraded.

APAR identifier

The APAR identifier is as follows:

PQ33429.

Query parallelism enhancements



Ability to limit degree of parallelism

- new DSNZPARM PARAMDEG
- consider if need to control virtual stored

Short static SQL running with parallelism

- performance improvements

[Click here for optional figure #](#)

YRDDPPPPUUU

6.3 Query parallelism enhancements

In this section we describe two enhancements related to query parallelism.

6.3.1 New feature to limit degree of parallelism

A new DSNZPARM, PARAMDEG, has been introduced; with this parameter, which can be set in the DSNTIP4 panel of the installation procedure, you can control the upper limit of the degree of parallelism for a parallel group. The value can be set between 0 and 254 where 0 means that there is no upper limit.

You should consider setting this parameter if you need to contain the usage of resources when running highly parallel queries. This function can be very useful when the DB2 environment is virtual storage constrained or you are encountering large increases in virtual storage when running parallelism.

APAR identifier

The APAR identifier is as follows:

PQ28414.

6.3.2 Short running static SQL running with parallelism

There have been performance enhancements for short-running static SQL queries that run in an environment with parallelism enabled.

The overhead of switching to sequential processing at run time once the host variable values are known has been reduced.

APAR identifier

The APAR identifier is as follows:

PQ25135.

Active log I/O performance



More than three concurrent active log readers

- DB2 assigns some to secondary copy
- I/O response time for read and write improved
- Another reason for dual logging

More efficient log writes

- Increase number of CIs written together
- Parallel CI writes

Improved instrumentation for log writes

Reducing contention for internal log write latch

6.4 Active log I/O performance

In the past, the main log read usage, apart from during DB2 system restart, was for recovering table spaces after media failure. Log read activity increased with data sharing, because we add the need to recover from logs after loss of a group buffer pool, if the new DB2 V6 function group buffer pool duplexing is not being used.

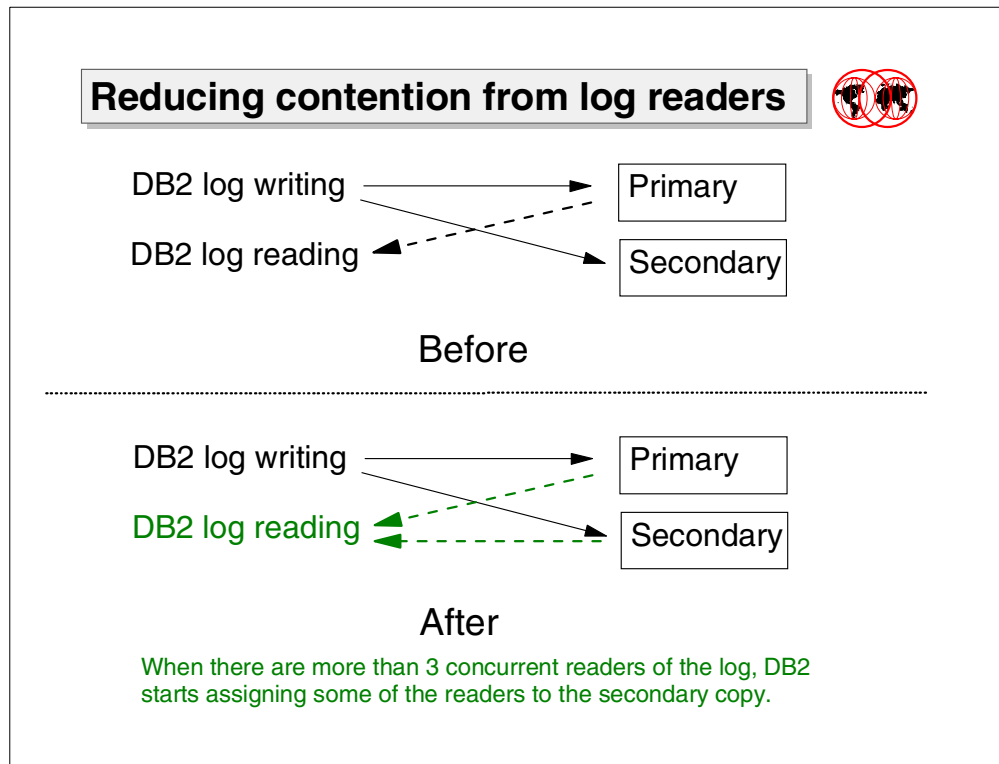
However, in recent years, there has been a significant increase in the level of read activity against the active log in many DB2 systems attributed to normal operational running. Specific examples of this include:

- Use of DPropR for data propagation
- Use of online reorganization
- Use of ISV products that read the logs for recovery or auditing purposes

DB2 support for reading the active log was not originally designed for this diverse and frequent usage. Several further improvements have been made to DB2 V6 logging, read and write activity, to cope with the evolution in the way in which the log has come to be used.

- Direct read requests to secondary rather than just the primary active log copy
- Improved log CI write processing
- Additional instrumentation
- Reduced log write latch time

We discuss these improvements in detail below.



6.4.1 Reducing contention from log readers

The diagram above shows how DB2 used to satisfy read and write requests to and from the active log in a dual logging environment (upper half of diagram). All readers used to be assigned to the primary copy of the active log. Writes were made to both log sets.

Log I/O contention has now been alleviated in circumstances when there are more than three concurrent readers of the DB2 active log. DB2 will satisfy some of the requests from the secondary copy of the active log (lower half of diagram). This improves read time and balances I/O. The response time of log writes should also therefore be improved.

We strongly recommend that you adopt dual logging for availability reasons, even if you are exploiting RAID devices. The logs are absolutely vital and protect the integrity of the entire subsystem and all of your data. The log pairs should be on different physical devices and use different I/O paths. These performance enhancements leverage the investment you have made in your dual logging infrastructure.

APAR identifier

The APAR identifier is as follows:

PQ25745.

DB2 log write improvements



Log writes of CIs combined

- Based on log write threshold
- Default value of 20 recommended
- Increase from 20 to 128 CIs per start I/O

Reduced serialization writing to primary and secondary

- only first CI written serially

Additional log write instrumentation

The log-write latch (class 19) is now held for less time.

6.4.2 DB2 log write improvements

In this section we have put together several log write enhancements. APAR PQ28857 improves log write performance and adds new instrumentation.

6.4.2.1 Log CI writes combined

The log write threshold provides a way to ensure that log records get externalized after a certain number of log control intervals (CIs) are placed into the log buffers. The default value of 20 is recommended. Previously, this number of CIs was always written as one start I/O. Now, these CIs will be combined with others into a single start I/O at times of high logging, up to a limit of 128 CIs per start I/O.

6.4.2.2 Log CI write parallelism

With dual logging, there is now less serialization between writing to the primary and writing to the secondary. Typically, only the first CI in a list to be written will be written serially, the rest will be written in parallel.

6.4.2.3 Instrumentation

There is more instrumentation in the logging statistics record. An addition six counters are recorded as follows.

QJSTLSUS

The number of times that a log manager request results in a suspend for a log record that is being written out to the log data sets. This is the sum of the waits recorded by IFCID 32 and IFCID 33 pairs.

QJSTLOGW

The total number of log write I/O requests (media manager calls). This is the sum of the waits recorded by IFCID 38 and IFCID 39 pairs. This value includes waits for copy1 and copy2 active log data set writes. This value should correspond to the active log write I/O activity in an RMF report.

QJSTCIWR

The total number of log CIs written. This value includes CI rewrites and copy1 and copy2 active log data set writes. If a given CI is rewritten five times, this counter is incremented by five. This counter, multiplied by 4 KB and divided by the statistics interval in seconds, represents the number of bytes per second of log data written to the active log data sets. When this value exceeds 1 MB/sec per log copy, attention should be paid to log data set I/O tuning.

QJSTSERW

The number of serial log write I/O requests. A serial log write I/O request occurs when DB2 rewrites a log CI that was previously written as a partial CI. In a dual logging environment, this value includes copy1 and copy2 active log data set writes. The difference between QJSTLOGW and QJSTSERW represents the number of parallel log write I/O requests. Typically, the first CI in a list of CIs to be written in one start I/O is written serially, and the remaining CIs are written in parallel to both copy1 and copy2 active log data sets. This value is meaningful only when DB2 runs in dual active log mode.

QJSTTHRW

The number of times that an asynchronous log write request was scheduled because the log write threshold was reached. This counter is provided primarily for an internal check. We recommend you use the default write threshold of 20 buffers.

QJSTBPAG

The number of times that a log output buffer was paged in before it could be initialized. The log write latch is held while a log buffer is being initialized. A nonzero value could indicate that the log output buffer size is too large or there is insufficient real storage to back the log output buffer size.

APAR identifier

The APAR identifier is as follows:

PQ28857.

6.4.2.4 Log write latch improvements

The code that moves log records into the log output buffer has been optimized to reduce the amount of time that the log-write latch (class 19) is held thus alleviating a potential bottleneck.

APAR identifier

The APAR identifier is as follows:

PQ30461.

Data sharing improvements



- Insert performance
- Remove CLOSE YES requirement
- Name class queue support
- Improved trace for synchronous requests

6.5 Data sharing improvements

In this section we describe the performance improvements related to data sharing.

