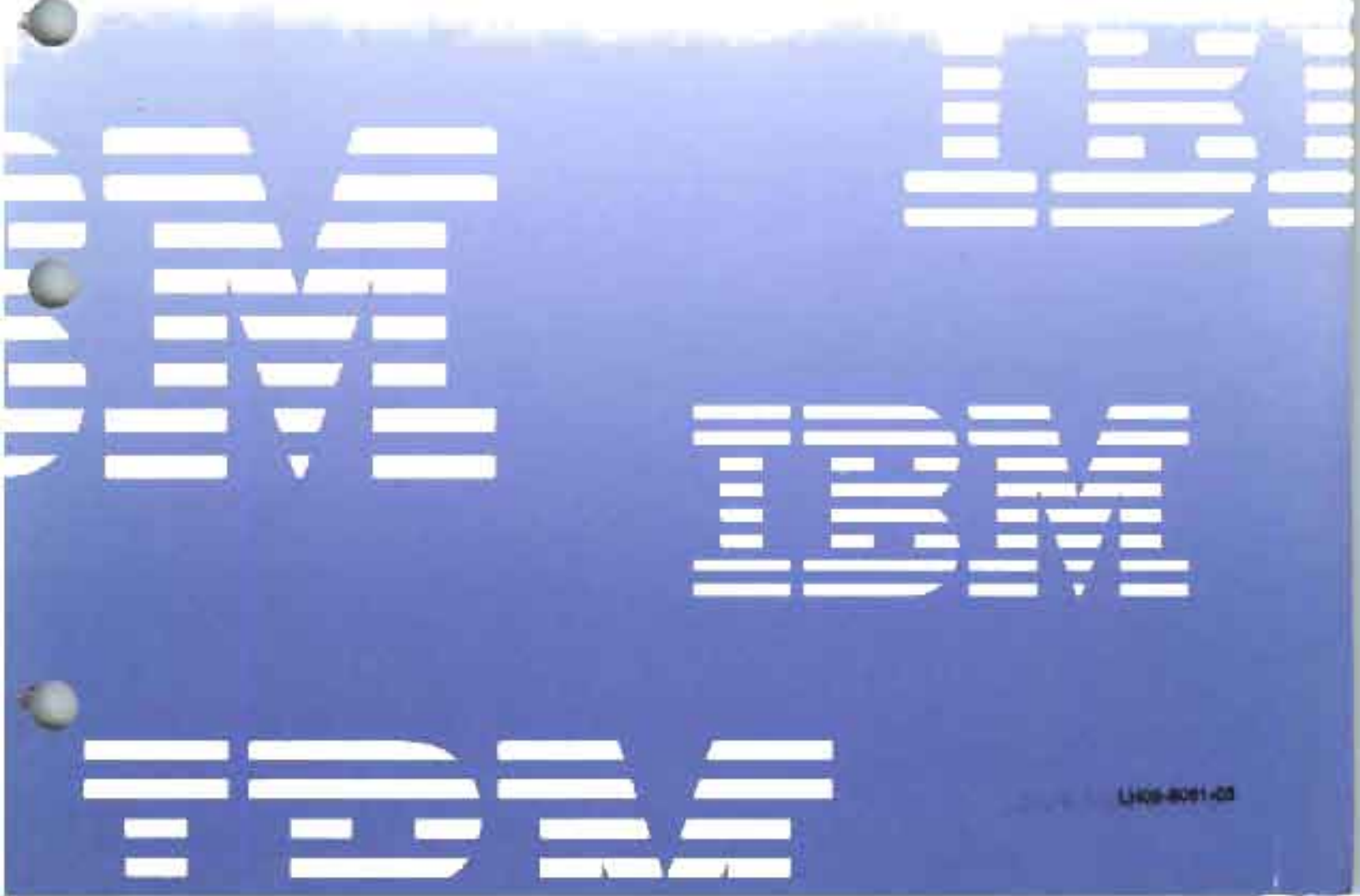


SOL/DS

**Diagnose Guide and Reference
for IBM VM Systems**

Version 3 Release 4







SQL/DS

LH09-8081-03

**Diagnosis Guide and Reference
for IBM VM Systems**

Version 3 Release 4

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page ix.

Fourth Edition (April 1993)

This edition applies to Version 3 Release 4, Modification Level 0, of the SQL/DS Program 5688-103 and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 Don Mills Road
North York, Ontario, Canada M3C 1V7

You can also send your comments by facsimile to (416) 448-6057 addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to torrcf@vnet.ibm.com; IBMLINK, to [toribm\(torrcf\)](mailto:toribm(torrcf)); IBM/PROFS, to [torolab4\(torrcf\)](mailto:torolab4(torrcf)); IBMMAIL, to [lbmmail\(calbmwt9\)](mailto:lbmmail(calbmwt9))

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

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

© Copyright International Business Machines Corporation 1987, 1993. 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.

IBM is a registered trademark of International Business Machines Corporation, Armonk, N.Y.

Contents

Notices	ix
Programming Interface Information	ix
Trademarks and Service Marks	x
About This Manual	xi
Purpose	xi
Audience	xi
Contents	xi
Components of the SQL/DS System	xii
Syntax Notation Conventions	xiv
Summary of Changes for SQL/DS Version 3 Release 4	xix
Enhancements, New Functions, and New Capabilities	xix
Support for the IBM DATABASE 2 AIX/6000 Database Manager	xix
Cascade Delete Enhancement for Referential Integrity	xix
Improved EXPLAIN Capabilities	xix
Host Structure Variables	xix
Removal of 512 Host Variable Restriction	xx
Enhanced SHOW STORAGE Command	xx
Usability Enhancements	xx
Package Dbspace Full Condition Handling	xx
The Connectable and Unconnected State	xx
Dual Logging Enhancement	xx
Reliability, Availability, and Serviceability Improvements	xx
Processing a DROP TABLE Statement	xx
Enhancement to COLDLOG Processing	xxi
Improved Storage Trace	xxi
Library Enhancements	xxi
SQL/DS Performance Tuning Handbook	xxi
Revised Manuals	xxi
Chapter 1. Introduction to Problem Diagnosis	1
Diagnosis Flowcharts	1
Chapter 2. SQL/DS Concepts	5
Introduction	6
SQL/DS Components in the Application Requester	7
Database Services Utility (DBSU)	7
Interactive SQL (ISQL)	7
Preprocessors (PREP)	7
VM Resource Adapter (VRA)	7
SQL/DS Components in the Application Requester and the Application Server	7
Data System Control (DSC)	7
Data Conversion (CONV)	8
Distributed Relational Resource Manager (DRRM)	8
SQL/DS Components in the Application Server	8
Work Unit Manager (WUM)	8
Relational Data System (RDS)	8
Database Storage Subsystem (DBSS)	9

The SQL/DS RDBMS in Single User Mode	10
The SQL/DS RDBMS in Multiple User Mode	10
When Using the SQL/DS-only Protocol	10
When Using the DRDA Protocol	11
Logical Unit of Work Concepts	11
Agent Handling Concepts	12
Agent Handling Functions	13
Allocating Users to Agent Structures	13
Dispatcher Components	14
Conceptual Overview of Prioritization Scheme	14
Conceptual Overview of the Fair Share Auditing Process	15
Finding "Deprived" Agents	15
Setting Fair Share Interval Size	16
Locating and Dispatching a Dispatchable Agent	16
Agent Processing at the End of an LUW	17
Communications Concepts	17
Concepts on the Application Requester	17
Concepts on the Application Server	18
Inter-User Communications Vehicle (IUCV) Protocol	18
Advanced Program-to-Program Communications/VM (APPC/VM) Protocol	20
Application Program Use of IUCV or APPC/VM	25
Package Management Concepts	25
RDIINs	26
Preprocessing	26
Execution-Time Processing	28
Package Cache Management	29
Repreprocessing	30
Authorization	30
Storage Management Concepts	31
Memory Management Concepts	31
Logical Storage Management Concepts	32
Physical Storage Management Concepts	36
Buffer Storage Management Concepts	39
Index Concepts	39
Basic Index Structure	39
Index Space Management	40
Invalid Indexes	41
Transient Indexes	41
Clustering Index	42
Clustered Indexes	42
Index Fragmentation	43
Sorting Concepts	44
Logging/Recovery Concepts	46
Locking Concepts	55
Specifying Isolation Levels	56
Locking Hierarchy	57
Lock Modes	58
Lock Durations	59
Lock Compatibility	59
Types of Internal Data Manipulation Calls	60
Locking for Different Internal DM Calls	60
Deadlock Detection	65
Escalation of Locks	65
Access to Private DBSPACES	66

Termination Concepts	66
Chapter 3. Reporting Defects	69
Developing the First (Two) Keyword(s)	70
Component Identification Keyword (PIDS)	70
Release Level Keyword (LVLS)	71
Developing the Remaining Keywords	72
Abnormal Termination	72
Message	74
First Failure Data Capture	76
No Response	80
Wait or Loop	80
Slow Response	81
Incorrect or Missing Output	81
Document	81
Additional Keywords	82
SQLCODES	82
SQL Statements	82
Start-up Parameters	82
Data Type	82
Application Type	83
EXECs	83
Application Program Generated SQLCODES	83
Invocation	83
Interactions	85
Reporting a Problem	85
Materials	85
Environments	87
Chapter 4. Functional Problems	95
System-Related Error Codes	95
SQL COMMAND FAILED (-901)	95
ROLLED BACK DUE TO A DEADLOCK (-911)	96
ROLLED BACK DUE TO EXCESSIVE (SYSTEM WIDE) LOCK REQUESTS (-912)	96
ROLLED BACK DUE TO EXCESSIVE LOCKS HELD FOR THIS LUW (-915)	96
Common User-related Error Codes	96
SQL COMMAND LIMITATION EXCEEDED (-101)	96
CREATOR.TABLE WAS NOT FOUND (-204)	97
INPUT VARIABLE DATA TYPE NOT COMPATIBLE WITH COLUMN (-301)	97
INPUT HOST VARIABLE TOO LARGE (-302)	98
AN INDICATOR VARIABLE IS MISSING (-305)	99
MISMATCH BETWEEN NUMBER OF HOST VARIABLES (-313)	99
Functional Deviations	99
Lockout with Cursor Stability	99
FETCH with Cursor Stability	99
Chapter 5. Diagnosing Performance Problems	101
Performance Analysis Glossaries	102
Glossary of Performance Index Headers	102
Glossary of Performance Indicator Terms	102
Glossary of Performance Terminology	103
Performance Problem Indexes	105
Application Function Indexes to Performance Problems	106

General Performance Problems	107
Data Authorization Performance Problems	108
Data Definition Performance Problems	108
Data Manipulation Performance Problems	109
Data Utilities Performance Problems	110
Recovery Control Performance Problems	111
Performance Problems by Performance Symptom	111
Agent Related Performance Problems	111
CPU Related Performance Problems	112
I/O Related Performance Problems	113
Locking Related Performance Problems	114
Storage Related Performance Problems	114
Special Case Performance Problems	115
Analysis of Performance Problems	115
Adjacent Key Locking in User Data	115
Agents Being Held	119
Bad Data Distribution	122
BLOCK I/O, APPC/VM and IUCV Not Resident	125
Blocking Suppression for INSERT CURSORS	125
Buffer Pool Too Big	126
Buffer Pool Too Small	128
CHARNAME Not Set Correctly	130
Checkpoint is Being Forced at End-LUW	130
CHKINTVL Too Big	131
CHKINTVL Too Small	132
CMS Work Unit Support Set On	134
Conflict in Catalog Key Locking	134
Conflict on Key Hash in User Data	136
CREATE INDEX Requires a Large Sort	138
Data Not Cached	139
Database Machine Favored Too Little	139
DBSPACE Scan Being Performed	141
Deadlocks	146
DRDA Protocol Used to Access an SQL/DS Database	147
DRDA Usage	147
ECMODE ON for Accounting	147
Excessive I/Os on INSERT	147
Excessive Locking in User Data	149
Frequent Checkpoints caused by SOSLEVEL	152
Hot Spot in the Catalog Tables	153
Hot Spot in User Tables	158
I/O Capacity Exceeded	160
I/O Not Balanced	161
Inaccurate Statistics	162
Index Disqualified	164
Index Maintenance	167
Index No Longer Highly Clustered	168
Indexes Are Fragmented	169
Inefficient Search	170
Inefficient SELECT List	174
Insufficient Indexing	174
Invalid Entities Exist	175
Large Tables Share Same DBSPACE	175
Lock Level Too High	178

Lock Level Too Low	179
Locks Held for Long Duration	180
Logging during Load	183
Long DBSS Calls Delaying Checkpoint	184
Missing Search Condition	185
Need a Highly Clustered Index	187
Need More CPU	188
Need More Real Storage	189
NLRB Parameters Too Large	190
NLRB Parameters Too Small	191
No Selective Index	192
One Database Machine Needs Too Much CPU	193
Package Needs Re-preprocessing	194
Package Cache Too Big or Threshold Too High	195
Package Cache Too Small or Threshold Too Low	196
Page Fault Serialization	197
Query Block Size Too Small	198
Range Predicate Used with Host Variables	199
Sequential Processing	200
Session Limit Exceeded	203
SET QDROP OFF USERS or SET QUICKDSP ON Not Used	203
SQL/DS Code Not Shared	204
Storage Pool Full	204
Synchronous APPC/VM Not Used	205
Too Few Agents	205
Too Many Agents	206
Too Many Joins	208
UPDATE STATISTICS by DATALOAD	210
Very Nonunique Index Key Prefix	211
Chapter 6. Recovering from DBSS Errors	213
Interpreting the Diagnostic Display	213
Action to Take for FORWARD Processing Failures	219
Action to Take for ROLLBACK Processing Failures	221
Action to Take for UNDO Processing Failures	221
UNDO Processing Failure During a Warm Start	221
UNDO Processing Failure During a Restore	222
Action to Take for REDO Processing Failures	223
REDO Processing Failure During a Warm Start	223
REDO Processing Failure During a Restore	224
Filtered Log Recovery	224
Extended Processing	225
Bypassing an UNDO WORK Failure	228
Rolling Back Committed Work	231
Filtered Log Recovery and Referential Integrity	237
Disabling a DBSPACE	240
Enabling a DBSPACE	241
Chapter 7. Recovering from Directory Verify Errors	243
Guidelines for Using Directory Verification	243
Recovery Actions for an Inconsistency	243
Chapter 8. Problem Isolation and Handling	247
System Problems	247

SQL/DS Database Machine Problems	247
SQL/DS Virtual Machine Dump Processing	249
Problem Isolation	250
SQL/DS Dumps	250
SQL/DS Link Maps and Access	252
Dump Navigation	253
Storage Layout after Initialization	256
Major Control Blocks	260
Locating SQL/DS Statements Associated with a System Error	262
DBSS OP Codes	265
Problem Isolation and The Trace Facility	267
Trace Facility	267
Trace in Storage	267
Using Trace for Deadlocks	267
Appendix A. RDIIN	273
Appendix B. Catalog Updates and References	283
Authorization	283
Interpretive Commands	286
Appendix C. SQL/DS Distributed Data Management (DDM) Command Support	297
How to Read the Tables	297
Command Tables	298
Reply Tables	298
Glossary	311
Bibliography	313
Index	315

Notices

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 licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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 Commercial Relations, IBM Corporation, Purchase, NY 10577.

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

Programming Interface Information

This publication is intended to help the customer to do diagnosis of SQL/DS* problems and primarily documents Diagnosis, Modification or Tuning Information.

Warning: Do not use this Diagnosis, Modification or Tuning Information as a programming interface.

However, this publication also documents Product-sensitive Programming Interface and Associated Guidance Information provided by the SQL/DS product.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of the SQL/DS product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Product-sensitive programming interface

Product-sensitive Programming Interface and Associated Guidance Information...

End of Product-sensitive programming interface

Trademarks and Service Marks

The following terms, denoted by an asterisk (*), used in this publication, are trademarks or service marks of IBM Corporation in the United States or other countries:

DATABASE 2	DB2	DB2/2
DB2/6000	Distributed Relational Database Architecture	DRDA
IBM	OS/2	OS/400
SQL/DS	VM/ESA	VM/XA
VTAM	AIX/6000	

About This Manual

Purpose

This manual is a task oriented publication. It provides background information which will allow you to better perform the tasks listed below. This manual should be used for one of these tasks:

1. Determining if there is a problem with the SQL/DS* RDBMS. This could be a defect, a functional problem, or a performance problem.
2. Gaining insight on what you can do to recover from certain situations or problems.
3. Developing a symptom string that describes a defect and reporting that defect along with the necessary documentation of the problem.

If your problem is occurring during distributed processing, you should also refer to the *Distributed Relational Database Problem Determination Guide*.

Wherever the term VM is used in this document, the reference applies to VM/SP, VM/XA*, or VM/ESA*. The term VM/SP refers to VM/SP Release 6 with or without High Performance Option (HPO). The term VM/XA refers to VM/XA SP Release 2. The term VM/ESA refers to VM/ESA Release 1.0 or above.

Audience

This manual is intended for persons responsible for diagnosing and fixing problems with the SQL/DS RDBMS.

Contents

SQL/DS Diagnosis Guide and Reference has eight chapters:

- Chapter One, "Introduction to Problem Diagnosis," gives a general description of the diagnosis task. It offers a "flowchart" as a means of directing you to the proper area within the manual.
- Chapter Two, "SQL/DS Concepts," introduces the primary characteristics of the Structured Query Language/Data System and discusses their use and interaction.
- Chapter Three, "Reporting Defects," shows you how to build a symptom string that describes a defect, how to report a problem, and how to list the documentation that should accompany your defect report.
- Chapter Four, "Functional Problems," describes analysis of problems that you might encounter in using the SQL/DS RDBMS. It gives possible causes and actions to take in resolving problems.
- Chapter Five, "Diagnosing Performance Problems," assists you in determining the ultimate cause of a performance problem. It includes analyses, corrective actions, and problem indexes that help you get to the right spot to perform the next item.

- Chapter Six, "Recovering from DBSS Errors," tells you how to interpret the diagnostic information following message ARI0126E and the actions to take to recover from the error(s).
- Chapter Seven, "Recovering from Directory Verify Errors," tells you when to use the Directory Verify function and how to recover from errors discovered by the Directory Verify function.
- Chapter Eight, "Problem Isolation and Handling," describes the tools you can use to correct a problem.
- Appendix A, "RDIIN," lists the RDIIN.
- Appendix B, "Catalog Updates and References," contains information about how catalogs are searched and updated when specific statements are executed.
- Appendix C, "SQL/DS Distributed Data Management (DDM) Support," contains information on the DDM commands that the SQL/DS product supports for DRDA* (Distributed Relational Database Architecture) level 1.
- Glossary contains definitions of SQL/DS terms.

Components of the SQL/DS System

Figure 1 on page xiii depicts a typical SQL/DS configuration with one database and two users.

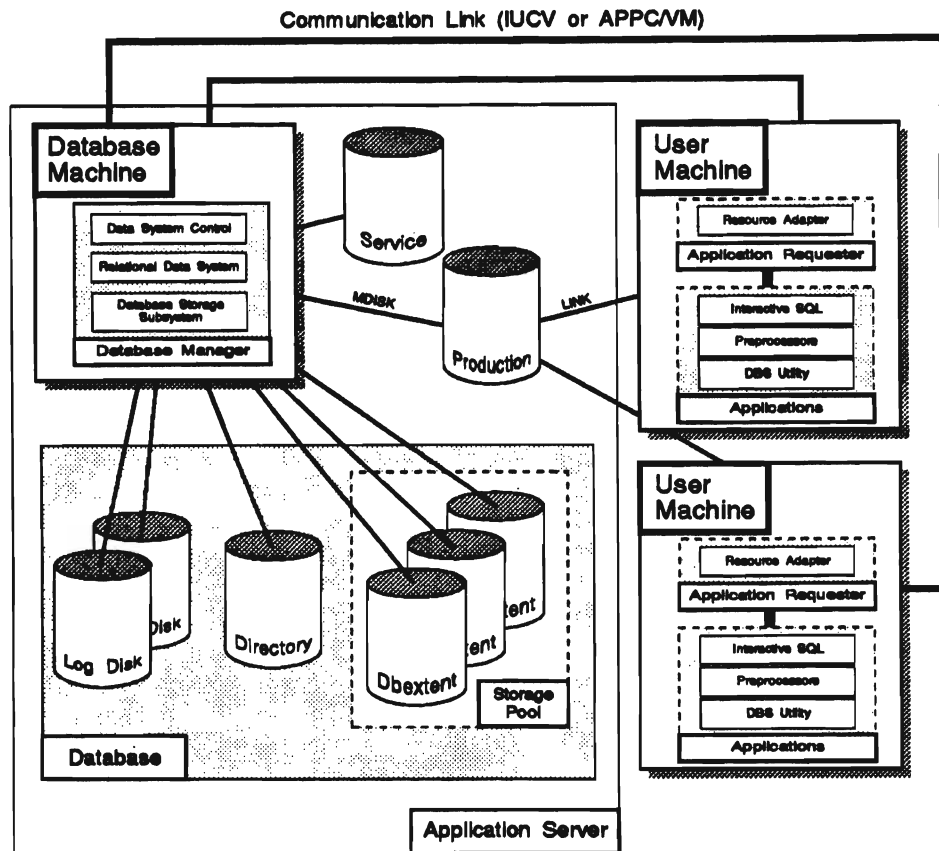


Figure 1. Basic Components of the SQLIDS RDBMS

The **database** is composed of:

- A collection of data contained in one or more *storage pools*, each of which in turn is composed of one or more *database extents (dbextents)*. A dbextent is a VM minidisk.
- A *directory* that identifies data locations in the storage pools. There is only one directory per database.
- A *log* that contains a record of operations performed on the database. A database can have either one or two logs.

The **database manager** is the program that provides access to the data in the database. It is loaded into the database virtual machine from the production disk.

The **application server** is the facility that responds to requests for information from and updates to the database. It is composed of the database and the database manager.

The **application requester** is the facility that transforms a request from an application into a form suitable for communication with an application server.

Syntax Notation Conventions

Throughout this manual, syntax is described using the structure defined below.



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

The  symbol indicates the beginning of a statement or command.

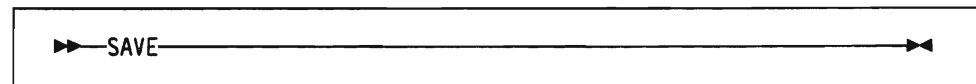
The  symbol indicates that the statement syntax is continued on the next line.

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

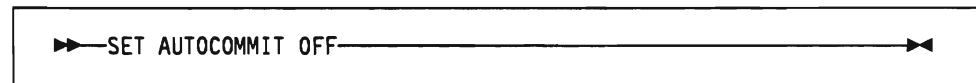
The  symbol indicates the end of a statement.

Diagrams of syntactical units that are not complete statements start with the  symbol and end with the  symbol.

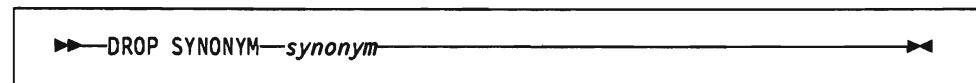
- Some SQL statements, Interactive SQL (ISQL) commands, or database services utility (DBS Utility) commands can stand alone. For example:



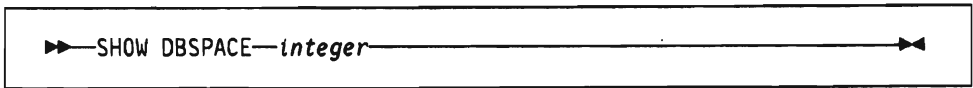
Others may be followed by one or more keywords and/or variables. For example:



- Keywords may have parameters associated with them which represent user-supplied names or values. These names or values can be specified as either constants or as user-defined variables called *host-variables* (*host-variables* can only be used in programs).



- Keywords appear in either uppercase (for example, SAVE) or mixed case (for example, CHARACTER). All uppercase characters in keywords must be present; you can omit those in lowercase.
- Parameters appear in lowercase and in italics (for example, *synonym*).
- If such symbols as punctuation marks, parentheses, or arithmetic operators are shown, you must use them as indicated by the syntax diagram.
- All items (parameters and keywords) must be separated by one or more blanks.
- Required items appear on the same horizontal line (the main path). For example, the parameter *integer* is a required item in the following command:



This command might appear as:

```
SHOW DBSPACE 1
```

- Optional items appear below the main path. For example:



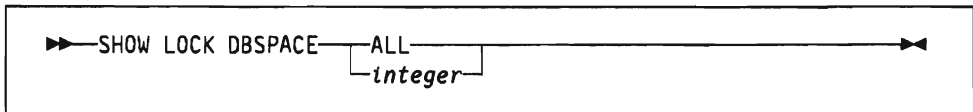
This statement could appear as either:

```
CREATE INDEX
```

or

```
CREATE UNIQUE INDEX
```

- If you can choose from two or more items, they appear vertically in a stack. If you must choose one of the items, one item appears on the main path. For example:



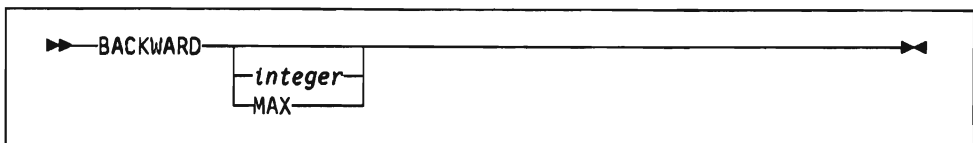
Here, the command could be either:

```
SHOW LOCK DBSPACE ALL
```

or

```
SHOW LOCK DBSPACE 1
```

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



Here, the command could be:

```
BACKWARD
```

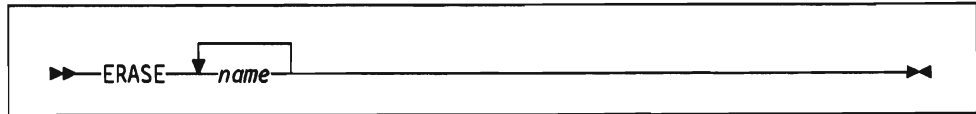
or

```
BACKWARD 2
```

or

```
BACKWARD MAX
```

- The repeat symbol indicates that an item can be repeated. For example:



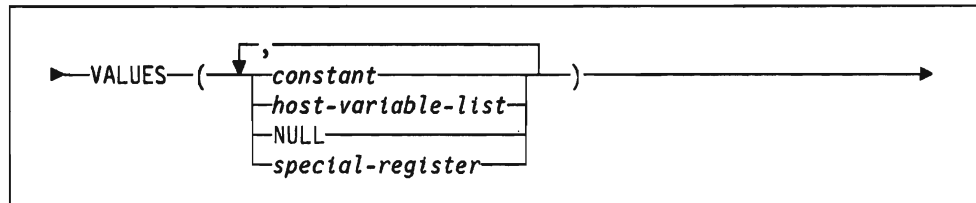
This statement could appear as:

ERASE NAME1

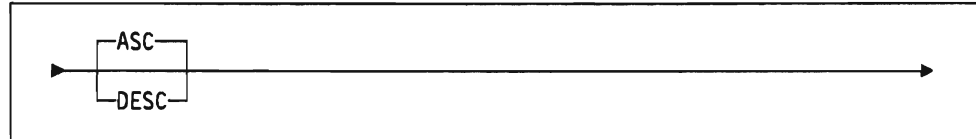
or

ERASE NAME1 NAME2

A repeat symbol above a stack indicates that you can make more than one choice from the stacked items, or repeat a choice. For example:

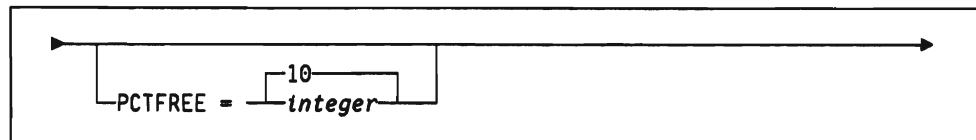


- If an item is above the main line, it represents a default, which means that it will be used if no other item is specified. In the following example, the ASC keyword appears above the line in a stack with DESC. If neither of these values is specified, the command would be processed with option ASC.

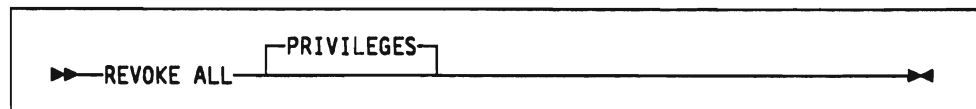


- When an optional keyword is followed on the same path by an optional default parameter, the default parameter is assumed if the keyword is not entered. However, if this keyword is entered, one of its associated optional parameters must also be specified.

In the following example, if you enter the optional keyword PCTFREE =, you also have to specify one of its associated optional parameters. If you don't enter PCTFREE =, the system will set it to the default value of 10.

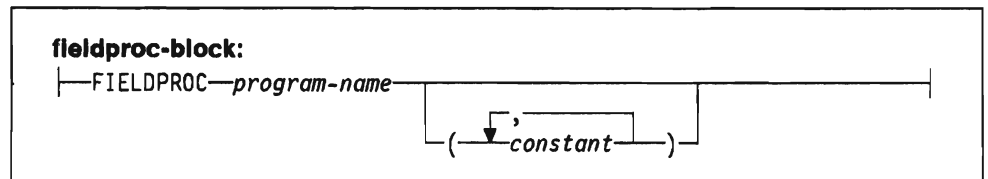
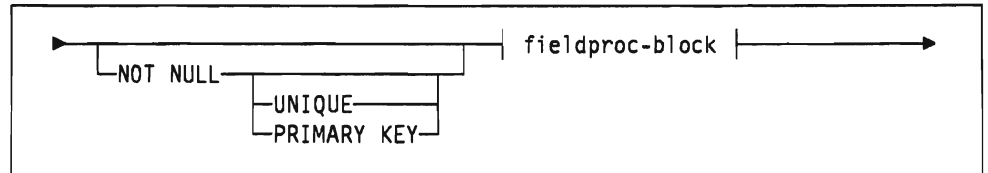


- Words that are only used for readability and have no effect on the execution of the statement are shown as a single uppercase default. For example:



Here, specifying either REVOKE ALL or REVOKE ALL PRIVILEGES means the same thing.

- Sometimes a single parameter represents a fragment of syntax that is expanded below. In the following example, **fieldproc-block** is such a fragment and it is expanded following the syntax diagram containing it.



Summary of Changes for SQL/DS Version 3 Release 4

This is a summary of the technical changes to the SQL/DS* Version 3 Release 4 database management system. All manuals are affected by some or all of the changes discussed here. This summary does not list incompatibilities between releases of the SQL/DS product; see either the *SQL Reference* or the *System Administration* manual for a discussion of incompatibilities. Version 3 Release 4 of the SQL/DS database management system is intended to run in the following VM environments:

- Virtual Machine/System Product (VM/SP) Release 6, with or without the High Performance Option (HPO)
- Virtual Machine/Extended Architecture* (VM/XA*) SP Release 2
- Virtual Machine/Enterprise Systems Architecture* (VM/ESA*) Release 1 or later.

Enhancements, New Functions, and New Capabilities

Support for the IBM DATABASE 2 AIX/6000 Database Manager

The IBM* DATABASE 2* AIX/6000* (DB2/6000*) database manager implements the DRDA* remote unit of work feature. Your VM SQL/DS database manager can function as an application server for your DB2/6000 database managers.

Cascade Delete Enhancement for Referential Integrity

Referential integrity ensures that references in one table to data in another table are always valid. In previous releases, you could specify cascade delete constraints to ensure that if a value was deleted from a parent table, the corresponding row of a dependent table would also be deleted.

Version 3 Release 4 expands the capabilities of the cascade delete constraint. When a row is deleted from a dependent table because of a

cascade delete, the delete rule that exists between the dependent table and any tables that are its dependents will be processed. For example, suppose a cascade delete constraint exists for Table_A that results in a row being deleted from Table_B. Table_B, in turn, might have a delete rule that then causes a value in Table_C to be set to NULL.

This enhancement can ensure integrity across several tables and reduce the programming effort needed to develop an application.

Improved EXPLAIN Capabilities

The EXPLAIN statement is used to analyze data manipulation statements to provide information about the structure, execution, and approximate cost of the SQL statement being analyzed.

In Version 3 Release 4, the EXPLAIN function has been enhanced as follows:

- Several new columns have been added to the existing EXPLAIN tables.
- The EXPLAIN function can now be used as a preprocessing option for all static SQL statements embedded in an application program.
- A DBS utility job file is shipped with the SQL/DS product to generate EXPLAIN tables, indexes, and views.

The enhanced EXPLAIN facility is more compatible with the DB2* EXPLAIN facility.

Host Structure Variables

Several of the host languages that are supported by the SQL/DS database manager let you define variables in structured formats. For example, a COBOL program might specify the following structure for a three line address.

```
04 ADDRESS
05 LINE-1
05 LINE-2
05 LINE-3
```

A programmer can specify ADDRESS to refer to all three lines.

The SQL/DS database manager has been enhanced to provide this kind of support for two levels of structured variables. If you code a host structure in an SQL declaration, you can use the name of that host structure in any SQL statement where you would otherwise code a list of host variables. This facility is available for SQL code embedded in application programs written in C, COBOL, PL/I and RPG. If you code a structure with more than two levels, all of the lowest two level substructures can be used as host structures by the SQL/DS database manager.

Removal of 512 Host Variable Restriction

The previous maximum number of host variables allowed in a program module was 512. This restriction has been removed. The number of host variables is now restricted only by the size of storage.

Enhanced SHOW STORAGE Command

A new SHOW STORAGE operator command is provided to let you determine the system load and avoid problems caused by insufficient storage. This command displays information about system storage currently being used and the maximum total storage usage.

The SHOW STORAGE command can be used together with the RESET HIGHSTOR command. By resetting the HIGHSTOR value, performing a function and then invoking the SHOW STORAGE command, you can determine the maximum storage needed to perform the function.

Usability Enhancements

Package Dbspace Full Condition Handling

Previously, when a package was dynamically reprocessed and a package dbspace full condition occurred, the reprocess would fail with an SQLCODE of -946 and the user would then need to explicitly recreate the package.

If this condition occurs with Version 3 Release 4, the database manager will automatically search

for a non-full package dbspace that can accommodate the reprocessed package. Manual intervention will only be required if all package dbspaces are full (if they already contain 255 packages or do not have enough free space to hold the additional package).

The Connectable and Unconnected State

When a severe error occurs, the logical unit of work is rolled back and the communication link is severed. This causes the application to enter a "connectable and unconnected" state. Previously, if the next SQL statement was not a CONNECT statement, SQL/DS caused the application to abend.

However, if this condition occurs in Version 3 Release 4, SQL/DS does not cause the application to abend, but issues a severe error code instead; SQLCODE of -900 and SQLSTATE of 51018.

Dual Logging Enhancement

If you are using dual logs and one is damaged, it can be replaced with a new minidisk. SQL/DS will then copy the good log to the new one without the necessity of doing a COLDLOG.

Reliability, Availability, and Serviceability Improvements

Processing a DROP TABLE Statement

When a DROP TABLE statement is executed, the rows of the specified table are not dropped immediately; instead, the table is marked for deletion by adding a row to the SYSDROP system catalog table. After the current logical unit of work (LUW) is committed, a new LUW is started and the table is dropped.

For each page that contains data for the table being dropped, an equal number of shadow pages must be retained until the delete operation is completed, and the LUW can be committed. If the table occupies a large number of pages, it is possible to run out of physical pages, or to encounter a storage pool full condition.

With this enhancement, checkpoints are possible as rows are deleted, freeing shadow pages more quickly. As well, the database manager will no longer abend if a storage pool full condition occurs during a DROP TABLE operation.

Enhancement to COLDLOG Processing

Previously, running a COLDLOG operation could destroy log data needed for recovery if any logical units of work were unresolved when the database was last shut down. In Version 3 Release 4, the operator will get a warning message if this condition exists and will be able to cancel the COLDLOG operation before any data is lost.

Improved Storage Trace

The storage trace facilities have been enhanced to provide relevant trace point information about the SQL/DS application server. The new facility provides consolidated trace information for system and working storage.

It can be invoked either during database startup with a new STARTUP command parameter or after the database is started with the TRACE operator command.

Library Enhancements

SQL/DS Performance Tuning Handbook

A new manual provides information to enable you to tune your SQL/DS system more effectively.

Revised Manuals

The *Database Administration* and *System Administration* manuals have been revised to provide you with better information.

Chapter 6 of the *Database Administration* manual and Chapter 8 of the *System Administration* manual have been removed; this information has been included in the *SQL/DS Performance Tuning Handbook*.

Chapter 1. Introduction to Problem Diagnosis

This chapter:

- Describes the tasks to be performed by persons involved in diagnosing problems that may be encountered with the SQL/DS RDBMS.
- Describes some tasks that you might use to solve particular types of problems.
- Offers guidance on using this book to effectively accomplish those tasks.
- Offers a "flowchart" to help you determine the correct chapter(s) and pages to use to perform a task for a given condition(s).

If you have used this book before, you can skip the introductory remarks and go directly to the flowchart or to the appropriate chapter.

You are here because someone has a problem using the SQL/DS RDBMS in a VM environment. The problem may not be one of failure. It might be one of poor performance, for example. In this case you would go to Chapter 5, "Diagnosing Performance Problems" on page 101 to attempt to isolate and diagnose the problem. Completion of a diagnosis task could result in your going to Chapter 3, "Reporting Defects" on page 69 to:

1. Systematically develop a keyword string to describe the problem.
2. Check whether this same problem is already documented and corrected.
3. Report the problem to IBM if it is a new one.

Using this book will expedite an IBM-supplied correction for a defect problem.

Use this book even when you are not thoroughly convinced that the SQL/DS RDBMS caused the problem. Instead, one of these might be causing the problem: a subtle user error, the current management of SQL/DS resources, or another IBM* product. If your problem is occurring during distributed processing, you should also refer to the *Distributed Relational Database Problem Determination Guide*.

This book also includes things you might want to do to aid in avoiding problems.

Diagnosis Flowcharts

Use the following flowchart to help you decide where you want to start using this book. For example, if you are experiencing a **performance problem** associated with a specific application or application function, this chart directs you to "Application Function Indexes to Performance Problems" on page 106 which will lead you through the path(s) to follow for diagnosing the problem. You may or may not need to return from one of the paths to the starting point. Need is determined by the path selected and/or the results of the task performed along the path.

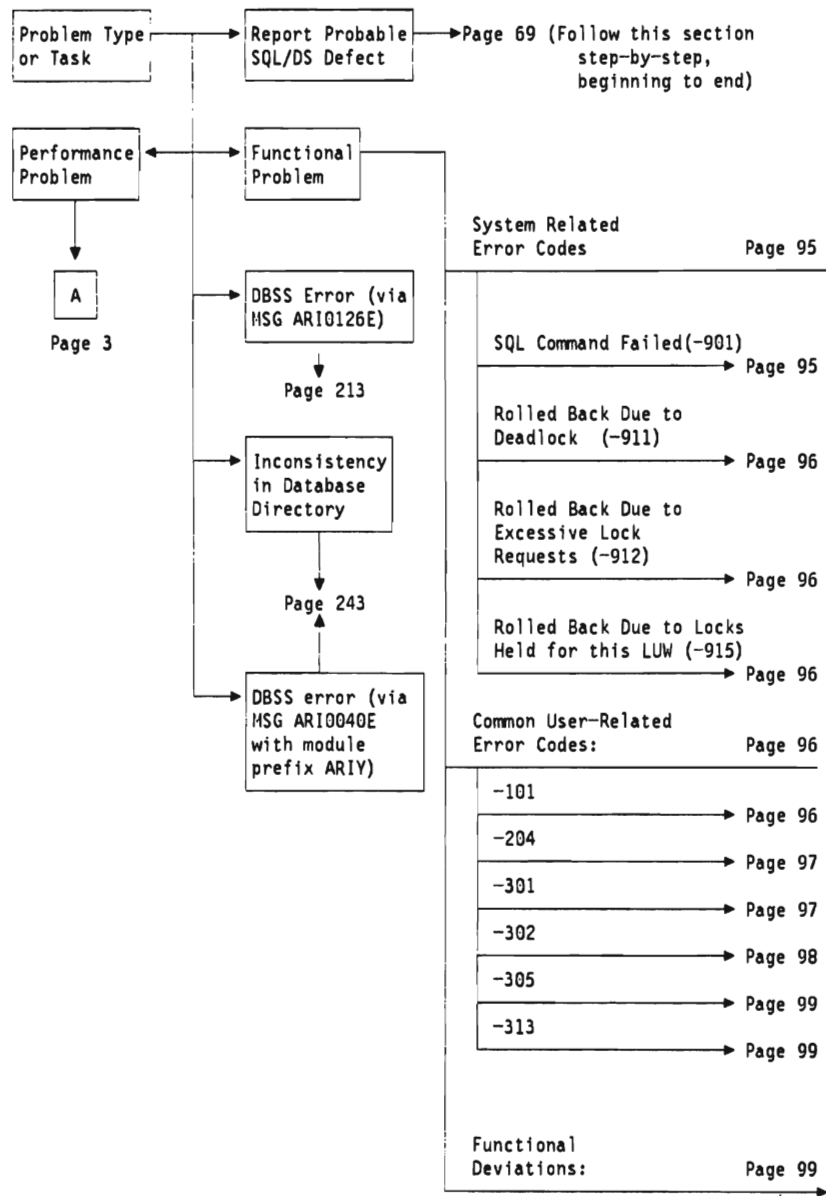


Figure 2 (Part 1 of 2). Diagnosis Flowchart

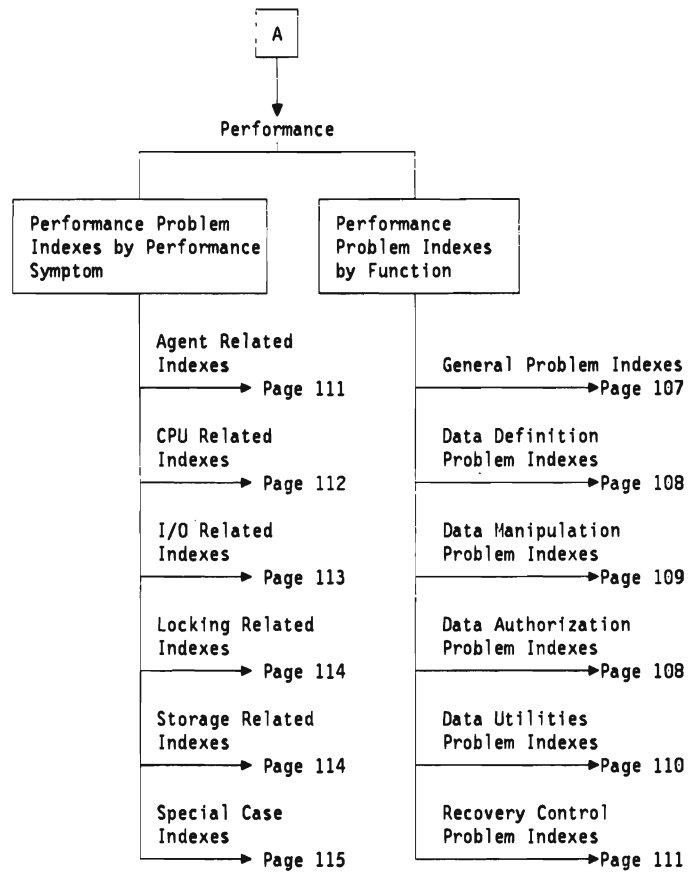


Figure 2 (Part 2 of 2). Diagnosis Flowchart



Chapter 2. SQL/DS Concepts

This chapter presents an overview of the SQL/DS RDBMS, including its purpose and function, and how it interacts with other programs. It is intended to provide background information to enable more efficient tuning, monitoring and problem determination. In particular, this chapter discusses the following concepts:

- SQL/DS Components
- Logical Unit of Work
- Agent Handling
- Mailbox/Communication
- Package Management
- Memory Management
- Logical Storage Management
- Physical Storage Management
- Buffer Storage Management
- Index
- Sorting
- Logging/Recovery
- Locking
- Termination.