6.5.1 Insert performance

A number of changes have been made to improve the performance of an insert-intensive workload running in a data sharing environment.

The number of times that the index page P-lock needs to be transferred back and forth between members has been reduced. Before being released, it will wait for a short time to see if there are any further latch requests for it.

To avoid repeated write I/O for pages that are frequently modified such as space map pages, the following changes have been made:

- For non-GBP-dependent objects, the vertical deferred write queue will be maintained in LRU rather than by clean-to-dirty LRSN.
- Where the VDWQ is set to zero, a write for 32 pages will not be scheduled until there are 40 pages on the queue

For member cluster table spaces, P-locks on data pages will be held past commit in the anticipation that the same member will modify them again.

0

APAR identifier

The APAR identifier is as follows:

PQ22910.

6.5.2 Remove CLOSE YES as requirement for data set physical close

DB2 dynamically tracks inter-system read write interest in group buffer pool dependent objects. Based on some criteria it will make the pageset or partitioning non group-buffer pool dependent by physically closing the data set which drops the P-lock. DB2 no longer requires the pageset or partition to be defined as CLOSE YES for this to happen.

Update for ALL users regarding the close rule

Over time, various recommendations have been made regarding the CLOSE rule for DB2 table spaces and indexes. Various changes within DB2 mean that the close rule no longer has as much of an impact as before. The only case where it does make a difference is when the maximum number of open data sets for the subsystem (DSMAX) is close to being reached. In this case DB2 will start closing the CLOSE YES data sets in least recently used sequence. If it needs to continue to close more data sets it will close the CLOSE NO data sets also in least recently used sequence.

APAR identifier

The APAR identifier is as follows:

PQ27637.

6.5.3 Name class queue support

With this function DB2 will make use of the name class queue support introduced with Coupling Facility Level 7. The Coupling Facility Control Code (CFCC) will organize the elements in the GBP into 'queues' based on DBID, PSID and partition number. This organization allows locating and purging these elements in a more efficient manner during pseudo-close or DB2 shutdown.

For those planning on installing or migrating the coupling facility to CFLEVEL= 7 or 8, please see APAR PQ35919, if using DB2 V6 with data sharing and APAR PQ23043/UQ35798 is installed. In this case you must ensure that the correct CFCC service levels are applied for CFLEVEL=7 or 8. For CFLEVEL=7, you need to ensure that CFCC service level 1.06 or above is applied. For CFLEVEL=8, you need to ensure that CFCC service level 1.03 or above is applied. If DB2 V6 data sharing and APAR PQ23043/UQ35798 runs with group buffer pools allocated in a CFLEVEL=7 coupling facility with a service level prior to 1.06, or CFLEVEL=8 with a service level prior to 1.03, then severe DB2 data integrity problems can be encountered. APAR PQ35919 contains ++HOLD data with further instructions.

APAR identifier

The APAR identifier is as follows:

PQ23043, PQ32199, PQ35919.

6.5.4 Improved trace for asynchronous requests

IFCID 329 is being added to track the wait time associated with GBP requests converted from synchronous to asynchronous. In addition castout trace IFCID 263 is being enhanced for CFLEVEL 7 functionality (Name Class Queues).

IFCID 329 quantifies the time spent while waiting on a GBP asynchronous CF request. IFCID 329 was added to accounting class 3, monitor class 3 and performance class 21. Additionally new Group Buffer Pool (GBP) counters were added to count the number of occurrences of these asynchronous requests.

APAR identifier

The APAR identifier is as follows:

PQ28722.

Chapter 7. Additional functional enhancements

Additional functional enhancements



Unicode client toleration support

IEEE float toleration

Controlling updates to partitioning key

Toleration of separator differences

New LANGUAGE bind option

New operator for NOT

DBPROTCL default change

Instrumentation enhancements

Unicode client toleration support



Requires Language Environment for OS/390 V2 R9

DB2 used to just search SYSSTRINGS for a translation table to do the required character conversion

However now DB2

- searches cache
- searches SYSIBM.SYSSTRINGS
- searches LE
- If no success, returns SQLCODE -322

Also provides support for other conversions supported by LE but not in SYSSTRINGS

7.1 Unicode client toleration support

All DB2 string data that is not defined FOR BIT DATA has an encoding scheme and is associated with a coded character set identifier (CCSID). Typically this is transparent to you because when you select string data from a table, the selected data follows the same encoding scheme as the table. DB2 support for character set conversions is defined in SYSIBM.SYSSTRINGS. The nature of the translation is specified by the TRANSTYPE column and available conversions include, EBCDIC to ASCII and ASCII to EBCDIC. Recent additions include provision support for the EURO symbol and the extended code page 300.

There is a requirement for client Unicode support. Language Environment (LE) for OS/390 Version 2 Release 9 has been enhanced to handle UNICODE conversions and DB2 uses this to support Unicode. Rather than providing a translation table in SYSIBM.SYSSTRINGS, DB2 has been enhanced to externalize the conversion request to LE. This brings UNICODE support for DB2 today and leverages facilities provided by LE so the probability of handling conversion requests has increased. To determine which extra conversions are supported, please refer to *OS/390 V2R9 C/C++ Programming Guide*, SC09-2362-05 under "Code set converters supplied" or "Universal coded character set converters".

For character conversions, provided that you have the right version of Language Environment, DB2 performs the following steps:

1. Searches the cache
2. Searches SYSIBM.SYSSTRINGS
3. Searches LE (provided all pre-requisites are satisfied)
4. If no success, return SQLCODE -332 (as before)

The prerequisites for UNICODE support in DB2 are OS/390 V2R9 for the required Language Environment and the following APARs.

APAR identifier

The APAR identifier is as follows:

PQ32782 and PQ32803.

IEEE Float toleration



DB2 stores floating point numbers in S/390 HFP format

Toleration provides for IEEE Binary Floating Point (BFP) host variables

For ASM, C, C++ only

Specify the `FLOAT(IEEE)` option on the precompile/compile

DB2 converts and stores the value as HFP format

7.2 IEEE float toleration

DB2 stores floating point numbers in the S/390 Hexadecimal Floating Point (HFP) format. Prior to this enhancement, SQL statements containing IEEE (Institute of Electrical and Electronics Engineers) Binary Floating Point (BFP) host variables receive an SQLCODE -20107 and the LOAD utility issues message `DSNT501I` with reason code `00E73005` because machine instructions must be available to DB2 to manipulate the variable.

DB2 toleration for BFP has been added. Applications written in ASM, C or C++ can now send and receive BFP data. The LOAD utility can also handle BFP numbers. DB2 continues to store its floating point numbers in HFP format but you can handle BFP numbers as follows

- To precompile/compile ASM, C or C++ programs that use BFP numbers, specify the `FLOAT(IEEE)` option. If the value of `HOST` is anything other than one of those three languages, DB2 ignores the value of `FLOAT`. DB2 will convert BFP numbers to HFP numbers before storing the data. `FLOAT(S390)` is the default. DB2 will not check that the host variable definition matches the pre-compiler option you have chosen so you will have to make sure the option you choose is appropriate for the program you have written.
- The LOAD utility has the new option `FLOAT(IEEE)`. DB2 expects floating point numbers to be in BFP format and will convert them to HFP as the data is loaded. If a conversion error occurs the record will be discarded. You cannot use the `FORMAT` option with `FLOAT(IEEE)`. The default is `FLOAT(S390)`.
- BFP host variables are not supported in user defined functions or triggers.

APAR identifier

The APAR identifier is as follows:

PQ30062, PQ30063, PQ28623.

Controlling updates to partitioning key



Restriction on updating partitioning key columns lifted in V5

Performance overhead if row moves across partitions

- drains data and index partitions and all NPIS
- highly disruptive

New DSNZPARM PARTKEYU

- YES - Updates permitted without restriction
- SAME - Updates permitted as long as row is not relocated to another partition
- NO - Updates are not permitted

Recommendations

- use YES only where you are sure updates are very low volume and the drains will not disrupt other transactions
- Use SAME if you have applications that need to update the key - rows cannot move across partitions
- Use NO to ensure possible failures are identified in testing

7.3 Controlling updates to partitioning key

The DB2 restriction on updating values in partitioning key columns was lifted in DB2 Version 5 with the functional enhancement introduced with maintenance (APAR PQ16946). In this refresh of Version 6, additional control over use of this facility has been introduced.

If an update of a key column results in the movement of the row, say from partition 3 to partition 52 of a 64 partition table space, DB2 performs the following step:

- Drains data partitions 3 to 52 inclusive
- Drains partitioning key partitions 3 to 52 inclusive
- Drains all non-partitioning indexes
- Updates what is needed to move data from partition 3 to partition 52

Clearly, these drains are highly disruptive for anything other than occasional updates of partitioning keys and unsuitable for high volume transaction workloads that resulted in the frequent movement of data to a new partition. Please refer to the DB2 manuals or to *DB2 Server for OS/390 Version 5 Recent Enhancements*, SG24-5421 for a description of this function.

You now can have control, at subsystem level, as to whether you want to enable update of partitioning keys and, if so, what updates are permitted. The behavior of partitioned key updates is controlled by DSNZPARM parameter PARTKEYU in panel DSNTIP4. The table below shows the range of valid values of PARTKEYU and the effect of each.

PARTKEYU	Effect of setting
YES	Update of partitioned key columns permitted without restriction. DB2 will request drains necessary to complete the update. Applications will be completely unaware that the table is in a partitioned table space.
SAME	Update of partitioned key columns is permitted if and only if the update does not require the updated row to relocate to another partition. An attempt to relocate a row will result in SQLCODE -904 and message <code>DSNT501I</code> with unavailable resource <code>CODE</code> , resource type <code>X'3000'</code> and reason code <code>00C900C7</code> .
NO	Update of partitioned key columns is not permitted. Applications will receive SQLCODE -151.

We recommend that you do not use YES unless you can be certain that updates of partitioning keys will be very low volume within the subsystem and the drains will not impact the workload.

The decision between SAME and NO depends on your applications. NO is the most straightforward and will work the same as prior to V5. The first time a program containing an update to a partitioning key is bound in development, the issue will be raised by a bind failure. The programmer will then change the update to a delete followed by an insert.

When SAME is specified, the issue will not be raised until the first time an update causes a row to change partitions. Depending on the set up of test tables and data this may not be until production or stress testing when the program fails with a -904. This needs to be weighed up against the flexibility that the SAME option provides. If you know your application will not update values causing the row to change partitions, the SAME option could be useful. It may mean that you can avoid changes to an application ported from another DBMS.

Please note that if you revert to disabling updates to partitioning keys, you need to check that there have been no programs implemented since V5 which update them. Otherwise they will fail with the resource unavailable SQLCODE.

APAR identifier

The APAR identifier is as follows:

PQ22653.

Additional enhancements



Toleration of separator differences

- DB2 accepts decimal point separator in dynamic SQL

New LANGUAGE bind option

- enables remote binds of COBOL programs containing hyphens

New operator for NOT

- ! has been added to ¬ and <> as symbols for NOT

DBPROTCL default change

- changed from DRDA to PRIVATE

Instrumentation enhancements

- additional dynamic statement cache statistics
- utility sub task count information

7.4 Toleration of separator differences

Support has been provided to allow a DB2 for OS/390 system which has been installed with comma as the default separator, to allow dynamic SQL which uses period as its separator.

APAR identifier

The APAR identifier is as follows:

PQ32872.

7.5 New LANGUAGE bind option

A new DRDA generic bind option called LANGUAGE has been introduced to help in the conversion of private protocol programs to DRDA access. It will enable you to remotely bind COBOL programs that use hyphens in cursor names, for example CURSOR-1.

Valid values for the new option are COBOL, COBOL2 and IBMCOBOL. The DRDA requester passes this information to the server at bind time so that it recognizes this value as valid.

APAR identifier

The APAR identifier is as follows:

PQ31209 and PQ29507.

7.6 New operator for NOT

Three new operators !=, !> and !< have been added to the DB2 syntax for NOT EQUAL, NOT GREATER THAN, and NOT LESS THAN respectively. These have been added for compatibility with other databases.

This supplements the existing not operators which are ~ , <> and NOT.

APAR identifier

The APAR identifier is as follows:

PQ27558.

Consideration for triggers

All triggers created before applying this APAR with ! used as the terminator in the trigger body will have to be dropped and recreated.

7.7 DBPROTCL default change

Please note that the default for the DBPROTCL, a new V6 DSNZPARM, has changed from DRDA to PRIVATE. This parameter was introduced as part of the DRDA support for three part name statements and provides a default for when the protocol is not specified at bind time. This change is to prevent customers inadvertently changing the remote access method specified by a package during migration. The previous default may also have led to packages becoming frozen during coexistence in a data sharing environment.

APAR identifier

The APAR identifier is as follows:

PQ25966.

7.8 Instrumentation enhancements

Dynamic statement cache statistics

IFCID 316 provides information and statistics for every statement in the cache. It has been enhanced to collect the following extra information which could be useful during tuning:

- Accumulated CPU and wait time values
- Indication of when the statistics interval began
- Indication that a RID list failure (statement reverted to table space scan) occurred

Detailed layout information can be found in SDSNSAMP(DSNDQW04)

APAR identifier

The APAR identifier is as follows:

PQ25652.

Instrumentation of utility sub task counts

The utility IFCID records QW0023 (utility start), QW0024 (utility phase change) and QW0025 (utility end) are modified to include sub task count information.

For IFCID 23, issued at start of utility, the QW0023R1 field will contain 0, or, for utilities that support a keyword to allow the user to control the number of sub tasks started (COPY and RECOVER), it will contain the number of sub tasks requested (for COPY and RECOVER this is twice the value specified by the PARALLEL keyword).

For IFCID 24, issued at utility phase changes, the QW0024R1 field will contain a count of all sub tasks started at the time the IFCID 24 is issued. This is a cumulative count.

For IFCID 25, issued at utility end, the QW0025R1 field will contain the total count of all sub tasks started during the utility execution.

APAR identifier

The APAR identifier is as follows:

PQ29243.

Appendix A. DB2 APARs cross references

In this appendix we provide two tables that represent a reference summary of maintenance that has been made available since GA in two areas: functional enhancements (described in the redbook) and performance related fixes (mostly regression avoidance). For details on the PTFs that comprise the RML, please refer to RETAIN, Information APAR II12343. Also always check RETAIN for the latest information on pre- or post-requisites: the following list, even if it has been carefully compiled, will not be accurate by the time you read it.

A.1 Functional enhancements

This table shows the APARs and PTFs for the enhancements that we have described in the redbook.

Function	APARs	PTFs	Access ability	Avail- ability	Exten- sibility	Manag eability	Perfor mance	Scala- bility
Cancel local threads faster outside DB2	PQ34465 PQ34466 PQ36702	UQ40219 UQ40220 UQ41780		X		X		
Catmaint - made optional	PQ30313	UQ34284		X		X		
Checkpage for copy	PQ25084	UQ38365		X		X		
Columns in order by not in select	PQ23778	UQ31707			X			
Data sharing - CF sync requests	PQ28722 PQ34764	UQ34769 UQ40107					X	
Data sharing - insert performance	PQ22910	UQ40244					X	
Data sharing - name class queue support	PQ23043 PQ32199 PQ35919	UQ35798 UQ38517 UQ40956					X	
Data sharing - new IMMEDWRITE option	PQ25337	UQ38882			X		X	
Data sharing -physical close rule	PQ27637	UQ32363				X	X	
DBPROTOCOL BIND default change	PQ25966	UQ34216				X		
DDF Suspend/Resume for DDL	PQ27123 PQ32920	UQ33969 UQ37904	X	X				
Decimal point enhancement	PQ32872	UQ38554			X			
Declared temporary tables	PQ32670 PQ35416	UQ39712 UQ40368			X			

Function	APARs	PTFs	Access ability	Avail- ability	Exten- sibility	Manag eability	Perfor mance	Scala- bility
Defer defining data sets	PQ30999 PQ34029 PQ34386 PQ34592 PQ34030	UQ38326 UQ41817 UQ40812 UQ39738 UQ40811				X		
Dynamic statement cache statistics	PQ25652	UQ39893				X	X	
EDM pool - new ZPARM	PQ31969	UQ40687					X	
External savepoints	PQ30439	UQ35648			X			
Extra block query DRDA	PQ30947	UQ36328						X
Faster data sharing member shutdown	PQ29907 PQ35845	UQ39752 UQ41226		X				
Global transaction support	PQ28487 PQ28611	UQ34479 UQ34478	X					
Identity column support	PQ30652 PQ30684 PQ36328 PQ36452	UQ38405 UQ36940 UQ43454 UQ42445			X			
IEEE float toleration	PQ30062 PQ30063 PQ28623	UQ34602 UQ34603 UQ33662			X			
IFI consolidation for data sharing	PQ29031 PQ25094	UQ35451 UQ40638				X		
Java stored procedures	PQ36011 PQ31846	UQ41672			X			
Langage bind option	PQ31209 PQ29507	UQ36980 UQ35735			X			
Log suspend/resume	PQ31492	UQ36695		X				
Logging - new write statistics	PQ28857	UQ33397				X		
Logging - read performance	PQ25745	UQ29949					X	
Logging - write performance	PQ30461	UQ34847					X	
Operators for not equal	PQ27558	UQ34625			X			
Parallelism - subtask count statistics	PQ29243	UQ33565					X	
Query parallelism ZPARM controller	PQ28414	UQ33443					X	
REXX stored procedures	PQ30219 PQ33133	UQ35973 UQ38510			X			
RML indicator	PQ34870	UQ41223				X		

Function	APARs	PTFs	Access ability	Avail- ability	Exten- sibility	Manag eability	Perfor mance	Scala- bility
Runstats - collection of uniform statistics	PQ21014	UQ28842					X	
Runstats - extra space statistics	PQ25091	UQ30738					X	
Volatile tables to use indexes	PQ33429	UQ38867				X		X
SQL procedures support	PQ29782 PQ30467 PQ30492 PQ33026 PQ33560 PQ24199	UQ34840 UQ34841 UQ39824 UQ40331 UQ40332 UQ38439			X			
SQLJ/JDBC driver	PQ36011	UQ41672					X	X
Star join support	PQ28813 PQ33666 PQ31326 PQ36206	UQ33085 UQ40281 UQ39733 UQ42008					X	X
Static SQL perf ZPARM	PQ25135	UQ33757					X	
Trigger performance	PQ34506	UQ40181					X	
Timestamp functions	PQ33564	UQ39444			X			
Unicode toleration	PQ32782 PQ32803	UQ37508 UQ37509			X			
Update partitioning keys ZPARM	PQ22653	UQ34417				X	X	
Update with subselect	PQ30383 PQ31272	UQ37262 UQ37245			X			

A.2 Performance related maintenance

The following list of APAR/PTFs is related to maintenance of interest for performance functions.