Introduction

Figure 3 depicts the SQL/DS components and shows where they are executed in the VM environment when using SQL/DS-only protocol.

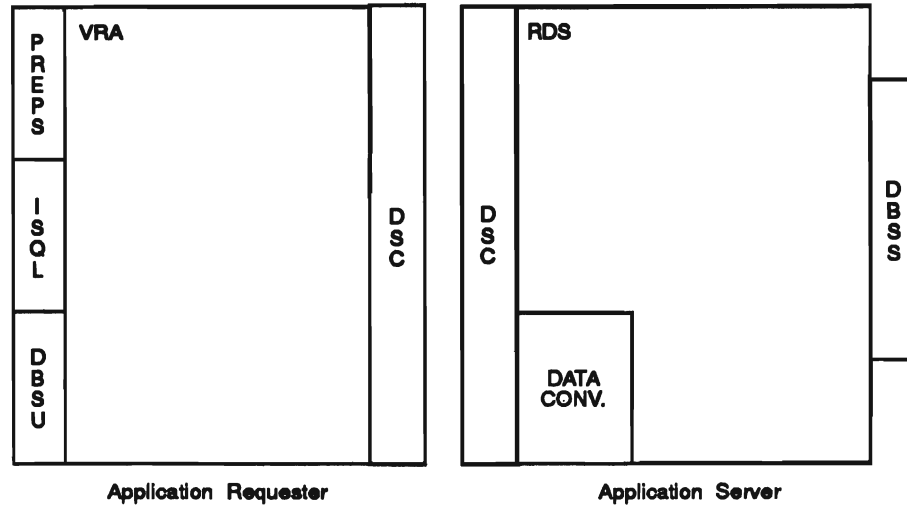


Figure 3. SQL/DS Structure and Components when using SQL/DS-only Protocol.

Figure 4 depicts the SQL/DS components and shows where they are executed in the VM environment when using DRDA protocol.

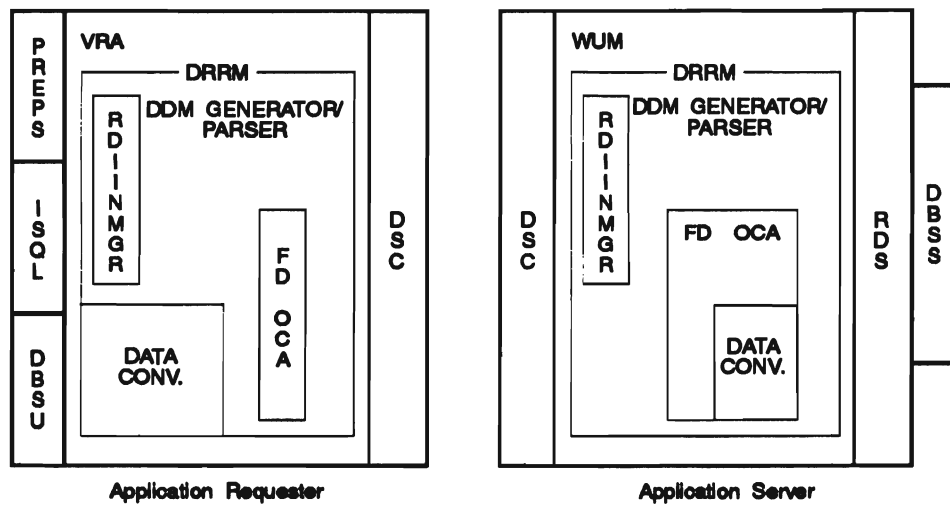


Figure 4. SQL/DS Structure and Components when using DRDA Protocol.

SQL/DS Components in the Application Requester

Database Services Utility (DBSU)

The SQL/DS DBS utility is an SQL/DS application program. It is usually run as a batch program and is used to load, unload and reload data and packages, and process SQL statements.

Interactive SQL (ISQL)

ISQL is an interactive query environment that allows you to access an application server from a display terminal.

Preprocessors (PREP)

The preprocessors compile the SQL statements in an application source program and create a modified copy of the source program so that it is suitable for language processing (compiling, assembling).

VM Resource Adapter (VRA)

The VM resource adapter communicates between the user application and an application server.

SQL/DS Components in the Application Requester and the Application Server

Data System Control (DSC)

The DSC component provides the following control services. The majority of these services are provided on the application server. Only the system-dependent routines and the communications services are provided on the application requester.

1. Initialization

DSC Initialization includes establishing the initial environment, and driving the initialization and recovery process of the other SQL/DS components.

2. SQL/DS System Services

DSC provides Operator, Message, Trace, and Storage services for the other SQL/DS components.

3. Agent Handling and Communications

DSC allows other SQL/DS components to handle multiple concurrent users and to communicate with them. The dispatcher function is provided for multi-threading multiple application requests in the SQL/DS database machine.

4. System-Dependent Routines

These routines shield the other SQL/DS components from the VM system functions, allowing them to be system-independent.

5. SQL/DS Termination

DSC termination manages normal (SQLEND) and abnormal termination of the SQL/DS database machine.

Data Conversion (CONV)

Data Conversion is used for performing CCSID conversion on character and graphic data. When the DRDA protocol is used, Data Conversion also performs conversion of numeric data from one representation to another.

Distributed Relational Resource Manager (DRRM)

The DRRM component is only invoked when the DRDA protocol is used. It resides in both the application server and the application requester. This component provides the following services:

1. DDM/FD:OCA Generator and Parser

The DRRM component in the application requester generates requests to be sent to an application server and parses replies received from an application server.

The DRRM component in the application server parses requests received from an application requester and generates replies to be sent to an application requester.

2. DDM/FD:OCA Dictionary

The dictionary provides the definition of DDM and FD:OCA terms. These are used by the DDM/FD:OCA Generator and Parser to build and interpret DRDA data streams.

3. RDIIN Manager

The RDIIN manager is invoked by the DRRM Parser and Generator to interpret or build the RDIIN structure. The RDIIN structure contains the SQL/DS-specific internal format of SQL queries and replies.

SQL/DS Components in the Application Server

Work Unit Manager (WUM)

The WUM component is only invoked when the DRDA protocol is used. It acts as the central control point between application requester and application server conversations. It tracks the status of those conversations, the current unit of work, and enforces the various DRDA protocol rules. It manages and provides the interface between the DRRM and RDS components.

Relational Data System (RDS)

The RDS component provides the following services:

1. Executives

The executives manage all work done in RDS:

- The invocation of the parser, optimizer, authorization, statement generator, access generator and interpreter
- The block storage needed for statement compilation
- The loading and storing of packages.

2. SQL/DS Parser

The parser performs the primary syntactical analysis of the user's SQL statement and converts it into parse tree format.

3. Optimizer

The optimizer attempts to choose the optimal path to fulfill a user DML SQL request, and represents it as an access plan for the access generator to implement. For more information on accessing data, see the *Performance Tuning Handbook*.

4. Gen-Time Access Generator

The Access Generator translates access plans generated by the optimizer into easily interpretable control blocks.

5. Run-Time Access Generator

The Access Generator reads the control blocks representing the SQL/DS DML and interprets them to perform the DML statement.

6. Statement Generator

The Statement Generator creates SQL/DS statements, turns them into easily interpretable control blocks, and adds them to the output of the Gen-Time Access Generator.

7. Authorization

This component manages authorization functions. It controls recording of privileges and ensures that only authorized operations are performed by SQL/DS users and applications.

8. Interpreter

The interpreter handles execution of data definition and of control SQL statements by translating the parser representation of the statements and then immediately executing the appropriate database operation.

Database Storage Subsystem (DBSS)

The DBSS component provides the following services:

1. DBSS Initialization

The DBSS initialization process initializes the Work, Trace, Lock, Storage, and Log components; opens the Directory, DBEXTENT(s), Logs, and DBSPACES; sets the system counters to zero; and performs filtered log recovery if necessary.

2. DBSS Data Control

Data control operations manage the definition and operation of objects within the database: DBSPACES, tables, and indexes.

3. DBSS Data Manipulation

The Data Manipulation component retrieves and updates data in the database.

4. Index Management

The Index component controls the space on index pages, and acts on behalf of data manipulation calls to maintain indexes when tables are modified.

5. Sort Component

The Sort component creates a sorted list of rows from an existing table. Sort order is based on values in one or more columns.

6. Update Statistics

The Update Statistics component gathers statistics on DBSPACES, tables, and indexes, which are used by the RDS optimizer in path selection.

7. Logging and Recovery

The Log component maintains a record of all changes completed by each logical unit of work, retaining old and new values for each updated object.

8. Work Component

The Work component controls the beginning and ending of LUWs (Logical Unit of Work). It commits updates to the database or schedules rollbacks of all work done by an LUW since the previous COMMIT or ROLLBACK.

9. Lock Component

The Lock component controls concurrent resource access.

10. Storage Component

The Storage component maps logical DBSPACES to physical DBEXTENTS on DASD, and does all database I/O.

The SQL/DS RDBMS in Single User Mode

In single user mode, DSC initializes the application server and creates three agent structures: the operator agent, the checkpoint agent, and the user agent. In addition to establishing the application server, DSC loads the application program, be it a user routine or an SQL routine (preprocessors, DBS utility, DBMS function, and so forth), and passes control to it.

The user application issues SQL requests, which are processed by the SQL/DS Resource Adapter. The Resource Adapter passes the requests directly to RDS. After a request has been processed, RDS first uses the DSC reply function to return the data to the application program, and then returns control to the Resource Adapter, which in turn passes control to the application program. When the application program finishes processing all of its SQL requests, it returns to the application server at the point where it was invoked by DSC, and control is then passed to the SQL/DS termination routine.

The DSC termination function then closes the database files and returns control to the host system.

DRDA protocol is not supported in single user mode.

The SQL/DS RDBMS in Multiple User Mode

When Using the SQL/DS-only Protocol

In multiple user mode, DSC initializes the application server and creates three system agent structures: the operator, checkpoint and recovery agents. One or more user agent structures, as specified by the NCUSERS initialization parameter, are also created. The requests are processed by the Resource Adapter which uses DSC mailbox functions to package the request into a message and invokes the DSC communication function to send the message to the SQL/DS machine. The DSC dispatcher function dispatches an agent to process the SQL request and passes the request to RDS for processing. The DSC reply function packages the data in a message and sends the data back to the user machine.

After the reply function is complete, the DSC receive function gets the next SQL request. This process continues until the SQL/DS operator issues a `SQLEND` command. After this command is issued, no new users are allowed to establish a connection with the application server. When the last connected user disconnects, control is passed to the DSC termination routine which closes the database files and returns control to the host system.

When Using the DRDA Protocol

The SQLIDS Application Requester: The SQL request is processed by the Resource Adapter. The Resource Adapter calls DRRM to convert the request from its SQL/DS internal format and generate the DRDA `DDM/FD:OCA` format suitable to be sent to a DRDA application server. The DDM portion contains the request information, and the FD:OCA portion contains the data and its format. The Resource Adapter then uses the DSC communication function to send the request to the application server for execution.

When the reply is received from the application server, DSC receives the reply and passes it to the Resource Adapter. The Resource Adapter then invokes DRRM. The reply received is in the DRDA `DDM/FD:OCA` format. DRRM parses this reply into the SQL/DS internal format, performing any necessary data conversion. Parsing involves verification of syntax and semantics, and data conversion.

The SQLIDS Application Server: The DSC dispatcher function calls the WUM component to process the request sent by the application requester. The WUM component calls DSC to receive the request. The request received is in the DRDA `DDM/FD:OCA` format. When a request is received, WUM then invokes DRRM to parse the request into the SQL/DS internal format. Parsing involves verification of syntax and semantics, and numeric data conversion. WUM then passes the request to RDS for processing. RDS returns the reply to WUM, but before sending it, WUM calls DRRM to generate the DRDA `DDM/FD:OCA` format of the reply. WUM then invokes the DSC communication function to send the reply back.

Logical Unit of Work Concepts

The logical unit of work (LUW) serves as the unit of consistency, allocation, and recovery for a user. Also, at the boundaries of a logical unit of work, the real agent structure is reset.

In general, a logical unit of work supports a set of one or more SQL statements. When these statements are issued by an application, a logical unit of work is automatically started. In general, the user application controls the length of the LUW by the `SQL COMMIT WORK` and `SQL ROLLBACK WORK` statements. There are other mechanisms that control the length of a logical unit of work. These are discussed under the headings "Explicit Termination of an LUW" on page 12 and "Implicit Termination of an LUW" on page 12.

Atomicity of SQL Statements:

SQL statements are atomic. This means that the SQL/DS database manager does not allow a statement to be only partially completed; for example, when a statement that makes several changes to the database is being processed, and an error occurs after some changes have already been made, the database

manager undoes those changes before returning to the calling application. The net effect is as if the SQL statement was never issued.

Note: Atomicity is not supported for objects stored in nonrecoverable storage pools. Also, when LOGMODE=N, the atomicity of a statement can only be enforced by rolling back the entire logical unit of work.

Agent Handling:

While a logical unit of work is in process, the user is associated with a real agent. In the SQL/DS RDBMS, the number of real agents is limited by the NCUSERS initialization parameter. Thus NCUSERS is the maximum number of active logical units of work at any one time.

End of LUW Processing:

Whenever a logical unit of work terminates, normally or abnormally, internal DBSPACES that were allocated are freed and any remaining locks are released.

Explicit Termination of an LUW:

There is no explicit invocation to start a unit of work. All explicit invocations have to do with termination or recovery of a logical unit of work. The following actions result in explicit termination of a logical unit of work:

- Commit. This is invoked when a user issues the SQL COMMIT WORK statement.
- Rollback. This is invoked when a user issues the SQL ROLLBACK WORK statement.

Implicit Termination of an LUW:

The following cause an implicit termination of a logical unit of work.

- The SQL/DS FORCE command. In general, this supports only rollback.
- Specific abnormal conditions. These conditions include log full, deadlock, and storage pool full.
- Return to CMS command line.
- The SQLRMEND exec

Agent Handling Concepts

The SQL/DS RDBMS uses a set of control blocks called an agent structure (or real agent) to service requests from multiple users to access a common database.

There are always two SQL/DS agent structures created: the Operator and the Checkpoint agents. (The initialization process is executed under the Operator agent. The checkpoint agent is activated whenever a checkpoint is to be taken.) In single user mode, there is also a User agent structure under which the user's SQL requests are executed. In multiple user mode, one or more real agent structures are allocated; the number is equal to the value of the NCUSERS initialization parameter. There is also a set of agent structures called "pseudo-agents." (See "Pseudo-Agent and Real Agent Structures" on page 13). The

number of pseudo-agents allocated is equal to the value of MAXCONN specified in the VM directory less the number of CMS DASDs used for the database and one for the connection to *IDENT. Also, in multiple user mode, a Ready/Recovery system agent structure is created.

The User agents are either general purpose or "in-doubt" agents. An agent is in-doubt when it has reached a point in processing where the work may either be committed or rolled back. If the database manager (or CICS) abnormally terminates at such a time, the next time it is restarted, the SQL/DS (or CICS) log is coordinated by way of the Ready/Recovery agent, and the in-doubt agents have the suspended work either committed or rolled back. Normally, agents are general purpose; however, in a Guest Sharing CICS environment, the general purpose agents can be in-doubt agents. There are also prototype agent structures created. This type of agent is not dispatchable and is used for storage purposes only.

Agent Handling Functions

Agent handling consists of:

- Allocating users to agent structures
- Dispatching agent structures
- Agent processing at the end of a logical unit of work (LUW).

Allocating Users to Agent Structures

There are differences in agent handling between single user mode and multiple.

In single user mode this process is quite simple. There are three agents created: the Operator, Checkpoint, and User agents. At initialization time, the Operator agent performs the initialization functions. When initialization is complete, it becomes dormant and passes control to the User agent, which is said to be "dispatched." The User agent executes until a checkpoint or archive is required, at which point the Checkpoint agent is dispatched, and the User agent waits until the checkpoint has been completed. When the checkpoint is complete, the User agent is redispached to continue processing. When the application program terminates, it returns control to the database for termination. Termination ensures that a final COMMIT or ROLLBACK is executed as necessary, based on the state of the last LUW.

In multiple user mode (MUM), when initialization has been completed, all the user agents are dormant (not allocated). When the user issues an SQL statement, a connection is established between the user machine and the database machine. This connection is made to a pseudo-agent structure by way of an IUCV or APPC/VM CONNECT.

Note: The IUCV or APPC/VM CONNECT results in an external interrupt in the database machine. The SQL/DS external interrupt handler allocates the pseudo-agent to the user and attempts to allocate a real agent, if available, to the pseudo-agent.

Pseudo-Agent and Real Agent Structures:

Each real agent requires approximately 110K bytes of storage. (This does not include dynamic storage requirements such as package storage.) Because it would not be practical in terms of storage and performance to allocate a real

agent for each user, pseudo-agents that use less than 600 bytes of storage were developed. Pseudo-agents allow many users to share (but not concurrently) a few real agent structures. Pseudo-agents are allocated to the real agent structures in FIFO order.

When a user virtual machine CONNECTs to the database machine, that user is allocated a pseudo-agent. The pseudo-agent is then placed in an "in use" queue until the user associated with that pseudo-agent sends a message to the database machine. That pseudo-agent is then allocated a real agent (assuming one is available). If all real agents are in use, any users having sent messages to the database machine have their pseudo-agents placed on a "wait" queue until a real agent is available. A real agent becomes available whenever an active user (one whose pseudo-agent already owns a real agent) completes a logical unit of work. At this point, the first waiting pseudo-agent is allocated to the newly available real agent. If the pseudo-agent that has just completed the unit of work sends another message, it is added to the end of the waiting pseudo-agent queue.

With Guest Sharing, the SQL/DS Online Resource Adapter, running under the control of CICS, can establish the number of communication links specified during Online Resource Adapter Initialization. Each of the links is associated with a pseudo-agent to which a real agent is permanently assigned. These pseudo and real agents are not available to other users until the Online Resource Adapter is terminated.

After real agents are connected to a user, they are dispatched in a nonpreemptive priority fashion. The following discussion describes the SQL/DS dispatching scheme for real agents.

Dispatcher Components

The SQL/DS dispatcher is a nonpreemptive dispatcher, which means agents must be willing to give up control of the processor. When agents give up control of the processor, they turn control of the processor over to the SQL/DS dispatcher. When given control, the dispatcher accomplishes the following actions:

- Agent Prioritization
- Fair Share Auditing
- Locating and dispatching a dispatchable agent.

The prioritization scheme ensures that agents are prioritized so that shorter LUWs are given preferential treatment over longer ones. Fair share auditing is a scheme that ensures that no LUW is denied a share of the processor indefinitely. Finally, among all the agents, the dispatcher is responsible for locating a dispatchable agent (that is, an agent that is not waiting for some event to occur— I/O completion, communications with a user, and so forth) and allowing the agent to use the processor (dispatching it).

Conceptual Overview of Prioritization Scheme

Prioritization of agents, which occurs after each dispatch, is based on the amount of referencing of database pages that agents do. Agents are sorted in ascending order in the dispatch queue based on their database page reference count values.

The priority dispatcher selects dispatchable agents by scanning the dispatch queue from the top to bottom in search of a dispatchable agent. Upon return to the dispatcher, the returning agent must undergo reprioritization in the dispatch queue.

The reprioritization process scans the dispatch queue to find the appropriate position in the dispatch queue for the returning agent. The process assumes that the dispatch queue is always sorted by agents' database page reference count. Thus, for performance reasons, the queue is scanned, beginning with the agent (in the dispatch queue) that follows the returning agent. By scanning the dispatch queue, the reprioritization process looks for the first agent with a database page reference count greater than the database page buffer reference count for the returning agent. If an agent with a database page buffer reference count is encountered before the end of the dispatch queue is reached, the returning agent is placed between the agent with the higher database page reference count and its predecessor. If the scan reaches the bottom of the dispatch queue, the returning agent is placed at the bottom of the dispatch queue. This mechanism ensures that the dispatch queue is always ordered. Note that reprioritization applies to user or general purpose agents only. "Special purpose" or "system" agents—operator and checkpoint—are not reprioritized after each dispatch; rather, they permanently reside at the top of the dispatch queue so that they receive the highest priority assigned to any agent.

After the above process has been completed, the reprioritization scheme is complete and the Fair Share Auditing process is invoked.

Conceptual Overview of the Fair Share Auditing Process

The function of the Fair Share Auditing process is to scan the dispatch queue bottom-up, find one "deprived" (see description of *deprived* below) agent, and enqueue that agent at the top of the dispatch queue (just below the system agents). Being enqueued at the top of the dispatch queue allows a deprived LUW the highest priority given to a general purpose agent for the next dispatch.

Finding "Deprived" Agents

When Fair Share Auditing is invoked, the "fair number" of buffer references that an agent should have received during the interval is computed. The fair number of buffer references is expressed by the formula:

$$\frac{\text{total-buffer-references-during-the-interval}}{\text{number-completed-LUWs} + \text{NCUSERS}}$$

The Fair Share Auditor scans the dispatch queue bottom-up and searches for the first general purpose agent that has an interval buffer reference count value less than the computed value of the above formula. If an agent is found that has an interval buffer reference count value less than the computed "fair number" it is called *deprived*.

Setting Fair Share Interval Size

Users can control the frequency of Fair Share Auditing by changing the value of the SQL/DS parameter DISPBIAS. The frequency of Fair Share Auditing (Fair Share Interval Size) is measured in dispatches. The table shown below describes the mapping from a particular DISPBIAS setting to a Fair Share Interval Size. For example, selecting DISPBIAS = 1 causes Fair Share Auditing to occur every seven dispatches, 2 causes it to occur every seventeen dispatches and so forth, as follows.

DISPBIAS	Fair Share Interval Size
1	7
2	17
3	39
4	97
5	263
6	753
7	2215
8	6593
9	19719
10	59089

The DISPBIAS setting can be used to control the way the dispatcher treats various LUWs. A setting of 10 causes short LUWs to be strongly favored and long LUWs to be strongly disfavored whereas a setting of 1 causes less favoritism among long and short LUWs. The default for DISPBIAS is 7.

After Fair Share processing has been completed, the function concludes by scheduling the time of the next auditing interval. After the next interval has been scheduled, the dispatcher starts scanning the prioritized dispatch queue from top to bottom in search of a dispatchable agent to actually be dispatched and receive control of the processor.

Locating and Dispatching a Dispatchable Agent

Agents are prioritized in the dispatch queue by the prioritization scheme. The Fair Share Auditor completes its fair share auditing process if a Fair Share Interval has expired. The dispatcher scans the agents starting from the top of the dispatch queue, testing its associated wait flags to determine whether the agent is dispatchable. An agent is dispatchable only if it is not waiting. The various wait state conditions are tested in the order listed below. The first agent encountered by the dispatcher that is dispatchable is given control of the processor (dispatched). When no agent is dispatchable, the database manager performs a system wait. If an agent is waiting, it is in one of eight wait states, which are mutually exclusive:

1. I/O Wait

In I/O wait, the agent is waiting for an I/O operation to be completed.

2. Inactive

This condition occurs only at initialization time and when the agent is not connected to a communication link.

3. Lock Wait

In lock wait, the agent is waiting to obtain a lock held by another agent.

4. Latch Wait

In latch wait, the agent is waiting for a latch on a page or block buffer which is in use by another agent who is reading into the buffer.

5. APPC/VM Wait

This wait state is entered by an agent when an APPC/VM RECEIVE or SENDDATA is not completed immediately. The APPC/VM function is complete when VM has moved the data into or out of the message buffer.

6. Communication Wait

In communication wait, the agent is waiting for a message to be sent from the user.

7. General Wait

This wait state condition is entered by other agents whenever the Checkpoint agent is dispatched.

8. Buffer Wait

This wait occurs when an agent requests a data page or directory block buffer to transfer data to or from, but none is available. The agent waits for one to become available.

The above wait state conditions and dispatchability of an agent are tested in the following order: I/O wait, lock wait, latch wait, APPC/VM wait, communication wait, general wait, and buffer wait.

Agent Processing at the End of an LUW

When a user completes an LUW (end-of-LUW), that user relinquishes ownership of the real agent (see "Pseudo-Agent and Real Agent Structures" on page 13). A user maintains ownership of a real agent only if no other users are waiting for a real agent, and the current user has issued another SQL request.

Communications Concepts

SQL/DS communications support is provided by IUCV in a VM/XA environment and by APPC/VM in a VM/SP and a VM/ESA environment. The communications concepts differ slightly depending on the selected protocol. When using the SQL/DS-only protocol, requests are sent and replies are received through *mailboxes*. When using the DRDA protocol, communication takes place in the form of request and reply *data streams*.

Concepts on the Application Requester

When the selected protocol is SQL/DS-only, the Resource Adapter uses DSC to package the user's SQL statement into a "mailbox." Then, DSC sends the mailbox to an SQL/DS application server and waits for a reply. The mailbox is stored in one buffer. One send action is required to transmit it. When the reply is detected by the Resource Adapter, it uses DSC to receive the mailbox. The Resource Adapter unpackages the reply and moves it to the user's application area. The mailbox is received in one or more receive actions using the same buffer.

When the selected protocol is DRDA, the Resource Adapter uses DRRM to convert the user's SQL statement from its SQL/DS internal format and package it into a DRDA request data stream. This format is suitable to be sent to a DRDA application server. The Resource Adapter then uses DSC to send the request data stream to an SQL/DS or non-SQL/DS application server and waits for a reply. The data stream is stored in a list of buffers and is transmitted with exactly one send action. When the reply is detected by the Resource Adapter, it uses DSC to receive the reply data stream. The Resource Adapter then invokes DRRM which converts the data stream into its SQL/DS internal format before it is moved to the user's application area. The data stream is received in one or more buffers.

Concepts on the Application Server

When the protocol selected is SQL/DS-only, the application server uses DSC to receive the mailbox. RDS then interprets the request in the mailbox and calls DBSS one or more times to process it. Having processed the request, RDS packages the reply in a mailbox and uses DSC to send it to the application requester. The entire mailbox or a section is transmitted with each send action. If the mailbox is large, several send actions are required to transmit it.

When the protocol selected is DRDA, the application server uses DSC to receive the request data stream. The DRRM component is used to convert the request from its DRDA format into the SQL/DS internal format. RDS then interprets the request and calls DBSS one or more times to process it. If the request generates any reply data rows, this data is contained in an output mailbox. DRRM must be used again to convert the reply into the form of a DRDA reply data stream before being sent back to the application requester by DSC. The reply data stream is stored in a list of buffers (if more than one buffer is required) and is transmitted with exactly one send action.

Inter-User Communications Vehicle (IUCV) Protocol

Figure 5 on page 19 shows how an SQL/DS application server uses IUCV in a VM/XA environment. Both the application requester and the application server use the HNDIUCV macro to enable IUCV communications. This allows the application requester to connect to the application server and communicate.

Note: VM does not queue subsequent connections after the maximum number of connections has been reached.

The application requester calls CMSIUCV CONNECT to establish an IUCV connection to the application server. This causes an external interrupt to be presented to the application server. VM passes control to the External Interrupt Handler (EIH). The EIH calls CMSIUCV ACCEPT to accept the connection and allocates a pseudo-agent to represent it. IUCV reflects the connection completion to the application requester.

At this point, the application requester calls IUCV SEND (with reply) and waits for the reply. This causes a message pending interrupt to be presented to the application server. EIH gets control and posts the RECEIVE ECB (RECB) associated with the pseudo-agent. If a real agent is not allocated to the pseudo-agent, one is allocated to it if available; otherwise, the pseudo-agent is placed in the wait queue. When a real agent is allocated to the pseudo-agent the SQL/DS Dispatcher moves it to the active queue and initiates execution.

When the request has been processed, the application server calls IUCV REPLY to send the reply to the application requester. The application requester posts the SEND ECB (SECB) and proceeds to process the next request (if one exists).

When all requests have been processed, the application requester calls CMSIUCV SEVER to sever the communications link. This causes a sever pending interrupt to be presented to the application server. The EIH receives control and calls CMSIUCV SEVER to sever its half of the link. The agent is reset and reassigned to another connection.

Having severed the link, the application requester uses the HNDIUCV macro to disable IUCV communications.

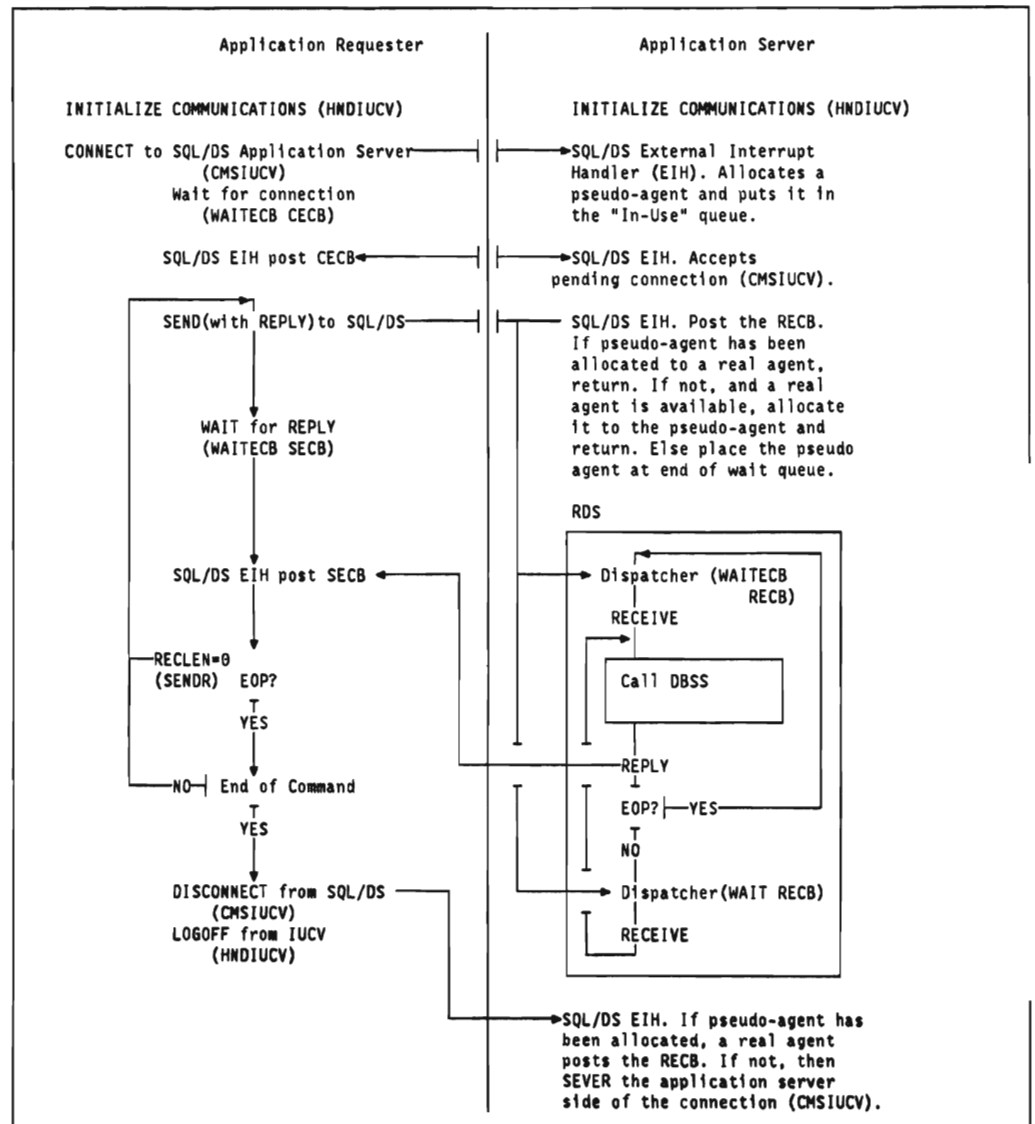


Figure 5. SQL/DS Use of the VM IUCV Functions

Advanced Program-to-Program Communications/VM (APPC/VM) Protocol

Figure 6 on page 23 and Figure 7 on page 24 show how an SQL/DS application server uses APPC/VM in a VM/SP and VM/ESA environment.

APPC/VM is not supported in a VM/XA environment. APPC/VM can be used when either the SQL/DS-only protocol or DRDA protocol is selected.

Both the SQL/DS application requester and the SQL/DS application server use HNDIUCV SET to enable APPC/VM communications. This enables an SQL/DS application requester to communicate with either an SQL/DS or non-SQL/DS application server. It also enables an SQL/DS application server to communicate with an SQL/DS or non-SQL/DS application requester.

The SQL/DS application server also establishes a connection to the *IDENT system service to identify itself as the application server for a particular resource (database). The connection is maintained until the application server is deactivated.

Note: VM does not queue subsequent connections after the maximum number of connections has been reached. But if an outbound connection is routed through an AVS gateway and all sessions are in use, AVS queues the connection until a session becomes available or until the session limit between the AVS gateway and the remote LU is increased. There is no time-out mechanism and a connection can be pended indefinitely. It is recommended that you define your session limit high enough to contain peak periods of usage.

The SQL/DS application requester calls CMSIUCV CONNECT to establish an APPC connection with an application server. This causes a connection pending interrupt to be presented to the application server. In the case of an SQL/DS application server, VM passes control to the External Interrupt Handler (EIH). The EIH calls CMSIUCV ACCEPT to accept the connection and allocates a pseudo-agent to represent it. APPC/VM reflects the connection completion to the application requester.

If an SQL/DS application requester chooses to use SQL/DS-only protocol to communicate with an SQL/DS application server in the TSAF collection or somewhere in the SNA network, the SQL/DS application requester uses the APPC/VM SENDCNF function to request an acknowledgement from the SQL/DS application server that the connection request has been accepted. The SQL/DS application server uses the APPC/VM SENDCNFD function to acknowledge its acceptance.

The SQL/DS application requester uses the APPC/VM SENDDATA function with the RECEIVE=YES option to transmit the individual request to the application server and waits for a reply. This causes an external interrupt to be presented to the application server. In the case of an SQL/DS application server, VM passes control to the EIH which posts the Receive ECB (RECB) associated with the pseudo-agent. If a real agent is not allocated to the pseudo-agent yet, one is allocated to it if available; otherwise, the pseudo-agent is placed in the wait queue. When a real agent is allocated to the pseudo-agent the SQL/DS Dispatcher moves it to the active queue and initiates execution.

For the first request in a logical unit of work (LUW), the SQL/DS application server uses the APPC/VM RECEIVE function to buffer the request. Initially it uses a default 1K buffer. If the default 1K buffer can not contain the entire request, overflow buffers are allocated and one or more calls to APPC/VM RECEIVE are

issued to buffer the entire request. Eventually, in both private and DRDA flows, the entire request is contained in one buffer. This buffer becomes the default receive buffer for the remainder of the logical unit of work or until an even larger request is received. Subsequent requests within the same LUW are automatically moved into this default buffer. If there is overflow, again the same procedures are used to buffer the entire request.

If an insufficient storage situation arises in the process of buffering the request, the SQL/DS application server uses the APPC/VM SENDERR function to purge the buffered portion of the request plus any portion pending to be received. Then it uses the APPC/VM RECEIVE function to return to Receive state so that it can process the next request.

After the request has been processed, the SQL/DS application server uses the APPC/VM SENDDATA function with RECEIVE=YES and RECEIVE=NO when appropriate to transmit the reply back to the application requester.

SQL/DS-only Protocol: The reply is stored in a 32K buffer (mailbox) in its entirety or a 32K segment at a time. If the reply is longer than 32K, several calls to APPC/VM SENDDATA are issued to transmit the entire reply to the application requester. The 32K buffer is simply reused to transmit a segment each time.

If the entire reply fits in the buffer, one call to APPC/VM SENDDATA suffices to transmit it. If an end-of-LUW condition is detected, RECEIVE=YES with a zero reply length is specified; otherwise RECEIVE=YES is specified with a reply length set to the length of the default buffer.

If the reply is segmented, APPC/VM SENDDATA is called several times with RECEIVE=NO and reply length set to zero. To send the last segment of the reply, APPC/VM SENDDATA is called with RECEIVE=YES and zero reply length (when end-of-LUW) or RECEIVE=YES and reply length set to the length of the default buffer (when not end-of-LUW).

DRDA Protocol: The reply is stored in a single linked-list of buffers. One call to APPC/VM SENDDATA with RECEIVE=YES always suffices to transmit the entire reply. If the agent is at the end-of-LUW, a zero reply length is specified to cause the application server to issue an APPC/VM RECEIVE to buffer the next request. Otherwise, a reply length equal to the length of the default buffer is specified to cause the next request to be moved automatically to the reply area. Synchronous communications are used to ensure that the buffer list is released only after the reply is received by the application requester. If while processing the request, the application server detects a severe error relating to its contents, APPC/VM SENDERR is called to purge the request and send an error indication to the application requester.

When the SQL/DS (or non-SQL/DS) application server issues an APPC/VM SENDDATA (or equivalent), an external interrupt is presented to the SQL/DS application requester. VM passes control to the EIH which posts the Send ECB (SECB). This action is treated as the completion of the original APPC/VM SENDDATA with RECEIVE=YES issued by the SQL/DS application requester to send the request to the application server. If the reply overflows the default reply buffer, then one or more calls to APPC/VM RECEIVE are issued to receive the entire reply. After the ECB has been posted, the application requester can continue processing. This involves moving the reply data to the application area and sending the next request.

When all requests have been processed, the SQL/DS application requester uses the CMSIUCV SEVER function to terminate the connection. The SQL/DS application server is presented with an external interrupt and informs the real agent, if one is allocated to the pseudo-agent, that the application requester has terminated its half of the connection. If a real agent has not been allocated to the pseudo-agent, the EIH uses the CMSIUCV SEVER function to terminate its half of the connection. Otherwise, the SQL/DS Dispatcher deallocates the real agent from the pseudo-agent and terminates its half of the connection likewise.

Finally, the SQL/DS application requester uses the HNDIUCV CLEAR function to disable APPC/VM communications.

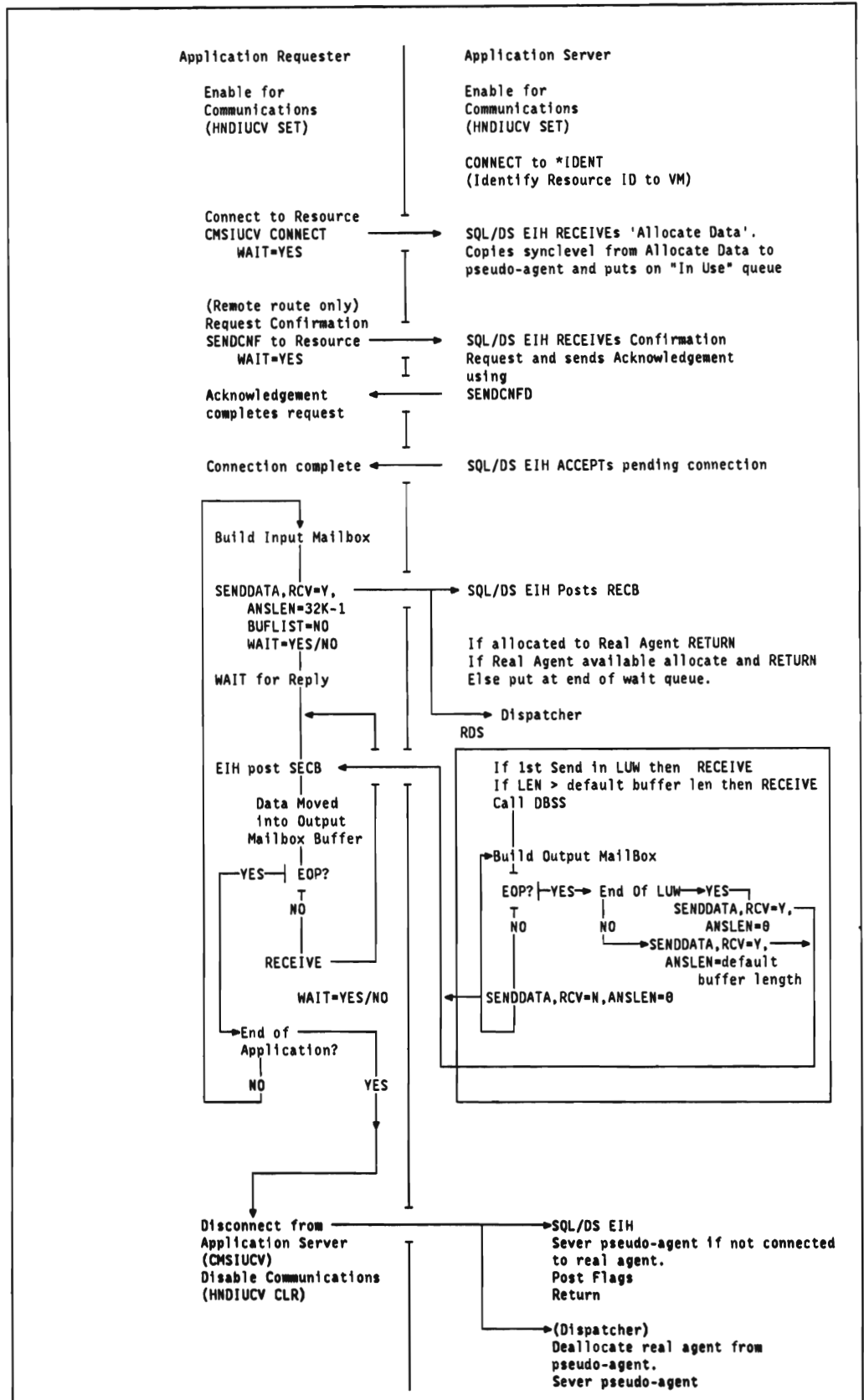


Figure 6. SQL/DS APPC/VM Communication Protocol with SQL/DS Flows.