APAR	PTF	Description
PQ34126	UQ38884	Performance improvement for set function table space scan path.
PQ32018	UQ37373	Excessive synchronous read I/O for workfiles when prefetch disabled due to too many concurrent logical workfiles.
PQ25135	UQ33757	Static SQL queries involving host variables can see significant performance degradation when running with DEGREE(ANY) if DB2 decides to execute the query sequentially.
PQ22497	UQ27840	An inefficient access path may be selected by DB2 for queries that involve the materialization of a table expression, view or workfile.
PQ22381	UQ26354	Inefficient access path selected for a query with a LIKE predicate on a VARCHAR column.
PQ24146	UQ31786	Enable parallelism for non-partitioning index (NPI) access to the first base table in a parallel group.
PQ29601	UQ34528	V6 RUNSTATS TABLESPACE utility shows an increase in CPU time usage when compared to V5.
PQ31735	UQ36271	Performance degradation during insert on large segmented tables.
PQ27838	UQ31619	Performance of index screening may be degraded when most of the keys do not qualify during the scan.
PQ27500	UQ34601	Performance improvement for the CALL statement USING DESCRIPTOR.
PQ32690	UQ39727	ABEND04E RC00C90105 and excess of getpages for the root index page uring an SQL update statement after a ALTER VARCHAR was done on a partition table space.
PQ35329	UQ41099	Performance degradation of the CALL statement when the stored procedure name is specified in a host variable.
PQ29600 PQ32937	UQ35205 UQ38019	Online Reorg fixes for reducing data unavailability for applications during the end of the LOG phase (UTRO) and the beginning of the SWITCH phase (UTUT).

Appendix B. Sample external user defined function

This sample external UDF converts a smallint into a variable length character.

Create function statement

```
CREATE FUNCTION CHAR_N (SMALLINT )
RETURNS VARCHAR(32)
SPECIFIC CHAR_N_SI
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME CHARNSI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 5
STAY RESIDENT YES
PROGRAM TYPE SUB
WLM ENVIRONMENT DB2YENV1
EXTERNAL SECURITY DB2
NO DBINFO;
C program listing
/*****
* Module name = CHARNSI
*
* DESCRIPTIVE NAME = Convert small integer number to a string
*
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5645-DB2
* (C) COPYRIGHT 1999 IBM CORP. ALL RIGHTS RESERVED.
*
* STATUS = VERSION 6
*
*
* Example invocations:
* 1) EXEC SQL SET :String = CHARN(number) ;
* ==> converts the small integer number to a string
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or higher
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: CHARNSI
* Purpose: See Function
* Linkage: DB2SQL
* Invoked via SQL UDF call
*****/
```

```

*
*
*      Input: Parameters explicitly passed to this function:
*      - *number      : a pointer to a small integer number
*                      to convert to a string
*
*      Output: Parameters explicitly passed by this function:
*      - *numString    : pointer to a char[32], null-termin-
*                      ated string to receive the reformatted
*                      number.
*      - *nullNumString : pointer to a short integer to receive
*                      the null indicator variable for *numString.
*      - *sqlstate     : pointer to a char[6], null-terminated
*                      string to receive the SQLSTATE
*      - *message      : pointer to a char[70], null-terminated
*                      string to receive a diagnostic message if
*                      one is generated by this function.
*
*
* Normal Exit: Return Code: SQLSTATE = 00000
*      - Message: none
*
*
* Error Exit: None
*
*
*      External References:
*      - Routines/Services: None
*      - Data areas       : None
*      - Control blocks    : None
*
*
* Pseudocode:
*      CHARNSI:
*      1) If input number is NULL, then return NULL, exit
*      2) Translate the small integer number to a string
*      3) Return output string
*
*
* *****/
*/
* Module name = CHARNSI
*
* DESCRIPTIVE NAME = Convert small integer number to a string
*
* *****/
*/
* *****/ C library definitions *****/
#pragma linkage(CHARNSI, fetchable)
#pragma runopts(POSIX(ON))
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <decimal.h>
#include <ctype.h>
*/
* *****/ Equates *****/
#define NULLCHAR '\0'
*/
* *****/ GREATN functions *****/
void CHARNSI
/* main routine
(

```

```

short int    *p1In,                /* First parameter address */
char         *pOut,                /* Output address          */
short int    *null1In,            /* in: indic var for null1In */
short int    *nullpOut,           /* out: indic var for pOut   */
char         *sqlstate,           /* out: SQLSTATE           */
char         *fnName,             /* in: family name of function*/
char         *specificName,       /* in: specific name of func */
char         *message             /* out: diagnostic message  */
);
/*****/
/*****/ main routine *****/
/*****/
void CHARNSI                /* main routine            */
(
short int    *p1In,                /* in: timestp1            */
char         *pOut,                /* out: timestp            */
short int    *null1In,            /* in: indic var for null1In */
short int    *nullpOut,           /* out: indic var for pOut   */
char         *sqlstate,           /* out: SQLSTATE           */
char         *fnName,             /* in: family name of function*/
char         *specificName,       /* in: specific name of func */
char         *message             /* out: diagnostic message  */
)
{
#define DEF_OUTPUT_LENGTH 32

/*****/ Local variables *****/
char         strect??( 100 ??); /* string reciever */
/*****/
* Initialize SQLSTATE to 00000 and MESSAGE to "" *
*****/
message[0] = NULLCHAR;
*nullpOut = 0; /* -1 if Null value returned */
memcpy( sqlstate,"00000",6 );
memset( pOut, NULLCHAR, DEF_OUTPUT_LENGTH);

/*****/
* Return NULL if at least one input parameter is NULL *
*****/
if (*null1In != 0)
{
*nullpOut = -1;
return;
}

/*****/
* Convert an integer to a string *
*****/

sprintf( pOut, "%-d", *p1In);

return;
} /* end of CHARNSI */

```

Appendix C. DB2 and REXX

C.1 Sample main program REXX code

```
/* rexx */
/* trace i */                               /* uncomment to activate REXX tracing */

/* Sample REXX program that interrogates PLAN_TABLE and SYSIBM.SYSTABLES
   to add to an exception table table space scans of greater than a
   specified number of pages

Input parameters ssid = DB2 subsystem to connect to
               creator = PLAN_TABLE owner
               NPAGES = number of pages we are prepared to
                       tolerate scanning

Output is to one of two tables
A) for table space scans of greater than NPAGES
   Rows found are written to exception table with definition as
   follows
CREATE TABLE EXCEPTION
(
    QUERYNO                INTEGER    NOT NULL,
    BIND_TIME              TIMESTAMP NOT NULL,
    APPLNAME               CHAR(8)    NOT NULL WITH DEFAULT,
    PROGNAME               CHAR(8)    NOT NULL WITH DEFAULT,
    TB_CREATOR             CHAR(8)    NOT NULL,
    TB_NAME                CHAR(18)   NOT NULL,
    TB_CARDINALITY         INTEGER    NOT NULL,
    TB_NPAGES              INTEGER    NOT NULL,
    DATETIME_DETECTED      TIMESTAMP NOT NULL WITH DEFAULT,
    CREATED_BY             CHAR(8)    NOT NULL WITH DEFAULT
    CURRENT SQLID)
IN <DBNAME>.<TSNAME>
;
B) for those scans for which there are no RUNSTATS
   rows are written to a control table which specifies the RUNSTATS
   statement required to collect STATS.
create table runstats
(
    control_stmt  varchar(254) not null,
    datetime     timestamp not null with default,
    created_by   char(8) not null with default
    current sqlid)
in <dbname>.<tsname>
;
*/

PARSE ARG ssid creator npages                /* read arguments */

/* check values of supplied arguments: supply defaults if none passed */
/* this parameter checking is rudimentary, designed to show you how to
   handle host variables in REXX procedures and REXX code when talking
   to DB2 than do anything too serious */

IF ssid = '' THEN ssid = "DB2Y";             /* our default subsystem */

IF creator = '' THEN creator = 'PAOLOR8';     /* this parameter represents
```

```

the owner of the plan
table we are using to
interrogate */

/* integer host variable */
/* when interrogate catalog
what's the maximum no.
of pages we are prepared
to table space scan */

IF npages = '' THEN NPAGES = -20;

CALL set_sqlstmt /* call REXX subroutine */
/* to build SQL statement */

/* OK, now ready to do the DB2 stuff */
SUBCOM DSNREXX /* set up host environment */
IF RC THEN /* is host command there */
    x = RXSUBCOM('ADD','DSNREXX','DSNREXX') /* no: so create it */

ADDRESS DSNREXX 'CONNECT' ssid /* connect to DB2 */
sqlcall="Called from MAIN: Connect to DB2" /* what are re trying to do*/
rc=check_sqlcode(SQLCODE); /* check SQL return code */

/* this routine accesses the catalog for which we are content to use
uncommitted read. Set the isolation level as required */
ADDRESS DSNREXX "EXECSQL SET CURRENT PACKAGESET='DSNREXUR'"
sqlcall="Called from MAIN: Set packageset" /* diagnostics */
rc=check_sqlcode(SQLCODE); /* check SQL return code */

/* first declare that we are to use a cursor. The choice of the cursor
name is NOT random. You MUST use predefined cursor and statement
names. See REXX Language Support manual */

ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
sqlcall="Called from MAIN: Declare C1 for S1"
rc=check_sqlcode(SQLCODE);

/* prepare the SQL statement stored in the sql_stmt host variable.
The choice of S1 is not a random one. You MUST use predefined
cursor and statement names. See REXX Language Support (rexrv6.pdf)*/
ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :SQL_STMT"
sqlcall="Called from MAIN: Prepare S1 using "||SQL_STMT
rc=check_sqlcode(SQLCODE);

/* the host variable NPAGES has been set, so execute the prepared
statement substituting the value in NPAGES for the ? parameter
marker in the S1 statement referred to by cursor C1 */
ADDRESS DSNREXX "EXECSQL OPEN C1 USING :NPAGES"
sqlcall="Called from MAIN: Open S1 using "||NPAGES
rc=check_sqlcode(SQLCODE); /* check SQL return code */

DO UNTIL (SQLCODE = 100)
/* start fetching the data - note UPPER case variables critical */
sqlcall="Called from MAIN: Fetch C1"
ADDRESS DSNREXX "EXECSQL FETCH C1 INTO ",
    ":REASON, :QUERYNO, :BIND_TIME, :APPLICATION_NAME, ",
    ":PROGRAM_NAME, :TB_CREATOR, :TB_NAME, :CARD, :ACTUAL_PAGES,",
    ":DBNAME, :TSNAME"
IF SQLCODE < 0 THEN /* any SQL error - handle */