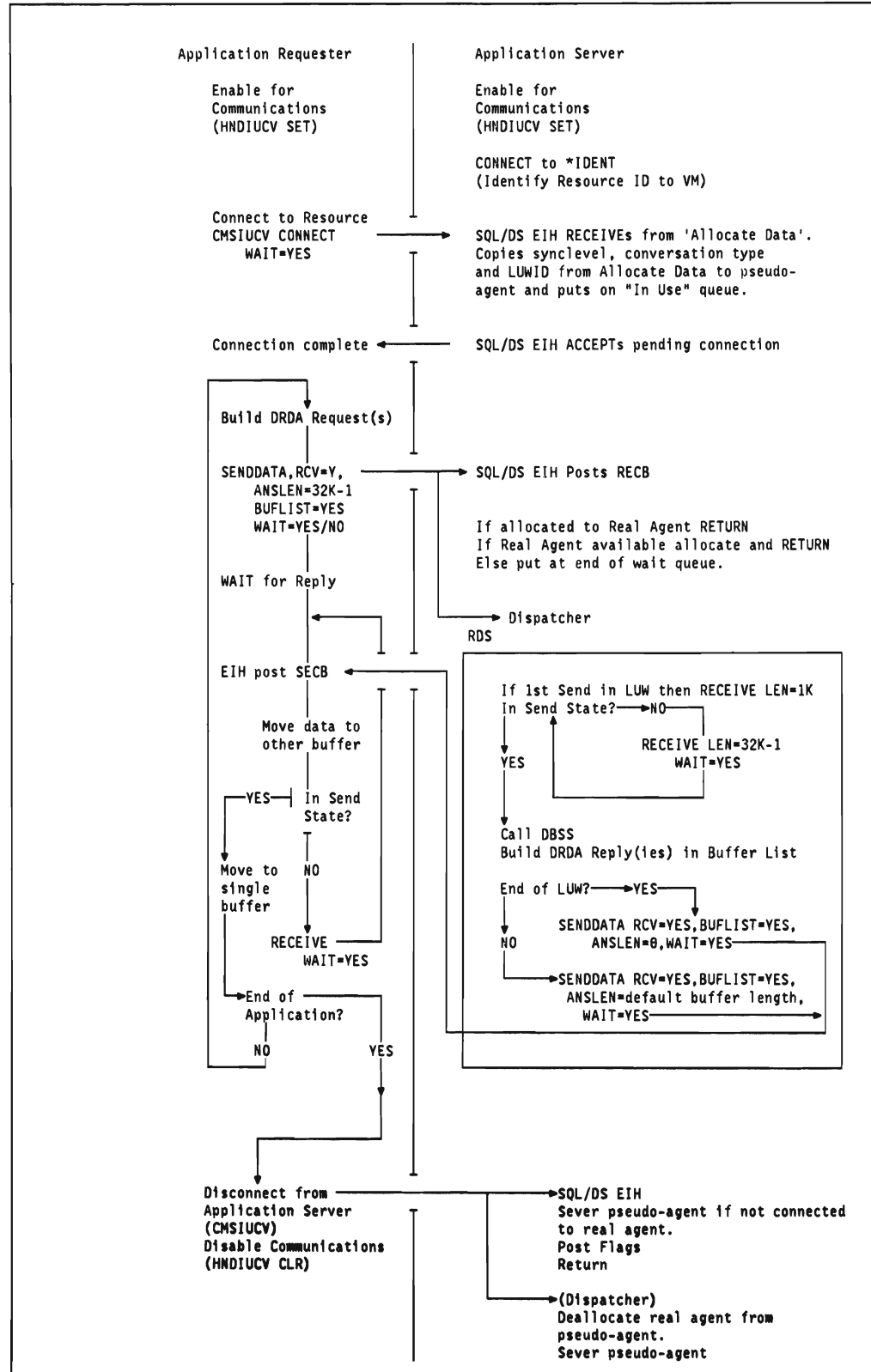


Figure 7. SQL/DS APPC/VM Communication Protocol with DRDA Flows.

Application Program Use of IUCV or APPC/VM

Use of IUCV or APPC/VM does not preclude application programs from using IUCV or APPC/VM while using the SQL/DS RDBMS. This applies to application programs running in both multiple user mode and single user mode. When invoking IUCV or APPC/VM, the application program must follow the same protocols that the SQL/DS RDBMS follows when it uses IUCV or APPC/VM:

1. Issue an HNDIUCV macro with the SET option to identify the application as an IUCV or APPC/VM program.
2. Issue a CMSIUCV macro with the CONNECT option to establish a connection to another virtual machine, and issue a CMSIUCV macro with the ACCEPT option to accept a connection from another virtual machine.
3. Perform application communications, as appropriate.
4. Issue a CMSIUCV macro with the SEVER option to terminate a connection with another virtual machine.
5. Issue an HNDIUCV macro with the CLR option to indicate that the application no longer uses IUCV or APPC/VM services.

In single user mode, the SQL/DS RDBMS uses IUCV to communicate to the VM DASD BLOCK I/O System Service. In this case, the CMS HNDIUCV and CMSIUCV macros are issued to establish the communication paths to the database devices. The Resource Adapter does not use IUCV or APPC/VM in single user mode, because it communicates directly with the application server.

In addition to using the CMS IUCV support, the user must also ensure that the MAXCONN value for the directory of the virtual machine is large enough to handle all the communication links that may be active at one time. In multiple user mode, the SQL/DS Resource Adapter requires only one communication link. In single user mode, the SQL/DS communication paths to the database devices are allocated prior to the application program being invoked. Therefore, the number of communication paths available to the application program is the MAXCONN value minus the number of database devices.

For further information on IUCV or APPC/VM, refer to the Connectivity Planning, Administration, and Operation manual for your IBM VM System Product.

Package Management Concepts

A package is the internal representation of an application program. It is made up of:

- A header, containing control information about the package, like the preprocessing options in effect.
- A series of sections. In general, a section is created for each DML or DDL statement in the application program. The section contains the internal form of the SQL statement.
- The statement itself (to be used for dynamic preprocessing).

RDIINs

The RDIIN is the control block passed to the application server by the application program. It contains all the information necessary for the application server to fulfill an application request. The RDIIN is received by the RDS component of the application server and then passed to the appropriate handler to perform the requested function.

When using the DRDA protocol, the RDIIN on the SQL/DS application requester is first translated into DDM commands by the RDIIN Manager before it is sent to the application server. Each DDM command received on the SQL/DS application server is then translated back into an RDIIN format by the RDIIN Manager.

Preprocessing

There are certain RDIIN calls that are specifically reserved for language pre-processor functions. The RDIIN call types for preprocessors and their functions are as follows:

- **Preprocessor Initialization Call:** results in the binding of the preprocessed program and its creator to an unused package. A row of the SYSACCESS table is updated and bound to the package being preprocessed.
- **SQL Statement Call:** results in the passed SQL statement being parsed and then, if appropriate, optimized and transformed into control blocks that describe how the statement is to be performed. Only data manipulation statements (SELECT, UPDATE, DELETE, and INSERT) are optimized and transformed into control blocks. In general, all other statements are considered interpretive and are stored in a parse tree format only.
- **Preprocessor Finish Call:** is issued by the preprocessor after all SQL statements in the application have been processed. It causes the finalization of the package in the case of successful preprocessing. The preprocessor then issues a COMMIT WORK to end the logical unit of work (LUW) started by the preprocessor initialization call. Unsuccessful preprocessing results in a rollback of the preprocessor logical unit of work.

There are four basic functions performed during SQL statement call processing: Parsing, Optimization, Access Generation, and Statement Generation.

- **Parsing**

The parsing function does the primary syntactical analysis of the user's SQL statement and converts it into an internal form called a parse tree. It is used either at preprocessing time (invoked by the preprocessor) or at run-time (for dynamic statements).

- **Optimization**

The objective of the optimization function is to prepare a single SQL statement for execution. Input consists of the parse tree for that statement, created by the parsing function. The following functions are performed during optimization:

- **Authorization checking:** The authorization control function is invoked to ensure that the requestor of an operation has the appropriate authority. Authorizations to perform specific operations on tables (for example, SELECT, INSERT, DELETE, UPDATE) are program dependencies and are recorded in the system table SYSTABAUTH.

- Symbol resolution: Names of tables, views, and columns are verified against the contents of the catalog. Internal identifiers of such are saved for use in preparing the run-time structures to communicate with DBSS. Program dependencies for object existence are stored in the SYSUSAGE system table.
- Semantic checking: The SQL statement is checked for validity of meaning. For example, the operands of each comparison operator are checked to see that they are comparable.
- Access path selection: A plan for executing each database request is determined by considering both the access paths (including indexes) available and statistics on the data to be accessed.

For more information on access path selection, see the *Performance Tuning Handbook*.

- Access Generation

During access generation the output from the optimization phase is transformed into control blocks that describe how the statement is to be executed. Only data manipulation statements (SELECT, UPDATE, DELETE and INSERT) require access generation.

- Statement Generation

Statement Generation occurs when an SQL/DS data manipulation statement is preprocessed. Currently, statements are generated to enforce referential integrity and the CHECK option in updateable views.

For referential integrity, the Statement Generator has three main functions:

1. It creates an SQL/DS data manipulation statement.
2. It changes the SQL statement into an SQL/DS internal representation (a section of a package) by invoking the optimizer, and access generator.
3. It combines the original SQL/DS statement with the internally generated statement.

For enforcing the CHECK option in updateable views, the Statement Generator does the same as for referential integrity except that it does not work on a complete data manipulation statement. Instead, it works only on the predicates that are to be checked. The predicates are defined in the views which are created with the CHECK option. In this case, the Statement Generator does the following:

1. It creates a SELECT statement for the predicates to be checked.
2. It changes the predicates of the SELECT statement into an internal representation (run-time tables which process the predicates) by invoking the parser, optimizer, and access generator.
3. It attaches the run-time tables of the predicates to the run-time block of the original SQL/DS statement.

Execution-Time Processing

An application program communicates with the database manager using code placed in the application by the SQL/DS preprocessor, which replaces the original SQL statements in the application. This code passes an RDIIN control block to the database manager identifying the function to be performed. Part of the RDIIN identifies the package and its section. The identified section within the package can be empty (an indefinite section) or can contain control blocks that describe how the statement is to be performed (a compiled section) or a parse tree (a parsed section or an interpretive section).

- **Static Statements**

Most statements embedded in an application program are static statements. These statements are converted into compiled sections or interpretive sections.

- **Parsed Section**

Parsed sections result from an SQL statement that causes a preprocessing warning (for example, it references a table that does not yet exist). For such statements, the SQL statement is parsed and the result is stored as a section in the package. The intent is to allow the user to correct the warning condition before the program is executed. When a parsed section is executed, it is handled somewhat like an EXECUTE IMMEDIATE statement from optimization through execution; the stored parsed data is treated as a dynamic statement starting at the optimization step because the parsing was already done at preprocessing time. If it is a frequently executed SQL statement, and if performance and storage requirements are important, consider reprocessing the program after the object is created or after the user is properly authorized.

- **Dynamic Processing**

Applications can be written to dynamically preprocess SQL statements using a PREPARE or EXECUTE IMMEDIATE statement. An application containing these statements has an indefinite (or empty) section created in the package for each different PREPARE statement and one for all EXECUTE IMMEDIATE statements.

When a package with an indefinite section is loaded, an actual section does not exist. As soon as a PREPARE or EXECUTE IMMEDIATE statement is executed, the actual section is generated and stored so that a subsequent user's EXECUTE (or an implicit one, in the case of EXECUTE IMMEDIATE) can be processed successfully.

- **Extended Dynamic Processing**

There are two types of packages created by Extended Dynamic Statements. These types are known as modifiable and nonmodifiable.

A nonmodifiable package cannot be altered after it is created and committed. Sections can be added to the package in a consecutive manner using an extended PREPARE statement, but the sections can be added only in the same logical unit of work in which the CREATE PROGRAM has been executed to create the package. The sections cannot be executed until the package is committed. This type of package is created in the same fashion as those created by the Assembler, C, PL/I, and COBOL preprocessors with a CREATE PROGRAM statement analogous to a PREP INIT Call, the EXTENDED PREPARE statement analogous to the PREP SQL Call, and the

COMMIT/ROLLBACK statement analogous to the PREP FINISH Call. The FORTRAN and RPG preprocessors create this type of package.

The other type of package is a modifiable package. This type of package may have sections added to or deleted from it using the extended PREPARE and DROP STATEMENT respectively. Additionally, the sections of the package can be executed in the same logical unit of work in which the package was created.

Note: The support for extended dynamic processing is restricted in the DRDA protocol environment. For more information on the restrictions, see the *System Administration* manual.

Package Cache Management

The RDS component manages and keeps track of the status of packages that have been loaded from the database into memory. The package cache, named PROGS, is used to hold information about all packages that have been loaded.

The package cache is an array of elements. The number of elements in the package cache is calculated using two initialization parameters, NPACKAGE and NCUSERS:

number of elements in package cache = NPACKAGE x NCUSERS

NPACKAGE defines the maximum number of packages in a logical unit of work. The default value of NPACKAGE is 10. NCUSERS represents the number of concurrent active users. The default value of NCUSERS is 5. The default number of elements in the package cache is 50.

During a logical unit of work, each package is loaded from the database into memory, if it is not already available in memory, and is assigned an element in the package cache. Each logical unit of work maintains its own chained list of loaded packages.

The package cache has a threshold. The purpose of the threshold is to control the amount of storage consumed by the loaded packages. The threshold is derived using the following calculation:

threshold = # of elements in the package cache x NPACKPCT / 100

The default value of NPACKPCT is 30, the default threshold is 15.

At the end of the logical unit of work, if the threshold is exceeded, the assigned elements that map to the loaded package are freed and returned to the package cache for use. This continues until either all elements assigned to a logical unit of work have been freed, or until the number of assigned elements drops below the threshold.

If the threshold is not exceeded, the package stays in memory but is no longer associated with the LUW that just completed.

Changing the initialization parameters NPACKAGE, NCUSERS and NPACKPCT can impact performance. For more information, see the *Performance Tuning Handbook* manual.

Repreprocessing

Repreprocessing, or "reprep," is initiated when a package is loaded and is found to be invalid. Execution of a RELOAD PROGRAM command will also force a reprep to occur. This reprep is transparent to the user's application except that, after the package is invalidated, extra time is required to reprep the program the first time the application is run. This is because during repreprocessing, each SQL statement stored in a package is preprocessed and the resulting generated code or parsed output replaces its predecessor. This is a significant function of package management.

When packages are created, an entry is made into the table SYSUSAGE for every database object on which the package depends. DBSPACES, tables, and indexes are examples of such objects.

When any object is dropped from the database, SYSUSAGE is scanned to determine which packages are dependent on the existence of the object. Packages that are found to have dependencies on the dropped object are marked invalid. SQL/DS also checks SYSTABAUTH to find and invalidate packages when the necessary privilege or authority of the creator of the package has been revoked. (Privileges granted to a package are those granted to the creator of the package.)

Marking a package invalid implies that its SYSACCESS table row is updated so that the column named VALID is updated to a value of N. Additionally, the entries in the package cache are affected. Occurrences of the affected package tied to an active logical unit of work have their cache entry marked invalid so that they are removed from the cache at the end of their logical unit of work.

The effect of marking the package invalid is that the package has to be loaded the next time it is needed by an application at the beginning of a logical unit of work. During the load process, the database manager recognizes that the package is invalid and initiates the repreprocessing. At successful completion of this transparent reprep, the package is marked valid again in its SYSACCESS row.

Authorization

Authorizations are classified into two categories, (1) privileges and (2) system authorities.

Privileges are the capabilities you possess to perform specific operations on tables and views (for example, SELECT, INSERT, DELETE, and UPDATE) or the RUN privilege for a package. Authorities are the levels of authorization you possess in the SQL/DS database (for example, CONNECT, RESOURCE, SCHEDULE, and DBA). Privileges and authorities are recorded in the following system catalog:

- SYSTABAUTH contains table and view privileges
- SYSCOLAUTH contains the column privileges when UPDATE is granted on specific columns and not on the whole table
- SYSPROGAUTH contains the RUN privilege for packages
- SYSUSERAUTH contains the authorities that a user has.

Privileges and authorities of static statements are checked at preprocessing time. Privileges and authorities of dynamic statements are checked at execution time.

When a package is preprocessed, the authorization component determines the RUN privilege that should be given to the creator. This depends on the creator's existing privileges and authorities as well as the type of statements in the package. Refer to the *Application Programming* manual for more detail.

Storage Management Concepts

This section discusses several different types of storage management concepts:

1. Memory Management

Virtual memory, or virtual storage, is the storage required during execution. It is required, for example, to hold packages, agent structures and dynamically assigned variables.

2. Logical Storage Management

The actual data in the database is stored in DBSPACES. A DBSPACE is a portion of the database that can contain one or more tables and their associated indexes. A DBSPACE, however, is not a physical space. It represents a logical allocation of space in a storage pool. Each DBSPACE is assigned to a specific storage pool.

3. Physical Storage Management

DBEXTENTS are the physical medium where the data in the database is stored. Each DBEXTENT is a VM minidisk. A storage pool is a collection of one or more DBEXTENTS.

Memory Management Concepts

Storage Services:

System storage is used for two things: stack storage and working storage. Storage is obtained and released by calling SQL/DS modules that execute the host system's storage request macros DMSFREE, DMSFRET, and CMSSTOR.

To judge current and future storage requirements, and to assist in problem determination, use the SHOW STORAGE command. This command displays information about the system storage currently in use as well as maximum total storage usage. For information on how to use this command to monitor and assess storage usage, see the *Performance Tuning Handbook*.

Stack Storage:

Stack storage functions allocate the dynamic storage required by a module during its execution, that is, storage required by its dynamically assigned variables. This storage is allocated when the module is invoked and deallocated when the module returns to its calling module.

When a called module requires more stack storage than is available in the current stack, a stack extension occurs, allocating a minimum of 12K of storage. This stack extension is added to the end of the stack chain and pointed to as the current stack. Subsequent calls to other modules cause them to obtain their

storage from the stack extension (unless another stack extension occurs). As each module returns to the caller, the stack extension is freed and removed from the stack chain.

Working Storage:

The working storage functions allocate storage for an agent structure. This storage is more permanent than stack storage; it is typically allocated and used across module calls. When an agent structure is created, it is allocated two initial working storage areas that are placed on two free queue chains. One queue is for requests for storage that must be below the 16M line. The other queue is for storage that can be above the 16M line. Initially, the entire working storage areas are available for suballocation. Additional working storage extensions are allocated and freed as required. At end-of-LUW, all remaining working storage extensions are returned to the host system.

When a module requests working storage, from one of the two areas, storage is removed from the appropriate working storage area and allocated to the requester. The remaining storage stays on that free queue chain. Whenever a module frees an area of working storage, it is returned to the working storage area from which it was suballocated by inserting it into the free queue chain.

Subsequent requests for working storage attempt to get storage from one of the working storage areas by navigating its free queue chain until it finds an area large enough to satisfy the request. If a large enough area cannot be found, a request is made for additional storage. This additional storage, called a working storage extension, is placed on the appropriate working storage extension chain and its free storage (if any) is added to the corresponding free queue chain. Additional working storage extensions are added to the end of this extension chain. In some cases if all the area in a working storage extension becomes free, it is returned to the system at that time. At end-of-LUW, all remaining working storage extensions are returned to the host system.

There is also prototype working storage. This storage is managed much like regular working storage but it is not freed at end-of-LUW. Prototype working storage extensions are chained to the working storage extension chain of the prototype agent rather than that of the agent requesting the storage.

Storage for each loaded package is also maintained in a working storage queue. All extensions and the initial pool are freed to the system when the package is purged from memory.

Logical Storage Management Concepts

A DBSPACE is logically divided into three sections: header pages, data pages and index pages. The size of each of these sections is determined when the DBSPACE is ACQUIRED. Logical pages from these sections are allocated from the top of each section when they are required.

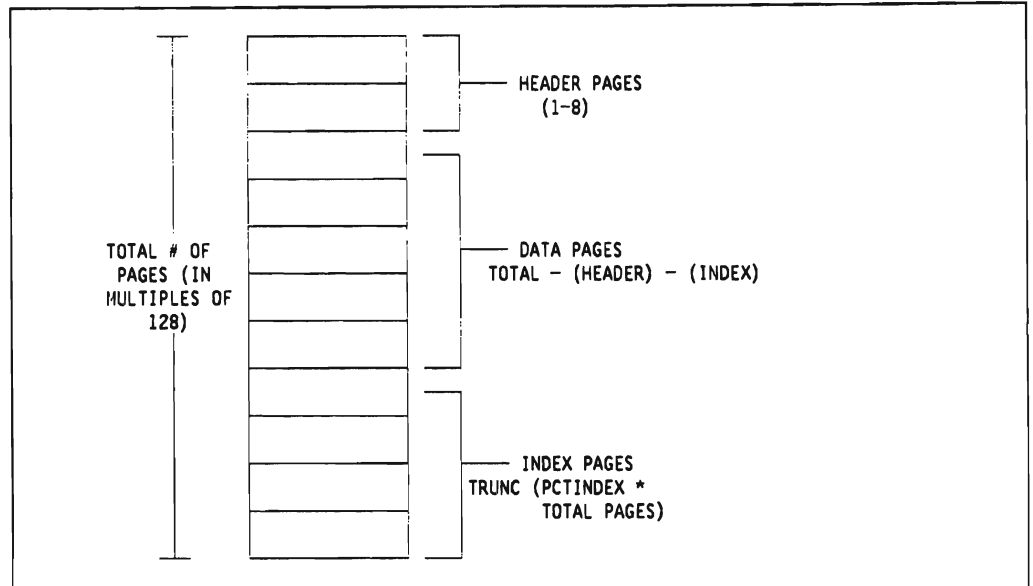


Figure 8. DBSPACE Structure (Logical View)

A page in a DBSPACE is 4096 bytes in size. Sixteen of these bytes are reserved for the page header. Since SQL/DS does not support rows that span DBSPACE pages (except when using long fields), the maximum row length is 4080 (including row overhead).

The following diagram shows the format of a data page. The page slots at the end of the page contain offsets to the rows on the page. A row is uniquely identified in a DBSPACE by a tuple identifier (TID) which consists of the page number of the page on which it resides and the number of the slot which points to the row.

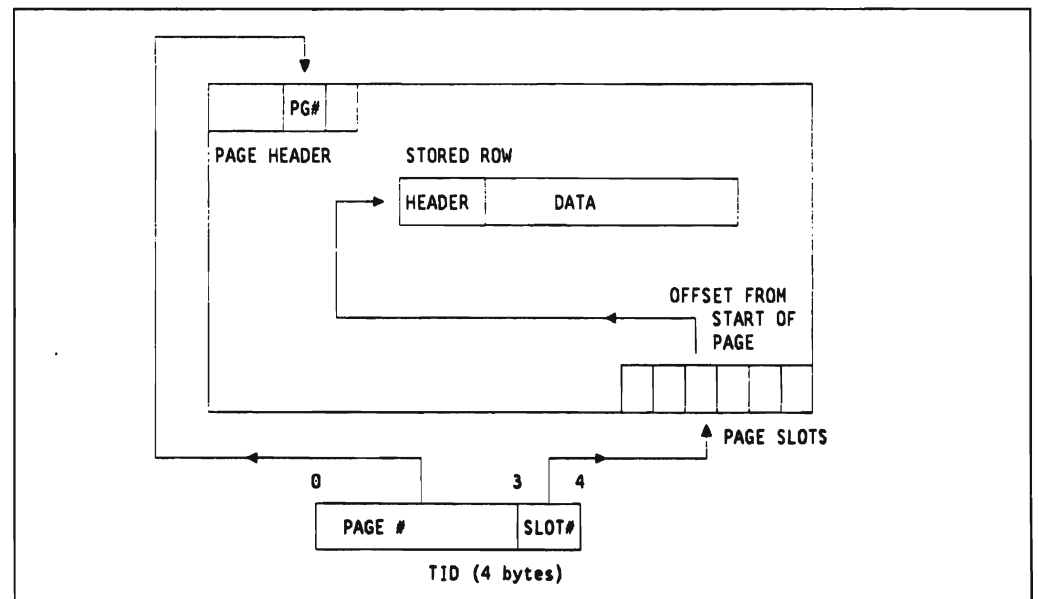


Figure 9. Layout of a DBSPACE Data Page

PCTFREE is the *minimum* free space to be reserved on each data page of the DBSPACE on an INSERT operation.

The following objectives are factors in free space management:

1. Preservation of at least PCTFREE free space on an INSERT
2. Minimize overflow due to row expansion
3. Minimize I/Os to find room for an INSERT.

The actual free space seldom works out to be exactly the PCTFREE specification. For INSERT operations, SQL/DS overallocates free space with one exception: SQL/DS always allows at least one row on a page regardless of the PCTFREE specification.

For example, a table with fixed length 3K byte rows always results in approximately 25% free space. Even if you specify PCTFREE = 10, you get 25% actual. SQL/DS can put only one 3K row on a 4K page. If you specify PCTFREE = 50, SQL/DS will still insert the 3K row even though the 50% free space request cannot be honored.

The purpose of PCTFREE is to minimize overflow due to row expansion. When UPDATE commands are executed on an existing row and the length of the row increases, the row could expand into the free space reserved with PCTFREE. If the expansion exceeds the free space, the page becomes full, and it causes an overflow. The row is relocated to a new page and a pointer chaining to the new location is set in the old page. If this row has to be moved again, the pointer in the original page is set to mark the newest location. Therefore, SQL/DS never reads more than two pages for one row, (except when using long fields).

To minimize I/O operations in finding room for an INSERT, the system must have some notion about the free space of the DBSPACE pages that can accommodate the row to be inserted. To achieve this, SQL/DS maintains summary information in the DBSPACE Page Map tables. DBSPACE Page Map tables are blocks in the Directory, each block having entries for 128 consecutive pages. In particular, each page map table entry contains a FREE CLASS designation that identifies the range of free space (bytes) available on the referenced page. The relevant free classes that a page might have are identified in Figure 10.

Figure 10. Free Classes

FREE CLASS	MIN FREE	MAX FREE
2	0	14
3	15	29
4	30	49
5	50	99
6	100	249
7	250	499
8	500	999
9	1000	1999
10	2000	4017
11	4018	4077
12	4078	4078

To determine whether or not a page qualifies for insertion of a row, SQL/DS looks at the FREE CLASS of the page and checks the following condition to qualify the page:

$$\text{ROWLENGTH} + \text{PCTFREE} * 40 \leq \text{MIN FREE}$$

(The value 40 is from 4080/100.)

If the condition is true, the page qualifies and is used for the insertion. That is, if the condition is true, insertion of the row does not compromise the current PCTFREE specification.

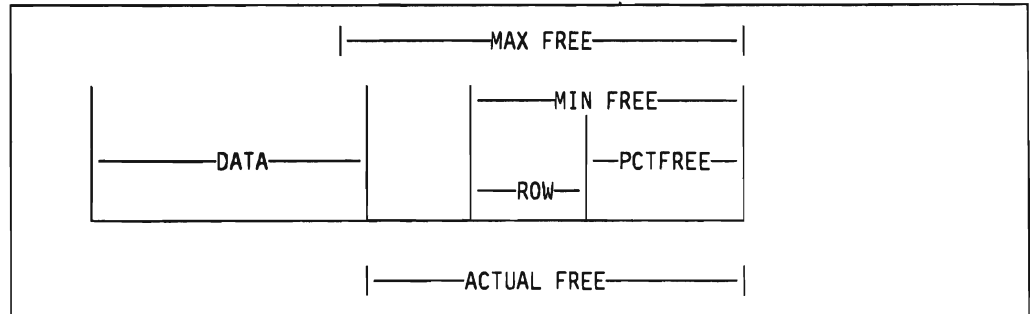


Figure 11. Qualifying a Page Based on FREE CLASS

Figure 11 illustrates the case where the row fits in the difference between MIN FREE and PCTFREE. Notice that the use of MIN FREE is a conservative check; the actual free space is somewhat larger.

Figure 12 illustrates the results of the insertion and an attempt to insert another row.

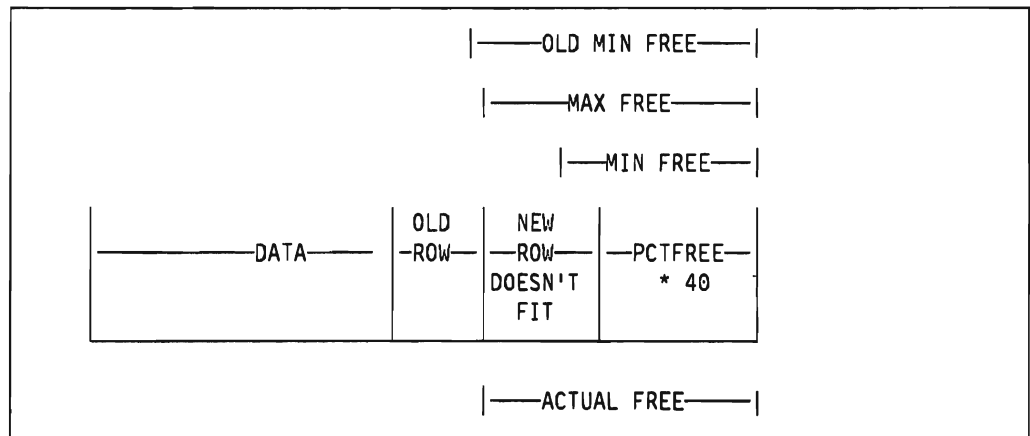


Figure 12. State of the Page after the Insertion

As a result of the insertion, the page had to be assigned a new FREE CLASS. In this illustration, the new FREE CLASS has a MAX FREE value that is one byte less than the old MIN FREE. Note that if another INSERT is attempted with a row as shown, the page no longer qualifies for an insertion ($\text{ROWLENGTH} > \text{MIN FREE} - \text{PCTFREE} * 40$).

Figure 12 also shows that the row would have actually fit without compromising the PCTFREE, even though this page did not qualify for the insertion. Because FREE CLASS does not identify the exact amount of actual free space, the data-

base manager cannot determine precisely whether a particular row fits. The result of this design is that the actual free space may be somewhat larger than the amount suggested by PCTFREE. The degree to which the database manager overallocates free space depends on the PCTFREE specification, the lengths of rows being inserted, and the FREE CLASS range involved.

Physical Storage Management Concepts

Directory, LOGs, and DBEXTENTS:

The Directory, LOGs, and DBEXTENTS are reserved minidisks with a blocksize of 512 bytes for the Directory and 4096 bytes for the LOGs and DBEXTENTS. These minidisks have CMS-like files that are in a format to be used with the VM BLOCKIO process. The VM BLOCKIO process is used to read and write records to these files. These minidisks are called reserved because they have been processed by the CMS RESERVE command. The RESERVE command specifies that the minidisk consists of a single CMS file, which is allocated using all available disk blocks. This CMS file cannot be processed by most CMS file system commands and must never be modified, except by the database manager. For an example of physical database concepts, see the *System Administration* manual.

Mapping of DBSPACES to DASD:

Logical DBSPACES must be mapped to physical DBEXTENTS on DASD. SQL/DS does this by maintaining a page map table, for each DBSPACE, which is used to map a given DBSPACE page to its location on DASD. The page map table is a collection of constant size blocks (512 bytes) in the directory. Each entry of a page map table is four bytes. Thus, the size of a DBSPACE is rounded up to the nearest multiple of 128 pages ($512/4 = 128$). Each logical page of a DBSPACE takes eight bytes of Directory space: four bytes for the current version of the page and four bytes for its shadow page. Shadow pages are discussed in "DBSPACE Recovery" on page 38.

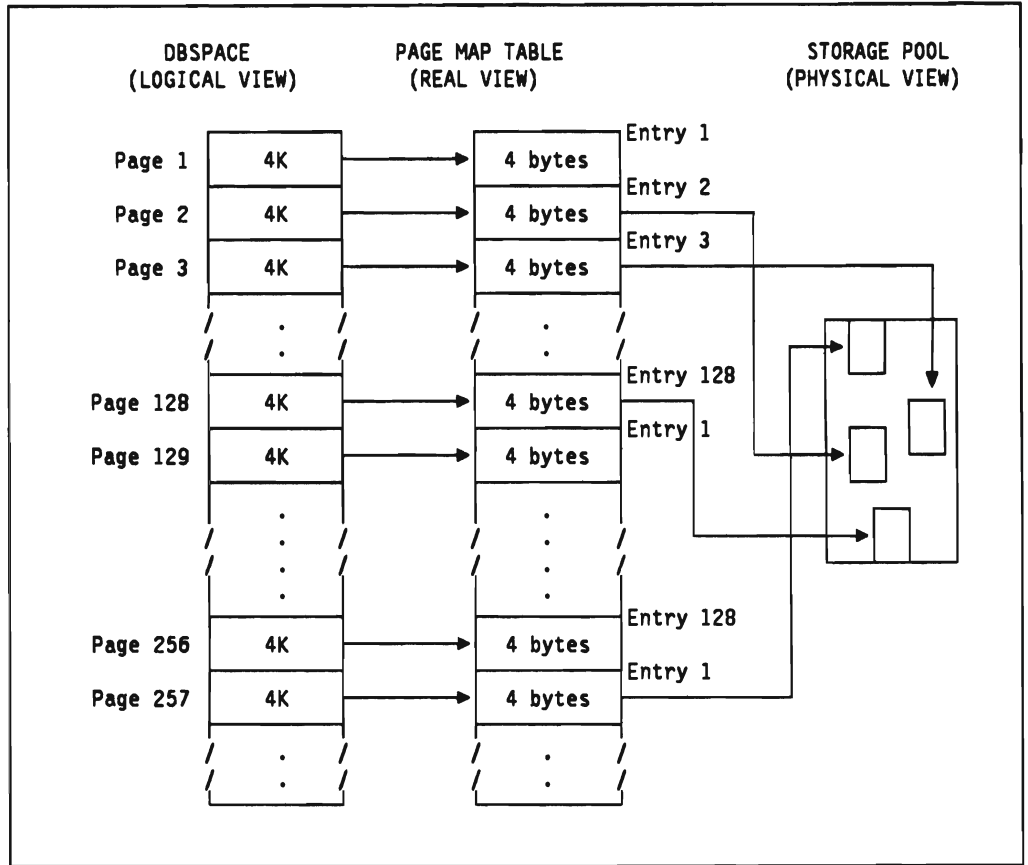


Figure 13. Mapping of DBSPACES to DASD

Logical To Physical Page Relationships: Physical page slots in the storage pool are allocated to the DBSPACES dynamically upon first reference. Once a logical page has had a physical page slot allocated to it, it will continue to have a physical page allocated, even if empty, until the DBSPACE is dropped. This is illustrated in the following example.

Example: A SHOW DBSPACE indicates number of pages occupied = 2000 and number of empty pages = 29000.

This means that 29000 pages are allocated and are all free space. New pages are required from the DBEXTENT when pages are needed for shadow page use. Shadow pages are given back to the storage pool at checkpoint time.

Example: A SHOW POOL indicates total pages in use = 32000

This example shows that 32000 pages are in use. Assuming that the only DBSPACE in this storage pool is the one in the previous example, only 31000 pages are actually assigned to the DBSPACE. This means that 1000 pages are in use as shadow pages, which will be released at checkpoint time. (For more information regarding shadow pages, refer to "DBSPACE Recovery" on page 38.)

Storage Pools:

A storage pool is a collection of one or more DBEXTENTS, which can be used to control the distribution of the database across DASDs. The maximum number of

storage pools for a given database is specified by the database generation keyword MAXPOOLS. A storage pool does not exist until a DBEXTENT is assigned to it. DBSPACES are assigned to a given storage pool when they are defined. That means when physical page slots are allocated to the DBSPACE, they are allocated from the storage pool to which the DBSPACE belongs. In addition, if the storage pool contains more than one DBEXTENT, a physical page slot may be allocated from any of these DBEXTENTS.

DBSPACE Recovery:

The DBSPACE recovery mechanism is the use of "shadow pages." Two page map table entries are associated with each permanent (not internal) DBSPACE. The entries are called "current" and "shadow." The shadow page contains the original page data at the time of the last checkpoint, and the current page contains all updates made to the page since the last checkpoint. (See "Logging/Recovery Concepts" on page 46 for a discussion of checkpoints.)

After a checkpoint, the current and shadow entries are identical. When there is a request to update a page, and it is the first request since the last checkpoint, a new physical page slot is allocated from the storage pool, and the current page map table entry is set to the new page location. When the page is written to DASD it is directed to the new location, whereas the shadow page and the shadow page map entries are left intact. At a checkpoint, the physical pages bound to the DBSPACE are brought up to date by writing out all buffer pages that have been updated. Effectively, the shadow page map entries are set equal to the current page map entries, and the physical pages in the shadow page map entries that have been changed are released.

Note that only one additional page is allocated for all updates made by any LUW since the last checkpoint. Also, the pages are physical pages from the storage pool and do not deplete the available data pages of the DBSPACE.

Reserved Pages:

Twenty pages are reserved in each storage pool to enable recovery from situations where the storage pool becomes full. In addition, if more than 20 new pages in a recoverable storage pool are used since the last checkpoint and logging is being performed, the database manager reserves one page in that storage pool for each new page used in excess of 20 pages. The reserved pages in excess of 20 are freed at checkpoint time. If an LUW requires a new page which would cause the number of reserved pages to exceed the number of free pages in the storage pool, a storage pool full condition occurs and the LUW is rolled back. During rollback the reserved pages may be used. If the number of free pages in the storage pool reaches 10 or less during rollback, a checkpoint is triggered. This enables the database manager to recover shadow pages and allows the rollback to continue without running out of pages in the storage pool.

For nonrecoverable storage pools, or when logging is not being performed (LOGMODE=N), a maximum of 20 pages are always reserved.

Buffer Storage Management Concepts

The Storage component manages the allocation of virtual storage buffers for data pages and Directory blocks. The number of page and Directory buffers are specified by the SQL/DS initialization parameters NPAGBUF and NDIRBUF respectively. Pages and blocks are fetched and fixed to a particular buffer until the agent is through referencing data and explicitly unfixes the page or block buffer for reuse. Modified pages or blocks are not written to DASD at "unfix" time but are written before the buffer is reused for another page or block. The buffer replacement strategy incorporated is a least recently used algorithm. This is predicated on the assumption that a recently used page is most likely to be referenced in the near future, thus eliminating the overhead of I/O to refetch it from DASD.

Buffer storage management works differently with the VM Data Spaces Support feature. For more information, see the *Performance Tuning Handbook* and the *VM Data Spaces Support* manual.

Index Concepts

This section describes the internal format of SQL/DS indexes. It also describes the two important aspects of indexes from the user's point of view: fragmentation and clustering.

Basic Index Structure

SQL/DS indexes are B-tree structures as illustrated in the following diagrams.

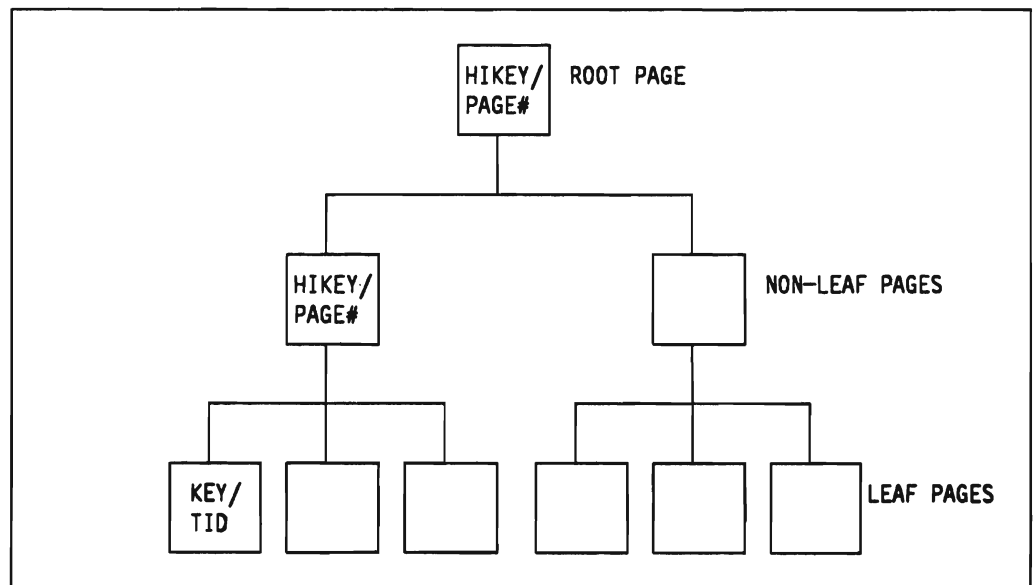


Figure 14. The Balanced Tree Index Structure for Unique Indexes

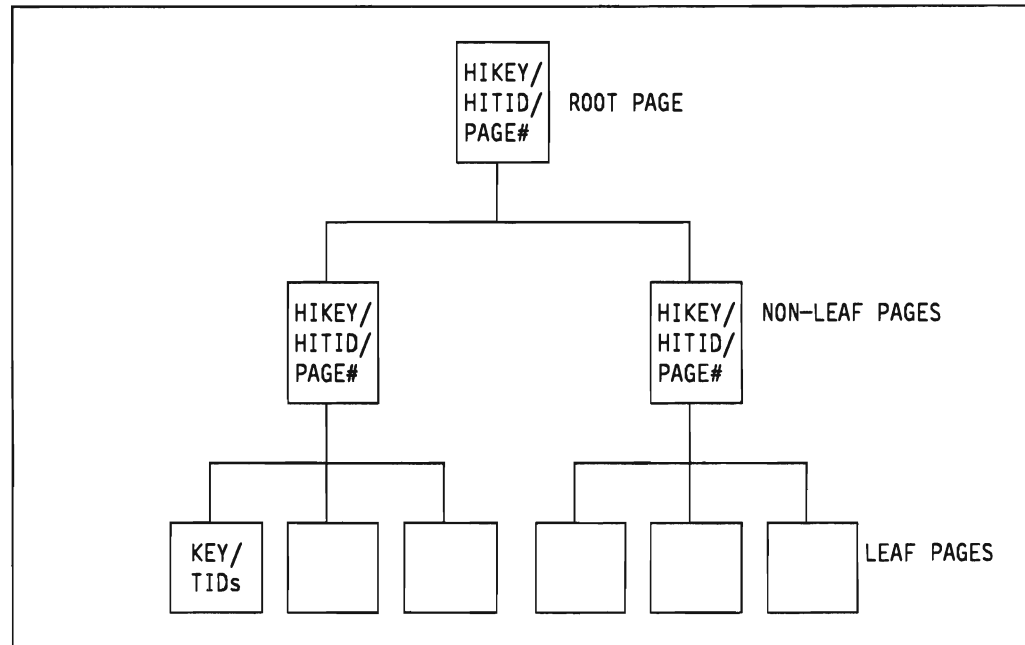


Figure 15. The Balanced Tree Index Structure for Nonunique Indexes

In Figure 14 on page 39, the root page would contain the high key value and page number for each of the nonleaf pages at the next (second) level. Similarly, the nonleaf pages would contain the high key value and page number for each of the leaf pages. The leaf pages would contain the uncompressed key and the TID of the row in the table.

In Figure 15, the root page would contain the high key value, the corresponding high TID value, and the page number for each of the nonleaf pages at the next (second) level. Similarly, the nonleaf pages would contain the high key value, the corresponding high TID value, and the page number for each of the leaf pages. The leaf pages would contain the uncompressed key and the TIDs of the rows in the table.

A balanced tree structure means that the number of pages read to traverse the index from the root page to any one of the leaf pages is the same. It does not necessarily mean that the keys are balanced across the pages.

Index Space Management

The Index component manages the space in its index pages. Unlike the Data Manipulation component, which searches for free space in an available data page to store a row, the Index component knows where the submitted key belongs. The keys are stored in a particular physical order (ascending or descending), and a particular key value has a specific position in an index page.

When an index page becomes full with key/TID pairs (or page numbers for nonleaf pages) and another key is to be inserted in the page, a new page is allocated and the full one is split. The low-key-range half of the page is moved to the new page and the high-key-range half is moved to the top of the previously full page.

If all keys that reside on an index page are deleted, the page is logically empty but cannot be reused for other key values. Index pages that have been allocated

to an index are normally reclaimed only when the index is either dropped or reorganized. The exception to this occurs when there are no more free pages available and a page is needed during ROLLBACK or UNDO WORK operations. In this case, an attempt is made to reclaim empty index leaf pages and merge partially full index pages to enable rollback to continue. Index pages are not reclaimed for DBSPACE SYS0001 (which contains the catalog tables), nor are the indexes on catalog tables marked invalid.

Invalid Indexes

An index can become invalid in the following ways.

- During a ROLLBACK or UNDO operation, if the database manager requires a free index page but is unable to reclaim any, the index is marked invalid. More than one index can become invalid during the LUW. SQL/DS rollback, UNDO, or REDO processing continues, but no updates are made to invalid indexes, and thus they no longer reflect the data. These indexes cannot be used until they have been reorganized, or, dropped and recreated.
- An index can be marked invalid if duplicates have occurred in a unique index. This can only happen if:
 - a checkpoint occurs during a searched UPDATE deferring checking of uniqueness,
 - a system failure occurs before the end of the statement, and
 - the database is started with an empty log.

At the end of initialization, any unique indexes that contain duplicates are marked invalid.

- An index can also be marked invalid if the following events occur in order:
 - A checkpoint occurs during a CREATE or REORGANIZE INDEX
 - A system failure occurs before the database manager can complete the CREATE or REORGANIZE statement
 - The application server is restarted with an empty log.

When an index is marked invalid, packages that use that index are not marked invalid; however, the packages will become invalid if the index is dropped. If the index is reorganized, the packages will remain valid.

Additional details about invalid indexes can be found under the SHOW INVALID command in the *Operation* manual.

Transient Indexes

An index can be marked transient in the following ways.

- An index is marked transient during a CREATE INDEX statement or REORGANIZE INDEX command. In this case, the index remains transient for the duration of the statement. When the index has been created or reorganized successfully, the index is marked valid.
- A unique index can be marked transient during a searched UPDATE statement where uniqueness checking is being deferred. In this case, the index remains transient for the duration of the LUW. The index is marked transient when the first duplicate is inserted. When the statement is completed, if duplicates still exist SQLCODE -803 is issued, and the UPDATE statement is rolled back. The index is marked valid at the end of the LUW.

Additional details about transient indexes can be found under the SHOW INVALID command in the *Operation* manual.

Clustering Index

A clustering index is used to determine placement of rows in pages of a DBSPACE. The first index created on a table is, by default, the clustering index. SQL/DS uses this index to attempt to put rows with similar index key values onto the same data pages.

When data is inserted into a table, there are two strategies for finding a place for the data in the DBSPACE: default logic and clustering index logic.

If the CLUSTERTYPE column in SYSTEM.SYSCATALOG for the table contains a "D," the default logic strategy is used. This strategy uses the value in the column CLUSTERROW in SYSTEM.SYSCATALOG for the table to determine the starting point to look for available space for the insert. The value in CLUSTERROW is a pointer to the end of the table. If the value in CLUSTERROW is significantly incorrect, the database manager has to do extra work to find a page that has sufficient free space to hold the row to be inserted. The value of CLUSTERROW can be significantly incorrect if UPDATE STATISTICS has not been executed recently or an application program that is doing the insert has not been preprocessed (prepped) recently. Because a preprocessed program that inserts with the default logic stores the value of CLUSTERROW in the package, you must periodically reprocess this kind of program to update the CLUSTERROW value in the package.

The clustering index strategy is used when there is a clustering column in SYSTEM.SYSCATALOG. This strategy attempts to place the new row on the same page as rows with similar key values. This determines the starting point to look for available space for the insert. If there is no available space on the pages at or near this starting point then the database manager must do additional work to find a page that has sufficient free space to hold the row to be inserted. Insufficient free space can occur because no free space was established for the DBSPACE or because inserts have used all the free space. If you reorganize the DBSPACE you can establish free space for inserts.

When you create a table, CLUSTERTYPE is set to "D" and CLUSTERROW is set to zero. When you create the first index on a table, CLUSTERTYPE is set to "I." If you REORGANIZE the clustering index it will remain the clustering index. If you drop the clustering index, CLUSTERTYPE is set back to "D." To establish a different index as the clustering index you must drop all indexes on the table, create the new clustering index as the first index, and then create any other indexes.

Clustered Indexes

An index is classified as either CLUSTERED or NOT CLUSTERED. An index is considered clustered if the data is physically stored in the DBSPACE in an order which closely matches the key sequence of the index. This means that the data can be sequentially retrieved using the index with a minimal number of I/Os.

Assume that all rows of a table are to be retrieved. A measure is taken of the data page referencing pattern that occurs in terms of data pages referenced. A

count is incremented whenever the immediately preceding reference to a row was to a different data page.

In the best case, the number of pages to be accessed is exactly equal to the number of pages occupied by that table within the DBSPACE. A data page is read, all the rows of the subject table in that page are retrieved, and then the next page is read. In this case, the pages are read sequentially - each page read only once.

A clustered index can be identified by either the CLUSTERRATIO or the CLUSTER column in the SYSINDEXES catalog table.

The CLUSTERRATIO value is used by the optimizer to choose a suitable index for access path selection. This value represents a percentage, with the two decimal places implied. The value is calculated by:

$$\text{CLUSTERRATIO} = 10000 * \frac{\text{ROWCOUNT} - \text{PAGE JUMPS}}{\text{ROWCOUNT} - \text{PAGE COUNT}}$$

where: PAGE COUNT = the number of pages the table occupies
PAGE JUMPS = the number of times a different data page is
referenced to access all the data in the table
in index order

The CLUSTER column, in addition to giving a general idea about whether or not the index is clustered, is also used to identify the clustering index for the table. If the value of the CLUSTER column is "F" or "C" then the index is clustered, and a value of "F" means that the index is also the clustering index. A value of "W" or "N" means that the index is no longer clustered, where "W" denotes the clustering index. The CLUSTER column will show an index to be clustered if the following is true:

#page jumps <= 110% of page count

Data is initially made clustered by loading it in the order of the clustering index. If the clustering index becomes unclustered, the data should be unloaded and reloaded to make it clustered again.

Index Fragmentation

A fragmented index is characterized by excessive amounts of free space in the index pages, which usually is spread unevenly among the pages. Free space distributed unevenly implies that index keys are also distributed unevenly. Indexes can become fragmented by insert, delete, and update activity on the table.

To help prevent index fragmentation, indexes should be created after the data has been loaded into the table, and an adequate PCTFREE value should be specified for the index.

If the index is created before the data is loaded, page splits occur and the index becomes fragmented when the data is loaded. In fact, if the data is loaded in clustering order, each index page of the clustering index has 50% free space.

If a sufficient PCTFREE value is specified for the index when it is created, subsequent inserts do fit on the existing index page, avoiding index page splits.

Indexes must either be reorganized or dropped and recreated to correct the fragmentation. If they are dropped and recreated, any packages with dependencies on them are marked invalid. In addition, if a clustering index is dropped, it no longer functions as the clustering index if there are other indexes on the table. In this case, all indexes would have to be dropped, the clustering index recreated, and then the rest of the indexes recreated. If indexes are reorganized, dependent packages are not marked invalid, and the clustering properties do not change.

Sorting Concepts

The SQL/DS Sort component sorts the rows of a table according to the values of one or more of the table's columns. A sort is performed whenever an index is created, or whenever an SQL statement is executed which requires that the manipulated rows be ordered (such as a SELECT statement with an ORDER BY clause) and no index providing that ordering exists, or the Optimizer does not use the index.

The Sort component can sort all the rows in the table (such as when creating an index on the table) or a subset of the rows (such as when ordering the results of a query with an ORDER BY clause and a predicate that retrieves only part of the table). It can sort the rows according to any or all columns in the table, in any combination of ascending or descending order. If DRDA protocol is used, sorting is always performed based on the application server encoding scheme. A maximum of 16 ordering columns can be used, and the resulting ordering key cannot exceed 255 bytes.

The result of a sort is an ordered list of rows. This list is stored in an INTERNAL DBSPACE. The rows can then be retrieved from the INTERNAL DBSPACE to build an index or to participate in the next phase of the execution of a query, as appropriate.

When a sort is performed implicitly as part of creating an index, the INTERNAL DBSPACE that holds the ordered list of rows is released before index creation is completed. Whenever a sort is performed for any other reason, the INTERNAL DBSPACE is not released until the current LUW is complete. The INTERNAL DBSPACE and the space it occupies on DASD are not available until the end of the LUW. In all cases, sorting requires either one or two additional INTERNAL DBSPACES to be used as work areas during the sort process, as described below. These INTERNAL DBSPACES are always released before sorting is completed.

The table whose rows are being sorted is never modified by the sorting. The Sort component scans the table and retrieves each row that must participate in the sort. For index creation, a DBSPACE scan is always used, and all the rows of the table are retrieved. In all other cases, the access path used and the number of rows retrieved depend on the SQL statement being processed. For each row retrieved by the scan, the appropriate column values are extracted and are encoded into a sort key so that they are easy to compare to other sort keys. The encoding of a sort key is similar to the encoding of a key in an index.

The data being sorted includes at least the ordering columns, but it also includes any other columns that must participate in the ordered result. For example, consider the following query:

```
SELECT COLUMN1, COLUMN2, COLUMN3
   FROM MYTABLE
   ORDER BY COLUMN1, COLUMN2
```

A sort key for this query consists of the encoded values of the ordering columns COLUMN1 and COLUMN2 and, appended to it, the corresponding (non-encoded) value of COLUMN3. The sort key together with the other columns (if any) is called the *sort row*. The resulting sorted list contains all the information requested in the select-list, avoiding the need to rescan MYTABLE to retrieve the values of COLUMN3.

Sorting is completed in successive passes. Because it is not generally feasible to retrieve all the required rows into virtual storage and sort them, the Sort facility retrieves some of the rows (enough to fill an internal sort buffer), sorts them, and then stores this sorted partial result in an INTERNAL DBSPACE. This continues until all the input rows are sorted. Now the partial results must be merged to create the final sorted list. This merging process is usually the most costly part of a sort, requiring several passes through the partial results with each pass reading and writing each sort row once. These multiple passes use two INTERNAL DBSPACES as sort work areas: one from which the partial results are read, and another where the merged results are stored. In the next pass, the partial results are read from the INTERNAL DBSPACE into which they were stored by the previous pass. The number of passes is determined by the number of partial results to be merged as well as the number of partial results that can be merged at one time. If the number of rows is small enough, or the data is essentially in the correct sequence to begin with, only one pass (and one INTERNAL DBSPACE work area) may be required.

Sort keys often need to be decoded during the creation of the final ordered list. In the example above, the sort key consisting of the values of COLUMN1 and COLUMN2 is decoded so that the column values can be returned to the user in the expected form. In the case of an ordered list that is used to build an index, decoding is usually not required. In some cases, decoding of the last sort key column is done to transform the sort key into an index key.

The result of a sort may not be what you expected if the DRDA protocol is used. For example, if the encoding schemes of the application requester and application server are different, sorting is done based on the encoding scheme of the application server. For more information, refer to the *SQL Reference* manual.

For information on calculating the size of the INTERNAL DBSPACE(s), and DASD space required to perform a sort, refer to the *Database Administration* manual.

Logging/Recovery Concepts

The database manager maintains a log of all the database changes affecting data in recoverable storage pools completed by each logical unit of work. For changes made in nonrecoverable storage pools, the following rules apply to the logging of updates:

- Data definition operations are logged and are thus recoverable. This includes: CREATE TABLE/INDEX, DROP TABLE/INDEX, REORGANIZE INDEX, ALTER TABLE, and ACQUIRE/ALTER/DROP DBSPACE. This ensures that SQL/DS catalog tables are consistent with the state of the database.
- Row update operations (UPDATE,DELETE,INSERT) are not logged and are therefore not recoverable.

Note: The adding and deleting of DBEXTENTS and the adding of DBSPACES to the database are not logged and are thus not recoverable. It is recommended that a database archive be taken immediately following these operations to ensure that your current database archive reflects the added DBSPACES or DBEXTENTS. For more information reference the *System Administration* manual.

The Log:

The SQL/DS log is a minidisk with a block size of 4096 bytes. The last two pages of the log are reserved for information to control the archiving and restoring processes.

The log is an integral part of the physical configuration of the database, and one must be defined because certain control information is written to the log even if no logging is specified. With single logging, any I/O error on the log causes the log component to terminate the application server.

Dual Logging:

The SQL/DS dual logging option protects the database from log failures due to DASD failures on the log devices. With dual logging, database updates are recorded in both logs. Ideally, one log should be an exact copy of the other. An unrecoverable error is unlikely to occur on both logs at the same time. The log component continues processing as long as it can read or write from either log.

When specifying the logs for dual logging, both must be identical in size.

The processing done by the log component for dual logging is as follows:

For a log write operation:

1. Write a log record to the first log; if an error occurs, issue an error message.
2. Write the same log record to the second log; if an error occurs, issue an error message.
3. If an error occurs on both write operations, terminate the procedure.

For a log read operation:

1. Read a log record from the first log; if an error occurs, issue an error message, and try the second log.

2. Read a log record from the second log; if an error occurs, issue an error message, and terminate the procedure.

Checkpoint:

The database manager takes periodic checkpoints. At a given point in time, taking a checkpoint involves recording the state information in the log and taking a "snapshot" of the database. This snapshot includes updates from completed logical units of work as well as updates from logical units of work that are still in progress. At the checkpoint, all updates are written to the database regardless of the state of their LUWs.

The SQL/DS RDBMS provides functions to recover the database to a consistent state with respect to logical units of work in the event of a system crash. In a *consistent state*, each logical unit of work is either completely reflected in the database (all updates) or is not present in the database (no updates).

A disk-oriented mechanism is used to recover from a "soft" failure (which causes the contents of memory to be lost), and is oriented toward frequent checkpoints and rapid recovery. This mechanism is dependent on the DBSPACE recovery functions and the log.

The DBSPACE recovery mechanism is the use of "shadow pages." Two versions of each DBSPACE are maintained: the "current pages" reflect all updates up to the current point in time and the "shadow pages" reflect the state of the database at the time of the last checkpoint. Note that, until a page is updated, its "current" and "shadow" versions are the same. The checkpoint process consists of making the current pages of the DBSPACES become the shadow pages and making the old shadow pages available for reuse. In addition, a special checkpoint log record is written to the log to synchronize the log with the state of the database. Another way of looking at a checkpoint is that a picture of the database is taken, regardless of the state of the LUWs in progress, and the fact that the picture was taken is recorded in the log.

A checkpoint is scheduled when:

- The number of log pages specified by the CHKINTVL initialization parameter have been written to the log.
- During rollback, the total number of free pages in a storage pool is less than or equal to 10. (This does not apply when LOGMODE = N.)
- The percentage of free pages in a storage pool reaches the minimum specified by the SOSLEVEL initialization parameter. (This does not apply when LOGMODE = N.)
- A DROP DBSPACE is processed.
- A COMMIT WORK is processed in single user mode with no logging (LOGMODE = N).
- Soft recovery processing is complete during startup.
- An archive is about to be taken and after the archive has completed successfully.
- During shutdown in multiple user mode (MUM) and single user mode (SUM).
- A log-full condition occurs.
- An LUW that has updated row data in a nonrecoverable storage pool is committed or rolled back. A checkpoint is scheduled and completed before the commit or rollback operation is complete. This ensures that all updates are

committed to the database and minimizes the user recovery effort in the event of a subsequent application or system failure.

When a checkpoint has been scheduled, no new access to DBSS is allowed; that is, no DBSS operations are started until the checkpoint process is complete. A checkpoint must wait until currently running DBSS operations (but not their logical units of work) are complete. Most long-running DBSS operations periodically exit from DBSS to allow a pending checkpoint to be executed.

When a soft failure occurs and the application server is restarted, the database is restored to the point of the last checkpoint by using DBSPACE page map tables that reflect the current pages at that point. The checkpoint log record, whose location on the log is saved in the Directory by the checkpoint, is obtained and used to synchronize the log with the state of the database at the checkpoint. Now the LUW recovery process can begin.

LUW Recovery:

The LUW recovery process determines the state of each LUW at the time of failure and at the time of checkpoint:

- If the LUW starts and ends before the checkpoint, no processing has to be done because all the updates are reflected in the database at the checkpoint.
- If the LUW starts before the checkpoint and commits work after the checkpoint but before the failure, those updates made after the checkpoint must be redone. The updates made prior to the checkpoint are reflected in the database by the checkpoint.
- If the LUW starts before the checkpoint and is not completed before the failure, those updates made prior to the checkpoint must be undone. The updates made after the checkpoint are not reflected in the database.
- If the LUW starts after the checkpoint and commits work before the failure, its updates must be redone. No updates are reflected in the database at the checkpoint.
- If the LUW starts after the checkpoint and is not completed before the failure, no recovery has to be done because none of the updates are reflected in the database.

The following diagram illustrates the LUW Recovery process for the five cases described above:

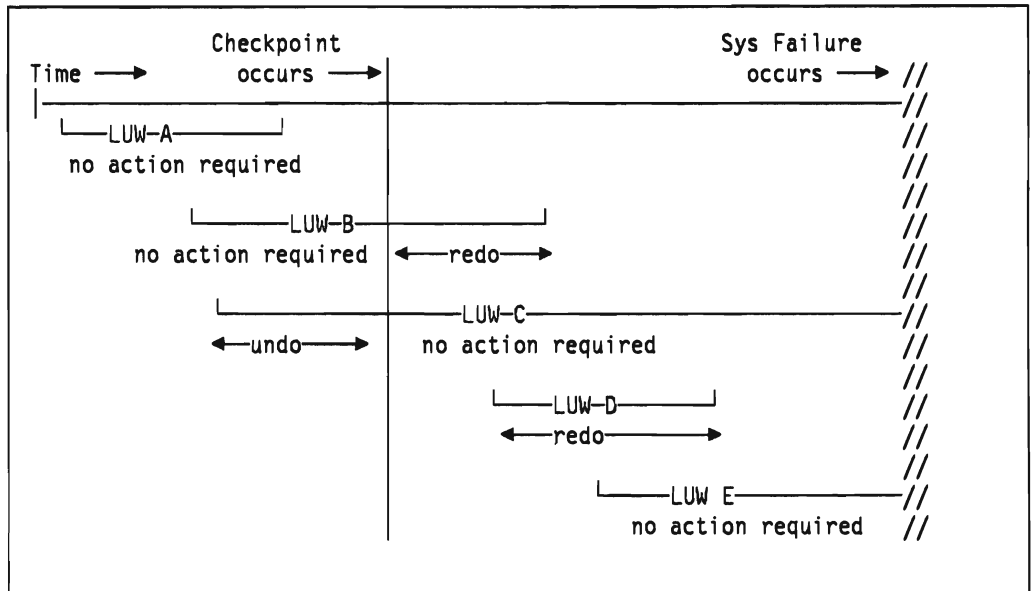


Figure 16. LUW Recovery Actions

Freeing Log Space:

This section describes how log space is freed when the database manager is not archiving (LOGMODE=Y). Freeing log space when the database manager is archiving (LOGMODE=A|L) is discussed in topic "Archiving" on page 52.

The SQL/DS log can be visualized as a straight line with the records for all logical units of work being written on it. (See Figure 17)

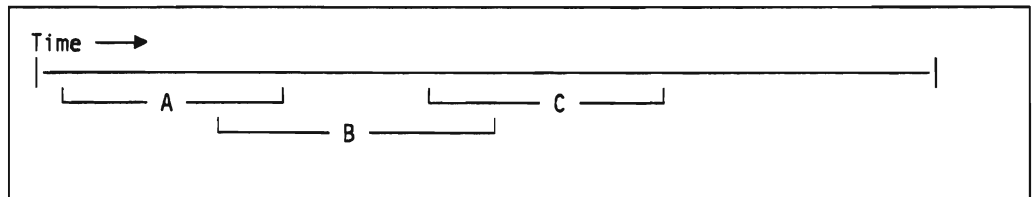


Figure 17. SQL/DS Log

Notes:

1. Logical unit of work B starts before logical unit of work A finishes. Logical unit of work C starts before B finishes.
2. The log is sequential. Records for B are interspersed with those for A, and so forth.

In time, the log fills up. When this happens, new records would overwrite those at the beginning of the log. (See Figure 18.)

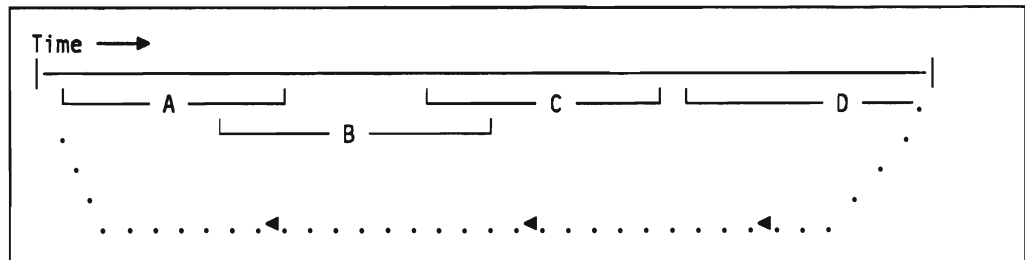


Figure 18. SQL/DS Log Wrap-Around

This "wrap-around" condition is undesirable and is prevented; overlaying of log records restricts the ability of the database manager to recover correctly. To prevent this situation, periodic checkpoints are taken.

Taking a checkpoint involves recording status information in the log and recording the current status of the database. Updates from all logical units of work are written to the database regardless of the state of the LUW. Log records for LUWs that end before the checkpoint are no longer needed and can be reclaimed, because the logical unit of work was complete when the updates were written to the database by the checkpoint. Log records for LUWs that were not ended before the checkpoint remain in the log file in case they need to be undone or redone (either because of an SQL/DS failure, or an application-specified ROLLBACK WORK).

Figure 19 shows a checkpoint at a time when only logical unit of work D is in progress. For this illustration, designate this checkpoint C1.

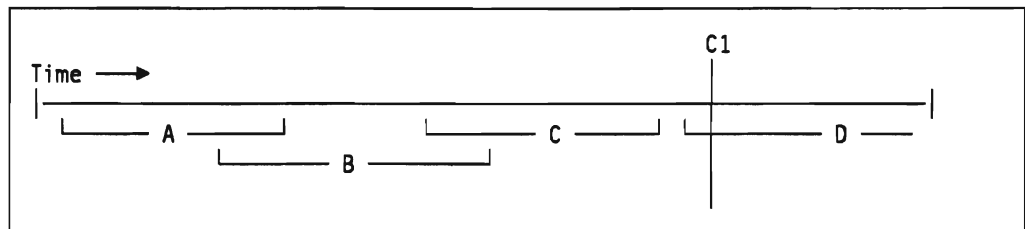


Figure 19. Checkpoint C1

After checkpoint C1 is taken, the database manager can reuse the log space formerly holding log records for logical units of work A, B, and C. (See Figure 20.)

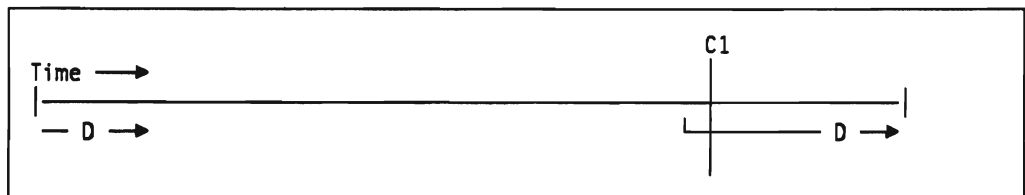


Figure 20. Logical Unit of Work D Wraps Around

As time passes, new logical units of work E and F occur. They start and end before D finishes. Another checkpoint, C2, occurs, as shown in Figure 21.

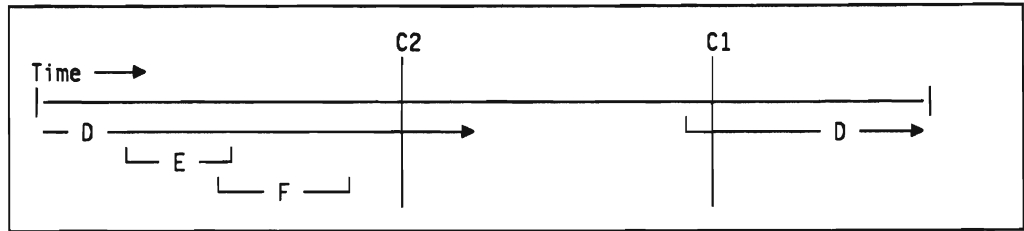


Figure 21. Checkpoint C2

Because of the sequential nature of the log, checkpoint C2 does not free any log space. Logical unit of work D is still in progress, and it started before logical units of work E and F. Log space used for E and F cannot be reclaimed until D finishes.

In a worst-case condition, logical unit of work D continues until the log is almost full. When the log reaches the percentage limit set by the SLOGCUSH initialization parameter (which defaults to 90%), a log-full message is issued to the operator. If log space cannot be made available, the longest running logical unit of work is rolled back so that log space can be reclaimed when the next checkpoint occurs. Then, when checkpoint C3 occurs, SQL/DS reclaims the entire log space. (See Figure 22.)

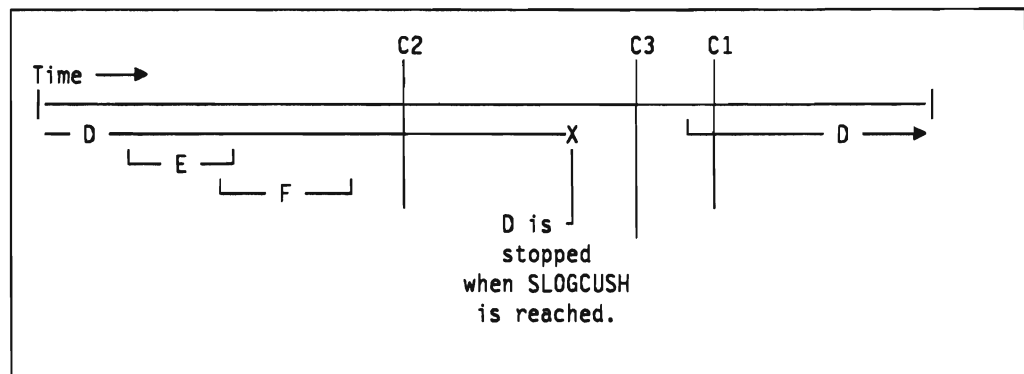


Figure 22. Checkpoint C3

Note: The situation described here rarely occurs if an adequate log is defined for the database.

There are alternatives to stopping work in progress. The obvious one is to have a log large enough to hold the longest logical unit of work expected. When installing the SQL/DS RDBMS for the first time, this size cannot be easily judged. It can be worthwhile to monitor the log usage by periodic use of the SHOW LOG operator command. This command can be issued from the SQL/DS operator console or by any ISQL user.

The database manager does not discriminate when it stops a logical unit of work. The work stopped can be a long-running program, or it can be an ISQL user who has gone home without signing off. Periodic use of SHOW LOG tells you how much log space remains so that you can begin to take action before SQL/DS begins stopping logical units of work. This action could be:

- In the case of the long-running ISQL user, contact the user and ask that person to end the logical unit of work.

- You can issue an SQL/DS FORCE operator command to immediately roll back the logical unit of work.

Archiving:

Archiving protects the database against media failures, such as a DASD head crash. The archive process copies either:

- An image of the entire database onto a magnetic tape (a database or user archive), or onto disk (a user archive).
- An image of the current SQL/DS log onto magnetic tape or disk (a log archive).

The LOGMODE SQL/DS initialization parameter indicates the type of archiving to be used. If LOGMODE=A, only database or user archiving is performed. If LOGMODE=L, database or user, and log archiving are performed. If LOGMODE=Y or N, archiving is not performed. For more information on LOGMODEs, see the *System Administration* manual.

The archiving process can be driven by an SQL/DS facility or by both an SQL/DS and a non-SQL/DS facility. When driven entirely by an SQL/DS facility, the archive process (database or log) has three steps:

1. The database manager takes a *base* checkpoint for the archive. All other SQL/DS work waits while the base checkpoint is being taken. For log archives, the database manager must wait until all active LUWs are complete and prevent any new LUWs from starting.
2. The database manager writes an image copy of the database (database archive) to tape or an image copy of the log (log archive) to tape or disk. Other SQL/DS work can be done while this archiving is occurring. If, however, a condition arises (during the image copying) that requires a checkpoint to be taken, this checkpoint (and all other SQL/DS work) must wait until the archive process is complete.
3. The database manager takes another checkpoint, called an *after* archive checkpoint. All other SQL/DS work waits while this checkpoint is being taken.

User archives are initiated by way of the SQLEND UARCHIVE command. During a user archive operation, the database manager is interrupted after Step 1 (above) and a non-SQL/DS facility is used to perform the actual database archive. After the non-SQL/DS utility has completed the user archive, step 3 (above) is performed on the subsequent warm start (after verification that the user archive was performed).

When an SQLEND ARCHIVE, SQLEND LARCHIVE, or SQLEND UARCHIVE command is used to shut down the database, the base checkpoint becomes the start of the log. Log space preceding the base checkpoint is freed.

Note: When using SQLEND UARCHIVE for a user archive, log space is not freed until the user archive is verified on the subsequent warm start.

The following situation applies to online database or log archives taken with the ARCHIVE or LARCHIVE commands. Figure 23 shows the log with its logical units of work (A through E) and the base and after-checkpoints.

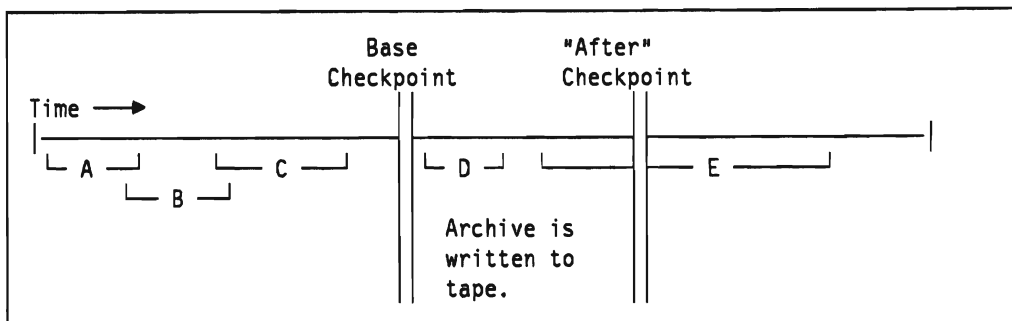


Figure 23. Base and "After" Checkpoints

At the base checkpoint, logical units of work A, B, and C are committed to the database and their log space could be freed. The image copy of the database or the log is then written to tape. While this copying is going on, logical unit of work D starts and finishes, and logical unit of work E starts but does not finish.

After the image has been written to tape, the database manager takes the *after* checkpoint. This checkpoint frees the log space used by logical units of work A, B, and C. Even though logical unit of work D has been committed to the database by the *after* checkpoint, the database manager does not free its log space because the base checkpoint serves as the starting point of the log. The reason for this arrangement is to protect against a media failure at some later time, as shown in Figure 24.

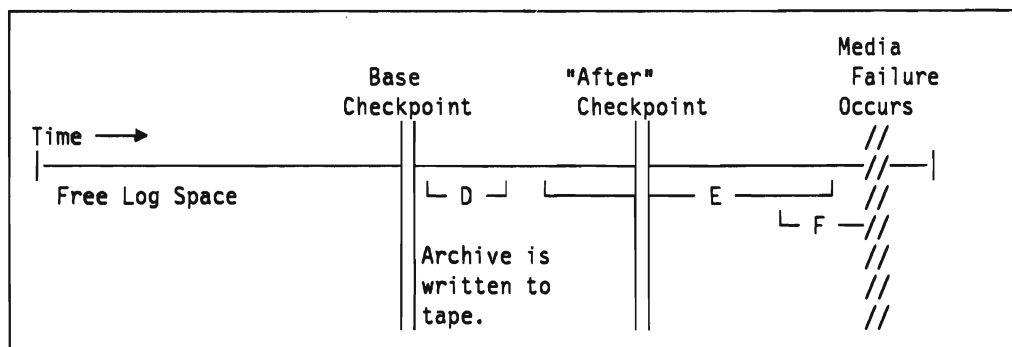


Figure 24. Media Failure Occurs after Archiving

Recovery from the situation shown in Figure 24 is done by restoring the database from the last database archive and, if LOGMODE=L, any subsequent log archives. The application server is started using the STARTUP=R initialization parameter. This restores the database to its state at the base checkpoint.

Because the archive tape or disk was created just after the base checkpoint, the database manager must apply all changes recorded in the log after the base checkpoint. This action includes the entire logical unit of work D. (For this reason, the log space used by D was *not* freed when the after-checkpoint was taken.) This action also includes the entire logical unit of work E. The changes made by logical unit of work F are not applied because it was not finished when the media failure occurred, (a partially completed logical unit of work is never committed) and since it occurred after the checkpoint it is not reflected in the database.

The situation in which the base checkpoint cannot be used as the start of the new log arises as shown in Figure 25. In this situation, an LUW spans the base checkpoint. This situation only applies to online database archives because for log archives to begin, all active LUWs have to be completed and no new LUWs are allowed to start.

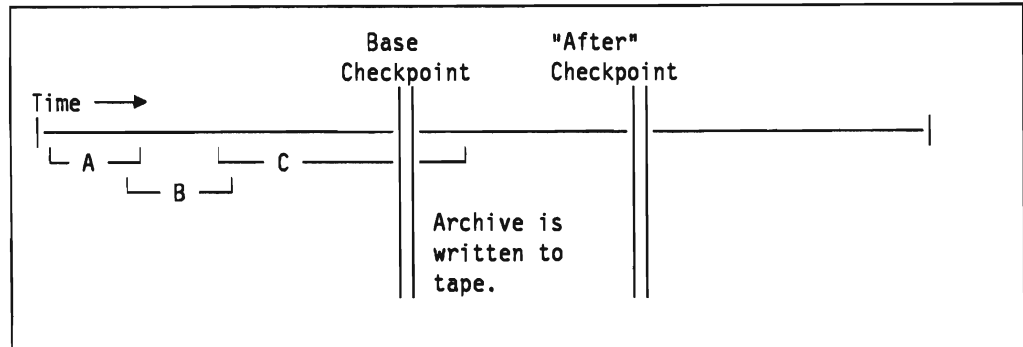


Figure 25. Base Checkpoint Occurs during Logical Unit of Work C

Here, the database manager cannot use the base checkpoint as the start of the log because some of the changes in logical unit of work C occur after the base checkpoint. In this situation, the database manager uses the start of logical unit of work C as the start of the log. (C was the "oldest" logical unit of work not finished at the base checkpoint.) Log space prior to the start of logical unit of work C is freed. If a media failure occurs later and application server has to be started by first recovering the database from the tape, the database manager can redo logical unit of work C using the log records.

When LOGMODE=A or L, a normal checkpoint does not free log space. The log space that can be freed is determined by the archive base checkpoint and freed by the archive "after" checkpoint. This is because the log is needed at least from the point of the last database or log archive when a media failure occurs. Consider checkpoint C in Figure 26.

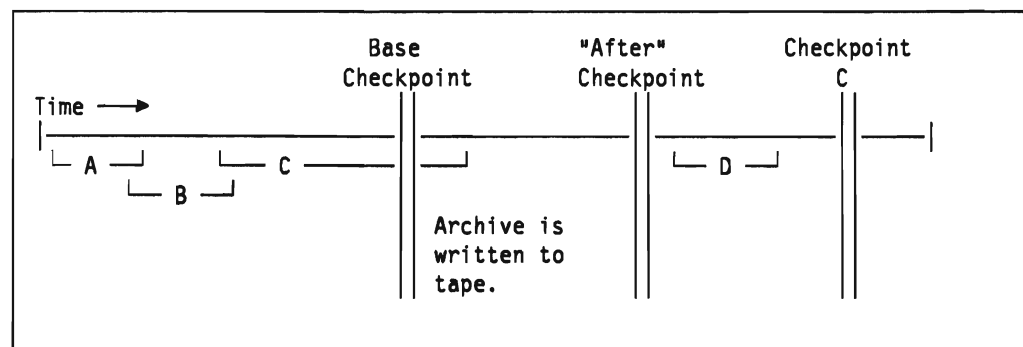


Figure 26. Checkpoint C Occurs after Archive Tape or Disk Written

Even though all logical units of work are finished at checkpoint C, the database manager cannot reclaim the log space of logical unit of work C; the log records are needed to enable recovery if a media failure occurs after the archive base checkpoint but before the next database or log archive. When the next database or log archive occurs, log space up to the *oldest* logical unit of work not yet finished at that time can be reclaimed.

Because normal checkpoints do not reclaim log space when LOGMODE=A or L, a database or log archive must be taken when the log approaches overflow. (If the log actually overflowed, log records needed for database recovery would be lost when they were overwritten. The log would then be useless.) When the log approaches overflow, as denoted by initialization parameter ARCHPCT, the database manager automatically activates the archive process if it is not already activated. If LOGMODE=A, this is a database archive. If LOGMODE=L, this is a log archive.

The database manager usually allows work to continue while the database or log archive tape is being written; however, if the log approaches overflow, as denoted by SLOGCUSH, while the archive tape is being written, the database manager suspends all other SQL/DS processing until the archive is complete.

Although normal checkpoints occurring while LOGMODE=A or L do not reclaim log space, they are valuable because:

- When data is changed, the database manager maintains a duplicate physical page called a shadow page to retain the state of that page *before* the change occurred. Committing a logical unit of work to the database does not free shadow pages, because they are needed in case recovery must be done. When a checkpoint occurs, all shadow pages are freed because those pages are no longer needed for recovery. (The log is sufficient for recovery after a checkpoint.)
- A checkpoint minimizes the updates that the database manager must do when recovering from a system failure. If a system failure does occur, the database manager recovers when the application server is started again by redoing the work done after the last checkpoint. The database manager does not redo all the work since the last database or log archive operation.

Locking Concepts

This section explains the type of locking done when specific DML and DDL statements are executed. This is a very important concept to understand when designing SQL/DS applications and in diagnosing locking problems.