```

```

rc=check_sqlcode(SQLCODE);          /* check SQL return code */

/* the particular reason we've got rows back from the plan_table
and the catalog is either:
a) stats not collected (REASON='NO STATISTICS')
b) ts-scan of more than :NPAGES (REASON=' LARGE SCAN')
We're going to check the value of the reason and where
- no statistics, generate RUNSTATS statements
- LARGE SCAN put data into an exception table */

IF SQLCODE ^= 100 THEN DO;
  IF REASON = 'NO STATISTICS' THEN
    CALL generate_runstats

    IF REASON = ' LARGE SCAN' THEN
      CALL create_exceptions
END;

END;

EXIT rc;

/* put the SQL statement we are to execute into a REXX variable */
/* it uses a REXX variable "creator" and will use a host variable */
/* npages - defined as a parameter marker(?) in the SQL statement */
/* NOTE: the SQL_STMT variable name must be UPPER case */
set_sqlstmt:
SQL_STMT =,
"SELECT ' LARGE SCAN' ,      ",
"      PLN.QUERYNO ,      ",
"      PLN.BIND_TIME ,      ",
"      PLN.APPLNAME ,      ",
"      PLN.PROGNAME ,      ",
"      TB.CREATOR ,      ",
"      TB.NAME ,      ",
"      TB.CARD ,      ",
"      TB.NPAGES ,      ",
"      TB.DBNAME ,      ",
"      TB.TSNAME ,      ",
"FROM SYSIBM.SYSTABLES TB,      /* creator - */
      creator".PLAN_TABLE PLN      /* this is the owner of the */
"WHERE PLN.CREATOR = TB.CREATOR      /* plan_table we set as a */
" AND PLN.TNAME = TB.NAME      /* REXX variable - passed as */
" AND PLN.ACCESTYPE='R'      /* parameter */
" AND TB.NPAGES > ?      /* NOTE: parameter marker */
"UNION ALL      ",
"SELECT 'NO STATISTICS' ,      ",
"      PLN.QUERYNO ,      ",
"      PLN.BIND_TIME ,      ",
"      PLN.APPLNAME ,      ",
"      PLN.PROGNAME ,      ",
"      TB.CREATOR ,      ",
"      TB.NAME ,      ",
"      TB.CARD ,      ",
"      TB.NPAGES ,      ",
"      TB.DBNAME ,      ",
"      TB.TSNAME ,      ",
"FROM SYSIBM.SYSTABLES TB,

```

```

        creator".PLAN_TABLE PLN          ",
"WHERE PLN.CREATOR = TB.CREATOR        ",
" AND PLN.TNAME      = TB.NAME          ",
" AND PLN.ACCESSTYPE='R'                ",
" AND TB.NPAGES < 0                     ";
RETURN;

/* rexx subroutine to generate the RUNSTATS statement required and
   then we'll insert this into a control table */
generate_runstats:
/* generate the control statement required */
CTL_STMT="RUNSTATS TABLESPACE " DBNAME"."TSNAME" TABLE("TB_CREATOR".",
        TB_NAME") COLUMN(ALL) INDEX(ALL) SHRLEVEL REFERENCE REPORT YES ",
        "UPDATE ALL"
/* build the SQL statement */
ISRT_STMT="INSERT INTO RUNSTATS (CONTROL_STMT) VALUES(?)"

/* now even though we are executing a singleton insert must declare
   a cursor - otherwise the prepare gives SQLCODE -504 */
sqlcall="Called from generate runstats: declare C4 for S4"
ADDRESS DSNREXX "EXECSQL DECLARE C4 CURSOR FOR S4"
rc=check_sqlcode(SQLCODE);                /* check SQL return code */

/* get DB2 ready to execute this SQL statement */
ADDRESS DSNREXX "EXECSQL PREPARE S4 FROM :ISRT_STMT"
sqlcall="Called from generate runstats: prepare S4 from "||ISRT_STMT
rc=check_sqlcode(SQLCODE);                /* check SQL return code */

/* and go using the CTL_STMT host variable set above */
ADDRESS DSNREXX "EXECSQL EXECUTE S4 USING :CTL_STMT"
sqlcall="Called from generate runstats: execute S4 using "||CTL_STMT
IF SQLCODE < 0 THEN
    rc=check_sqlcode(SQLCODE);                /* check SQL return code */

RETURN;

create_exceptions:
/* generate the required insert statement */
ISRT_STMT="INSERT INTO PAOLOR8.EXCEPTION ",
        "(QUERYNO, BIND_TIME, APPLNAME, PROGNAME, TB_CREATOR,",
        "TB_NAME, TB_CARDINALITY, TB_NPAGES)",
        " VALUES (?,?,?,?,?,?,?,?)"

/* declare a cursor for this */
sqlcall="Called from generate runstats: declare C3 for S3"
ADDRESS DSNREXX "EXECSQL DECLARE C3 CURSOR FOR S3"
rc=check_sqlcode(SQLCODE);                /* check SQL return code */

/* get DB2 ready to execute this SQL statement */
ADDRESS DSNREXX "EXECSQL PREPARE S3 FROM :ISRT_STMT"
sqlcall="Called from generate runstats: prepare S3 from "||ISRT_STMT
rc=check_sqlcode(SQLCODE);                /* check SQL return code */

/* and go using the host variables set above */
ADDRESS DSNREXX "EXECSQL EXECUTE S3 USING ",
        ":QUERYNO, :BIND_TIME, :APPLICATION_NAME, ",
        ":PROGRAM_NAME, :TB_CREATOR, :TB_NAME, :CARD, :ACTUAL_PAGES,"
sqlcall="Called from generate excption: execute S3 using host variables"

```

```

IF SQLCODE < 0 THEN
    rc=check_sqlcode(SQLCODE);          /* check SQL return code */
RETURN;

/* check value of SQLCODE. If 0, fine just leave. Otherwise display
the error messages, issue a ROLLBACK and exit with the bad code */
check_sqlcode:
IF SQLCODE = 0 THEN RETURN 0;
SAY "Error detected at " sqlcall
SAY "SQLCODE = "SQLCODE
SAY "RETCODE = "RETCODE
SAY "SQLSTATE = "SQLSTATE
SAY "SQLERRMC = "SQLERRMC
SAY "SQLERRP = "SQLERRP
SAY "SQLERRD ="SQLERRD.1',' ,
                ||SQLERRD.2',' ,
                ||SQLERRD.3',' ,
                ||SQLERRD.4',' ,
                ||SQLERRD.5',' ,
                ||SQLERRD.6

SAY "SQLWARN ="SQLWARN.0',' ,
                ||SQLWARN.1',' ,
                ||SQLWARN.2',' ,
                ||SQLWARN.3',' ,
                ||SQLWARN.4',' ,
                ||SQLWARN.5',' ,
                ||SQLWARN.6',' ,
                ||SQLWARN.7',' ,
                ||SQLWARN.8',' ,
                ||SQLWARN.9',' ,
                ||SQLWARN.10

ADDRESS DSNREXX "EXEC SQL ROLLBACK"
exit SQLCODE