Because the SQL/DS RDBMS is a concurrent-user system, locking techniques have been employed to resolve various synchronization problems, both at the logical level of objects (like tables) and at the physical level of pages.

At the logical level, the database manager must try to ensure that two concurrent logical units of work (LUWs) do not read the same value and then try to write back the updated values. If these LUWs are not synchronized, the second overwrites the first, and the effect of one update is lost.

At the physical level of pages, locking techniques are required to ensure that the database gives correct results. For example, a data page may contain several rows from one or more tables. Even if no logical conflict occurs between two LUWs (because each is accessing different tables or different rows in the same table) a problem can occur at the physical level. If, for example, one LUW causes an access to a row on some page, while another LUW updating a second row on the same page causes data compaction (because of lack of contiguous free space), row locations are reassigned within the page.

Specifying Isolation Levels

The user can improve performance by specifying a lower *isolation level* for appropriate applications. An isolation level is the degree of independence that one SQL/DS application has from another.

There are two isolation levels: *repeatable read* and *cursor stability*. The user can set or change the isolation level when preprocessing. The ISOLATION parameter can be used to establish the isolation level for a program. The user specifies ISOLATION(RR) for repeatable read and ISOLATION(CS) for cursor stability. An alternative to setting the isolation level during preprocessing would be to allow the program to set and change isolation levels during the running of the program. This requires that ISOLATION(USER) be specified when the program is preprocessed. If the user wants to change the isolation level that has already been set, that user must preprocess, compile(assemble), and link the program again.

The DRDA architecture defines two additional isolation levels, Uncommitted Read (UR), and Read Stability (RS). The SQL/DS application server, upon receiving a request at UR, will escalate it to CS and proceed without informing the application requester. Similarly, RS will be escalated to RR.

Repeatable Read:

Repeatable read (RR) ensures that within a logical unit of work a user can repeatedly read the same row of data without having it changed by some other user. With repeatable read, the user is completely isolated from interference by other applications. The price of this high degree of isolation is a reduction in concurrency; other users must wait until the logical unit of work is complete before they can access data being used under repeatable read conditions (unless the data is not being modified).

The following rules specify the isolation that repeatable read provides:

1. A logical unit of work cannot modify any data that another active logical unit of work has modified. *Modify* includes SQL INSERT, PUT, DELETE, or UPDATE.
2. A logical unit of work cannot see (read) any data that another active logical unit of work modifies.
3. A logical unit of work cannot modify any data that another active logical unit of work reads.

In terms of the data it reads or modifies, a logical unit of work is "unaware" of the existence of any other concurrent logical unit of work.

Cursor Stability:

Cursor stability (CS) places a lock on the row or page of data the user's cursor is pointing to. Rows in a table, or pages in a DBSPACE, that the user has already read are subject to change by other users. This means that more than one user can work on the same data at the same time. It also means that the data can appear to be inconsistent. For example, it is possible for a user to issue the same query twice within a logical unit of work and get different results. Users must be very careful when deciding to use cursor stability for their applications.

Note that cursor stability applies only to tables in PUBLIC DBSPACES with PAGE or ROW level locking. Tables in PRIVATE DBSPACES or PUBLIC DBSPACES with DBSPACE level locking always have the repeatable read isolation level.

Cursor stability provides the following:

1. Another logical unit of work cannot modify any data the user's active logical unit of work has modified. "Modify" implies SQL INSERT, PUT, DELETE, or UPDATE.
2. Another logical unit of work cannot see (read) any data the user's active logical unit of work has modified.

Note: When the database manager uses a DBSPACE scan (does not use an index) to access a table in a DBSPACE with ROW level locking using isolation level cursor stability, the effect is similar to repeatable read: no other logical unit of work can update the table until the logical unit of work performing the DBSPACE scan ends. Also, if one logical unit of work has updated a table, another logical unit of work (using cursor stability) cannot access that table with a DBSPACE scan until the updating logical unit of work ends. This reduced concurrency for DBSPACE scans does not apply for tables in DBSPACES with PAGE level locking, or when accessing through indexes.

Guidelines for Selecting an Isolation Level:

The effects of using cursor stability can be very subtle. Specific guidelines for selecting isolation levels are in the appropriate SQL/DS manuals. For guidelines on selecting an isolation level in application programs, see the *Application Programming* manual. For guidelines that apply to the DBS utility, see the *Database Services Utility* manual. For ISQL guidelines, see the *ISQL Guide and Reference* manual.

Isolation Level and Updates:

Note that the isolation level does not affect the duration of the locks held on data that has been inserted, deleted, or updated in an LUW. Locks on this data are always held until the end of the LUW, regardless of the isolation level.

Locking Hierarchy

The locking protocol uses a locking hierarchy that allows conflicts to be detected at the highest level. The locking hierarchy used is shown in Figure 27.

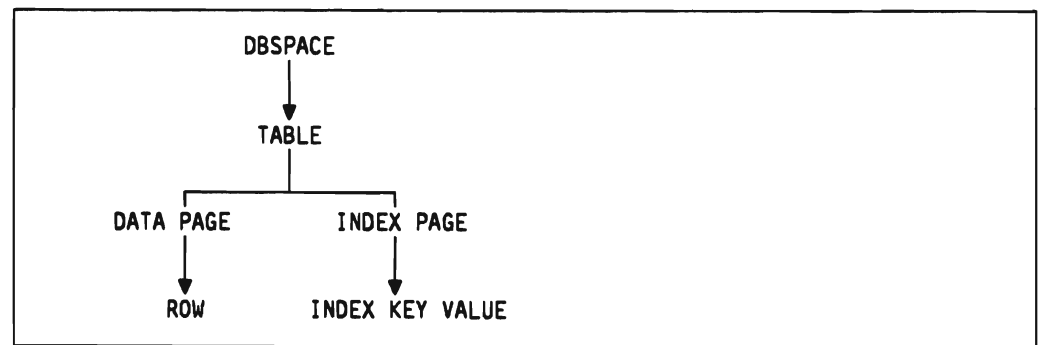


Figure 27. Locking Hierarchy

The protocol ensures that an agent issues lock requests in the order of the hierarchy. Thus, an agent accessing a row in table X of DBSPACE Y, would first have to obtain a lock on DBSPACE Y, then obtain a lock on table X, then obtain a lock on the page on which the row resides, and finally, obtain the lock on the row.

Note: After the first row is locked, it is not necessary to get the DBSPACE and table locks again, unless the required lock mode changes.

Before rows are accessed by an agent, the agent must first obtain a lock on the DBSPACE that identifies the agent's intentions within that DBSPACE. Any other agent does the same thing. For example, if one agent obtains EXCLUSIVE use of the DBSPACE, other agents are locked out immediately when they attempt to obtain any kind of lock on the DBSPACE.

Note: If a DBSPACE is defined to have PAGE level locking, row locks are not obtained. Having a lock on the appropriate page guarantees that other agents do not conflict on the row in question.

Lock Modes

There are six modes in which a data object may be locked:

IS	Intent Share
IX	Intent Exclusive
S	Share
U	Update
SIX	Share with Intent Exclusive
X	Exclusive

Intent modes describe low level locking intentions at a higher level in the hierarchy. For example, an agent wanting to read data in a DBSPACE that has PAGE level locking needs to obtain SHARE locks on the pages read. To do this, the agent must first obtain INTENT SHARE locks on the DBSPACE and table.

SHARE locks are obtained for read operations at the lowest level of the locking hierarchy. They can also be obtained at higher levels. For example, a lock escalation can promote SHARE locks on pages to one SHARE lock on the DBSPACE, or an SQL LOCK TABLE statement can be used to obtain a SHARE lock on a table. For more information on lock escalation, see "Escalation of Locks" on page 65.

EXCLUSIVE locks are obtained for UPDATE operations at the lowest level of the locking hierarchy. They can also be obtained at higher levels. For example, a lock escalation can promote EXCLUSIVE locks on pages to one EXCLUSIVE lock on the DBSPACE, or an SQL LOCK TABLE statement can be used to obtain an EXCLUSIVE lock on a table.

Note: PRIVATE DBSPACES are locked with SHARE or EXCLUSIVE locks. With DBSPACE level locking, the lower level locks are not obtained.

UPDATE locks are obtained at the lower levels of the locking hierarchy for read operations with an intent to update. A lock escalation can promote UPDATE LOCKS on pages to one share lock on the DBSPACE. If an UPDATE lock is

requested at the low level, the agent must first obtain INTENT SHARE locks at the higher levels.

Lock Durations

The length of time that a lock is held can be one of the following:

INSTANT The lock is acquired and then freed immediately.

SHORT The lock is held for the duration of the DBSS operation.

MEDIUM The lock is held over multiple DBSS operations, but can be freed before the end of the LUW.

LONG The lock is held until the end of the LUW.

UPDATE locks are not held for the duration requested. Instead, they are downgraded to SHARE locks (RR) or released (CS) when the agent has moved past the data. For repeatable read, the SHARE locks are then held for the requested duration.

Lock Compatibility

The matrices in Figure 28 through Figure 30 on page 60 indicate the modes that are compatible with each other. Yes means the requested lock is compatible with the held lock (and therefore is granted). No means the request is denied or the requesting agent is put in a LOCK WAIT. The lock component chooses whether the agent is to wait for the lock or to be given control immediately if the lock is not available.

Figure 28. Compatibility of Row Lock Modes

		MODE OF LOCK B		
MODE OF LOCK A	S	U	X	
S	Yes	Yes	No	
U	Yes	No	No	
X	No	No	No	

Figure 29. Compatibility of Page & Table Lock Modes

		MODE OF LOCK B					
MODE OF LOCK A	IS	IX	S	U	SIX	X	
IS	Yes	Yes	Yes	Yes	Yes	No	
IX	Yes	Yes	No	No	No	No	
S	Yes	No	Yes	Yes	No	No	
U	Yes	No	Yes	No	No	No	
SIX	Yes	No	No	No	No	No	
X	No	No	No	No	No	No	

Figure 30. Compatibility of DBSPACE Lock Modes

MODE OF LOCK B					
MODE OF LOCK A	IS	IX	S	SIX	X
IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No

Types of Internal Data Manipulation Calls

Following is a list of the basic internal data manipulation operations performed by the database manager. These are to be used in conjunction with Figure 31 to understand the locking done for each operation.

OPEN SCAN: The OPEN SCAN operation causes a scan (the internal equivalent of a cursor) to initiate access to:

- Rows in a table through a specified index. Index keys and key search conditions are supplied to support selective searching. This is an INDEX scan.
- Rows in a table in their inserted order without an index. Column values and column search conditions can be supplied to support selective searching. This is a DBSPACE scan.
- Rows in a sequential list (for example the output of a sort operation).

NEXT: causes the "next" row in an opened scan to be retrieved. For scans, optional search arguments can be supplied and are used to determine whether rows in the scan "qualify" as the desired next row. If a row does not qualify, the scan continues to the next row.

CLOSE SCAN: terminates the scan.

FETCH: retrieves a table row by way of either its row identifier or an index key. It does not use an open scan.

INSERT: inserts a row into a table and inserts a corresponding key into each index on the table.

DELETE: deletes a specified row from a table and deletes the corresponding key from each index on the table.

UPDATE: replaces one or more columns in a specified row with user supplied values and updates the affected indexes on the table.

Locking for Different Internal DM Calls

The following is a table of the locking that is done for the different internal data manipulation calls.

Figure 31 (Page 1 of 2). Locking for Different Internal DM Calls

SQL FUNCTION	DBSS FUNCTION	LOCKED OBJECT	PAGE LOCKING	ROW LOCKING	DBSPACE LOCKING
INSERT	INSERT	DBSPACE TABLE PAGE ROW	LONG IX LONG IX LONG X none	LONG IX LONG IX SHORT X LONG X	LONG X none none none
DELETE	DELETE	DBSPACE TABLE PAGE ROW	LONG IX LONG IX LONG X none	LONG IX LONG IX SHORT IX LONG X	LONG X none none none
UPDATE	UPDATE	DBSPACE TABLE PAGE PAGE ROW	LONG IX LONG IX LONG X LONG X none	LONG IX LONG IX SHORT IX SHORT X LONG X	LONG X none none none
DML with unique index and equal predicate (3).	FETCH	DBSPACE TABLE PAGE ROW IPAGE IKEY	LONG IS LONG IS LONG S none LONG S none	LONG IS LONG IS SHORT IS LONG S SHORT S LONG S	LONG S none none none none none
OPEN/FETCH with DB SCAN Repeatable Read	OPEN/NEXT DB SCAN	DBSPACE TABLE PAGE ROW	LONG IS LONG S MEDIUM S none	LONG IS LONG S SHORT S none	LONG S none none none
OPEN/FETCH with DB SCAN Cursor Stability	OPEN/NEXT DB SCAN	DBSPACE TABLE PAGE ROW	LONG IS LONG IS MEDIUM S none	LONG IS LONG S SHORT S none	LONG S none none none
OPEN/FETCH with INDEX SCAN Repeatable Read	OPEN/NEXT INDEX SCAN	DBSPACE TABLE PAGE ROW IPAGE KEY	LONG IS LONG IS LONG S (4) none LONG S none	LONG IS LONG IS SHORT IS LONG S (4) SHORT S LONG S	LONG S none none none none none
OPEN/FETCH with INDEX SCAN Cursor Stability	OPEN/NEXT INDEX SCAN	DBSPACE TABLE PAGE ROW IPAGE KEY	LONG IS LONG IS MEDIUM S none MEDIUM S none	LONG IS LONG IS SHORT IS MEDIUM S SHORT S INSTANT S	LONG S none none none none none
DML with unique index and equal predicate. Update only Repeatable Read (1) (3)	FETCH using INDEX	DBSPACE TABLE PAGE ROW IPAGE IKEY	LONG IS LONG IS LONG U (2) none LONG S none	LONG IS LONG IS SHORT IS LONG U (2) SHORT S LONG S	LONG S none none none none none
DML with unique index and equal predicate. Delete only Repeatable Read (1) (3)	FETCH using INDEX	DBSPACE TABLE PAGE ROW IPAGE IKEY	LONG IS LONG IS LONG U (2) none LONG U none	LONG IS LONG IS SHORT IS LONG U (2) SHORT S LONG U	LONG S none none none none none

Figure 31 (Page 2 of 2). Locking for Different Internal DM Calls

SQL FUNCTION	DBSS FUNCTION	LOCKED OBJECT	PAGE LOCKING	ROW LOCKING	DBSPACE LOCKING
OPEN/FETCH with DB SCAN Repeatable Read Update/Delete	OPEN/NEXT DB SCAN	DBSPACE TABLE PAGE ROW	LONG IS LONG U MEDIUM U none	LONG IS LONG U SHORT S none	LONG S none none none
OPEN/FETCH with DB SCAN Cursor Stability Update/Delete	OPEN/NEXT DB SCAN	DBSPACE TABLE PAGE ROW	LONG IS LONG IS MEDIUM U none	LONG IS LONG U SHORT S none	LONG S none none none
OPEN/FETCH with INDEX SCAN Repeatable Read Update and Format 2 delete	OPEN/NEXT INDEX SCAN	DBSPACE TABLE PAGE ROW IPAGE KEY	LONG IS LONG IS LONG U none LONG S none	LONG IS LONG IS SHORT IS LONG U SHORT S LONG S	LONG S none none none none none
OPEN/FETCH with INDEX SCAN Cursor Stability Update and Format 2 delete	OPEN/NEXT INDEX SCAN	DBSPACE TABLE PAGE ROW IPAGE KEY	LONG IS LONG IS MEDIUM U none MEDIUM S none	LONG IS LONG IS SHORT IS MEDIUM U SHORT S INSTANT S	LONG S none none none none none
OPEN/FETCH with INDEX SCAN Repeatable Read Format 1 delete only	OPEN/NEXT INDEX SCAN	DBSPACE TABLE PAGE ROW IPAGE KEY	LONG IS LONG IS LONG U none LONG U none	LONG IS LONG IS SHORT IS LONG U SHORT S LONG U	LONG S none none none none none
OPEN/FETCH with INDEX SCAN Cursor Stability Format 1 delete only	OPEN/NEXT INDEX SCAN	DBSPACE TABLE PAGE ROW IPAGE KEY	LONG IS LONG IS MEDIUM U none MEDIUM U none	LONG IS LONG IS SHORT IS MEDIUM U SHORT S INSTANT S	LONG S none none none none none

- (1) U locks are only used for calls of this type when all the search arguments are sargable. This guarantees that if a row is found, it is updated and prevents U locks from being held until the end of the LUW when the row is not being updated.
- (2) These U locks are always upgraded to X locks on the next DBSS call involving the scan for which they have been acquired.
- (3) Locks on data pages and rows are acquired only if the page must be accessed. This is the case only where there are search arguments that are not part of the index key or when domains are requested.
- (4) Locks on data rows or pages will only be acquired if data page access is required. This is the case only where there are search arguments which are not part of the index key or when domains are requested and are not part of the index key or when a variable length domain is requested.

Note: There are three additional lock types that are not mentioned in the previous tables:

1. Rollback Lock: a special system lock, acquired during rollback at the beginning of a particular operation that requests page locks during the rollback.
2. Internal DBSPACE Lock: a lock on an internal DBSPACE.

- Database Lock: a special system lock that is acquired in IX mode at the beginning of each LUW. It is acquired in X mode at checkpoint time if a log archive is to be performed. This prevents new LUWs from starting when a log archive is scheduled.

These locks may appear as lock type SYS (rollback lock), INT (internal-DBSPACE lock), or DB (database lock) in the output of the SHOW LOCK operator command.

Locking on Indexes: SHARE and EXCLUSIVE locks are obtained on the pages and keys of an index. The root and leaf pages that are traversed to obtain a submitted key are locked for either a SHORT, MEDIUM or LONG duration depending on the operation locking level (page or row), and isolation level (repeatable read or cursor stability). Non-leaf pages are never locked.

With row level locking the successor key in the index is locked for an index insert or delete, as well as the inserted or deleted key. Similarly, with page level locking, the successor page may be locked if the successor key is on the next page. This is referred to as "adjacent key locking."

Detailed locking on indexes is described in the following tables for each of the more common internal data manipulation operations.

OPEN SCAN, NEXT and FETCH – REPEATABLE READ

LOCKED OBJECT	PAGE LOCKING	ROW LOCKING
ROOT PAGE	SHORT SHARE	SHORT SHARE
LEAF PAGE	LONG SHARE	SHORT SHARE
KEY	none	LONG SHARE

OPEN SCAN, NEXT and FETCH – CURSOR STABILITY

LOCKED OBJECT	PAGE LOCKING	ROW LOCKING
ROOT PAGE	SHORT SHARE	SHORT SHARE
LEAF PAGE	MEDIUM SHARE	SHORT SHARE
KEY	none	INSTANT SHARE

OPEN SCAN, NEXT and FETCH – for DELETE – REPEATABLE READ

LOCKED OBJECT	PAGE LOCKING	ROW LOCKING
ROOT PAGE	SHORT SHARE	SHORT SHARE
LEAF PAGE	LONG UPDATE	SHORT SHARE
KEY	none	LONG UPDATE

OPEN SCAN, NEXT and FETCH – for DELETE – CURSOR STABILITY

LOCKED OBJECT	PAGE LOCKING	ROW LOCKING
ROOT PAGE	SHORT SHARE	SHORT SHARE
LEAF PAGE	MEDIUM UPDATE	SHORT SHARE
KEY	none	INSTANT SHARE

INDEX DELETE (DELETE or UPDATE for Deleted key)

LOCKED OBJECT	PAGE LOCKING	ROW LOCKING
ROOT PAGE	SHORT SHARE	SHORT SHARE
LEAF PAGE	LONG EXCLUSIVE	SHORT EXCLUSIVE
DELETED KEY	none	INSTANT/LONG EXCLUSIVE
SUCCESSOR KEY	none	LONG EXCLUSIVE/none

INDEX INSERT (INSERT or UPDATE for Added key)

LOCKED OBJECT	PAGE LOCKING	ROW LOCKING
ROOT PAGE	SHORT SHARE	SHORT SHARE
LEAF PAGE	LONG EXCLUSIVE	SHORT EXCLUSIVE
INSERTED KEY	none	LONG EXCLUSIVE
SUCCESSOR KEY	none	INSTANT EXCLUSIVE

INDEX UPDATE WITH DEFERRED CHECKING OF UNIQUENESS

LOCKED OBJECT	PAGE LOCKING	ROW LOCKING
ROOT PAGE	SHORT SHARE	SHORT SHARE
LEAF PAGES OF INSERTED AND DELETED KEYS	LONG EXCLUSIVE	SHORT EXCLUSIVE
INSERTED KEY	none	LONG EXCLUSIVE
DUPLICATE OF INSERTED KEY	none	MEDIUM EXCLUSIVE
SUCCESSOR OF INSERTED KEY	none	INSTANT EXCLUSIVE
DELETED KEY	none	INSTANT EXCLUSIVE
SUCCESSOR OF DELETED KEY	none	LONG EXCLUSIVE/none

Notes:

1. For OPEN SCAN, NEXT, and FETCH, index locking occurs only for the index used (if any) to retrieve the rows.
2. For UPDATE and DELETE operations, OPEN SCAN and NEXT are used to locate the rows to be updated or deleted.
3. UPDATE causes index locking as described above for any index where a key column value was updated.
4. INSERT and DELETE cause every index on the table to be updated.

Deadlock Detection

The database manager performs deadlock detection prior to placing any agent into a lock wait. An example of a deadlock is: agent A holds resource X and agent B wants resource X while holding resource Y, which agent A wants. There is an impasse, which the system removes by rolling back the youngest LUW. Agents that are in the process of being rolled back are never chosen as deadlock "victims."

Escalation of Locks

In managing the lock requirements of a LUW, the database manager uses internal control blocks called lock request blocks (LRBs). Each time a lock is acquired one or more LRBs are used. The number of LRBs that can be held by any given agent is defined by the SQL/DS initialization parameter NLRBU. The sum of the number of LRBs held by all agents cannot exceed the limit defined by the SQL/DS initialization parameter NLRBS. When either of these limits is reached, lock escalation is initiated for the agent that caused the limit to be exceeded.

Lock escalation is the act of trading low level locks (page, row, table, index page, or key value locks) for the appropriate DBSPACE lock for one of the DBSPACES in which the victim agent holds locks. The DBSPACE chosen is the one in which the agent holds the most locks.

Note that the lock manager is selective about the locks it escalates. A request for data in DBSPACE X does not necessarily cause escalation to go after a lock on DBSPACE X.

If the agent holds any EXCLUSIVE locks in the DBSPACE, an EXCLUSIVE lock is requested on the DBSPACE chosen. Otherwise, a SHARE lock is requested. If the DBSPACE lock cannot be granted, the system checks for a possible deadlock. If no deadlock is found, the DBSPACE lock request is queued. After the DBSPACE lock is granted, the lower level locks are freed, except those which are required to support the atomicity of SQL statements. For more information on atomicity of SQL statements, see the *Database Administration* manual. When calculating NLRBU, you will need to allocate extra LRBs when applications contain SQL statements which affect multiple rows (such as UPDATE).

As the user resumes access to the DBSPACE (which is now locked at the DBSPACE level), lower level locks are not required and are not obtained. Thus, for any given LUW, the user can escalate only once on a particular DBSPACE. Or another way of looking at it, the maximum number of times an LUW can be escalated is the number of DBSPACES accessed during that LUW.

Access to Private DBSPACES

More than one user can have concurrent access to a private DBSPACE, but for read operations only. That is, multiple users can hold a shared lock on the DBSPACE.

Termination Concepts

The application server can be terminated normally or abnormally. Normal termination occurs in single user mode (SUM) when the application program returns to the application server. In multiple user mode (MUM), normal termination occurs when the SQL/DS operator issues an SQLEND command without the QUICK keyword, and after all users have disconnected their communication link with the application server. If DVERIFY was specified, the directory is then verified. For SQLEND with ARCHIVE or LARCHIVE, the appropriate archive operation is performed.

During normal termination in the VM environment, the database manager severs the IUCV connection to the *IDENT system service to relinquish ownership of the database as a resource, so new users are not able to establish a connection.

Abnormal termination occurs when the database manager detects an internal error, a resource limitation, a hardware error, a program check (or similar situation), or the operator issues an SQLEND QUICK command, as follows:

- In the case of an internal error, the detecting module issues the message:

```
ARI0040E SQL/DS system error occurred-ARlxxxx nn
```

ARlxxxx is the name of the module detecting the error and nn is the point within the module where the error was detected. This message is accompanied by the SQL/DS mini-dump and a dump of the database machine according to the DUMPTYPE initialization parameter specification.

- Whenever the database manager cannot obtain sufficient resources, usually storage, it issues a limit error message, and normally there is an accompanying message preceding it:

```
ARI0039E SQL/DS limit error occurred-ARlxxxx nn
```

where ARlxxxx is the name of the module detecting the limit error and nn is the detection point within the module. Because the accompanying message indicates the cause of the error, no dump is taken.

- Whenever a hardware error is detected, the database manager issues the message:

```
ARI0041E System hardware error occurred-ARlxxxx nn
```

where ARlxxxx is the module detecting the hardware error and nn the error detection point. This message is normally preceded by a message indicating the cause of the error. No dump is taken for a hardware error.

Note: For limit and hardware errors, the message

```
ARI0042I SQL/DS reason code is n1-X'n2'
```

is issued where n1 (decimal representation) and n2 (hexadecimal representation) is the host system return code associated with the failure.

If a program check occurs, control is given to the SQL/DS abnormal termination routines. The abnormal termination exit is established by way of a DMSABN command. The action taken depends upon the environment (SUM or MUM) and the time when the condition occurs (in DBSS, RDS, and so forth). The action can be as simple as passing a return code to the application program, or as drastic as terminating the database machine. These actions are described in "System Problems" on page 247.



Chapter 3. Reporting Defects

This chapter introduces you to the concepts that you need in order to build a symptom string that describes a defect. You are here because someone has a problem with the SQL/DS product and you are fairly certain that it was being used correctly at the time of the failure. However, if a non-SQL/DS application requester or server was accessed at the time the problem was encountered, you should also refer to the *Distributed Relational Database Problem Determination Guide*. A problem reported on an SQL/DS application requester could, for example, be caused by a non-SQL/DS application server.

This chapter will help you determine whether the SQL/DS failure has been previously documented and corrected. If it has not, the chapter will help you communicate with IBM* support personnel to isolate and correct the problem.

Specifically this chapter will help you to:

- Systematically develop a set of "keywords" to describe the failure.
- Use these keywords to identify your problem when contacting the IBM support center for assistance.
- Gather the necessary documentation to aid IBM support personnel in determining and correcting the problem.

Using this chapter will therefore expedite your getting an IBM-supplied correction for the problem.

For your convenience, a set of blank forms has been included (the first is on page 89) to aid you in constructing a symptom string to describe your problem. Figure 32 on page 70 is the hierarchy of keywords that are required for developing a keyword string to describe your problem. The two topmost elements are required for any problem you might encounter. The first (component id) is already completed on the forms provided, and you need only to place a checkmark for the second element (release level). Depending on what your symptom is, follow the branch for the third-level element (ABNORMAL TERMINATION, NO RESPONSE, etc.) that describes your symptom, specifying the information requested on the appropriate form. A separate form is provided for each of the third-level elements.

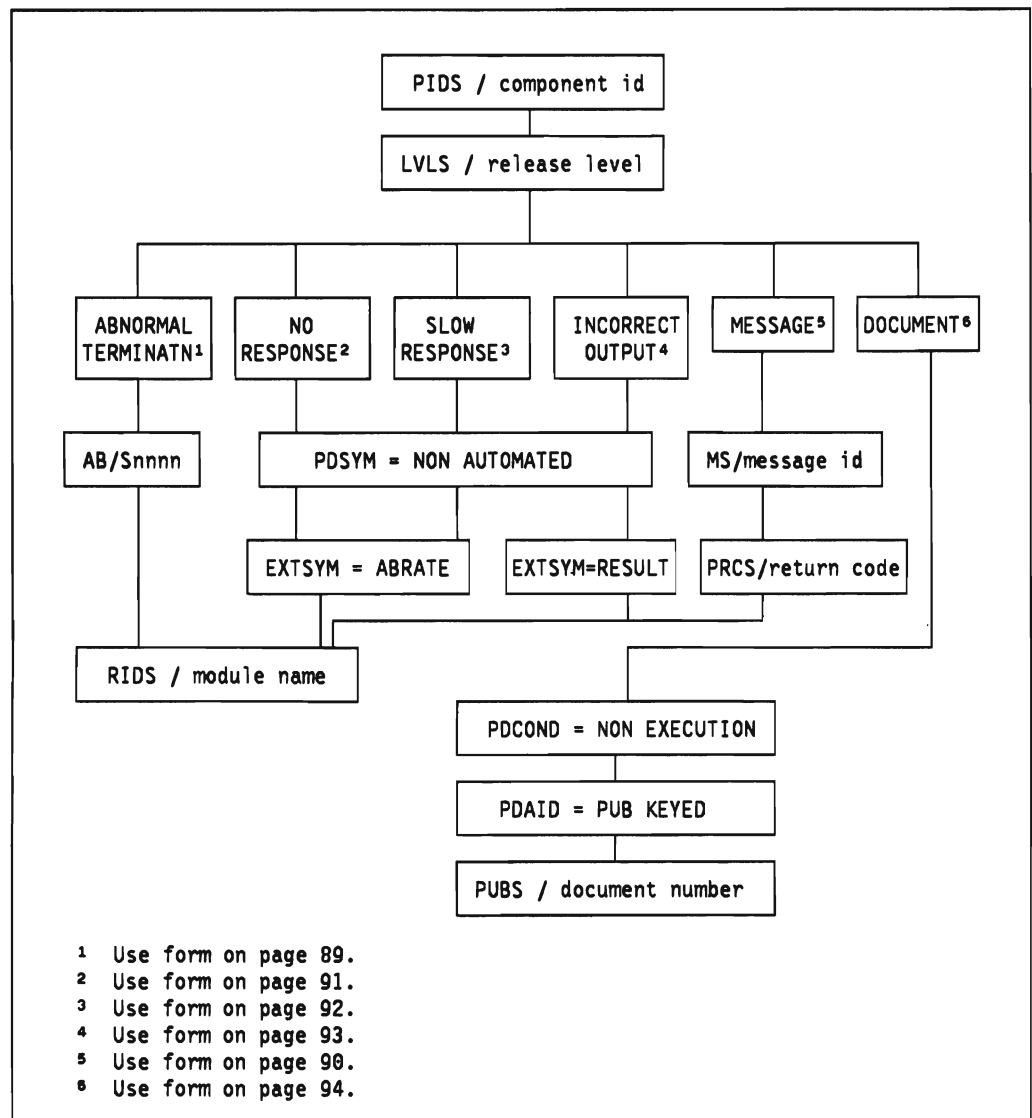


Figure 32. Anatomy of a Keyword String

Developing the First (Two) Keyword(s)

All keyword strings contain keywords of the following two types:

- Component Identification Keyword (PIDS)
- Release Level Keyword (LVLS)

Component Identification Keyword (PIDS)

The first keyword in the string holds the number by which IBM support personnel identify the SQL/DS database manager as the component detecting the error. This is included on each of the forms. (If you choose not to use one of the forms provided in this book, you must supply: PIDS/568810301.)

Release Level Keyword (LVLS)

The second keyword in the string shows the Component Level Code number for the release. For previous releases and this release, check the appropriate level on the form or (if you do not choose to use the form) specify the element as follows:

Version 2 Release 2 :

- For all problems which are not specific to NLS Language type, use LVLS/220
- For NLS specific problems, use the keyword for the language in the list below

English Upper Case: LVLS/221
French : LVLS/222
German : LVLS/223
Italian : LVLS/224
Spanish : LVLS/225
Japanese : LVLS/226
Korean : LVLS/227
Chinese : LVLS/228

Version 3 Release 1 :

- For all problems which are not specific to NLS Language type, use LVLS/310
- For NLS specific problems, use the keyword for the language in the list below

English Upper Case: LVLS/311
French : LVLS/312
German : LVLS/313
Italian : LVLS/314
Spanish : LVLS/315
Japanese : LVLS/316
Korean : LVLS/317

Version 3 Release 2 :

- For all problems which are not specific to NLS Language type, use LVLS/320
- For NLS specific problems, use the keyword for the language in the list below

English Upper Case: LVLS/321
French : LVLS/322
German : LVLS/323
Italian : LVLS/324
Spanish : LVLS/325
Japanese : LVLS/326
Korean : LVLS/327

Version 3 Release 3 :

- For all problems which are not specific to NLS Language type, use LVLS/330
- For NLS specific problems, use the keyword for the language in the list below

English Upper Case: LVLS/33A
French : LVLS/33B
German : LVLS/33C
Italian : LVLS/33D
Spanish : LVLS/33E
Japanese : LVLS/33F
Korean : LVLS/33G
Chinese : LVLS/33N

Version 3 Release 4 :

- For all problems which are not specific to NLS Language type, use LVLS/340
- For NLS specific problems, use the keyword for the language in the list below

English Upper Case: LVLS/34A
French : LVLS/34B
Italian : LVLS/34D
Spanish : LVLS/34E
Japanese : LVLS/34F
German : LVLS/34H
Chinese : LVLS/34J

Developing the Remaining Keywords

Abnormal Termination

To complete the keyword string for an abnormal termination, the system abend code is necessary, and can be obtained from the SQL/DS minidump provided when the abend occurred. The keyword is specified as follows:

AB/Snnnn
RIDS/module name
ADRS/

where nnnn is the system abend code; module name is obtained from the SQL/DS minidump. For example, the keyword for an operation exception would be AB/S00C1.

Figure 33 is an example of a minidump of an abend situation. The line displayed before the ARI0032I message is the SQL/DS-generated symptom string.

```
DMSITP141T Operation exception occurred at 800FB852 in routine ARISDBBT

SQL/DS MINIDUMP FOLLOWS:
IF DUMPTYPE = N NO DUMP IS PRODUCED

SQL/DS ABEND SAVEAREA :

ADDR  OFFSET  DUMP DATA
007870F0 000000 00000000 00000000 00787000 000FB7E0 * ..... *
00787100 000010 000FB3E0 000FB850 004F0151 004EF152 * .....&.|...+1. *
00787110 000020 00000000 00000001 004E48C4 000FB520 * .....+D.... *
00787120 000030 001DEAB0 004EE153 804ED154 000FB850 * .....+...+J....& *
00787130 000040 000FB850 004ED19E 03EC1E00 800FB852 * ...&.+J..... *
00787140 000050 00000000 00000000 00000000 00000000 * ..... *
00787150 TO 007871C0 SUPPRESSED LINE(S) SAME AS ABOVE .....
007871C0 0000D0 00000000 00000000 * ..... *
```

Figure 33 (Part 1 of 2). SQL/DS Minidump, Example for Abend

ABTERM CODE 0C1 AT 000FB850

PROGRAM OLD PSW IS : 03EC1E00 800FB852

GPR 0 = 00787000 000FB7E0 000FB3E0 000FB850
 GPR 4 = 004F0151 004EF152 00000000 00000001
 GPR 8 = 004E48C4 000FB520 001DEAB0 004EE153
 GPR 12 = 804ED154 000FB850 000FB850 004ED19E

FAILURE AT OFFSET +000307F0 IN ARISQLDS PROGRAM (007F1000)
 FAILURE AT OFFSET +00000190 IN ARIYM00 93.090

CALLED FROM OFFSET +0000E9C2 IN ARIXRDS PROGRAM (0066A000)
 CALLED FROM OFFSET +0000036A IN ARIXEDB 93.090

SUMMARY OF USERS

DS2CVT	RDCVT	YRSSCVT
00787000	00787888	00787540

CURRENTLY RUNNING DCE 001800A0

USERID	PROGRAM NAME	DSCAREA	YTABLE1	RDAREA	DCE
001 OPERATOR		001DE8A0	001DEAB0		001800A0
002 CHECKPT		001F7D80	001F7F90		001801D8
003 RECOVERY		001F9230	001F9440	00202BF0	00180310
004 SQLUSR5		00206388	00206598	0020FCE0	00180448

NOTE: USERID AND/OR PROGRAM NAME MAY BE RESIDUAL AND MAY NOT NECESSARILY
 BE THE CURRENT VALUES

STORAGE NEAR FAILURE :

ADDR OFFSET DUMP DATA

```
000FB830 000000 00000039 00000000 00000000 001DEAB0 * ..... *
000FB840 000010 000FB800 004E4900 000FB5B8 000FB724 * .....+..... *
000FB850 000020 000FB870 000FB790 00000000 00000000 * ..... *
000FB860 000030 00000000 00000000 00000000 00000000 * ..... *
000FB870 000040 003A0000 C1D9C9F0 F0F6F3C9 40E896A4 * ....ARI0063I Y.. *
```

SYMPTOM STRING:

AB/S00C1 PIDS/568810301 RIDS/ARIYM00 ADRS/00000190

ARI0032I SQL/DS has terminated.
 ARI0042I SQL/DS reason code is 193 - X'C1'
 ARI0043I SQL/DS return code is 516.

Figure 33 (Part 2 of 2). SQL/DS Minidump, Example for Abend

Note: The situation here was that module ARIXEDB called module ARIYM00,
 and an operation exception occurred at offset 00000190 in ARIYM00.

Message

Use the following keywords to describe message-associated problems, such as:

- The user received the SQL/DS "system error" (ARI0040E) message.
- The user received message ARI2909I indicating a First Failure Data Capture dump is produced. For details see the next section, "First Failure Data Capture" on page 76.
- The user had a problem with a message of format ARInnnnx.

The following keywords should be specified for the ARI0040E messages:

MS/message id
RIDS/module name
PRCS/return code

where message id is ARI0040E;
module name is ARlxxxx from the message text;
return code is the nn from the message text.

Figure 34 on page 75 is an example minidump of an SQL/DS system error. The last line displayed in the minidump is the SQL/DS-generated symptom string.

The following keywords should be specified for messages having a format of ARInnnnx (excluding ARI0040E and ARI2909I):

MS/message id
PRCS/return code

where message id is the ARInnnnx, return code is the return code from the message text if available. Pad the return code with zeroes on the left hand side.

```

ARI0040E SQL/DS system error occurred - ARICMUD 03

SQL/DS MINIDUMP FOLLOWS:
IF DUMPTYPE = N NO DUMP IS PRODUCED

SQL/DS ABEND SAVEAREA :

ADDR  OFFSET  DUMP DATA

007870F0 000000 00000000 00000000 00787000 000FB7E0 * ..... *
00787100 000010 000FB3E0 000FB850 004F0151 004EF152 * .....&|.+.1. *
00787110 000020 00000000 00000001 004E48C4 000FB520 * .....+.D.... *
00787120 000030 001DEAB0 004EE153 804ED154 000FB850 * .....+.+.J....& *
00787130 000040 000FB850 004ED19E 03EC1E00 800FB852 * ...&+.J..... *
00787140 000050 00000000 00000000 00000000 00000000 * ..... *
00787150 TO 007871C0 SUPPRESSED LINE(S) SAME AS ABOVE .....
007871C0 0000D0 00000000 00000000 * ..... *

GPR 0 = 00000000 009C91F0 0082C130 00B8D000
GPR 4 = 0082C134 00000001 00000000 00822B10
GPR 8 = 00000001 007E8270 0081EF30 0082C008
GPR 12 = 009C8668 0082C008 4E9C9166 0092EE58

FAILURE AT OFFSET +00003162 IN ARIXRDS PROGRAM (009C6000)

CALLED FROM OFFSET +00000796 IN ARIXRDS PROGRAM (009C6000)
CALLED FROM OFFSET +000004DE IN ARIXERD 93.090 PLX

SUMMARY OF USERS

DS2CVT      RDCVT      YRSSCVT
00B8D000    00B8D660    00B8D3A0

CURRENTLY RUNNING DCE 007E8270

USERID      PROGRAM NAME      DSCAREA  YTABLE1  RDAREA    DCE
001 OPERATOR      00806688  00806858                007E8048
002 CHECKPT      00809EC0  0080A090                007E8100
003 RECOVERY     0080C628  0080C7F8  008172D8  007E81B8
004 SQLUSR5      0081ED60  0081EF30. 0082BA10  007E8270

NOTE: USERID AND/OR PROGRAM NAME MAY BE RESIDUAL AND MAY NOT NECESSARILY
      BE THE CURRENT VALUES

SYMPTOM STRING:
MS/ARI0040E      PIDS/568810301      RIDS/ARICMUD      PRCS/00000003
  
```

Figure 34. SQLIDS Minidump, Example for SQL/DS System Error

Note: The situation here was that module ARIXERD called module ARICMUD. ARICMUD then detected an internal system error at error detection point 3.

First Failure Data Capture

First Failure Data Capture support is only provided when using the DRDA protocol. With this support, it may not be necessary to re-create an error in order to report it to the IBM support center. All relevant diagnostic information, control blocks and data streams are captured at the error detection point in the VM console log of the application requester or application server and held in the print queue.

This support is provided on both the application requester and the application server, but the formats are different. It is invoked automatically when an error in the communication data stream is detected, or an internal error is detected. At that time, you will receive message ARI2909I which tells you that the error occurred and that data is being captured for the dump. When you receive message ARI2910I, you will know that the data capture process has completed successfully.

The information captured includes:

- System information
 - LU 6.2 logical unit of work identifier
 - time and date
 - communication information
 - symptom string
 - probable cause string
- Failure point specific data area
- Control blocks
- request/reply data streams

The number of control blocks and data streams depend on the point at which the error was detected. These are used by IBM Support personnel for diagnosis.

To complete the keyword string for a First Failure Data Capture, the following keywords should be specified:

MS/message id
RIDS/module name
PRCS/return code

where message id is ARI2909I,
module name is ARlxxxx from the dump,
return code is the nn.

The keywords are found in the symptom string of the dump.

Following is a modified example of an application requester First Failure Data Capture dump. This type of dump is typically several pages long.


```
##### BEGIN FIRST FAILURE DATA CAPTURE DUMP #####

APPLICATION REQUESTER DUMP

LUWID = SNANETID.*IDENT.A36973962095.0001   DATE/TIME = 02-05-91/15:15:52

EXTNAM:   JONES
RDBMS:    PAYROLL1           SQLDS/VM V3.3.0
PACKAGE:  JONES             .PAYROLL           SECTION: 0001
LU:       *IDENT
TPN:      PAYROLL1   X('D7C1E8D9D6D3D3F1')

SYMPTOM STRING:  MS/ARI2909I PIDS/568810301 RIDS/ARITFQA PRCS/01

PROBABLE CAUSE OF FAILURE:
NULL POINTER

##### DATA AREA BLHBUF #####

00000010 00000000 000064B0                * .... *

##### END DATA AREA #####

##### CONTROL BLOCK DUMP FOR VMCBLOCK #####

00386DC8 00000000 E5D4C3C2 D3D6C3D2 00000128 00000000 * VMCBLOCK..... *
00386DD8 00000010 00000000 00000000 0F00000F 00000000 * ..... *
00386DE8 00000020 00000000 00000000 00000000 00000000 * ..... *
00386DF8 TO 00386EE8 SUPPRESSED LINE(S) SAME AS ABOVE ....
00386EE8 00000120 00000000 00000000                * ..... *

##### END CONTROL BLOCK DUMP #####

##### CONTROL BLOCK DUMP FOR BLHEADER #####

002F5000 00000000 D9D4C2E4 C6C64040 00000030 00007FFF * RMBUFF .....". *
002F5010 00000010 002F5030 0000002D 00000000 00319000 * ..&..... *
002F5020 00000020 00329060 00D50000 00000000 00000000 * ...-N..... *

##### END CONTROL BLOCK DUMP #####

##### REPLY DATA STREAM #####

002F5030 00000000 002DD003 03C50027 24130019 00100676 * .....E..... *
002F5040 00000010 D0020004 0971E054 0001D000 010671F0 * .....0 *
002F5050 00000020 E0000000 0A147AFF 00000000 0C                * .....:..... *

##### END DATA STREAM DUMP #####

##### END FIRST FAILURE DATA CAPTURE DUMP #####
```

Figure 35. Application requester First Failure Data Capture Dump

Following are descriptions of all possible fields contained in an application requester dump. In general, a First Failure Data Capture Dump may contain all or only some of these fields.

LUWID This field contains an LU 6.2 Logical Unit of Work Identifier.

DATE/TIME Date and time in MM-DD-YY/HH:MM:SS format.

EXTNAM Name of the user task on the application requester.

RDBMS Name of the application server the application requester is connected to. This includes 18 byte application server name, the product name of the application server and the version of the product the application server is running.

PACKAGE and SECTION These parameters appear on the same line. The PACKAGE parameter is in the format "*collection-id.package-name*". The SECTION parameter contains the section in the package that is currently being executed.

LU Logical unit name. This identifies the gateway used to establish the connection to the application server. With an SQL/DS application requester, the LU is either *IDENT (for a local or TSAF collection) or an AVS gateway name. With a non-SQL/DS application requester, the LU is the LU name of the application server.

TPN Transaction Program Name. This is the name used to identify the connection to the application server on the target network.

SYMPTOM STRING The symptom string consists of a set of keywords and values.

PROBABLE CAUSE OF FAILURE: The line following this parameter should contain a short description of why the error occurred.

The parameters PACKAGE, SECTION, LU, and TPN may or may not appear in a given dump. They do not appear if the current DDM request does not concern a specific package, and the LU and TPN may not appear if the connection is local.

Following is a modified example of an application server First Failure Data Capture dump. This type of dump is typically several pages long.

```
##### BEGIN FIRST FAILURE DATA CAPTURE DUMP #####

APPLICATION SERVER DUMP

LUWID = SNANETID.*IDENT.A3DD074FE6BF.0001    DATE/TIME = 05-08-91/14:15:12

VM USERID: JONES          SQLID: JONES
EXTNAM:   JONES.1
REQUESTER: SQLDS/VM      V03.03.0    AT TORVMLB8
PACKAGE:  JONES          .PAYROLL          SECTION: 0001

SYMPTOM STRING:  MS/ARI2909I PIDS/568810301 RIDS/ARIWDDM PRCS/04

PROBABLE CAUSE OF FAILURE:
INVALID CORRELATION ID IN DDM HEADER

##### DATA AREA ASPCORID #####

007E393A 00000000 0008                * ..                *

##### END DATA AREA #####

##### DATA AREA DDMSVCOR #####

007D6300 00000000 FFFFFFFF            * ....             *

##### END DATA AREA #####

##### CONTROL BLOCK DUMP FOR VMQ #####

00798FD8 00000000 E5D4D8C5 D3C5D440 00000078 00799188 * VMQELEM .....jh *
00798FE8 00000010 00000000 00000000 00000000 00000000 * .....           *
00798FF8 00000020 00000000 00000000 00000000 00000000 * .....           *
00799008 00000030 00000000 007983F0 00799050 00B58410 * .....c0...&..d. *
00799018 00000040 50204004 00000000 00000000 00000000 * &. ....           *
00799028 00000050 00000000 00000000 00000000 00000000 * .....           *
00799038 00000060 00000000 A3DD0762 10E9C800 A3DD0750 * ....t....ZH.t..& *
00799048 00000070 0B57E800 02000000                * ..Y.....        *

##### END CONTROL BLOCK DUMP #####

##### REQUEST DATA STREAM #####

007CCED0 00000000 0020D001 0008001A 200F0016 2110E2D8 * .....SQ *
007CCEE0 00000010 D3D4C1C3 C8D44040 40404040 40404040 * LDBA          *

##### END DATA STREAM DUMP #####

##### END FIRST FAILURE DATA CAPTURE DUMP #####
```

Figure 36. Application server First Failure Data Capture Dump

Following are descriptions of all possible fields contained in an application server dump. In general a First Failure Data Capture dump may contain all or only some of these fields.

LUWID This field contains an LU 6.2 Logical Unit of Work Identifier.

DATE/TIME Date and time in MM-DD-YY/HH:MM:SS format.

VM USERID This is the VM userid received by the application server. Note that in a remote connection, this may not be the original userid on the originating system due to userid translation.

SQLID This is the SQL/DS user id of the application requester.

EXTNAM Name of the user task on the application requester.

REQUESTER This is the type of application requester connected to the application server. This includes the product name of the application server, the version of the product the application server is running, and the server name of the application requester.

PACKAGE and SECTION These parameters appear on the same line. The PACKAGE parameter is in the format "*collection-id.package-name*". The SECTION parameter contains the section in the package that is currently being executed.

SYMPTOM STRING The symptom string consists of a set of keywords and values.

PROBABLE CAUSE OF FAILURE: The line following this parameter should contain a short description of why the error occurred.

The PACKAGE and SECTION parameters may not appear in a given dump. They do not appear if the current DDM request does not concern a specific package.

No Response

No Response can be either a wait or loop condition. Before contacting IBM support center, you should go to Chapter 5, "Diagnosing Performance Problems" on page 101 and determine if your situation may be caused by a performance problem within your system.

Wait or Loop

For a wait or loop condition, the following keywords should be specified:

PDSYM = NON AUTOMATED
EXTSYM = ABRATE
RIDS = module name

The module name (identified above by RIDS =) should be provided if it can be determined. If the specific module name cannot be determined, it is recommended you use the following names depending on which environment you are running:

ISQL - ARIISQL
DBS Utility - ARIDBS
SQL/DS - ARISQLDS

Slow Response

Slow response is a performance related problem that you feel is caused by the SQL/DS product and not due to application design. Before contacting the IBM support center, you should go to Chapter 5, "Diagnosing Performance Problems" on page 101 and determine if your situation may be caused by a performance problem within your system. For a slow response problem the following keywords should be specified:

PDSYM = NON AUTOMATED
EXTSYM = ABRATE
RIDS = module name

The module name (identified above by RIDS =) should be provided if it can be determined. If the specific module name cannot be determined, it is recommended you use the following names depending on which environment you are running:

ISQL - ARIISQL
DBS Utility - ARIDBS
SQL/DS - ARISQLDS

Incorrect or Missing Output

When incorrect or incomplete output is delivered to the user, specify the following keywords:

PDSYM = NON AUTOMATED
EXTSYM = RESULT
RIDS = module name

The module name (identified above by RIDS =) should be provided if it can be determined. If the specific module name cannot be determined, it is recommended you use the following names depending on which environment you are running:

ISQL - ARIISQL
DBS Utility - ARIDBS
SQL/DS - ARISQLDS

Document

When you find incorrect or misleading information in an SQL/DS publication, or cannot find information that should be there, the keywords below should be specified. Before you do this, however, consider the simpler alternative: Describe your problem on the Reader's Comment Form in the back of the book with the apparent defect. Then mail the completed form to IBM according to the form's instructions. Only extremely severe defects warrant the procedure in this section, such as when a documented procedure causes permanent damage to the database.

The following keywords should be specified for severe documentation problems:

PDCOND = NON EXECUTION
PDAID = PUB KEYED
PUBS/document number

where document number is the xxnn-nnnn-vv from the publication's cover.

Additional Keywords

Additional keywords can be used to describe failures related to SQLCODES, SQL Statements, Start-up Parameters, Data Types, Application Languages, and EXECs. These keywords will provide more information to expedite the search for previously identified problems and resolutions.

SQLCODES

If the failure is related to an SQLCODE, use the following keywords:

PRCS/SQLCODE (absolute value)
PRCS/SQLERRD1 (absolute value)
PRCS/SQLERRD2 (absolute value)
PRCS/SQLERRP

Note: The code must be 8 digits long, and left padded with zeroes.

For the following SQLCA values:

SQLCODE -901, SQLERRD1 -1, SQLERRD2 -30, SQLERRP ARIXRSS

You would use these keywords:

PRCS/00000901, PRCS/00000001, PRCS/00000030, RIDS/ARIXRSS

SQL Statements

Use all SQL/DS reserved keywords that are related to the failure. For an SQL statement the following keywords should be specified.

PCSS/keyword

For example, if the failing statement was:

```
SELECT * FROM SQLDBA.ACTIVITY -  
        WHERE ACTNO IN (85,95)
```

then the symptom string would be:

PCSS/SELECT PCSS/WHERE PCSS/IN

Start-up Parameters

If the failure is related to the value of a start-up parameter, use the following keyword:

PCSS/PARAMETER
PCSS/n (where n is the parameter value)

For example, if the failure only occurs in single user mode, (SYSMODE=S), then the keywords would be:

PCSS/SYSMODE
PCSS/S

Data Type

If the failure is dependant on the field data type, use the keyword:

FLDS/DATA TYPE

Here is a list of data types and keywords which can be used:

SMALL INTEGER	FLDS/SMALLINT
INTEGER	FLDS/INTEGER
DECIMAL	FLDS/DECIMAL
DECIMAL(5,2)	FLDS/DECIMAL VALU/C0502
FLOATING POINT	FLDS/FLOAT
CHARACTER	FLDS/CHAR
VARYING CHARACTER	FLDS/VARCHAR
VARYING GRAPHIC	FLDS/VARGRAPHIC
LONG VARCHAR	FLDS/LONG FLDS/VARCHAR
LONG VARGRAPHIC	FLDS/LONG FLDS/VARGRAPHIC
HOST VARIABLE	FLDS/HOSTVAR
DATE	FLDS/DATE
TIME	FLDS/TIME
TIMESTAMP	FLDS/TIMESTAMP

Note: The keyword DECIMAL should be used for table columns defined as NUMERIC.

Application Type

If the failure is dependant on the language of the application program, use one of these keywords:

RIDS/PLI	RIDS/FORTRAN
RIDS/ASSEMBLER	RIDS/COBOL
RIDS/C	RIDS/RPG

EXECs

If the failure occurs as a result of running an EXEC, use the following keyword:

RIDS/exec name

For example, if the failure occurred while running a COLDLOG (EXEC SQLLOG), the keyword would be:

RIDS/SQLLOG

Application Program Generated SQLCODES

An interface is provided allowing application programs to format a symptom string which can be used as another search argument to identify application problems.

Invocation

The interface is composed of two modules that can be called from an application program. The modules receive an SQLCA and return a formatted symptom string in the form of five character strings.

For FORTRAN application programs, the invocation is:

```
---> CALL ARISSMF(SQLCOD,SQLERP,S1,S2,S3,S4,S5) /* FORTRAN */
```

For application programs of other languages, the invocations are as follows:

```

====> CALL ARISSMA(SQLCA,S1,S2,S3,S4,S5);          /* PL/I          */
====> CALL 'ARISSMA' USING SQLCA S1 S2 S3 S4 S5.    /* COBOL        */
====> ARISSMA(&sqlca,S1,S2,S3,S4,S5)              /* 'C'          */
====> CALL ARISSMA,(SQLCA,S1,S2,S3,S4,S5),VL      /* Assembly Language */
====> CALL ARISSMA(SQLCA,S1,S2,S3,S4,S5)          /* RPG          */
  
```

(Note: For Assembly Language and RPG, this is pseudocode only.
 See the *Application Programming* manual for syntax.)

Module	
ARISSMA	ARISSMA can be called by any PL/I, RPG, Assembler, COBOL and 'C' application program.
ARISSMF	ARISSMF can be called by any FORTRAN application program.
Input	
SQLCA	The SQLCA causing the error.
Output	Length Symptom string for
SQLCSTR1	13 SQLCODE
SQLCSTR2	13 SQLERRD1
SQLCSTR3	13 SQLERRD2
SQLCSTR4	12 SQLERRP (part 1)
SQLCSTR5	14 SQLERRP (part 2)

Figure 37. Symptom String invocation

The values returned by ARISSMA and ARISSMF are as follows:

- SQLCSTR1** PRCS/nnnnnnnn; where n is the decimal representation of the absolute value of the SQLCODE, right justified, padded with 0's, for a total length of 8 digits.
- SQLCSTR2** PRCS/nnnnnnnn; where n is the decimal representation of the absolute value of the SQLERRD1, right justified, padded with 0's, for a total length of 8 digits.
- SQLCSTR3** PRCS/nnnnnnnn; where n is the decimal representation of the absolute value of the SQLERRD2, right justified, padded with 0's, for a total length of 8 digits.
- SQLCSTR4** FLDS/SQLERRP. This value is always returned in the string.
- SQLCSTR5** VALU/Caaaaaaa; where a is left justified, padded by blanks, and is the module name provided in field SQLERRP.

Suppose the SQLCA fields have the following values when the error occurred:


```
SQLCODE = -901  
SQLERRD1 = -160  
SQLERRD2 = -33  
SQLERRP = ARIXOEX
```

then the values of the strings will be:

```
SQLCSTR1 ==> PRCS/00000901  
SQLCSTR2 ==> PRCS/00000160  
SQLCSTR3 ==> PRCS/00000033  
SQLCSTR4 ==> FLDS/SQLERRP  
SQLCSTR5 ==> VALU/CARIXOEX
```

Interactions

An application program can format a symptom string by calling ARISSMA or ARISSMF, passing the SQLCA structure, and the 5 output strings.

The application program can then display or write the symptom string to a file as required.

No messages are issued by ARISSMA and ARISSMF. The symptom strings are built based on the SQLCA received.

Reporting a Problem

This section describes the types of material that can be sent to the IBM change team when the problem reported to the IBM support center (using the keyword string constructed) is a unique problem. The type of material required depends on the problem encountered.

If your problem is occurring during distributed processing, you should also refer to the *Distributed Relational Database Problem Determination Guide* for further information on the types of material to collect.

Materials

This is the list of materials. See the Environments section for which material is necessary for your problem.

- Environment
 - Operating system including version and mode, where applicable (for example, VM/ESA 1.0 XA mode).
 - Other environment specifics (for example, fullscreen or VTAM*).
 - SQL/DS release level.
 - Indication as to whether or not the DRDA code is installed.
 - Protocol parameter being used by the application server (AUTO or SQLDS) and the application requester (AUTO, SQLDS, or DRDA).
 - Identification of the SQL products involved (for example, SQL/DS application server and OS/400* Database Manager application requester).
 - Virtual storage size.
 - CHARNAME (or CCSIDs) being used by the application server and the application requester. If the CHARNAME being used is not an IBM-supplied CHARNAME, provide specifics for that CHARNAME (that is, the CCSID(s), the conversion table(s) and the entry into the SYSCHESETS catalog table).

- Maintenance History
 - VM PUT service level applied.
 - System configuration changes.
- SQL/DS User Tables
 - Tables used as input to the failing request.
 - Table definitions, index definitions, and optionally table data.
 - Obtained via an SQL SELECT statement issued from ISQL or DBS Utility.
- Failing SQL Statement or Sequence of Statements
 - Is necessary for some problems.
- The Entire SQLCA
 - From the SQL/DS user application.
 - Obtaining the *entire* SQLCA requires coding in the application program.
- SQL Codes
 - Usually indicate errors in use of the SQL language.
 - May appear in SQL/DS error messages.
 - May also appear in SQLCA control block in user application.
- Output from EXPLAIN statement
 - Is necessary for some problems.
 - Obtained via an SQL SELECT statement issued from ISQL or DBS Utility.
 - For details on the EXPLAIN statement refer to the *Database Administration* manual.
- SQL/DS Accounting Record Information
 - Is necessary for some problems.
 - For more information on accounting records, refer to the *System Administration* manual.
- VM Dump
 - Taken by the operator for loops and waits.
 - Either the SQL/DS machine or the user machine must be dumped.
- Database Machine Dump
 - Optionally taken by the database manager for abnormal terminations and detected system errors (DUMPTYPE = F in SQLSTART).
 - Output to the SQL/DS operator console and/or job listing.
- SQL/DS Minidump
 - Taken by the database manager for abnormal terminations in the database machine.
 - Output to the SQL/DS operator console and/or job listing.
- Job Output Listings
 - From the database machine.
 - From the SQL/DS user application.
- First Failure Data Capture Dump
 - From the print queue

- SQL/DS Operator Console
 - Error messages.
 - SQL return codes.
 - Output from the SQL/DS SHOW SYSTEM command.
 - Output from the SQL/DS SHOW CONNECT command.
- Terminal Input and Output
 - SQL or ISQL commands entered.
 - Error messages.
 - Display results.
 - Output from SQL/DS SHOW SYSTEM command.
- SQL/DS System Catalog
 - Obtained via SQL SELECT statement issued from ISQL or DBS Utility.
 - For names of the system catalog tables and their contents refer to the *SQL Reference* manual.
- SQL/DS Trace Output
 - For details on Trace usage and output refer to the *Operation* manual.
- Instruction Trace
 - For small loops.
 - Use hardware instruction step or PER trace.
- SQL/DS Error Messages
 - May appear on operator console, job listing, or ISQL terminal.
 - Record message id numbers, as well as exact message text.
- VM PER Trace
 - For large loops.

Environments

This section describes what materials would be required for the different types of failures in each environment. For the following descriptions, it is assumed that the failure occurred in multiple user mode. If the application server is running in single user mode, the database machine and the user machine will be one and the same.

Note: Maintenance history is required for all problems. Items in square brackets [] indicate optional materials.

PROBLEM TYPE	PROBLEMS IN THE DATABASE MACHINE	PROBLEMS IN THE USER MACHINE	ISQL PROBLEMS
ABNORMAL TERMINATION	SQL/DS Minidump SQL/DS Machine Dump First Failure Data Capture Dump* [SQL/DS Operator Console] [SQL/DS Trace Output]	VM Dump JOB Output Listings First Failure Data Capture Dump* [SQL/DS Operator Console]	VM Dump Terminal Input and Output [SQL/DS Operator Console]
MESSAGE	SQL/DS Operator Console SQL/DS Error Messages First Failure Data Capture Dump* [SQL/DS Machine Dump] [SQL/DS Trace Output]	The Entire SQLCA SQL/DS Error Messages First Failure Data Capture Dump*	
WAIT	VM Dump SQL/DS SHOW SYSTEM Command [SQL/DS Operator Console] [SQL/DS Trace Output]	SQL/DS SHOW SYSTEM Command VM Dump JOB Output Listings [SQL/DS Operator Console]	SQL/DS SHOW SYSTEM Command VM Dump Terminal Input and Output [SQL/DS Operator Console]
LOOP	Instruction Trace (small loops) VM PER Trace (large loops) SQL/DS SHOW SYSTEM Command VM Dump [SQL/DS Operator Console] [SQL/DS Trace Output]	Instruction Trace (small loops) VM PER Trace (large loops) SQL/DS SHOW SYSTEM Command VM Dump JOB Output Listings [SQL/DS Operator Console]	Instruction Trace (small loops) VM PER Trace (large loops) SQL/DS SHOW SYSTEM Command Terminal Input and Output [SQL/DS Operator Console]
INCORRECT OUTPUT	Application Server CHARNAME Value	SQL Codes SQL/DS Error Messages JOB Output Listings Application requester CHARNAME Value [SQL/DS System Catalog] [SQL/DS User Tables] [SQL/DS Trace Output]	Terminal Input and Output SQL/DS Error Messages [SQL/DS System Catalog] [SQL/DS User Tables]

Figure 38. Materials for Defect Problems

* The First Failure Data Capture Dump is only available when using the DRDA protocol. Even then it is not always available. When it is present, it is required information that must be sent to the IBM change team when reporting a problem.

SYMPTOM KEYWORD STRING FOR **ABNORMAL TERMINATIONS**
(for the SQL/DS Licensed Program)

SECTION I (PRODUCT IDENTIFICATION)

COMPONENT ID: 568800401 (for Version 2 Release 2)
COMPONENT ID: 568810301 (for Version 3 Release 1, 2, 3, 4)

LEVEL CODE: __LVLS/220 (for Version 2 Release 2, English) (check one)
 __LVLS/___ (for Version 2 Release 2, other languages)
 __LVLS/310 (for Version 3 Release 1, English)
 __LVLS/___ (for Version 3 Release 1, other languages)
 __LVLS/320 (for Version 3 Release 2, English)
 __LVLS/___ (for Version 3 Release 2, other languages)
 __LVLS/330 (for Version 3 Release 3, English)
 __LVLS/___ (for Version 3 Release 3, other languages)
 __LVLS/340 (for Version 3 Release 4, English)
 __LVLS/___ (for Version 3 Release 4, other languages)

SECTION II (PROBLEM IDENTIFICATION)

ABEND CODE: AB/S_____

MODULE NAME: RIDS/_____

SECTION III (PROBLEM RESOLUTION)

SERVICE RESPONSE: __ RET (Return)
(check one) __ PER (Program Error)
 __ DOC (Documentation error)
 __ USER (User Error)
 __ NTF (No trouble found)

APAR: _____ (APAR number)

SERVICE: PUT/_____ (PUT level)
 PTF = _____ (Corrective Fix)

NOTES:

This page may be reproduced without written permission from IBM

SYMPTOM KEYWORD STRING FOR MESSAGE
(for the SQL/DS Licensed Program)

SECTION I (PRODUCT IDENTIFICATION)

| COMPONENT ID: 568800401 (for Version 2 Release 2)
| COMPONENT ID: 568810301 (for Version 3 Release 1, 2, 3, 4)

LEVEL CODE: __LVLS/220 (for Version 2 Release 2, English) (check one)
 __LVLS/___ (for Version 2 Release 2, other languages)
 __LVLS/310 (for Version 3 Release 1, English)
 __LVLS/___ (for Version 3 Release 1, other languages)
 __LVLS/320 (for Version 3 Release 2, English)
 __LVLS/___ (for Version 3 Release 2, other languages)
 __LVLS/330 (for Version 3 Release 3, English)
 __LVLS/___ (for Version 3 Release 3, other languages)
| __LVLS/340 (for Version 3 Release 4, English)
| __LVLS/___ (for Version 3 Release 4, other languages)

SECTION II (PROBLEM IDENTIFICATION)

MESSAGE ID: MS/_____ (ARInnnnX)
MODULE NAME: RIDS/_____ (if MS/ARI0040E)

RETURN CODE: PRCS/_____ (Return code or absolute
 value of SQLCODE in decimal)

SECTION III (PROBLEM RESOLUTION)

SERVICE RESPONSE: __ RET (Return)
(check one) __ PER (Program Error)
 __ DOC (Documentation error)
 __ USER (User Error)
 __ NTF (No trouble found)

APAR: _____ (APAR number)

SERVICE: PUT/_____ (PUT level)
 PTF = _____ (Corrective Fix)

NOTES:

This page may be reproduced without written permission from IBM

SYMPTOM KEYWORD STRING FOR NO RESPONSE
(for the SQL/DS Licensed Program)

SECTION I (PRODUCT IDENTIFICATION)

| COMPONENT ID: 568800401 (for Version 2 Release 2)
| COMPONENT ID: 568810301 (for Version 3 Release 1, 2, 3, 4)

LEVEL CODE: __LVLS/220 (for Version 2 Release 2, English) (check one)
 __LVLS/___ (for Version 2 Release 2, other languages)
 __LVLS/310 (for Version 3 Release 1, English)
 __LVLS/___ (for Version 3 Release 1, other languages)
 __LVLS/320 (for Version 3 Release 2, English)
 __LVLS/___ (for Version 3 Release 2, other languages)
 __LVLS/330 (for Version 3 Release 3, English)
 __LVLS/___ (for Version 3 Release 3, other languages)
| __LVLS/340 (for Version 3 Release 4, English)
| __LVLS/___ (for Version 3 Release 4, other languages)

SECTION II (PROBLEM IDENTIFICATION)

PD SYMPTOM: PDSYM = NON AUTOMATED

EXTERNAL SYMPTOM: EXTSYM = ABRATE

MODULE NAME: RIDS/_____

SECTION III (PROBLEM RESOLUTION)

SERVICE RESPONSE: __ RET (Return)
(check one) __ PER (Program Error)
 __ DOC (Documentation error)
 __ USER (User Error)
 __ NTF (No trouble found)

APAR: _____ (APAR number)

SERVICE: PUT/_____ (PUT level)
 PTF = _____ (Corrective Fix)

NOTES:

This page may be reproduced without written permission from IBM

SYMPTOM KEYWORD STRING FOR SLOW RESPONSE
(for the SQL/DS Licensed Program)

SECTION I (PRODUCT IDENTIFICATION)

COMPONENT ID: 568800401 (for Version 2 Release 2)
COMPONENT ID: 568810301 (for Version 3 Release 1, 2, 3, 4)

LEVEL CODE: __LVLS/220 (for Version 2 Release 2, English) (check one)
 __LVLS/___ (for Version 2 Release 2, other languages)
 __LVLS/310 (for Version 3 Release 1, English)
 __LVLS/___ (for Version 3 Release 1, other languages)
 __LVLS/320 (for Version 3 Release 2, English)
 __LVLS/___ (for Version 3 Release 2, other languages)
 __LVLS/330 (for Version 3 Release 3, English)
 __LVLS/___ (for Version 3 Release 3, other languages)
 __LVLS/340 (for Version 3 Release 4, English)
 __LVLS/___ (for Version 3 Release 4, other languages)

SECTION II (PROBLEM IDENTIFICATION)

PD SYMPTOM: PDSYM = NON AUTOMATED

EXTERNAL SYMPTOM: EXTSYM = ABRATE

MODULE NAME: RIDS/_____

SECTION III (PROBLEM RESOLUTION)

SERVICE RESPONSE: __ RET (Return)
(check one) __ PER (Program Error)
 __ DOC (Documentation error)
 __ USER (User Error)
 __ NTF (No trouble found)

APAR: _____ (APAR number)

SERVICE: PUT/_____ (PUT level)
 PTF = _____ (Corrective Fix)

NOTES:

This page may be reproduced without written permission from IBM

SYMPTOM KEYWORD STRING FOR INCORRECT OR MISSING OUTPUT
(for the SQL/DS Licensed Program)

SECTION I (PRODUCT IDENTIFICATION)

COMPONENT ID: 568800401 (for Version 2 Release 2)
COMPONENT ID: 568810301 (for Version 3 Release 1, 2, 3, 4)

LEVEL CODE: __LVLS/220 (for Version 2 Release 2, English) (check one)
 __LVLS/___ (for Version 2 Release 2, other languages)
 __LVLS/310 (for Version 3 Release 1, English)
 __LVLS/___ (for Version 3 Release 1, other languages)
 __LVLS/320 (for Version 3 Release 2, English)
 __LVLS/___ (for Version 3 Release 2, other languages)
 __LVLS/330 (for Version 3 Release 3, English)
 __LVLS/___ (for Version 3 Release 3, other languages)
 __LVLS/340 (for Version 3 Release 4, English)
 __LVLS/___ (for Version 3 Release 4, other languages)

SECTION II (PROBLEM IDENTIFICATION)

PD SYMPTOM: PDSYM = NON AUTOMATED

EXTERNAL SYMPTOM: EXTSYM = RESULT

MODULE NAME: RIDS/_____

SECTION III (PROBLEM RESOLUTION)

SERVICE RESPONSE: __ RET (Return)
(check one) __ PER (Program Error)
 __ DOC (Documentation error)
 __ USER (User Error)
 __ NTF (No trouble found)

APAR: _____ (APAR number)

SERVICE: PUT/_____ (PUT level)
 PTF = _____ (Corrective Fix)

NOTES:

This page may be reproduced without written permission from IBM

SYMPTOM KEYWORD STRING FOR DOCUMENTATION PROBLEMS
(for the SQL/DS Licensed Program)

SECTION I (PRODUCT IDENTIFICATION)

COMPONENT ID: 568800401 (for Version 2 Release 2)
COMPONENT ID: 568810301 (for Version 3 Release 1, 2, 3, 4)

LEVEL CODE: __ LVLS/220 (for Version 2 Release 2, English) (check one)
 __ LVLS/___ (for Version 2 Release 2, other languages)
 __ LVLS/310 (for Version 3 Release 1, English)
 __ LVLS/___ (for Version 3 Release 1, other languages)
 __ LVLS/320 (for Version 3 Release 2, English)
 __ LVLS/___ (for Version 3 Release 2, other languages)
 __ LVLS/330 (for Version 3 Release 3, English)
 __ LVLS/___ (for Version 3 Release 3, other languages)
 __ LVLS/340 (for Version 3 Release 4, English)
 __ LVLS/___ (for Version 3 Release 4, other languages)

SECTION II (PROBLEM IDENTIFICATION)

PD CONDITION: PDCOND = NON EXECUTION

PD AID: PDAID = PUB KEYED

PUBLICATION: PUBS/_____ (document number)

SECTION III (PROBLEM RESOLUTION)

SERVICE RESPONSE: __ RET (Return)
(check one) __ PER (Program Error)
 __ DOC (Documentation error)
 __ USER (User Error)
 __ NTF (No trouble found)

APAR: _____ (APAR number)

SERVICE: PUT/_____ (PUT level)
 PTF = _____ (Corrective Fix)

NOTES:

This page may be reproduced without written permission from IBM

Chapter 4. Functional Problems

This chapter describes analysis of functional problems that might be encountered in the use of the SQL/DS product. Problems are defined in this chapter by symptoms; a symptom might be an SQL/DS error code or a functional deviation.

For each symptom identified, possible causes of the problem are listed along with the corresponding actions that should be taken. In some cases the action will involve use of problem determination facilities or suggestions to further isolate and identify the problem. This chapter is meant to supplement the *Messages and Codes* manual. If an error code is sufficiently described in that manual so that no additional suggestions are necessary, that code will not be listed in this chapter.

This chapter is organized into the following sections:

- System-Related Error Codes
- Common User-related Error Codes
- Functional Deviations

System-Related Error Codes

The error codes listed here usually require action by the Systems Programmer. Those for which the probable cause is 'INTERNAL SYSTEM ERROR' will usually require the Systems Programmer to record appropriate data and contact the designated IBM support group.

SQL COMMAND FAILED (-901)

SQL CODE	RDS CODE (SQLERRD1)	DETECTED BY (SQLERRP)	REASON
-901	-100	ARIXSUT	Alert the system programmer. The SQL/DS database manager cannot use the STORE CLOCK value provided by the operating system because of any of the following reasons: 1. The clock value represents elapsed time instead of time of day, 2. The clock is not in an operational state, 3. The clock is in an error state.

Note: For System Error related SQL Code -901 occurrences, Diagnostics should be run by the System Programmer before contacting the IBM support group.

ROLLED BACK DUE TO A DEADLOCK (-911)

If this occurs frequently, have the system programmer investigate via SHOW LOCKS and the guidelines in Chapter 5, "Diagnosing Performance Problems" on page 101 where the bottleneck is, to determine whether design or system alterations would alleviate the problem.

EXCESSIVE DEADLOCKS: An unusually large number of deadlocks occur between your applications.

Deadlocks are inherent to a DBMS with concurrent access. The SQL/DS database manager does deadlock detection before placing any user in a lock wait. The classic example of a deadlock situation is: User A holds resource X, which User B wants, while User B is holding resource Y, which user A wants. There is an impasse which the deadlock detector removes by rolling back the LUW which started last.

If the applications are accessing the same table(s), then the deadlocks are most likely due to the sequence of access to those tables. If the deadlocks are occurring between two applications that are accessing different user data, then the contention is with the system catalogs.

ACTIONS: To try to theorize the possible points of contention that would result in a deadlock would be futile in this publication. Therefore, it is recommended that the Trace facility be used to collect locking information to determine the points of contention resulting in a deadlock. To do this, refer to "Using Trace for Deadlocks" section in Chapter 8, "Problem Isolation and Handling" on page 247.

ROLLED BACK DUE TO EXCESSIVE (SYSTEM WIDE) LOCK REQUESTS (-912)

If this happens frequently, the system programmer should monitor lock requests and refer to guidelines in Chapter 5, "Diagnosing Performance Problems" on page 101 of this book. (See "NLRB Parameters Too Small" on page 191.)

ROLLED BACK DUE TO EXCESSIVE LOCKS HELD FOR THIS LUW (-915)

Either the number of lock request blocks per user is too small (check with the system programmer), or the locks are at too low a level (check with the system or application programmers), or the program should be reviewed to see if all accesses are necessary or if COMMIT WORKs issued within the program could free up some of the locks. (See "NLRB Parameters Too Small" on page 191 in Chapter 5, "Diagnosing Performance Problems" on page 101.)

Common User-related Error Codes

SQL COMMAND LIMITATION EXCEEDED (-101)

Occurring when using ISQL, DBSU or when preprocessing an application program, or executing a PREPARE or EXECUTE IMMEDIATE SQL statement, this error is usually because of an internal SQL/DS storage limitation. The storage limitation was caused by one of the following:

1. The query is too long.

An SQL command cannot exceed 8192 characters. It is highly unlikely that you have exceeded this limit, but you should check to make sure. Do not count extra blanks and do not worry about the expansion of a view.

2. Too many columns.

3. Too many predicates.

4. View expansion of the request.

ACTIONS:

1. **Shorten the query.** If inspection of the query shows that it is more than 8192 bytes in length, then it must be shortened by using table labels, by using views with shorter column names or by reorganizing the formulation of the query.

2. **Break up the query.** If the query references too many columns, then it may be necessary to break it up into multiple requests. If the end result does not reference too many columns, then you may be able to do the desired function in a series of steps using tables for holding intermediate results.

3. **Simplify the query.**

If none of the above are indicated, then you probably have more predicates than the SQL/DS database manager can handle or some internal storage space has been exceeded. For a list of -101 SQLCODES and an explanation indicating which limit is being exceeded, see the Messages and Codes manual.

CREATOR.TABLE WAS NOT FOUND (-204)

This error may be generated because the table really does not exist, or because CREATOR was not specified and the userid by which the user was connected to the database is appended as the creator.

To resolve this, create the table, connect with appropriate userid, or define a synonym for the table so that "CREATOR" need not be coded.

INPUT VARIABLE DATA TYPE NOT COMPATIBLE WITH COLUMN (-301)

The database manager was attempting to do conversion of an input host variable to the data type of a column or expression in an SQL statement. Possible reasons for getting the -301 SQLCODE are:

DATA TYPE OF TARGET COLUMN OR EXPRESSION	DATA TYPE OF INPUT HOST VARIABLE WAS NOT:
FLOAT	NUMERIC: FLOAT, DECIMAL, INTEGER, SMALLINT
DECIMAL	NUMERIC: FLOAT, DECIMAL, INTEGER, SMALLINT
INTEGER	NUMERIC: FLOAT, DECIMAL, INTEGER, SMALLINT
SMALLINT	NUMERIC: FLOAT, DECIMAL, INTEGER, SMALLINT
CHAR	CHAR, VARCHAR, OR LONG VARCHAR
GRAPHIC	GRAPHIC, VARGRAPHIC, OR LONG VARGRAPHIC
VARCHAR	CHAR, VARCHAR, OR LONG VARCHAR
VARGRAPHIC	GRAPHIC, VARGRAPHIC, OR LONG VARGRAPHIC
DATE	CHAR, VARCHAR
TIME	CHAR, VARCHAR
TIMESTAMP	CHAR, VARCHAR

Figure 39. Possible Reasons for -301 SQLCODE

INPUT HOST VARIABLE TOO LARGE (-302)

The database manager is attempting to convert an input host variable to the data type of a column or expression in an SQL statement. The length of the input host variable is too long if the target is graphic or character. The numeric value of the input host variable is too large if the target column or expression is numeric. Possible reasons for getting the -302 SQLCODE are:

DATA TYPE OF TARGET COLUMN OR EXPRESSION	HOST VARIABLE:
DECIMAL	VALUE TOO LARGE
INTEGER	VALUE TOO LARGE
SMALLINT	VALUE TOO LARGE
FIXED CHAR OR GRAPHIC	LENGTH TOO LONG
VARYING CHAR OR GRAPHIC	LENGTH TOO LONG

Figure 40. Possible Reasons for -302 SQLCODE

AN INDICATOR VARIABLE IS MISSING (-305)

If a NULL value is fetched, an indicator variable must be supplied with the output host variable. See "Using Indicator Variables" in the *Application Programming* manual.

This indicator variable should be checked before manipulating data retrieved by the select.

MISMATCH BETWEEN NUMBER OF HOST VARIABLES (-313)

The number of input host variables supplied in the SQLDA or host variable list on an OPEN, PUT, or EXECUTE SQL statement does not match the number of host variables specified in the original Data Manipulation Language (DML) SQL statement. The original statement may be a SELECT statement specified in a DECLARE CURSOR or PREPARE. Or it may be an INSERT, UPDATE, or DELETE specified in a PREPARE or EXECUTE IMMEDIATE.

This condition most likely occurs when an application prepares (via PREPARE) an SQL statement that contains "?" parameters. The application must supply the host variables to be substituted for the "?" on a subsequent SQL statement.

The number of host variables in the SQLDA descriptor or host variable list supplied in the EXECUTE, OPEN, or PUT statements must be the same as the number of question marks ("?") in the original statement.

Functional Deviations

Following are situations which do not result in SQL/DS error codes, but which may represent a problem.

Lockout with Cursor Stability

When SQL/DS uses a DBSPACE scan to access a table in a DBSPACE with row level locking using isolation level cursor stability, the effect is the same as repeatable read. That is, no other LUW can update the table until the LUW performing the DBSPACE scan ends. Also, if one LUW has updated a table, another LUW cannot access that table with a DBSPACE scan until the updating LUW ends.

Possible actions that can be taken to avoid the lockout situations are:

- Ensure that the table is accessed via an index.
- Alter the DBSPACE specifying page level locking.
- Place the table in a DBSPACE with page level locking.

FETCH with Cursor Stability

When the isolation level is CS, the use of a DBSS FETCH is prevented at run time for statements that can be updated. Instead, a DBSS OPEN SCAN is performed. In this case, isolation level RR may prove to be more efficient to allow a DBSS FETCH operation and avoid the overhead of opening an internal scan. See "Types of Internal Data Manipulation Calls" on page 60, for a description of OPEN SCAN and FETCH.



Chapter 5. Diagnosing Performance Problems

This chapter covers performance problem diagnosis. It takes problem symptoms and leads the reader through problem isolation and problem determination activities to identify the ultimate cause of the problem. If your problem is occurring during distributed processing, you should also refer to the *Distributed Relational Database Problem Determination Guide*.

Sometimes it may not be obvious which kind of performance problem you have. The first step in performance problem diagnosis is isolating the problem to a particular type of problem. Once you have isolated the problem to a specific type, you then do the analysis for that type of problem.

This chapter is organized into the following sections:

1. Performance Analysis Glossaries

This section provides three glossaries of terms used in this chapter. The glossaries may be used for reference to assist you in interpretation of the performance problem indexes and problem analysis sections.

2. Performance Problem Indexes

The *performance problem indexes* provide a list of possible problems based on what functions are being performed and what symptoms are being observed. There are two sets of indexes: the first set is organized by application function, and the second is organized by performance symptom.

The performance problem indexes serve as an index into the specific problem descriptions and their corrective actions. By using the *application function indexes* starting on page 106, you can find the problems that might be causing specific application functions to perform poorly. By using the *performance symptom indexes* starting on page 111, you can find the problems that might be causing a specific performance symptom to occur.

3. Analysis of Performance Problems

For each performance problem, a problem description is provided and a list of possible corrective actions is identified. The problem description will identify the cause of the problem and secondary symptoms that should help confirm that you do or do not have that problem.

The list of possible corrective actions includes possible circumventions to the problem, as well as actions that might completely eliminate the problem.

Performance Analysis Glossaries

Three glossaries are provided in this section to assist in your interpretation of the performance problem indexes and problem descriptions. The three glossaries are:

1. Performance Index Headers

This glossary explains how to interpret the columns in the performance problem indexes.

2. Performance Indicator Terms

This glossary explains how to interpret some of the column values that appear in the performance problem indexes.

3. Performance Terminology

This glossary provides a quick reference to terminology used in the performance problem descriptions.

Glossary of Performance Index Headers

Figure 41 lists the definitions of the headers used in the performance problem indexes. It is important that you understand what each of the columns of the indexes represent.

INDEX HEADER	DESCRIPTION
APPLICATION FUNCTION	This refers to the SQL function being performed when the performance problem occurs. It is <i>not necessarily</i> the function which caused the problem.
PAGE	A <i>page reference</i> to the description of the problem listed in the index line.
PERFORMANCE INDICATOR	A <i>performance indicator</i> is a performance symptom that indicates the possible existence of a problem.
POSSIBLE PROBLEM	A <i>possible problem</i> is a performance problem that might exist for the conditions shown in the index.