```

C.2 JCL to invoke REXX main program

```

/*
/* EMPTY TABLES BEFORE WE START. THIS PROVES REXX CODE IS
/* POPULATING THEM WITH DATA
/*
//S0010 EXEC PGM=IKJEFT01
//STEPLIB DD DSN=DSN610.SDSNEXIT,DISP=SHR
//          DD DSN=DSN610.SDSNLOAD,DISP=SHR
//          DD DSN=DB2V610Y.RUNLIB.LOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DB2Y)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP61)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
DELETE FROM RUNSTATS;
DELETE FROM EXCPTION;
/*

```

```

/* RUN REXX PROGRAM - NOTE SDSNLOAD IS IN STEPLIB
/*
//S0020 EXEC PGM=IKJEFT1B,PARM='%DB2REXXM'
//STEPLIB DD DSN=DSN610.SDSNLOAD,DISP=SHR << NEED THIS FOR DSNREXX
//SYSEXEC DD DSN=PAOLOR8.DB2.EXEC,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD DUMMY
/*
/* SELECT FROM TABLES
/*
//S0030 EXEC PGM=IKJEFT01
//STEPLIB DD DSN=DSN610.SDSNEXIT,DISP=SHR
// DD DSN=DSN610.SDSNLOAD,DISP=SHR
// DD DSN=DB2V610Y.RUNLIB.LOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DB2Y)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP61)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
SELECT * FROM RUNSTATS;
SELECT * FROM EXCPTION;

```

C.3 REXX stored procedure

```

/* rexx */

trace i

SAY 'hi there'

SUBCOM DSNREXX /* set up host environment */

IF RC THEN /* is host command there */

x = RXSUBCOM('ADD','DSNREXX','DSNREXX') /* no: so create it */

ISRT_STMT="INSERT INTO PAOLOR8.RUNSTATS (CONTROL_STMT) VALUES(?)"

ADDRESS DSNREXX "EXECSQL DECLARE C2 CURSOR FOR S2"

s="Called from DB2REXXR: declare C2 for S2"

rc=check_sqlcode(SQLCODE); /* bad - return to caller */

/* get DB2 ready to execute this SQL statement */

ADDRESS DSNREXX "EXECSQL PREPARE S2 FROM :ISRT_STMT"

s="Called from DB2REXXR: prepare S2 from "||ISRT_STMT

```

```

rc=check_sqlcode(SQLCODE);                      /* bad - return to caller */

/* and go using the CTL_STMT host variable set above */

CTL_STMT='what a fine day we are having 0001'

ADDRESS DSNREXX "EXECSQL EXECUTE S2 USING :CTL_STMT"

s="Called from DB2REXXR: execute S2 using "||CTL_STMT

rc=check_sqlcode(SQLCODE);                      /* bad - return to caller */

EXIT SQLCODE;

/* subroutine to check SQLCODE and if bad pass control to caller */

check_sqlcode:

    trace i

    IF SQLCODE = 0 THEN RETURN 0;                /* all ok then o back */

    SAY "Error detected at " s                    /* where did it go wrong */

    SAY "Passing control back to CALLER..."

    EXIT SQLCODE                                /* leave this stored proc*/

RETURN SQLCODE                                  /* don't forget */

```

C.4 Create procedure statement for REXX SP

```

CREATE PROCEDURE ADMF001.SPA4
( INOUT RETCODE INTEGER)
FENCED
RESULT SET 0
EXTERNAL NAME TESTREXX
LANGUAGE REXX
NOT DETERMINISTIC
MODIFIES SQL DATA
NO DBINFO
NO COLLID
WLM ENVIRONMENT WLMENV2
PARAMETER STYLE GENERAL WITH NULLS
ASUTIME NO LIMIT
STAY RESIDENT NO
PROGRAM TYPE MAIN
SECURITY DB2
COMMIT ON RETURN NO
RUN OPTIONS 'STACK(128K,128K,ANY,KEEP),ALL31(ON),HEAP(,,ANY)';

```

C.5 COBOL program calling REXX SP

```

IDENTIFICATION DIVISION.

```

```

PROGRAM-ID.    TESTPGM.

AUTHOR. Neil Toussaint.

DATE-WRITTEN.  4/11/2000.

*****

* REMARKS: CALLING STUB PROGRAM TO INVOKE REXX STORED PROCEDURE
*           ILLUSTRATIVE PURPOSES ONLY!!
*****

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.


EXEC SQL INCLUDE SQLCA END-EXEC.


01  WS-INT1                PICTURE S9(9) COMP VALUE +0.
01  WS-EYE-CATCHER        PICTURE X(60) VALUE SPACES.
01  WS-OUTPUT-SQLCODE.

03  WS-INT2                PICTURE  +(16) .


PROCEDURE DIVISION.

A100-START.


MOVE 'IN A100-START' TO WS-EYE-CATCHER.


EXEC SQL

    CALL ADMF001.SPA4 (:WS-INT1)

END-EXEC.


MOVE SQLCODE TO WS-INT2

DISPLAY 'SQLCODE=' WS-INT2

MOVE WS-INT1 TO WS-INT2

DISPLAY 'WS-INT1 (RETURN VALUE) =' WS-INT2

```



```
GOBACK.
```

```
EXIT.
```

The program was prepared using the sample JCL procedure DSNHICOB provided as part of the Installation Verification Procedure jobs and bound as a package. The COBOL program was executed under TSO as follows

```
DSN SYSTEM(DB2Y)
RUN  PROGRAM(TESTPGM) PLAN(TESTPGM)
```

C.6 WLM SP started task source JCL

```
//DB2YWLM  PROC RGN=0K,APPLENV=WLMENV2,DB2SSN=DB2Y,NUMTCB=8

//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//          PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB DD DISP=SHR,DSN=DB2V610Y.RUNLIB.LOAD
//          DD DISP=SHR,DSN=CEE.SCEERUN
//          DD DISP=SHR,DSN=DSN610.SDSNLOAD
//SYSEXEC DD DSN=DB2V610Y.SRCLIB.DATA,DISP=SHR
//DSSPRINT DD SYSOUT=A
//UTPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSPRINT DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUDUMP DD SYSOUT=A
//SYSTSPRT DD SYSOUT=A
//SYSOUT DD SYSOUT=A
```

Output from REXX SAY statements and trace diagnostics is routed to SYSTSPRINT. Sample output from DB2YWLM SYSTSPRT after running our our REXX SP follows.

```
3 *-* SAY 'hi there'
>L>  "hi there"
hi there
5 *-* SUBCOM DSNREXX
>L>  "SUBCOM"
>L>  "DSNREXX"
>O>  "SUBCOM DSNREXX"
+++ RC(1) +++
6 *-* IF RC
>V>  "1"
*-* THEN
7 *-* x = RXSUBCOM('ADD','DSNREXX','DSNREXX')
>L>  "ADD"
>L>  "DSNREXX"
>L>  "DSNREXX"
>F>  "0"
....
```

C.7 WLM configuration

```
Extract from couple data set
Definition data set . . . : none
Definition name . . . . . db2yrexex (Required)
Description . . . . . Stored procedures for DB2Y
Select one of the
```

```

following options. . . . . 9__ 1. Policies
2. Workloads
...
9. Application Environments
10. Scheduling Environments

```

Selecting option 9

9. Application Environments

```

Appl Environment Name . . WLMENV2
Description . . . . . Stored procedures for DB2
Subsystem type . . . . . DB2
Procedure name . . . . . DB2YWLM
Start parameters . . . . DB2SSN=DB2Y,NUMTCB=8,APPLENV='WLMENV2'
Limit on starting server address spaces for a subsystem instance:
No limit

```

Note:

And don't forget to activate the chosen service policy using Utilities from main WLM screen. Ours is called DAYTIME (8-8).

Having done that check SYSLOG for confirmation message

```
IWM001I WORKLOAD MANAGEMENT POLICY DAYTIME NOW IN EFFECT
```

C.8 Commands to manipulate WLM and SP

In this section we list the most common WLM commands and related output:

```
/d wlm
```

```

RESPONSE=SC63
IWM025I 01.38.15 WLM DISPLAY 135
ACTIVE WORKLOAD MANAGEMENT SERVICE POLICY NAME: DAYTIME
ACTIVATED: 2000/03/28 AT: 15:42:57 BY: PAOLOR8 FROM: SC63
DESCRIPTION: from 8 till 8
RELATED SERVICE DEFINITION NAME: db2yrex
INSTALLED: 2000/03/28 AT: 15:42:03 BY: PAOLOR8 FROM: SC63
WLM VERSION LEVEL: LEVEL008
WLM FUNCTIONALITY LEVEL: LEVEL003
WLM CDS FORMAT LEVEL: FORMAT 3
STRUCTURE SYSZWLM_WORKUNIT STATUS: DISCONNECTED

```

```
/d wlm,applenv=wlmenv2
```

```

RESPONSE=SC63
IWM029I 01.37.34 WLM DISPLAY 131
APPLICATION ENVIRONMENT NAME STATE STATE DATA
WLMENV2 AVAILABLE
ATTRIBUTES: PROC=DB2YWLM SUBSYSTEM TYPE: DB2

```

```
/=db2y dis proc(*.*)
```

```

DSNX940I =DB2Y DSNX9DIS DISPLAY PROCEDURE REPORT FOLLOWS -
----- SCHEMA=ADMF001

```

PROCEDURE	STATUS	ACTIVE	QUEUED	MAXQUE	TIMEOUT	WLM_ENV
SPA3	STARTED	0	0	0	0	WLMENV2
SPA4	STARTED	0	0	1	0	WLMENV2

----- SCHEMA=PAOLOR7

PROCEDURE	STATUS	ACTIVE	QUEUED	MAXQUE	TIMEOUT	WLM_ENV
TESTSPA3	STARTED	0	0	0	0	WLMENV2

----- SCHEMA=SYSPROC

PROCEDURE	STATUS	ACTIVE	QUEUED	MAXQUE	TIMEOUT	WLM_ENV
DSNACCMD	STARTED	0	0	0	0	
DSNWZP	STARTED	0	0	0	0	

DSNX9DIS DISPLAY PROCEDURE REPORT COMPLETE

DSN9022I =DB2Y DSNX9COM '-DISPLAY PROC' NORMAL COMPLETION

Appendix D. Special notices

This publication is intended to help system programmers, database administrators, and application developers in understanding, assessing, and utilizing the new functions of DB2 UDB Server for OS/390 Version 6. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 UDB Server for OS/390 Version 6. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 UDB Server for OS/390 Version 6 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

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, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating

environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

This document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain 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.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

IBM ®

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Appendix E. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 289.