Figure 41. Glossary of Index Header Terms

Note: The **APPLICATION FUNCTION** column in the indexes refers to functions that can *show* poor performance as a result of the listed problem.

Glossary of Performance Indicator Terms

Figure 42 on page 103 lists the definitions of *performance indicator* terminology used in the performance problem indexes. The indicator terms are abbreviated references to problem symptoms and may not be self-explanatory. Figure 42 provides a brief definition of symptoms implied by the indicator terms.

PERFORMANCE INDICATOR	DESCRIPTION
COMMUNICATION WAITS INDICATOR	There is a <i>high frequency</i> of communication waits, or communication waits are of <i>long duration</i> . (Note: A communication wait is the time an SQL/DS agent spends in-LUW waiting for the application requests)
CONSISTENTLY HIGH RESPONSE TIME INDICATOR	The response time is consistently higher than expected.
ESCALATES INDICATOR	There is a <i>high frequency</i> of lock escalations, as measured by COUNTER ESCALATE and COUNTER LOCKLMT. (NOTE: ESCALATE counts <i>successful</i> escalations, and LOCKLMT counts <i>unsuccessful</i> escalations.)
HIGH CPU USAGE INDICATOR	The amount of CPU time for an application is <i>high</i> .
HIGH I/O INDICATOR	The number of I/O's to database DASD devices is <i>high</i> given the application or work being done on the system. (Note: HIGH I/O usually also implies HIGH CPU USAGE).
HIGH I/O UTILIZATION INDICATOR	A <i>high</i> device or channel utilization on any one device or channel.
INDICATOR	This refers to <i>performance indicators</i> used in the indexes into the performance problem descriptions.
LINK WAITS INDICATOR	There is a <i>high frequency</i> of link wait conditions, or link waits are of <i>long duration</i> .
LOCK WAITS INDICATOR	There is a <i>high frequency</i> of lock wait conditions, or lock waits are of <i>long duration</i> .
LOG I/O'S INDICATOR	There is a <i>high number</i> of log I/O's occurring for the application or system.
LOW CPU UTILIZATION INDICATOR	The percentage of time the CPU is busy is <i>low</i> . A CPU Utilization of 30 or 40% would be considered low.
LOW I/O UTILIZATION INDICATOR	A <i>low</i> device or channel utilization on any one device or channel.
PAGING INDICATOR	Page faults are occurring on the system at a noticeable rate. (Could be considered a <i>high</i> paging rate).
PERIODIC HIGH RESPONSE TIME INDICATOR	Unexplained phenomenon that result in users getting widely varying, unpredictable response times for repeated executions of the same query or application.

Figure 42. Glossary of Index Indicator Terms

Note: The HIGH I/O indicator also implies some level of unnecessary CPU overhead as well. Thus, HIGH I/O problems can also be investigated in situations where your CPU usage is high.

Glossary of Performance Terminology

Figure 43 on page 104 defines some of the terms used in the performance problem indexes and performance problem descriptions. You may want to familiarize yourself with some of the terms before using this chapter, or you may simply use this figure as a reference.

PERFORMANCE TERM	DESCRIPTION
BUFFER HIT RATIO	This is the ratio of "looks in the page buffers" to actual page reads (LPAGBUFF divided by PAGEREAD).
BUFFER POOL THRASHING	This is term for the condition when data for an application must be read from DASD every time the application references it.
COMMUNICATION WAIT	A communication wait occurs when an active (IN-LUW) agent returns control to the application machine and is waiting for the next request from the application.
DATA AUTHORIZATION COMMANDS	This refers to SQL commands that control data authorization (that is, GRANT and REVOKE).
DATA DEFINITION LANGUAGE (DDL)	This refers to SQL commands for maintaining data definitions (CREATE, ALTER, ACQUIRE and DROP).
DATA MANIPULATION LANGUAGE (DML)	This refers to SQL commands for reading and maintaining user tables (DELETE, INSERT, SELECT and UPDATE).
ESCALATE	A lock <i>escalation</i> occurs when the database manager trades several low level locks (page or row and key locks) for one DBSPACE lock. Lock escalations are not always successful.
INDEX ELIGIBLE PREDICATE	This is a predicate that can be used as an argument of an index scan in the DBSS component.
KEY LOCKING	This refers to locking of key hash values on non-unique key accesses. Key locking is done in DBSPACES where row level locking is in effect.
LINK WAIT	A <i>link wait</i> is a wait condition that occurs when an application needs an agent, but all the agents are in use by other applications. The application <i>waits</i> until an agent frees up.
LINK WAIT RATIO	Refers to the ratio of number of users in a link wait state to the NCUSERS value.
LOADING	This is used in the generic sense and refers to both the DBS Utility DATALOAD and RELOAD functions.
LOCK LEVEL	This refers to the amount of data locked when SQL/DS requests SHARE (S) or EXCLUSIVE (X) locks. SQL/DS supports DBSPACE, page and row level locking.
LOCK WAIT	A <i>lock wait</i> is a wait condition that occurs when an application requests a "piece" of data being used by one or more other applications for a conflicting purpose (For example, Read access conflicts with update access).
LOCK WAIT RATE	This is the rate at which lock waits are occurring in terms of the number of lock waits per LUW. (LOCKWAIT divided by BEGINLUW).
PREPROCESS	This refers to the execution of the SQL/DS preprocessors (PL/1, COBOL, C, RPG, FORTRAN or Assembler).
QCE	This refers to the Query Cost Estimate displayed by ISQL for SELECT commands.
RECOVERY CONTROL COMMANDS	This refers to SQL commands that control data recovery (that is, COMMIT WORK and ROLLBACK WORK).
SELECTIVE INDEX	This is an index that matches an index eligible predicate of a particular query. Note: This does not refer to the data characteristics of the index. It is meaningful only in the context of a specific query.
UNLOADING	This is used in the generic sense and refers to both the DBS Utility DATAUNLOAD and UNLOAD functions.
VERY NONUNIQUE INDEX	This is an index which has very few different key values (compared to the number of rows in the indexed table).
VERY NONUNIQUE KEY PREFIX	Refers to an index for which the first 6-bytes of the key values are very nonunique.

Figure 43. Glossary of Performance Analysis Terms

Performance Problem Indexes

The performance problems in this chapter are indexed by *application function* and *performance symptom*. If the performance problem can be observed when using specific applications or application functions, then you should look directly to "Application Function Indexes to Performance Problems" on page 106.

If the problem is not observed with any particular application or application function, but you *can* observe some *symptoms* of poor performance (such as lock waits, high CPU usage, etc.), then you should start with the "Performance Problems by Performance Symptom" on page 111. Access to the performance problem diagnosis information in this chapter is diagrammed below:

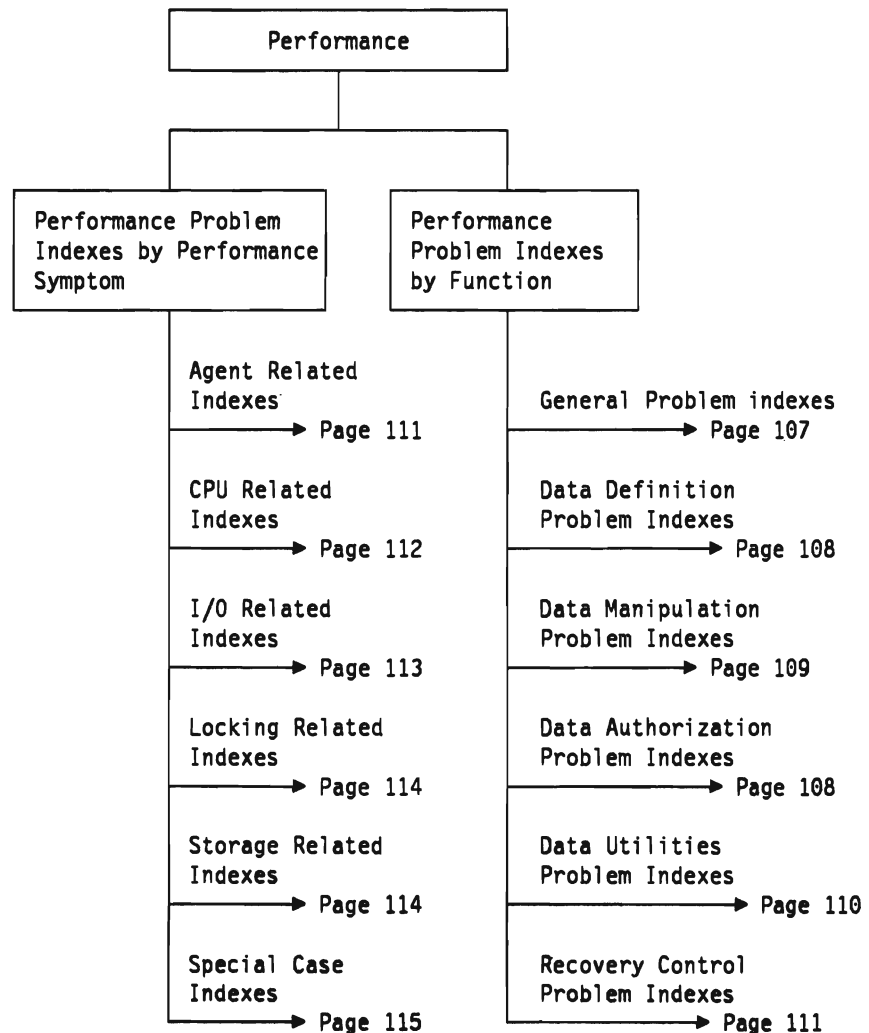


Figure 44. Diagnosis and Recovery Flowchart - Performance

Application Function Indexes to Performance Problems

APPLICATION FUNCTION INDEXED	FUNCTION	PAGE
General (for any SQL)	ANY SQL	107
Data Authorization	GRANT REVOKE	108
Data Definition	ACQUIRE ALTER CREATE DROP	108
Data Manipulation	DELETE INSERT SELECT UPDATE	109
Data Utilities	LOADING PREPROCESSOR UNLOADING UPDATE STATISTICS	110
Recovery Control	COMMIT WORK ROLLBACK WORK	111

Figure 45. Indexes For Performance Problems by Application Function

The following sections provide six problem indexes by application function. The functions covered by each of the indexes are identified in Figure 45.

If all users are experiencing performance problems, you might want to start with the "General Performance Problems" on page 107. If only the users issuing data definition commands are experiencing performance problems, you should start with "Data Definition Performance Problems" on page 108. However, for completeness, you may want to review the problems in "General Performance Problems" on page 107 as well. General problems are not all repeated in the other indexes.

If you are experiencing performance problems with a specific *function* (such as DATALOADs), then you should start with the index that corresponds to that function. In the case of DATALOAD, you would start with the "Data Utilities Performance Problems" on page 110.

Note: The functions used to index the problems are the *functions being performed when the problem was observed*. They are *not necessarily* the functions that are *causing* the performance problem. You do not have to know the function that is causing your problems to use these indexes. You need to know only the function or functions that are impacted by the problem.

General Performance Problems

Figure 46. General Performance Problems.

APPLICATION FUNCTION	PERFORMANCE INDICATOR	POSSIBLE PROBLEM	PAGE
ANY SQL	ESCALATES	NLRB Parameters Too Small	191
	HIGH CPU USAGE	Need More CPU	188
		Invalid Entities Exist	175
		CHARNAME Not Set Correctly	130
		BLOCK I/O, APPC/VM and IUCV Not Resident	125
		SQL/DS Machine Favored Too Little	139
		ECMODE ON for Accounting	147
		CMS Work Unit Set On	134
		DRDA protocol to Access an SQL/DS Database	147
	HIGH I/O and/or HIGH CPU USAGE	Buffer Pool Too Small	128
		Sequential Processing	200
		Indexes are Fragmented	169
		CHKINTVL Too Small	132
Data not Cached		139	
Frequent Checkpoints caused by SOSLEVEL	152		
HIGH I/O UTILIZATION	I/O Capacity Exceeded	160	
	I/O Not Balanced	161	
LINK WAITS	Agents Being Held	119	
	Too Few Agents	205	
	Session Limit Exceeded	203	
LOCK WAITS	Locks Held for Long Duration	180	
	NLRB Parameters Too Small	191	
	Conflict in Catalog Key Locking	134	
	Hot Spot in the Catalog Tables	153	
LOW CPU UTILIZATION	One DB Machine Needs Too Much CPU (Multiprocessor systems only)	193	
PAGING	Page Fault Serialization	197	
	SET QDROP OFF USERS/SET QUICKDSP ON	203	
	SQL/DS Code Not Shared	204	
	Buffer Pool Too Big	126	
	Too Many Agents	206	
	NLRB Parameters Too Large	190	
	Need More Real Storage	189	
PERIODIC HIGH RESPONSE TIME	CHKINTVL Too Big	131	
	Checkpoint Being Forced at End-LUW	130	
	Long DBSS Calls Delaying Checkpoint	184	
	Buffer Pool Too Big	126	
	Sequential Processing	200	
	Logging during Load	183	
	Storage Pool Full	204	
	DRDA protocol to Access an SQL/DS Database	147	
	DRDA usage	147	

Note: The possible problems listed in Figure 46 are problems that can be experienced when using *any* SQL functions. That is, most of the problems listed here will impact any SQL work you would be doing. However, it should be noted that some of the problems will impact some types of SQL work more than others. For specific functions that are particularly sensitive to a general problem, the general problem is repeated in the index for the specific function.

For example, although "Conflict in Catalog Key Locking" can impact any SQL work; SQL DDL, SQL authorization commands and preprocessing will be particularly sensitive to this problem. Thus, "Conflict in Catalog Key Locking" is repeated in the indexes for those functions. On the other hand, SQL DML com-

mands are less sensitive to the problem, and therefore, the problem is not repeated in the index for SQL DML problems.

Data Authorization Performance Problems

Figure 47. Data Authorization Performance Problems.

APPLICATION FUNCTION	PERFORMANCE INDICATOR	POSSIBLE PROBLEM	PAGE
SQL AUTH - GRANT - REVOKE	LOCK WAITS	Conflict in Catalog Key Locking Hot Spot in Catalog Tables	134 153
Also See: ANY SQL	*****	Index on: General Performance Problems	107

Data Definition Performance Problems

Figure 48. Data Definition Performance Problems.

APPLICATION FUNCTION	PERFORMANCE INDICATOR	POSSIBLE PROBLEM	PAGE
SQL DDL - ACQUIRE - ALTER - CREATE - DROP	LOCK WAITS	Conflict in Catalog Key Locking Hot Spot in Catalog Tables	134 153
CREATE INDEX	HIGH I/O and/or HIGH CPU USAGE	Create Index Requires a Large Sort Large Tables Share Same DBSPACE DBSPACE Scan Being Performed Frequent Checkpoints caused by SOSLEVEL	138 175 141 152
DROP TABLE	HIGH I/O and/or HIGH CPU USAGE	Large Tables Share Same DBSPACE DBSPACE Scan being performed	175 141
Also See: ANY SQL	*****	Index on: General Performance Problems	107

Data Manipulation Performance Problems

Figure 49. Data Manipulation Performance Problems.

APPLICATION FUNCTION	PERFORMANCE INDICATOR	POSSIBLE PROBLEM	PAGE
SQL DML - DELETE - INSERT - SELECT - UPDATE	ESCALATES	Lock Level Too Low	179
	HIGH I/O and/or HIGH CPU USAGE	Inaccurate Statistics	162
		Insufficient Indexing	174
		Need a Highly Clustered Index	187
		Index No Longer Highly Clustered	168
		No Selective Index	192
		Index Disqualified	164
		Indexes are Fragmented	169
		Very Nonunique Index Key Prefix	211
		Bad Data Distribution	122
		Inefficient Search	170
		Missing Search Condition	185
		Large Tables Share Same DBSPACE	175
		DBSPACE Scan Being Performed	141
		Package Needs Re-Preprocessing	194
Range Predicate Used With Host Vars	199		
Blocking Suppression for INSERT CURSORS	125		
LOCK WAITS	Lock Level Too High	178	
	Lock Level Too Low	179	
	Excessive Locking in User Data	149	
	Hot Spot in User Tables	158	
	Conflict on Key Hash in User Data	136	
	Adjacent Key Locking in User Data	115	
SQL DML - DELETE - INSERT - UPDATE	HIGH I/O and/or HIGH CPU USAGE	Index Maintenance	167
SQL DML - SELECT - INSERT FMT 2	HIGH CPU	Inefficient SELECT List	174
	HIGH I/O and/or HIGH CPU USAGE	Too Many Joins	208
SQL DML - SELECT	CONSISTENTLY HIGH RESPONSE TIME	Query Block Size Too Small	198
Also See: ANY SQL	*****	Index on: General Performance Problems	107

Data Utilities Performance Problems

Figure 50. Data Utilities Performance Problems.

APPLICATION FUNCTION	PERFORMANCE INDICATOR	POSSIBLE PROBLEM	PAGE
LOADING	ESCALATES	Lock Level Too Low	179
	HIGH I/O and/or HIGH CPU USAGE	Index Maintenance	167
		UPDATE STATISTICS by DATALOAD	210
		Frequent Checkpoints caused by SOSLEVEL	152
		Blocking Suppression for INSERT CURSORS	125
LOCK WAITS	Excessive Locking in User Data	149	
	Lock Level Too Low	179	
Log I/O's	Lock Level Too High	178	
	Conflict on Key Hash in User Data	136	
	Adjacent Key Locking in User Data	115	
	Log I/O's	Logging during Load	183
UNLOADING	ESCALATES	Lock Level Too Low	179
	HIGH CPU USAGE	Inefficient SELECT List	174
	HIGH I/O and/or HIGH CPU USAGE	Large Tables Share Same DBSPACE	175
		DBSPACE Scan Being Performed	141
		Insufficient Indexing	174
		Need a Highly Clustered Index	187
Index No Longer Highly Clustered		168	
No Selective Index		192	
Index Disqualified		164	
Very Nonunique Index Key Prefix		211	
Inefficient Search		170	
Missing Search Condition		185	
Too Many Joins	208		
Inaccurate Statistics	162		
Bad Data Distribution	122		
LOCK WAITS	Excessive Locking in User Data	149	
	Lock Level Too Low	179	
	Lock Level Too High	178	
	Conflict on Key Hash in User Data	136	
	Adjacent Key Locking in User Data	115	
CONSISTENTLY HIGH RESPONSE TIME	Query Block Size Too Small	198	
PREPROCESSOR	ESCALATES	NLRB Parameters Too Small	191
	LOCK WAITS	Conflict in Catalog Key Locking	134
NLRB Parameters Too Small		191	
UPDATE STATISTICS	HIGH I/O and/or HIGH CPU USAGE	Large Tables Share Same DBSPACE	175
		DBSPACE Scan Being Performed	141
Also See: ANY SQL	*****	Index on: General Performance Problems	107

Recovery Control Performance Problems

Figure 51. Recovery Control Performance Problems.

APPLICATION FUNCTION	PERFORMANCE INDICATOR	POSSIBLE PROBLEM	PAGE
RECOVERY CONTROL - COMMIT - ROLLBACK	PERIODIC HIGH RESPONSE TIME	Checkpoint Being Forced at End-LUW Storage Pool Full	130 204
Also See: ANY SQL	*****	Index on: General Performance Problems	107

Performance Problems by Performance Symptom

Figure 52. Indexes For Performance Problems by Symptom

PERFORMANCE AREA	SYMPTOM INDEXED	PAGE
Agent Related Problems	Communication Waits Link Waits	111
CPU Related Problems	High CPU Usage Low CPU Utilization	112
I/O Related Problems	High I/O's High I/O Utilization Log I/O's	113
Locking Related Problems	Escalates Lock Waits	114
Storage Related Problems	Paging	114
Special Case Problems	Periodic High Response Times Communication Delays	115

The following sections provide six problem indexes by performance symptom. The symptoms covered by each of the indexes are identified in Figure 52.

If you are experiencing performance problems with a specific symptom, then you should start with the corresponding index. For example, if the primary symptom of the problem you want to solve is lock waits, then you should start with the index for "Locking Related Performance Problems" on page 114.

Agent Related Performance Problems

Figure 53. Agent Related Performance Problems.

PERFORMANCE INDICATOR	APPLICATION FUNCTION	POSSIBLE PROBLEM	PAGE
COMM WAITS	ANY SQL	Agents Being Held Locks Held for Long Duration	119 180
LINK WAITS	ANY SQL	Too Few Agents Agents Being Held Locks Held for Long Duration	205 119 180

CPU Related Performance Problems

Figure 54. CPU Related Performance Problems

PERFORMANCE INDICATOR	APPLICATION FUNCTION	POSSIBLE PROBLEM	PAGE		
HIGH CPU USAGE	ANY SQL	Indexes are Fragmented	169		
		Buffer Pool Too Small	128		
		BLOCK I/O, APPC/VM and IUCV Not Resident	125		
		Database Machine Favored Too Little	139		
		SET QDROP OFF USERS/SET QUICKDSP ON	203		
		CHKINTVL Too Small	132		
		Sequential Processing	200		
		Logging during Load	183		
		Need More CPU	188		
		Synchronous APPC/VM Not Used	205		
CHARNAME Not Set Correctly	130				
Package Cache Too Big/Threshold Too High	195				
Package Cache Too Small/Threshold Too Low	196				
SQL DDL - CREATE INDEX		Create Index Requires a Large Sort	138		
		Frequent Checkpoints caused by SOSLEVEL	152		
SQL DDL - CREATE INDEX - DROP TABLE SQL DML - DELETE - INSERT FMT 2 - SELECT - UPDATE UNLOADING UPDATE STATISTICS		Large Tables Share Same DBSPACE	175		
		DBSPACE Scan Being Performed	141		
		SQL DML - DELETE - INSERT FMT 2 - SELECT - UPDATE		Package Needs Re-Preprocessing	194
				Range Predicate Used With Host Vars	199
		SQL DML - DELETE - INSERT FMT 2 - SELECT - UPDATE UNLOADING		Insufficient Indexing	174
				Need a Highly Clustered Index	187
				Index No Longer Highly Clustered	168
				No Selective Index	192
				Index Disqualified	164
				Very Nonunique Index Key Prefix	211
Inaccurate Statistics	162				
Bad Data Distribution	122				
SQL DML - SELECT - INSERT FMT 2 UNLOADING		Inefficient SELECT List	174		
		Too Many Joins	208		
LOADING		UPDATE STATISTICS by DATALOAD	210		
		Frequent Checkpoints caused by SOSLEVEL Blocking Suppression for INSERT CURSORS	152 125		
LOADING SQL DML - DELETE - INSERT - UPDATE		Index Maintenance	167		
LOW CPU UTILIZATION		One DB Machine Needs Too Much CPU (Multiprocessor systems only)	193		

I/O Related Performance Problems

Figure 55. I/O Related Performance Problems.

PERFORMANCE INDICATOR	APPLICATION FUNCTION	POSSIBLE PROBLEM	PAGE
HIGH I/O	ANY SQL	Indexes are Fragmented	169
		Buffer Pool Too Small	128
		CHKINTVL Too Small	132
		Sequential Processing	200
		Too Many Agents	206
		Data not cached	139
		Package Cache Too Big/Threshold Too High	195
		Package Cache Too Small/Threshold Too Low	196
	SQL DDL - CREATE INDEX	Create Index Requires a Large Sort Frequent Checkpoints caused by SOSLEVEL	138 152
	SQL DDL - CREATE INDEX - DROP TABLE SQL DML - DELETE - INSERT FMT 2 - SELECT - UPDATE UNLOADING UPDATE STATISTICS	Large Tables Share Same DBSPACE DBSPACE Scan Being Performed	175 141
SQL DML - INSERT	Excessive I/Os on INSERT	147	
SQL DML - DELETE - INSERT FMT 2 - SELECT - UPDATE	Package Needs Re-Preprocessing Range Predicate Used With Host Vars	194 199	
SQL DML - DELETE - INSERT FMT 2 - SELECT - UPDATE UNLOADING	Insufficient Indexing Need a Highly Clustered Index Index No Longer Highly Clustered No Selective Index Index Disqualified Very Nonunique Index Key Prefix Inaccurate Statistics Bad Data Distribution Inefficient Search Missing Search Condition	174 187 168 192 164 211 162 122 170 185	
SQL DML - SELECT - INSERT FMT 2 UNLOADING	Too Many Joins	208	
LOADING	UPDATE STATISTICS by DATALOAD Frequent Checkpoints caused by SOSLEVEL Blocking Suppression for INSERT CURSORS	210 152 125	
LOADING SQL DML - DELETE - INSERT - UPDATE	Index Maintenance	167	
HIGH I/O UTILIZATION	ANY SQL	I/O Capacity Exceeded I/O Not Balanced	160 161
LOG I/O	LOADING	Logging during Load	183

Locking Related Performance Problems

Figure 56. Locking Related Performance Problems.

PERFORMANCE INDICATOR	APPLICATION FUNCTION	POSSIBLE PROBLEM	PAGE
ESCALATES	ANY SQL	NLRB Parameters Too Small	191
	LOADING SQL DML UNLOADING	Lock Level Too Low Excessive Locking in User Data	179 149
LOCK WAITS	ANY SQL	NLRB Parameters Too Small Locks Held for Long Duration Agents Being Held Conflict in Catalog Key Locking Hot Spot in Catalog Tables Too Many Agents	191 180 119 134 153 206
	SQL DDL - CREATE INDEX - DROP TABLE SQL DML UNLOADING UPDATE STATISTICS	DBSPACE Scan Being Performed Large Tables Share Same DBSPACE	141 175
	LOADING	UPDATE STATISTICS by DATALOAD	210
	LOADING SQL DML UNLOADING	Adjacent Key Locking in User Data Conflict on Key Hash in User Data Excessive Locking in User Data Lock Level Too High Lock Level Too Low	115 136 149 178 179
	SQL DML	Hot Spot in User Tables	158
	SQL DML - SELECT - INSERT FMT 2 UNLOADING	Too Many Joins	208
	DEADLOCKS	ANY SQL	Excessive Deadlocks

Storage Related Performance Problems

Figure 57. Storage Related Performance Problems.

PERFORMANCE INDICATOR	APPLICATION FUNCTION	POSSIBLE PROBLEM	PAGE
PAGING	ANY SQL	SET QDROP OFF USERS or QUICKDSP ON	203
		SQL/DS Code Not Shared	204
		Too Many Agents	206
		Buffer Pool Too Big	126
		NLRB Parameters Too Large	190
		Page Fault Serialization	197
		Need More Real Storage	189
		Package Cache Too Big/Threshold Too High	195
		Package Cache Too Small/Threshold Too Low	196

Special Case Performance Problems

Figure 58. Special Case Performance Problems.

PERFORMANCE INDICATOR	APPLICATION FUNCTION	POSSIBLE PROBLEM	PAGE
PERIODIC HIGH RESPONSE TIME COMMUNICATION DELAYS	ANY SQL	CHKINTVL Too Big	131
		Buffer Pool Too Big	126
		Checkpoint Being Forced at End-LUW	130
		Long DBSS Calls Delaying Checkpoint	184
		CHKINTVL Too Small	132
		Sequential Processing	200
		Logging during Load	183
		Storage Pool Full	204
		Frequent Checkpoints caused by SOSLEVEL DRDA usage	152 147
CONSISTENTLY HIGH RESPONSE TIME	SQL DML - SELECT UNLOADING	Query Block Size Too Small	198

Analysis of Performance Problems

Adjacent Key Locking in User Data

Problem Description: Adjacent key locking in user data refers to locking done "around" a key that is locked for access. This problem addresses locking conflicts on access to *user tables* that occur as a result of adjacent key locking. That is, the locking conflict is in the *indexes*, rather than the data rows (data pages), *and* the conflicting users are using "neighboring" keys, not the same keys (or same key hash values).

Adjacent key locking is discussed in "Locking Concepts" on page 55. Basically, when the database manager updates a key, an EXCLUSIVE lock is obtained on the next higher key, as well as the key being updated. Similarly, when an index scan is performed (read access) and if all matching rows are retrieved (that is until SQLCODE 100 returned), the database manager will read one key beyond the last key requested (obtaining a SHARE lock on both the key requested and the next higher key). In the case of repeatable read this lock is held until the end of the LUW.

One of the results of adjacent key locking is that you can experience locking conflicts on indexes even when the keys addressed are different. If they are consecutive (in key sequence), a lock conflict can occur.

The primary symptom of the "Adjacent Key Locking in User Data" problem is *lock waits*, where the lock waits are occurring on *index pages* or *key hash* values. Note that the "Adjacent Key Locking in User Data" problem can occur under either row (key) or page level locking. The problem is not just a row level locking phenomenon. An adjacent *index page* is locked under page level locking when an adjacent key is in the next index page.

Another characteristic of the "Adjacent Key Locking in User Data" problem is a relatively small number of lock requests. That is, if your applications are obtaining many locks in the index, your problem probably goes beyond the conflicts that might be occurring on adjacent keys. You probably should be investi-

gating one of the other lock wait problems, such as "Excessive Locking in User Data" on page 149.

Note: This section specifically addresses adjacent key locking problems in *user* DBSPACES. For a discussion of adjacent key locking problems in the SQL/DS catalog DBSPACE, see "Conflict in Catalog Key Locking" on page 134. Adjacent key locking problems in the SQL/DS catalog DBSPACE will look just like the "Adjacent Key Locking in User Data" problem, except that the DBSPACE involved will be DBSPACE 1.

A problem with symptoms that are identical to the "Adjacent Key Locking in User Data" problem is the "Conflict on Key Hash in User Data" problem. In observing SHOW LOCK results showing lock conflict on indexes, you will not be able to tell whether the conflict is on users going after "adjacent keys" or the same keys (or same key hash values). Knowledge of the conflicting applications will be necessary to distinguish these two problems. For a discussion of conflicts on the *same* keys (or same key hash values), see "Conflict on Key Hash in User Data" on page 136.

A problem with *similar* symptoms is the "Hot Spot in User Tables" problem. For a discussion of this problem, see "Hot Spot in User Tables" on page 158.

Possible Actions:

ACTION	PAGE
Decrease Lock Level	Below
Use Cursor Stability	Below
Change Key Structures	Below
Drop Unnecessary Indexes	Below
Use Redundant Data	Below
Use Multiple LUWs	Below
Re-Design Application	Below
For problems with similar symptoms, see:	
- "Conflict on Key Hash in User Data"	136
- "Hot Spot in User Tables"	158
If lock conflicts in <i>catalogs</i> , see:	
- "Conflict in Catalog Key Locking"	134
- "Hot Spot in the Catalog Tables"	153

Figure 59. Adjacent Key Locking in User Data - Actions

There are four basic approaches to resolving a lock contention problem due to adjacent key locking in user data:

1. Reduce locking done by applications
 - a. Decrease Lock Level

If the contention is in a DBSPACE with page level locking, changing the DBSPACE to row level locking may help. Adjacent key locking will still occur, but only the adjacent key will be locked (not the entire page of the adjacent key).

b. Use Cursor Stability Isolation Level

Another easy action that reduces the potential lock conflict is to use the Cursor Stability Isolation Level wherever possible. This will reduce the locking done by applications, and possibly reduce or eliminate the incidents of adjacent key locking conflicts. Adjacent key locking will still be done under Cursor Stability, but the key locks with which they are conflicting may be released sooner (maybe even before the conflict occurs).

Note: This action applies only when adjacent key locking done by updaters is conflicting with *read* access through the index. With cursor stability, the readers may release their SHARE locks before the "updaters arrive."

c. Use Multiple LUWs

The use of multiple LUWs to reduce adjacent key locking conflicts uses the same principle as the use of the cursor stability isolation level. By issuing COMMIT WORK (or ROLLBACK WORK) commands more frequently, locks that are conflicting are held for shorter periods of time. As a result, the conflicts might not even occur, but if they do, the lock waits will be shorter.

2. Change the index such that keys are not adjacent

Another way of reducing locking done by applications is to look for alternate access paths for some of the applications, such that the index accesses do not conflict. Specifically, two alternatives should be considered:

a. Change Key Structures

Since the conflict is on adjacent keys, one approach would be to change the index definition (re-create the index) such that the key values have a different sequence. In this way, you would be changing the definition of what key value is the "adjacent key" value.

For example, in an index on PROJNO and DEPTNO columns of the PROJECT table, the key value (AD3100,D01) would be adjacent to the key value (AD3110,D21). However, in an index on DEPTNO and PROJNO, the adjacent key value for (D01,AD3100) would be (D01,MA2100). Thus, by redefining the index you could avoid conflicts between keys (AD3100,D01) and (AD3110,D21).

b. Drop Unnecessary Indexes

Of course, the most effective way to avoid the adjacent key locking conflicts is to drop the index in which the conflict occurs. The indexes in which conflicts are occurring should be reviewed for their necessity. It may be better to use a less efficient index and avoid lock contention, than to use the current index and experience lock wait problems.

3. Change the table designs such that conflicting actions don't occur

The third approach to adjacent key locking problems is to review the data required by your applications, and consider alternative data designs. Specifically, you might consider alternative designs that would mean your conflicting applications (users) would not be accessing the same tables with conflicting requests.

One variation of this approach is to use redundant data to avoid the lock contention problems. Three types of redundant data designs you can try are:

a. Table Splitting

This refers to creating multiple tables from one table that is the source of the lock contention. By splitting the tables based on type of reference (read or write), and the mapping of columns used by the conflicting applications, you may be able to achieve a design such that the rows locked by the conflicting applications are in different tables.

b. Stored Results

This refers to storing the same data in multiple tables. This could be complete duplication of a table, duplication of some of the rows, or duplication of column information.

For example, storing intermediate query results in a separate table, rather than running multiple queries against a "production" table can reduce lock contention in the production table. Another example would be to maintain "popular" column information in multiple tables, rather than using joins to get the information from a single source table.

Note, for example, that CREATOR and TNAME appear in SYSTEM.SYSCOLUMNS as well as SYSTEM.SYSCATALOG. From a storage consumption point of view, it would have been "cheaper" to use the internal table ID information for supporting joins of these two catalogs. However, CREATOR and TNAME are almost always desired in queries to SYSTEM.SYSCOLUMNS. Thus, those columns are "duplicated," thus avoiding SYSCATALOG contention on accesses to SYSCOLUMNS.

c. Transaction Tables

This refers to doing data entry and editing in a "mirror" copy of the target (production) table, and later batch replacing rows in the production table. The use of transaction tables avoids lock contention by consolidating update access to the production table in one, less frequently executed, batch application. This can be very effective at alleviating lock contention due to continuous data entry and edit activity.

4. Change applications such that actions don't conflict

The last approach to resolving adjacent key locking problems is to change the applications such that the accesses they make to the data do not conflict.

If you are not sure that the lock contentions are due to *adjacent keys*, then you should also review "Conflict on Key Hash in User Data" on page 136 and "Hot Spot in User Tables" on page 158. Not all of the solutions to the "Adjacent Key Locking in User Data" problem will work if you have one of these problems.

If the lock contention problems are in the catalog DBSPACE (DBSPACE 1), then you should be reviewing "Conflict in Catalog Key Locking" on page 134 and "Hot Spot in the Catalog Tables" on page 153. Solutions to these problems are quite different from lock contention problems in user DBSPACES.

Agents Being Held

Problem Description:

This problem refers to agents being held for long periods of time and the ill effects this can cause. If agents are held for long periods of time, then users of the system may experience long *link waits* or long *lock waits*.

If response time is long or erratic in multiple user mode, but OK when only one user is on the system, then the problem could be agents held by other users. This can occur when the system is extremely busy, or as a result of users (or applications) holding their agents while they are not actively using them.

The most likely symptom of the "Agents Being Held" problem is *link waits*. A link wait occurs when a user (or application) is waiting for an agent in the database machine. Link waits could be due to applications (or users) that are not freeing agents when they should.

If the users are experiencing link waits, but no other performance problems, then you may not have enough agents. This is particularly true if you are not experiencing lock wait or paging problems. If this is the case, refer to "Too Few Agents" on page 205.

Another possible symptom of the "Agents Being Held" problem is *lock waits*. A lock wait occurs when a user (or application) is waiting for data being used by another user. If the other user is active (holding its agent) for a long period of time, the lock waits on that user will be a problem to the waiting users. If the lock waits are more of a problem than the link waits, see "Locks Held for Long Duration" on page 180.

Another symptom to look for is *communications waits*. A communications wait occurs when the SQL/DS agent is waiting for communications from the user's machine. If agents are spending a lot of time in communications waits, then the agents could be being held unnecessarily. (Note: Communications waits can be detected using the SHOW ACTIVE or SHOW LOCK ACTIVE SQL/DS operator commands).

The user or application that causes or contributes to the "Agents Being Held" problem will not necessarily notice a performance problem. Users that are holding agents could be doing almost any SQL function. However, some common examples of situations that could cause the agents being held problem are:

1. ISQL Users running with AUTOCOMMIT set OFF

When an ISQL user runs with AUTOCOMMIT set OFF, the agent that services its requests is held between commands (until a COMMIT WORK, ROLLBACK WORK or CANCEL command is issued). If commands are issued from a routine, this would typically not be a problem. However, if the user is entering commands 'by hand', there could be substantial delays between commands. During those delays, the agent is just sitting there doing nothing, but no other user can use it.

2. Conversational Programs

A conversational program is a program that 'talks' to the user of the program during a logical unit of work. That is, it issues reads to the terminal without committing the work it was doing. In this case, the agent is held for

the application while it is waiting for the application user to respond to the terminal read. Again, this could be a long time.

3. Batch-Online Processing

Long running batch programs can also cause agents to be held for long periods of time. Sometimes this is necessary, but sometimes it can be avoided. Examples of long running batch programs that could cause a problem include:

- Batch programs without periodic COMMIT WORKs
- Large DBS Utility DATALOADs without the COMMITCOUNT option.

Note: The "Agents Being Held" problem can affect *any* SQL application or user. The contention is for SQL/DS agents. If an agent is not available, it does not matter what function the user wants to perform, the user must wait.

Possible Actions:

ACTION	PAGE
Increase NCUSERS	Below
Use Multiple LUWs	Below
Use AUTOCOMMIT ON	Below
Use Routines for AUTOCOMMIT OFF Processing	Below
Use Pseudo-Conversational Programming	Below
Use COMMITCOUNT On DATALOAD Processing	Below
If no lock or communication waits, see also: - "Too Few Agents"	205
If also experiencing lock waits, see: - "Locks Held for Long Duration"	180

Figure 60. Agents Being Held - Actions

There are two basic approaches to addressing the "Agents Being Held" problem:

1. Increase the number of available agents

The simplest approach to resolving an "Agents Being Held" problem is to increase the number of available agents (by increasing NCUSERS) and the corresponding connection "ports" to SQL/DS. That is, increasing NCUSERS and possibly increasing MAXCONN. Refer to the *System Administration* manual and the *Performance Tuning Handbook*.

Increasing NCUSERS may not be practical due to virtual storage limitations on your system or the number of users that will be holding agents. A large number of agents may result in a high paging rate which can be detrimental to all applications. If increasing the connections to the database machine is not practical, then you need to review your applications and their need to hold agents.

2. Reduce the length of time agents are required

If increasing NCUSERS is not practical, then you need to see what can be done to reduce the length of time agents are being held. By reducing the

amount of time agents are held, you may be able to increase the amount of time your existing agents will be available.

Even if the total *amount* of time an agent is needed is not reduced, reducing the *duration* of each agent holding period will tend to minimize the occurrence of long link waits by allowing other users to get processing time on a more frequent basis.

The way you reduce the length of time agents are held is by having the users and applications release the agents as soon as possible. Some of the ways of doing this include:

a. Use of Multiple LUWs

This means issuing COMMIT WORK or ROLLBACK WORK commands whenever practical, rather than doing work as one large LUW. If an application does not need to be one LUW, then COMMIT WORK commands should be used.

b. Use of COMMITCOUNT in DATALOAD processing

A special case of "use of multiple LUWs" is the use of the COMMITCOUNT option on DATALOADs.

c. Use of AUTOCOMMIT ON in ISQL Sessions

A special case of "use of multiple LUWs" is the use of the AUTOCOMMIT ON during ISQL sessions. With AUTOCOMMIT set to OFF, agents are held while the user views command output and enters his/her next command. This time could be used to process requests from other users. (Note: The use of AUTOCOMMIT ON causes ISQL to COMMIT WORK, and therefore release locks, at the end of many commands. However, COMMIT WORK commands are not automatically issued when the user is in display mode or input mode).

If AUTOCOMMIT OFF is needed for multiple command LUW processing from ISQL, it is recommended that the commands be executed from routines. In this way there is minimal delay between the completion of one command and the start of the next.

d. Use of Pseudo-Conversational Programming Techniques

This refers to programming such that terminal read operations are not done while the application has an LUW in progress. More specifically, it means issuing the COMMIT (or ROLLBACK) WORK statement as the last SQL statement before issuing a terminal read request.

If you are not experiencing lock waits or communications waits, then you may not have enough agents. See "Too Few Agents" on page 205 for more information on this case.

If you are also experiencing lock waits, then the lock waits could be a factor contributing to why the agents are being held for long periods of time. You may want to treat the problem as a lock wait problem, rather than a link wait problem. See "Locks Held for Long Duration" on page 180 for more information on this case.

Bad Data Distribution

Problem Description:

If the statistics in the SQL/DS catalogs do not accurately reflect the actual characteristics of the data, or if a query involves a predicate which prevents the SQL/DS optimizer from taking those characteristics into account, the optimizer may choose an inefficient access path. This will not cause the SQL request to fail, but the response time experienced may be longer than expected.

Inaccurate statistics might exist if the statistics have never been updated or are out of date (See "Inaccurate Statistics" on page 162 for more information on this case). However, it could also reflect the fact that the data, itself, defies characterization. The optimizer takes account of non-uniformity of distribution of data values of certain columns by maintaining percentile points for 10%, 50% and 90%, together with statistics on the frequency of the two most common values. These statistics are maintained only for columns which are the first column of an index. They are held in the SYSCOLSTATS catalog table.