- *How to Build Java Stored Procedures: DB2 UDB Gets Wired With SQLJ and JDBC*, SG24-5945
- *DB2 UDB for OS/390 Version 6 Performance Topics*, SG24-5351
- *DB2 UDB for OS/390 Version 6 Management Tools Package*, SG24-5759
- *DB2 Server for OS/390 Version 5 Recent Enhancements - Reference Guide*, SG24-5421
- *DB2 for OS/390 Capacity Planning*, SG24-2244
- *Getting Started with DB2 Stored Procedures: Give Them a Call through the Network*, SG24-4693-01
- *Developing Cross-Platform DB2 Stored Procedures: SQL Procedures and the DB2 Stored procedure Builder*, SG24-5485
- *The Integration of Java and DB2 UDB by Example*, SG24-5945
- *DB2 for OS/390 and Continuous Availability*, SG24-5486
- *Parallel Sysplex Configuration: Cookbook*, SG24-2076-00
- *DB2 for OS390 Application Design for High Performance*, GG24-2233
- *Using RVA and SnapShot for Business Intelligence Applications with OS/390 and DB2*, SG24-5333
- *IBM Enterprise Storage Server Performance Monitoring and Tuning Guide*, SG24-5656
- *Java Programming Guide for OS/390*, SG24-5619.

E.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

E.3 Other resources

These publications are also relevant as further information sources:

- *DB2 UDB for OS/390 Version 6 What's New*, GC26-9017-02
- *DB2 UDB for OS/390 Version 6 Application Programming and SQL Guide*, SC26-9004
- *DB2 UDB for OS/390 Version 6 SQL Reference*, SC26-9014-01
- *DB2 UDB for OS/390 Version 6 Administration Guide*, SC26-9003-01
- *DB2 UDB for OS/390 Version 6 Installation Guide*, SC26-9008-01
- *DB2 UDB for OS/390 Version 6 Command Reference*, SC26-9006-01
- *DB2 UDB for OS/390 Version 6 Utility Guide and Reference*, SC26-9015-01
- *DB2 UDB for OS/390 Version 6 ODBC Guide and Reference*, SC26-9005-01
- *DB2 UDB for OS/390 Version 6 Application Programming and SQL Guide*, SC26-9004-01
- *DB2 UDB for OS/390 Version 6 Messages and Codes*, SC26-9011-01
- *DB2 UDB Version 6 Application Development Guide*, SC09-2845
- *OS/390 V2R8.0 MVS Planning: Workload Management*, GC28-1761-09
- *OS/390 V2R8 MVS Programming: Resource Recovery*, GC28-1739-05
- *DB2 Performance Monitor for OS/390 Version 6 Using the Workstation On-line Monitor*, SC26-9170
- *DB2 Performance Monitor for OS/390 Version 6 Installation and Customization*, SC26-9171
- *DB2 UDB Installation and Configuration Supplement V6*, GC09-2857
- *Java Application Programming Guide and Reference for Java*, SC26-9018-01
- *OS/390 V2R8.0 MVS Programming: Assembler Services Guide*, GC28-1762-06
- *OS/390 V2R9 C/C++ Programming Guide*, SC09-2362-05
- *Understanding SQL's Stored Procedures: A Complete Guide to SQL/PSM*, Jim Melton, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-461-8

E.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- <http://www.ibm.com/s390/corner/> Java on S/390
- <http://www.software.ibm.com/data/db2/> DB2 Family
- <http://www.software.ibm.com/data/db2/os390/> DB2 for OS/390
- <http://www.ibm.com/software/db2os390/downloads.html> DB2 downloads

- <http://www.ibm.com/solutions/businessintelligence/teraplex/index.htm> Teraplex Center

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

[illegible]

First name	Last name
------------	-----------

Company

Address _____

City	Postal code	Country
------	-------------	---------

Telephone number	Telefax number	VAT number
------------------	----------------	------------

☐ Invoice to customer number☐ Credit card number

Credit card expiration date	Card issued to	Signature
-----------------------------	----------------	-----------

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Abbreviations and acronyms

AIX	Advanced Interactive eXecutive from IBM	EBCDIC	extended binary coded decimal interchange code
APAR	authorized program analysis report	ECS	enhanced catalog sharing
ARM	automatic restart manager	ECSA	extended common storage area
ASCII	American National Standard Code for Information Interchange	EDM	environment descriptor management
BLOB	binary large objects	ERP	enterprise resource planning
CCSID	coded character set identifier	ESA	Enterprise Systems Architecture
CCA	client configuration assistant	FDT	functional track directory
CFCC	coupling facility control code	FTP	File Transfer Program
CTT	created temporary table	GB	gigabyte (1,073,741,824 bytes)
CEC	central electronics complex	GBP	group buffer pool
CD	compact disk	GRS	global resource serialization
CF	coupling facility	GUI	graphical user interface
CFRM	coupling facility resource management	HPJ	high performance Java
CLI	call level interface	IBM	International Business Machines Corporation
CLP	command line processor	ICF	integrated catalog facility
CPU	central processing unit	ICF	integrated coupling facility
CSA	common storage area	ICMF	internal coupling migration facility
DASD	direct access storage device	IFCID	instrumentation facility component identifier
DB2 PM	DB2 performance monitor	IFI	instrumentation facility interface
DBAT	database access thread	IRLM	internal resource lock manager
DBD	database descriptor	ISPF	interactive system productivity facility
DBID	database identifier	ISV	independent software vendor
DBRM	database request module	I/O	input/output
DCL	data control language	ITSO	International Technical Support Organization
DDCS	distributed database connection services	IVP	installation verification process
DDF	distributed data facility	JDBC	Java Database Connectivity
DDL	data definition language	JFS	journaled file systems
DLL	dynamic load library manipulation language	JVM	Java Virtual Machine
DML	data manipulation language	KB	kilobyte (1,024 bytes)
DNS	domain name server	LPAR	logically partitioned mode
DRDA	distributed relational database architecture	LOB	large object
DTT	declared temporary tables		
EA	extended addressability		

LPL	logical page list
LRSN	log record sequence number
LSFM	xxxxxxx
LVM	logical volume manager
MB	megabyte (1,048,576 bytes)
OBD	object descriptor in DBD
ODBC	Open Data Base Connectivity
OS/390	Operating System/390
PAV	parallel access volume
PDS	partitioned data set
PSID	pageset identifier
PSP	preventive service planning
PTF	program temporary fix
PUNC	possibly uncommitted
QMF	Query Management Facility
RACF	Resource Access Control Facility
RBA	relative byte address
RID	record identifier
RRS	resource recovery services
RRSAF	resource recovery services attach facility
RS	read stability
RR	repeatable read
SDK	software developers kit
SMIT	System Management Interface Tool
SP	stored procedure
SRB	system resource block
TCB	task control block
WLM	workload manager

Index

Numerics

-1 171
15 tables in a join 208
225 222
25 222
9 222

A

additional space statistics 199
APARs cross reference 261
AS IDENTITY 56
ATOMIC 131
ATTN 167
auxiliary table 40

B

BACKODUR 164
backup procedure 159
bind time 208
built-in function exploitation 21
built-in function or application program 16
built-in function or UDF 17
built-in functions 14

C

CALL 132
cancel thread 185
CANCEL(n) 180
cardinality 223
CASE 132
CAST function 24
CASTOUT(NO) 188
 considerations 189
 performance 189
check_sqlcode 119
CHECKPAGE 196
 message 197
 recommendations 198
 scope 196
 why using it 197
CICS 61, 95, 143
CLASSPATH 145
COBOL program 123
colon 12
columns in order by 99
combining SQLJ and JDBC 145
compound statement
 example 132
compound statements 131
connection pooling 53
CONTINUE 134
COPY 196
CREATE PROCEDURE 138
CREATE TRIGGER 27
creator 121

D

data warehouse 203
DB2 PM 3, 192
DB2 REXX support 116
DDF 95
DDF command options 183
DDF suspend 177
DDL operation 177
defer defining data sets 170
 benefits 173
 performance 172
 restrictions 174
DEFINE NO 170
 operational issues 175
 performance 172
 restrictions 174
denormalized 204
density 223
DFDSS DUMP 159
dimension table 205
distributed functions
 performance 53
DSMAX 248
DSN1COPY 174, 196
DSN1PRNT 174
DSN8WLMP 140
DSNELI 123
DSNHSQL 141
DSNREXCS 120
DSNREXX 118
DSNREXX CONNECT 122
DSNTEJ63 141
DSNTIJSQ 141
DSNTIP4 241
DSNTIPR 178
DSNTPSM 139
DSNTPSMP 137
DSNZPARM 241

E

EDM pool 194
EDMBFIT 194
ELSE 132
ESS 156
 performance 49
 read performance 50
ET/390 151
EXIT 135
EXPLAIN 33
Extra Query Block support 54

F

fact table 205, 219
faster cancel thread 184
faster data sharing member shutdown 188
FlashCopy 156

- flat file 204
- functional enhancement
 - areas 4
- functional enhancements
 - summary 4

G

- GENERATED 77
- GET DIAGNOSTICS 133
- global transactions 100
 - considerations 112
 - flow between participants 110
 - passing the XID 108
 - problem solved 106
 - sample scenario 101
 - XID token 107
- GOTO 133
- group scope 193
- GRS 48

H

- High Performance Java 143
- host variables preceded with a colon 13
- HPGRBRBA 165

I

- identity columns 56
 - advantages 77
 - availability 77
 - concurrency 77
 - recoverability 77
 - uniqueness 77
- IF 132
- IFI consolidation 192
- II12100 12
- INACTIVE THREADS 178
- INCLUDE SQLCA 119

J

- Java stored procedures 147
 - coding considerations 149
 - definition 148
 - preparing 151
 - restrictions 149
 - results set 150
 - running 153
 - using SQLJ 152
- Java Virtual Machine 143
- JAVAENV 153
- JDBC 142
- JSPDEBUG 153

L

- language support 115
- LBACKOUT 164
- LCASE(string) 15
- LEAVE 132

LOBs

- auxiliary table 40
- delete 46
- insert 45
- overview 40
- performance considerations 39
- processing 41
- read performance 43
- recommendations 47
- update 45
- write performance 45
- log I/O performance 243
- log instrumentation 245
- log rates 52, 247
- log read 244
- log write 245
- log write latch 246
- LOGONLY 165
- LOOP 132

M

- maximum number of tables in a join 222
- migraton 10
- missing colons
 - sample program 13

N

- next number 60
- non uniform statistics 199
- normalization 205
- normalizing data 47
- NPAGES 121, 239
- NPGTHRS 239
- NPI 264

O

- objective 1
- offsite recovery 156
- operational enhancements 155
- OW38843 113
- OW39220 113

P

- PARAMDEG 241
- PARSE ARG REXX 122
- PCLOSEN 165
- PCLOSET 165
- performance 201
- performance measurements
 - sources 7
- P-locks 189
- PQ17740 11
- PQ21014 199
- PQ23043 248, 249
- PQ23778 99
- PQ24199 141
- PQ25084 198
- PQ25091 199

PQ25094 193
 PQ25135 242
 PQ25745 244
 PQ26922 12
 PQ27022 113
 PQ27123 183
 PQ27461 113
 PQ28414 241
 PQ28487 113
 PQ28611 113
 PQ28813 207, 222, 238
 PQ28857 246
 PQ29031 193
 PQ29782 141
 PQ29907 189
 PQ30219 125
 PQ30383 98
 PQ30390 12
 PQ30439 81
 PQ30461 246
 PQ30467 141
 PQ30492 141
 PQ30652 11, 78
 PQ30684 11, 78
 PQ30999 176
 PQ31272 98
 PQ31326 222, 238
 PQ31492 169
 PQ31846 154
 PQ31969 195
 PQ32199 249
 PQ32387 113
 PQ32670 96
 PQ32782 253
 PQ33026 141
 PQ33028 98
 PQ33133 125
 PQ33429 240
 PQ33560 141
 PQ33666 238
 PQ34029 176
 PQ34030 176
 PQ34118 99
 PQ34199 11
 PQ34321 143
 PQ34386 176
 PQ34465 186
 PQ34466 186
 PQ34506 38
 PQ34592 176
 PQ34849 113
 PQ34972 198
 PQ35416 96
 PQ35845 189
 PQ35919 248, 249
 PQ36011 146, 154
 PQ36206 207, 209, 222, 238
 PQ36328 72, 78
 PQ36405 11
 PQ36452 78

PQ36702 186
 PQ36815 11
 PSM 126
 PUNC 162

Q

query parallelism 241

R

recoverability offsite 165
 recovery 164
 REDO logging 48
 REFERENCING clause 30
 refresh 2
 release incompatibilities checklist 11
 REPEAT 132
 RESIGNAL 132
 RESTP 164
 RETURN 132
 REXX and DB2
 sample program 269
 REXX and stored procedures 122
 REXX procedure 13
 REXX program
 coding conventions 120
 handling errors 118
 host environment 118
 isolation level 120
 RMF 167
 ROLLBACK 118
 ROW_COUNT 133
 ROWID 40, 78
 RRSF 100
 run time 208
 Runstats 199
 RVA 50, 156

S

SDSNLOAD 118
 SDSNSAMP(DSNTEPS) 199
 SET 132
 SET LOG RESUME
 actions and messages 161
 SET LOG SUSPEND
 actions and messages 160
 diagnose problems 166
 messages 161
 recommendations 162
 set log suspend
 operational considerations 166
 Shark 49
 SIGNAL 132
 SnapShot 156, 163
 snowflake schema 206
 SPACE column 171
 SQL stored procedure
 allowed control statements 132
 eye catcher 135

- SQL Stored Procedure language 126
- SQL stored procedures
 - considerations 129
 - creation 130
 - debugging 136
 - handling errors 134
 - preparing 137
 - reasons for using them 128
- SQL_STMT 120
- SQLCA 118
- sqlcall 119
- SQLCODE 118
- SQLJ 142
- SQLJ/JDBC driver
 - using it 144
- SQLJ/JDBC driver support 142
- ssid 119
- STAR 48
- star join 202
 - access path 213, 228
 - characteristics for good performance 227
 - efficient indexes 224
 - fact table index design 223
 - good environment 221
 - how it works 217
 - missing index column 230
 - missing key predicate 228
 - performance results 235
 - query parallelism 209
 - sample scenario 210
 - support 207
 - the 9 conditions 222
 - the conditions 219
 - visual explain 214, 232
- star schema 203
- static SQL 241, 243
- STOP DDF MODE(SUSPEND) 179
- STOP DDF MODE(SUSPEND) CANCEL(n) 182
- STOP DDF MODE(SUSPEND) WAIT(n) 181
- Stored Procedure Builder 127
- STOSPACE 171
- SUBSTR 43
- SUBSTR(LOB,1,200) 41
- suspend updates 156
- SYSIBM.SYSCOLDIST 199
- SYSIBM.SYSINDEXPART 170
- SYSIBM.SYSTABLEPART 170
- SYSIBM.SYSTABLESPACE 170

T

- TCP/IP 53
- team vii
- techniques to create new key 59
- Temporary Storage Queue 61
- TESTREXX 122
- THEN 132
- thread pooling 178
- thread termination processing 184
- time out 105
- timestamp 60

- transition tables 30, 31
- transition variables 29, 31
- trigger
 - coding considerations 33
- triggers
 - AFTER 27
 - BEFORE 27
 - coding considerations 35
 - overview 28
 - performance 37
 - performance considerations 26
 - row trigger 33
 - scope of a trigger 33
 - statement trigger 33
 - transition tables 30
 - transition variables 29
- type 1 driver 142
- type 2 driver 142

U

- UDF 25
 - based on built-in functions 21
 - efficiency 19
 - options 23
 - performance considerations 14
 - sourced 24
- UDF or built-in function 17
- UNDO 159
- UPDATE SET 97
- update with subselect 97
 - conditions 97
 - self referencing 98
- user defined function
 - sample 265

V

- VDWQ 247
- Volatile tables 239
- VSAM 170

W

- WAIT(n) 180
- WHENEVER 134
- WHILE 132
- WLM 19
- WLM commands 125
- WLM environment 124, 140
- WLM ENVIRONMENT parameter 123

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at <http://www.redbooks.ibm.com/>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number	SG24-6108-00
Redbook Title	DB2 UDB Server for OS/390 Version 6 Technical Update
Review	<div></div> <div></div> <div></div> <div></div> <div></div> <div></div>
What other subjects would you like to see IBM Redbooks address?	<div></div> <div></div> <div></div>
Please rate your overall satisfaction:	<input type="radio"/> Very Good <input type="radio"/> Good <input type="radio"/> Average <input type="radio"/> Poor
Please identify yourself as belonging to one of the following groups:	<input type="radio"/> Customer <input type="radio"/> Business Partner <input type="radio"/> Solution Developer <input type="radio"/> IBM, Lotus or Tivoli Employee <input type="radio"/> None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	<div></div> <div><input type="radio"/> Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.</div>
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. http://www.ibm.com/privacy/yourprivacy/

SG24-6108-00

Printed in the U.S.A.