For other columns, only the following statistics are maintained:

- number of distinct values
- second-lowest value
- second-highest value

and for these columns the optimizer assumes an "even" distribution of data values.

The optimizer takes this distribution into account when considering the selectivity of certain predicates such as equality, range, BETWEEN. However, in the case where the predicate involves a column with a non-uniform distribution of data, then the optimizer takes that non-uniformity into account only if the value against which the column is being compared is a literal value, not a host variable. If it is a host variable, then an even distribution is assumed. (And, as mentioned above, if the column is not the first column of an index, then an even distribution is assumed). For more information on problems associated with predicates containing host variables, refer to "Range Predicate Used with Host Variables" on page 199.

For example, if the catalog statistics indicate that 75% of the employees in the EMP_ACT table are assigned to activity number 95, then the optimizer will assume that a predicate of:

```
ACTNO=95
```

will select 75% of the rows in the table.

Similarly if 90% of the employees are assigned to an activity number in the range 30 to 95, then the optimizer will recognize that a predicate of:

```
ACTNO>95
```

will select no more than 10% of the rows in the table.

However, if the catalog statistics indicate that there are 20 different project numbers in the EMP_ACT table, the optimizer will assume that a predicate of:

```
PROJNO=DD3000
```

will select only 5% of the rows in the EMP_ACT table since PROJNO is not the first column in any index and so an even distribution of values is assumed. If the actual distribution is not uniform, for example if 90% of the employees are assigned to project number DD3000, then the optimizer will be making a bad assumption. In cases such as this, the optimizer may choose to use an index that will be very inefficient, or it may fail to choose an index that, for most cases, would work fine.

Note: - the distributions in the above examples are not the same as the distributions in the supplied SQL/DS sample tables.

In general, whenever, for any of the reasons described above, the optimizer does not correctly take account of the true distribution of data values, problems can arise. Consider the following situations:

1. False sense of High Selectivity

A column may appear to be highly selective (based on ROWCOUNT divided by COLCOUNT), but actually have very low selectivity. This is the case cited above. An index on project number would look very attractive in the presence of queries based on project numbers. Unfortunately, if most of your queries use PROJNO=DD3000 just to limit the answer set to that particular project, the optimizer might choose an index on PROJNO over an index on ACTNO (if there were fewer activities than project numbers).

2. False sense of Low Selectivity

Equally frustrating is the case where the optimizer has a false sense of *low* selectivity. If most of your queries are on other project numbers, the optimizer might choose the index on ACTNO over an index on PROJNO (if there are *more* activities than projects).

The optimizer can be similarly misled when considering other predicates such as range predicates and join conditions.

The bad data distribution problem can extend to indexes as well. That is, in considering access through multiple column indexes, the optimizer will consider:

$$\frac{\text{ROWCOUNT}}{\text{FIRSTKEYCOUNT}}$$

and,

$$\frac{\text{ROWCOUNT}}{\text{FULLKEYCOUNT}}$$

Again, if the distributions of key values are not "even," the optimizer can get a false sense of either high or low selectivity.

Note: You can sometimes benefit from bad data distributions. However, most of the time they will hurt your performance more than help it. Thus, it is recommended that you avoid bad data distributions unless you really think you know what you are doing.

Possible Actions:

ACTION	PAGE
Make the optimizer take account of non-uniformity	Below
Smooth Key Distributions	Below
Hide Bad Column in the Index	Below
Avoid Bad Distributions in Columns	Below
For cases with Bad Query Cost Estimates, also see: - "Inaccurate Statistics"	162
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 61. Bad Data Distribution - Actions

There are several ways of approaching problems due to "Bad Data Distribution":

1. Make the optimizer take account of non-uniformity

Change your query and/or table to meet the conditions described earlier:

- a. column in predicate must be first column of an index
- b. value in predicate must be a literal

The second condition may require use of Dynamic SQL instead of Static SQL; this incurs an overhead cost in itself, so a trade-off is involved.

2. Special index design to achieve uniform distribution.

This technique affects the "accuracy" of the SQL/DS index selection algorithms.

If key values of an index are "uneven," you can achieve a smoothing effect by adding another column to the index. By adding a column, you can make the distribution of values more even for *whole* keys. However, unless the column added is made the *first* column of the index, it will have no effect on the distribution inferred from FIRSTKEYCOUNT.

3. Redefine Columns with better distributions

Actual distribution of column values in columns is less of a concern. However, it can have some effect on performance in cases where the optimizer has choices among columns to be used as arguments defining how to scan an index (called "key domains") or used as search arguments (called SARGs) after the index scan has found a "candidate" row.

Thus, you may want to reconsider the design of your table such that you get column statistics that more accurately reflect an "even" distribution of values. For example, if you have a table that contains date information (month, day and year), you have a choice of storing the date as one column, two columns or three columns. If the distribution of months, days and years is even, then you can store the date any of the possible ways without concern. On the other hand, if all possible dates are valid, but most of the months are June or December, you may want to combine month with day (or all three).

Before you consider the actions for correcting for bad data distributions, you should verify that your system is currently operating with up-to-date statistics. See "Inaccurate Statistics" on page 162.

You may also want to investigate other possible causes of the high I/O symptom before taking the more radical step of changing the table (column) definitions. Refer to "I/O Related Performance Problems" on page 113 for the complete list of problems that could give you excessive database I/O's.

BLOCK I/O, APPC/VM and IUCV Not Resident

In 370 mode operation, DMKBIO and all high use IUCV modules are placed in the resident CP nucleus by default.

In XA and XC mode operation, HCPBIO and all IUCV modules are placed in the resident CP nucleus by default. If they are not placed in the resident CP nucleus, unnecessary CPU overhead will be caused. (VM/XA does not support APPC/VM.)

Possible Actions:

ACTION	PAGE
Make BLOCK I/O, APPC/VM and IUCV Modules Resident	Below

Figure 62. Block I/O, APPC/VM and IUCV Not Resident - Actions

Make these modules part of the CP resident nucleus. This can be done by moving these modules above DMKCPE or HCPCPE in the CP load list (CPLOAD EXEC) and regenerating the CP nucleus. This process is explained in the *VM/SP Planning Guide and Reference*. Consult your operating system programmer.

Note: These changes are only necessary if your CP nucleus has been previously modified by your operating system programmer.

Blocking Suppression for INSERT CURSORS

Problem Description:

Note: This problem is not applicable in a DRDA environment. Blocking is not supported for PUT statements in this environment.

Two problems may occur:

1. A program receives a "W" in SQLWARNA while performing an INSERT CURSOR (that is, programs that use OPEN, PUT, and CLOSE).
2. You are running a DBS utility DATALOAD and receive the following message: "ARI8002I Blocked INSERT processing was suppressed for &1 &2.."

These problems occur even though the data row involved is small (for example, a few hundred bytes) compared to the 8K-byte block. Normally, blocking should have been done.

Possible Actions:

Blocking for INSERT CURSOR will be suppressed when two rows cannot fit into an 8K block. This does not mean two times the row size of the rows to be inserted. It is derived using the following calculation:

Minimum mailbox size to perform blocking =
2 * row size +
length of RDIIN structure +
size of SQLDA +
length of the mailbox header

For a table with a large number of columns, the SQLDA grows in size quickly and carries a heavy weight in the above calculation.

For more information about INSERT blocking and suppressed blocking, see the *Performance Tuning Handbook* manual.

Buffer Pool Too Big

Problem Description:

Generally speaking, you want to run with the buffer pools (NPAGBUF and NDIRBUF) as large as you can afford to make them. This is because you can realize substantial savings in DASD I/O costs. DASD I/O's are "expensive" in terms of both CPU time and I/O wait time.

However, you *can* make the buffer pools too big. How big "too big" is will depend on the configuration of your system, and the workload on your system. More specifically, "too big" depends on how much "paging" is being done by your system.

Database I/O is better than page fault I/O. Database I/O is more expensive than paging I/O in terms of CPU cycles required, but SQL/DS processing does not stop for database I/O. Since the database manager can process other requests while doing database I/O (but not during page faults), you want to keep the buffer pools small enough to avoid high paging.

The primary symptom of "Buffer Pool Too Big" is a relatively high *paging rate*. The effect of the high paging rate will be generally poor performance (response time) for *all* SQL applications and queries. Performance will be worse for the less frequently used functions. The single user response time may be better than multiple user response time, but if there is enough non-SQL activity on the system, even this case will perform poorly.

There are other problems that can also cause paging problems. Some of these are listed in Figure 63 on page 127. However, you can distinguish "Buffer Pool Too Big" problems from these other paging problems by looking at your *Buffer Hit Ratio*. Buffer hit ratio (BHR) is the ratio of "looks in the page buffer" (LPAGBUFF) to actual DASD read operations (PAGEREAD).

This can be determined from COUNTER information as follows:

$$\frac{\text{LPAGBUFF}}{\text{PAGEREAD}}$$

A Buffer Hit Ratio of 7 would mean that 6 out of 7 requests for data pages are being satisfied out of the page buffer pool. For most environments, this would be considered a very good buffer hit ratio.

On the other hand, a buffer hit ratio between 1 and 2 would mean that your buffer pool is not saving you database I/O's very often. As a result, if your buffer pool is large, it probably isn't worth it. (Note: The buffer hit ratio cannot be less than 1).

Possible Actions:

ACTION	PAGE
Decrease Page Buffers	Below
Add More Real Storage	Below
If your buffer hit ratio is not so good, see:	
- "Too Many Agents"	206
- "NLRB Parameters Too Large"	190
- "SET QDROP OFF USERS or SET QUICKDSP ON Not Used"	203
- "SQL/DS Code Not Shared"	204

Figure 63. Buffer Pool Too Big - Actions

For the "Buffer Pool Too Big" problem, there are basically two ways of approaching the problem:

1. Reduce the demand for Real Storage

If your buffer hit ratio is good, you can probably afford to reduce the size of your buffer pools without causing major problems with your applications/queries. In fact, to the extent that reducing the size of the buffer pools reduces the paging on your system, your users should see a response time improvement.

If your buffer hit ratio is not impressive, then reducing the size of the buffer pool will only make it worse. However, if your buffer pool is also large, you may be able to decrease its size without making your buffer hit ratio significantly worse. Thus, decreasing the size of the buffer pool can be a reasonable alternative even when your buffer hit ratio is not so impressive.

There is no easy way to estimate the optimal size for your buffer pool. You must use trial and error. Decrease the size of your buffer pool until you see a significant degradation in your buffer hit ratio. At that point, you probably have reached the best trade-off between buffer pool size and the paging on your system.

2. Get more Real Storage

At the point where database I/O's increase sharply, you should consider other alternatives, such as adding more real storage to your system.

Note: Before you consider adding more real storage, you may want to investigate the related paging problems listed in Figure 63.

Buffer Pool Too Small

Problem Description:

The buffer pools maintained by the database manager (and established by the NPAGBUF and NDIRBUF initialization parameters) are intended to minimize the need to access DASD every time data is needed. The larger the buffer pools, the less likely DASD accesses will be required. Conversely, the smaller the buffer pools, the more likely it will be that the database manager will have to perform actual DASD I/O to get the data.

The buffer pool sizes needed will depend on the nature of your SQL workload. There is no one size that suits all situations. The defaults for NPAGBUF and NDIRBUF are reasonable defaults for some environments. However, they may not be appropriate for yours. In particular, for any given workload, there will usually be a specific size where **buffer pools are too small**.

When buffer pools are too small, you will experience an unusually high number of database I/O's. This is due to something known as **buffer pool thrashing**. Buffer pool thrashing is a condition where an application that references the same page(s) of data multiple times has to do database I/O each time because other users have 'stolen' its buffers for their own data. As a result, each user (or application) keeps dragging in the same pages of data over and over again until they are done. Just when one user has established its data in the buffer pools, another user gets control and replaces the data with its own.

Buffer pool thrashing, of course, means your applications are doing more database I/O than they really need to. If the buffer pools were large enough, all users (agents) would be able to have all (or most) of their data in the buffer pools at the same time.

In addition to experiencing a high number of I/O's, you should also be able to observe a very poor *buffer hit ratio*. That is,

LPAGBUF
PAGEREAD

will be a very small value (For example, less than 2).

Note: A low buffer hit ratio can also occur when you have a lot of applications that do large, sequential accesses to data (such as large sorts, CREATE INDEX on large tables, DBSPACE scans, etc.). In these cases, the large, sequential applications are **flooding the buffer pools** with large amounts of data. As a result, the smaller applications will get their buffers 'stolen'. For more information on this problem, see "Sequential Processing" on page 200.

In summary, you probably have the "Buffer Pool Too Small" problem when you have all of the following conditions:

1. Few (if any) large, sequential applications,
2. A large number of database I/O's, and
3. A very poor buffer hit ratio

Note: The "Buffer Pool Too Small" problem will usually occur when the applications are performing SQL data manipulation operations. If the applications are performing data definition operations (such as, CREATE INDEX or DROP TABLE),

loading/unloading or large reports, then you probably have the "Sequential Processing" problem. For more information on this problem see "Sequential Processing" on page 200.

Possible Actions:

ACTION	PAGE
Increase Page Buffers	Below
Decrease NCUSERS	Below
Use Redundant Data to Avoid I/O's	Below
Re-design Application	Below
For other problems with a poor buffer hit ratio, see: - "Sequential Processing"	200
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 64. Buffer Pool Too Small - Actions

There are basically three ways of approaching the "Buffer Pool Too Small" problem:

1. Increase the size of the buffer pools

The first approach is obvious. However, under your current workload, it may not be practical to increase the size of the buffer pools to the size necessary to avoid buffer pool thrashing. Thus, the first approach may, in fact, not be appropriate for your situation. If the size is impractical, you need to investigate the other possible actions. If you do increase the buffer pool size, be aware that it may also increase system paging I/O.

2. Decrease the number of users needing buffers

If increasing the size of the buffer pools is not practical, the next thing to consider is reducing the contention for buffers by reducing the number of users that are using the buffer pools. That is, reduce the number of users that can be active at the same time.

For more information on problems relating to the number of users and reducing the number of users, see "Too Many Agents" on page 206.

3. Decrease the number of buffers needed

In some cases, neither of the first two approaches will be desirable and it will be necessary to consider a third approach. Specifically, you can also reduce buffer contention by reducing the number of buffers each application requires. This can be done in one of two ways:

a. Data design changes

In the first case, the objective is to get the highly referenced data stored on as few pages as possible. That is, if the average application accesses 10 pages of data, you want to try to get the average down to say 5 pages by reorganizing or restructuring the data. Clustering data, lowering freespace allocations, dropping and recreating indexes,

changing the mappings of tables to DBSPACES, and even some uses of redundant data can be used to do this.

b. Application design changes

In the second case, the objective is to get the applications to be more specific in their data requests. That is, you would review your applications to make sure they are not referencing more data than they really need to. Some of the techniques discussed under "Inefficient Search" on page 170 might apply.

CHARNAME Not Set Correctly

Problem Description:

The CHARNAME parameter specifies the CCSIDs to be used as the defaults for either the application server or the application requester. If the CHARNAMEs of the application server and the application requester are different, then in a request the elements of predicates may have different CCSIDs. In any predicate where the elements of the predicate have different CCSIDs, the database manager usually has to perform a CCSID conversion. This changes any sargable predicate into a residual predicate. Both the data conversion and the sargability have an impact on performance. For more information on the performance impact of CCSIDs, see the *Performance Tuning Handbook*.

Different CHARNAMEs may be unavoidable to ensure data integrity. For example, in some situations the application server and the application requester must be set differently to reflect the code page of the terminal. However, the CHARNAME of the application server and the application requester should be the same unless there is a *specific reason* they should be different.

For additional information on CCSIDs, see the *SQL Reference* manual. For details on choosing a CHARNAME, or CCSID conversion, see the *System Administration* manual.

Checkpoint is Being Forced at End-LUW

Problem Description:

Normally, an SQL/DS checkpoint is taken every time CHKINTVL log pages have been filled (CHKINTVL is an SQL/DS initialization parameter). There are two cases, however, where a checkpoint is forced at the end of every logical unit of work (LUW):

1. The SQL/DS application server is being run in single user, nolog mode (LOGMODE=N). A checkpoint is forced even if the LUW is read-only.
2. An LUW modifies data in one or more nonrecoverable DBSPACES.

Under normal, intended use of nolog mode or nonrecoverable DBSPACES, these forced checkpoints have no significant performance effect. They can however, significantly degrade performance if such LUWs occur frequently.

Possible Actions:

ACTION	PAGE
Avoid Short LUWs	Below
Run with Logging	Below

Figure 65. Checkpoint Being Forced at End-LUW - Actions

When running in single user mode with LOGMODE=N, package the work into one or a few LUWs. If this is not practical, you should consider running with logging in effect (LOGMODE = Y, A or L).

In general, the only modifications to nonrecoverable DBSPACES should be due to the bulk loading of data from an external source. These bulk load operations should be packaged into one or a few LUWs to minimize checkpoint overhead. Read-only use of nonrecoverable DBSPACES involves no special performance considerations, since in that case there is no forced checkpoint at end-LUW. If you need to make updates to the data once it has been loaded, you probably should be using a recoverable DBSPACE instead.

CHKINTVL Too Big

Problem Description:

If the checkpoint interval is set too high, the result can be unacceptably long response time delays each time a checkpoint occurs. This is because:

1. The time required to process a checkpoint increases as CHKINTVL increases.
2. All users must wait while a checkpoint is in progress.

This condition can be verified by issuing the SHOW ACTIVE operator command each time a long, unexpected delay is being experienced. If you frequently see "CHECKPOINT AGENT IS PROCESSING A CHECKPOINT" as a response, checkpoint processing delays may be a significant factor and a reduction in the checkpoint interval should be considered.

If you often see "CHECKPOINT AGENT IS WAITING TO START CHECKPOINT" as a response, the database manager is in the process of quiescing activity in the DBSS in preparation for doing a checkpoint. This problem is addressed under "Long DBSS Calls Delaying Checkpoint" on page 184. If the results usually show that the checkpoint agent is inactive, checkpoint-related delays are not the problem.

Possible Actions:

ACTION	PAGE
Reduce CHKINTVL	132
If buffer pool is very large, see: "Buffer Pool Too Big"	126
If delays are during quiesce for checkpoint, see: "Long DBSS Calls Delaying Checkpoint"	184
If checkpoint is inactive during delays, see: "Sequential Processing"	200

Figure 66. CHKINTVL Too Big - Actions

Reduce the CHKINTVL initialization parameter. Proceed with caution, however, since there are significant advantages to keeping CHKINTVL as high as is practical. Review the discussion under "CHKINTVL Too Small."

CHKINTVL Too Small

Problem Description:

The checkpoint interval, as specified by the CHKINTVL initialization parameter, is too short. This results in significant additional database I/O and associated processing overhead.

A short checkpoint interval also increases the likelihood that a long-running DBSS call (for example, to implement a DROP TABLE) will delay the initiation of the checkpoint process. This delay will add to the response time of all other users that concurrently have an SQL statement in progress. See "Long DBSS Calls Delaying Checkpoint" on page 184 for a discussion of this problem.

Possible Actions:

ACTION	PAGE
Increase CHKINTVL	Below
Decrease Logging Rate	Below
See "Long DBSS Calls Delaying Checkpoint"	184
If high log I/O, see: "Logging during Load"	183

Figure 67. CHKINTVL Too Small - Actions

Consider increasing the CHKINTVL initialization parameter. With proper allowance for the additional secondary storage requirements (see below), many installations will find that the optimum CHKINTVL setting is in the 50-300 range. Installations with large, randomly modified databases will be in the lower end of that range, while installations with small databases or large databases having a relatively low frequency of random modifications will tend to be in the upper end of that range.

Increasing CHKINTVL will realize the following advantages:

- The overhead associated with the checkpoint process will decrease. This can have a very beneficial effect on overall performance.
- The frequency of checkpoint-related response time delays is reduced. There are two different types of delays:
 1. The time it takes to quiesce all DBSS activity before the checkpoint can be started.
 2. The time it takes to do the actual checkpoint.

The advantages of increasing CHKINTVL should be weighed against the following adverse side effects:

- It will take more time to restart the application server after a system failure.
- If you are running with LOGMODE=Y, the possibility of a log full condition is increased because the database manager only reclaims log space as part of checkpoint processing. This consideration does not apply if you are doing archiving (LOGMODE=A) because in that case log space is reclaimed only when the database is archived.
- Checkpoints will be less frequent, but when a checkpoint *does* occur, checkpoint processing will take longer (quiesce time stays the same). This delays all users who currently have an SQL statement in progress.

This will be an important effect if the workload includes a significant amount of random data modifications over a relatively large area (for example, more than 100 megabytes). This will be an unimportant effect if the database is small or there is very little random data modification activity. Bulk sequential data modifications generally do not pose a problem.

- Secondary storage requirements will increase.

Whenever a page in the database is modified and it has to be written out to secondary storage, the database manager does not overlay the original copy of that page, but instead writes it out to a free page in that same storage pool. The original, unmodified copy is referred to as a shadow page.

The shadow page mechanism is useful in facilitating system recovery, but it does require that sufficient empty pages be available in each storage pool. The secondary storage occupied by shadow pages is only reclaimed when the database manager takes a checkpoint. Therefore, the amount of extra space that must be set aside in each storage pool increases in proportion to the CHKINTVL setting.

Of these side effects, it is the secondary storage requirements that usually require the most attention. See "Estimating Database Storage" in the *System Administration* manual.

You can check secondary storage availability by issuing the SHOW DBEXTENT or SHOW POOL operator command.

Since an SQL/DS checkpoint is triggered by the number of log pages that have been filled, another way to reduce the checkpoint rate is to reduce the rate at which log pages are filled. This may be useful in conjunction with a moderate increase in CHKINTVL if larger CHKINTVL settings are precluded due to the side effects cited above.

One way to slow down the logging rate is to run big DATALOADs and other bulk update operations off-line in single user mode with LOGMODE=N. Another technique is to bulk load data into nonrecoverable DBSPACES. This is applicable if the data is read-only and can be easily reconstructed from an external source in the event of a DASD failure. See "Nonrecoverable Storage Pools" in the *System Administration* manual for guidelines.

CMS Work Unit Support Set On

Problem Description:

When the database machine is running in a VM/SP or VM/ESA environment, it is preferable that CMS work unit support be set OFF. Applications running with this support set ON use more of the CPU because each call to the Resource Adapter results in a CMS call to query the work unit identifier. The default parameter, set in the SQLINIT EXEC, is ON. Details on the SQLINIT EXEC are in the *Database Administration* manual.

CMS work unit support does not apply to VM/XA SP.

Conflict in Catalog Key Locking

Problem Description:

SQL Data Definition and Control statements result in update, insert and/or delete operations on the catalog tables. Thus, they acquire EXCLUSIVE locks in the catalog DBSPACE. This can result in conflicts with almost any other kind of SQL activity.

In an attempt to minimize the contention in the catalog tables, the database manager does row and key level locking in the catalog DBSPACE. However, conflicts can arise. As with row (and key) locking on user tables, the following types of conflicts can arise:

1. Conflicts on Key hashes on non-unique indexes

As a result of key level locking, conflict can arise when "names" of objects (tables, indexes, programs, etc.) hash to the same value. If you observe locking conflicts between objects with dissimilar "names," the problem is probably a conflict on the hash values of the keys.

This applies only to non-unique indexes. See Figure 68 on page 135 to find the catalogs that have non-unique indexes. For more information on key hashing conflicts see "Conflict on Key Hash in User Data" on page 136.

2. Adjacent Key locking

Adjacent key locking conflicts can also arise in the catalogs. If you observe locking conflicts between objects with similar "names," the problem is probably an adjacent key locking problem.

For more information on adjacent key locking problems see "Adjacent Key Locking in User Data" on page 115.

3. Multiple statements contending for the same catalog entries.

For more information on this conflict, see "Hot Spot in the Catalog Tables" on page 153.

In order to understand the type of catalog key locking problem you have, you need to know the indexing done on the catalogs. Figure 68 identifies the indexes on the SQL/DS catalog tables.

CATALOG	UNIQUE	INDEX	COLUMNS
SYSACCESS	U	IACCESS	TNAME,CREATOR,TABTYPE
SYSCATALOG	U	ICAT	TNAME,CREATOR
SYSCCSIDS	U	ICCSIDS	CCSID
SYSCHARSETS	U	ICHARSETS	NAME
SYSCOLAUTH	- - -	ICOLAUTH1 ICOLAUTH2 ICOLAUTH3	CREATOR,TNAME,COLNAME,GRANTEE TIMESTAMP,COLNAME GRANTEE,TIMESTAMP,COLNAME
SYSCOLSTATS	U	ICOLSTAT	TNAME,CREATOR,CNAME
SYSCOLUMNS	U	ICOL	TNAME,CREATOR,CNAME
SYSDBSPACES	- - U	IDBSPACE IDBSPACE2 IDBSPACE3	OWNER,DBSPACENAME DBSPACETYPE,OWNER,NPAGES DBSPACENO
SYSDROP	-	IDROP	DBSPACENO,QUALF
SYSFIELDS	U	IFLD	TNAME, CREATOR, CNAME
SYSINDEXES	- U	IINDX IINDX2	TNAME,CREATOR INAME,ICREATOR
SYSKEYCOLS	-	ISYSKEYCOLS1	TNAME,TCREATOR,KEYTYPE,KEYNAME,KEYORD
SYSKEYS	- -	ISYSKEYS1 ISYSKEYS2	TNAME,TCREATOR,KEYTYPE REFTNAME,REFTCREATOR
SYSLANGUAGE	- -	SYSLANGINDEX SYSLANGIDINDEX	LANGUAGE LANGID
SYSOPTIONS	U	IOPTIONS	SQLOPTION
SYSPROGAUTH	- -	IPROGAUTH1 IPROGAUTH2	CREATOR,PROGNAME,GRANTEE,RUNAUTH GRANTOR,GRANTEE,CREATOR,PROGNAME
SYSSTRINGS	U	ISTRINGS	INCCSID,OUTCCSID
SYSSYNONYMS	U	ISYN	USERID,ALTNAME
SYSTABAUTH	- - - - -	ITABAUTH1 ITABAUTH2 ITABAUTH3 ITABAUTH4 ITABAUTH5	GRANTEE,TCREATOR,TTNAME,GRANTEETYPE GRANTOR,SCREATOR,STNAME GRANTOR,GRANTEE,SCREATOR,STNAME,GRANTEETYPE SCREATOR,STNAME TCREATOR,TTNAME
SYSUSAGE	- -	IUSAGE IUSAGE2	BNAME,BCREATOR,BTYPE DNAME,DCREATOR,DTYPE
SYSUSERAUTH	U	IUSERAUTH	AUTHOR,NAME
SYSVIEWS	U	IVIEWS	VIEWNAME,VCREATOR,SEQNO

Figure 68. Indexes on SQL/DS Catalog Tables

As can be seen from Figure 68, there are many ways in which a key locking conflict might arise. For example, the USER1.INVENTORY key would presumably be adjacent to the USER2.INVENTORY key in SYSTEM.SYSCATALOG and SYSTEM.SYSTABAUTH, and possibly SYSTEM.SYSINDEXES and SYSTEM.SYSUSAGE. Thus, creating an index on USER2.INVENTORY (which is an SQL Data Definition statement inserting into SYSINDEXES) could conflict with an ad hoc query on USER1.INVENTORY (which references SYSINDEXES for index information on USER1.INVENTORY). The conflict, in this case, would be an "adjacent key lock" conflict in IINDX (SYSINDEXES).

Key hashing conflicts are less obvious. If SHOW LOCK WANTLOCK indicates a conflict on a catalog key hash, and there is no evidence that the keys involved

are similar, then the conflict is probably a key hash conflict. This can only occur for non-unique indexes.

Possible Actions:

ACTION	PAGE
Schedule jobs with SQL data definition and control statements in sequence	Below
For other catalog locking problems, see: - "Hot Spot in the Catalog Tables"	153

Figure 69. Conflict in Catalog Key Locking - Actions

You cannot change the indexing on the catalogs. Even if you add indexes, the database manager will not use them for Data Definition or Data Authorization statements. To avoid the contention in the catalog tables, schedule the contending jobs so that they run in sequence.

Conflict on Key Hash in User Data

Problem Description:

This section applies only to non-unique indexes. For unique indexes, the tuple identifier (TID) is used as the gatename.

When a DBSPACE is defined to have row level locking (LOCKMODE = "T"), locking on non-unique index data in the DBSPACE is done at the 'key' level. With key level locking, locks are not actually obtained on individual key values. Instead, they are obtained on a 4-byte hash of the key values. Different key values can hash to the same 4-byte hash value, resulting in the same "gatename." That is, for locking purposes, the key values will look like they are the same. This, of course, is the "Conflict on Key Hash in User Data" problem.

The "Conflict on Key Hash in User Data" problem will look like any other locking conflict problem. That is, you will see lock waits. In addition, the lock waits you will see, will be in the indexes.

A problem with almost identical symptoms is "Adjacent Key Locking in User Data" on page 115.

A special case of conflicts on key hashing is the "Conflict in Catalog Key Locking" problem. This case is where the key hash conflict occurs on non-unique indexes on the catalog tables. See "Conflict in Catalog Key Locking" on page 134 for more information.

Normally the key hash conflicts occur in user data and show up as conflicts between data manipulation statements, or conflicts between data manipulation statements and load or unload operations. However, the conflict can also occur between data manipulation statements (DELETE, INSERT, SELECT and UPDATE) and data definition or control statements (CREATE, DROP, GRANT or REVOKE) in some cases. In particular, dynamic SQL data manipulation statements (for example, SELECT from ISQL or the DBS Utility) will require references to the catalogs. This can conflict with data definition or data control statements which update the catalogs. Since the catalog DBSPACE has row level (and therefore

key level) locking, conflicts on key hashes in the catalogs can occur. More detail on these cases are provided in "Conflict in Catalog Key Locking" on page 134.

Possible Actions:

ACTION	PAGE
Change Key Structure to Avoid Key Conflicts	Below
Increase Lock Level	Below
For other problems with similar symptoms, see: - "Adjacent Key Locking in User Data" - "Hot Spot in User Tables"	115 158
If lock waits are in catalog DBSPACE, see: - "Conflict in Catalog Key Locking" - "Hot Spot in the Catalog Tables"	134 153
For other lock wait problems, see: - "Locking Related Performance Problems"	114

Figure 70. Conflict on Key Hash in User Data - Actions

Conflicts on Key hash values are not easily avoided. There are basically two approaches that can be considered:

1. Changing the structure of the key

Since the hash value for keys is a function of the key values, one approach to avoiding the conflict is to change the key values. That is, you can redefine the index to be a different combination of columns, or a different sequence of the same columns. By doing this, you may not totally eliminate hash value conflicts, but can change the frequency of conflicts. It can also change the characteristics of which keys conflict.

Another way to eliminate key hash conflicts is by redefining the index as unique instead of non-unique.

2. Increasing the lock level

Since key hash conflicts occur only with key level locking, another approach would be to use page level locking. Under page level locking, index pages are locked instead of the key hash values. This means locks on "similar" keys will conflict, but this may be more desirable than the conflict on key hash values.

You might also review the problems with similar symptoms (see Figure 70 for the list) before concluding that your problem is the key hash problem. The differences between these problems and the key hashing problem are very subtle. However, the solutions can differ significantly.

If the key locking problems are in the catalog DBSPACE (DBSPACE 1), then you should refer to the problems dealing with catalog locking (see Figure 70 for the list).

CREATE INDEX Requires a Large Sort

Problem Description:

CREATE INDEX processing includes a sort of all index key values. This step can be time-consuming for a large table, especially if the keys are large.

Possible Actions:

ACTION	PAGE
Define Fewer Key Columns	Below
Make the Key Column(s) Smaller	Below
Spread Sort Across a Number of Devices	Below
Ensure NPAGBUF is Large Enough	Below
REORGANIZE INDEX	169
Also see: "DBSPACE Scan Being Performed"	141

Figure 71. CREATE INDEX Requires a Large Sort - Actions

1. Define fewer Key Columns

In the case of a multicolumn key, check to see if any of the columns can be eliminated from the index definition.

2. Make the Key Columns smaller

Consider the feasibility of redefining the key columns so as to make them smaller. For example, a part number might currently be stored in a CHAR(12) column. If all valid part numbers are numeric, you might be able to store the part number in binary form in a 4-byte INTEGER column. This will result in about a 3:1 reduction in the volume of data that has to be sorted, which reduces sort time.

3. Spread Sort across a Number of Devices

For a very large sort, it is frequently the case that the time required to do the sort is dominated by DASD seek time. This situation is indicated if, during CREATE INDEX processing on a dedicated system, CPU utilization is not high (less than 50 percent) and channel utilization is low (less than 5 percent).

This problem can be alleviated by putting the internal DBSPACES into their own storage pool which is associated with a number of DBEXTENTS, each on a separate drive.

The size of each DBEXTENT is important because SQL/DS will not store pages in the next-defined DBEXTENT in a storage pool until all previously defined DBEXTENTS have been filled. When you are deciding on how big to make each DBEXTENT, consider the following:

- The number of bytes that have to be sorted to accomplish a given CREATE INDEX can be estimated by multiplying key length + 8 by the number of rows in the table.

- The amount of space that is used during sorting is approximately twice this number.

Suppose that you wish to spread sort processing across six devices and that you have estimated the number of bytes to be sorted for your largest CREATE INDEX to be N. One approach, then, would be to make the size of the six DBEXTENTS 0.1N, 0.2N, 0.3N, 0.4N, 0.5N, and 0.6N respectively. By graduating the sizes, a wider range of sort sizes will result in I/O activity that is spread across two or more devices. Actually, the sixth (last defined) DBEXTENT should be made extra large so that unusually large sorts can still be accommodated.

4. Ensure NPAGBUF is large enough

The merge phase of a large sort uses 25% of the page buffers, up to a maximum of 64 buffers. If NPAGBUF is less than 256, the merge phase can have many unnecessary merge passes.

Data Not Cached

Problem Description:

DASD or Expanded Storage caching can improve SQL/DS performance by providing faster access to data than can be obtained with normal DASD I/O. Caching support varies by operating system. Consult the operating system documentation for your installation for instructions on how to utilize caching.

Caching is frequently used by the operating system to benefit all users. Consult your System Programmer to see if more cache resource can be made available.

Possible Actions:

ACTION	PAGE
Read-only Applications	Below

Figure 72. Data Not Cached - Actions

1. Read-only Applications

Caching is of most use for applications that are read-only. It is not recommended that the directory and logs be candidates for caching. However, the most important primary read-only database extents should be made eligible for caching.

Database Machine Favored Too Little

Problem Description: The SQL/DS database machine was run with default dispatching priority or with too little favoring, resulting in a disproportionate increase in the response time experienced by SQL/DS users during periods of high processor utilization.

If all virtual machines are run with default dispatching priority, the CP scheduler will attempt to distribute the CPU equally among them. A database machine, however, may be supporting a large number of users, whereas a user virtual machine just has to support one user. In such situations, truly equitable distribution of the CPU resource requires that the database machine be given a dispatching priority that is better than the default.

VM tries to optimize system throughput by monitoring the execution status of virtual machines. When a virtual machine becomes idle, VM drops it from the run list. The virtual machine's page and segment tables are scanned, and resident pages are invalidated and put on the flush list. If this cycle of queue dropping and reactivation is executed repeatedly, the overhead of invalidating and revalidating the virtual machine's pages can become large.

Possible Actions when using VM/SP and VM/ESA 370 Feature

ACTION	PAGE
Increase SET PRIORITY and/or SET FAVORED	Below
SET QDROP OFF USERS	203

Figure 73. Actions under VM/SP and VM/ESA 370 Feature

VM/SP and VM/ESA 370 Feature offer two different CP operator commands to change the dispatching priority associated with a virtual machine: SET PRIORITY and SET FAVORED. Priority can be raised by using one or both of these commands. SET PRIORITY requires less overhead than SET FAVORED, but SET FAVORED can have a larger effect.

The exact values to use can be determined only by trial and error. First try "SET PRIORITY userid 1," and if this does not have a large enough effect, additionally issue "SET FAVORED userid 50." After observing the effect of this combination, adjust the SET FAVORED percentage value upward or downward as necessary.

For further information, see "Performance" in VM/SP Administration for VM/SP or in VM/ESA CP Planning and Administration for 370.

Note: Priority 1 is the best; larger values mean less priority.

Possible Actions when using VM/XA SP and VM/ESA ESA Feature

ACTION	PAGE
Increase SET SHARE	Below
SET QUICKDSP ON	Below
SET RESERVED	Below

Figure 74. Actions under VM/XA SP and VM/ESA ESA Feature

Increase SET SHARE: The VM operator can use the SET SHARE command to control the percentage of system resources a user receives. These system resources include processors, real storage, and paging I/O capability. The SQL/DS database machine is usually a heavily loaded machine. If more system resources can be allocated, users will usually benefit from better system response.

A virtual machine receives its proportion of any scarce resource according to its share setting. An ABSOLUTE share allocates to a virtual machine an absolute percentage of all available system resources. A RELATIVE share allocates to a virtual machine a portion of the total system resources minus those resources allocated to virtual machines with an ABSOLUTE share. Also, a virtual machine with a RELATIVE share receives access to system resources with respect to

other virtual machines with RELATIVE shares. If the SQL/DS RDBMS is the major application on your VM system, you may want to specify an ABSOLUTE share; otherwise, specify a RELATIVE share.

SET QUICKDSP ON: An SQL/DS database machine usually requires critical system response time and it is always expected to be in service to all user machines. Having the database machine always available in the dispatch list is one of the most important aspects of improving the database machine response time.

VM offers a system operator command, SET QUICKDSP ON, to allow designated virtual machines to not wait in the eligible list when they have work to do. Therefore, a virtual machine with a QUICKDSP setting is assigned an eligible list class of E0 and is added to the dispatch list immediately.

When a virtual machine has both QUICKDSP and SHARE settings, it does not wait in the eligible list and may spend a proportionally greater percentage of time in the dispatch list. Because of this, a QUICKDSP virtual machine's SHARE setting is spread out over a longer period of time. It is advisable to give a QUICKDSP virtual machine a higher SHARE setting than normal.

SET RESERVED: To work with SET SHARE and SET QUICKDSP ON, users should also consider using SET RESERVED to obtain the best effects under some conditions.

A virtual machine that has pages reserved with the SET RESERVED command gets to hold the reserved pages essentially 100% of the time, when the virtual machine is dormant. Thus, no paging delays are incurred (normally) when the virtual machine wants to use storage.

The CP scheduler allows a QUICKDSP virtual machine to enter the dispatch list regardless of the virtual machine's storage requirements and the system's current storage load. Also, when a virtual machine is dormant it is likely that CP will steal some or all of its pages. When it leaves the dormant state, this virtual machine will need its working set to be brought back into storage. This heavy paging requirement will hurt the virtual machine's performance. The solution may be to use SET RESERVED for at least some of the virtual machine's pages.

For further information, see "Performance" in VM/XA SP or VM/ESA ESA Feature Planning and Administration.

DBSPACE Scan Being Performed

Problem Description:

A DBSPACE scan is one of the "access paths" available for accessing a table. A DBSPACE scan does exactly that. All active data pages in the DBSPACE are read in search of the desired rows. This frequently results in some rather unpleasant performance characteristics. In particular, you will probably observe:

1. A high number of database I/O's

It usually requires a substantial number of database I/O's (to read all the active data pages).

2. A high CPU usage

In addition to the I/O wait time implied, a substantial number of CPU cycles are used in doing the I/O's.

3. Lock wait conditions

Furthermore, since all active pages are going to be read, the database manager usually gets a *table* lock to do the scan, even when the DBSPACE is defined to have row or page level locking. The only exception to this is when the scan is done under the cursor stability isolation level in a DBSPACE defined to have page level locking.

Clearly, you do not want to use DBSPACE scans any more often than is necessary. However, for many requests, a DBSPACE scan is the only reasonable way for the database manager to proceed.

For data manipulation operations (DELETE, INSERT, SELECT and UPDATE), the database manager may have other access paths from which to choose (such as, Index scans). In these cases, DBSPACE scans are usually the least efficient access path. However, if the proper indexes or clustering have not been set up, it may be the most efficient path *available*. Nonetheless, DBSPACE scans are usually unattractive.

Some operations are done **only** through DBSPACE scans because the DBSPACE scan is the only reasonable way to do them. In particular, CREATE INDEX, DROP TABLE and UPDATE STATISTICS are *always* done with DBSPACE scans. In any of these cases, the table in question might be quite small. But if it is in a DBSPACE with a large number of active pages, all the pages will be read, and you will get a lot of database I/O's. In cases where a CREATE INDEX is preceded by DROP INDEX, the DBSU REORGANIZE INDEX command could be used, and the DBSPACE scan is avoided.

For data manipulation statements, DBSPACE scans could be occurring because of the lack of alternatives to choose from. If you are experiencing DBSPACE scans on data manipulation statements, refer to some of the other problems documented in this manual. Specifically, you might want to look at problems that can result in DBSPACE scans, as listed in Figure 75.

Possible Actions:

ACTION	PAGE
Try to Avoid DBSPACE Scans	Below
Minimize the Impact of Scans	Below
CREATE INDEX Before Loading	Below
Run Offending Jobs Off-hours	Below
For other possible DBSPACE scan problems, see:	
- "Package Needs Re-preprocessing"	194
- "Bad Data Distribution"	122
- "Inaccurate Statistics"	162
- "Index Disqualified"	164
- "Index No Longer Highly Clustered"	168
- "No Selective Index"	192
- "Inefficient Search"	170
- "Insufficient Indexing"	174
- "Large Tables Share Same DBSPACE"	175
- "Missing Search Condition"	185
- "Need a Highly Clustered Index"	187
- "Range Predicate Used with Host Variables"	199
- "Too Many Joins"	208
- "Very Nonunique Index Key Prefix"	211
For similar CREATE INDEX problems, see:	
- "CREATE INDEX Requires a Large Sort"	138
REORGANIZE INDEX	169
For similar UPDATE STATISTICS problems, see:	
- "UPDATE STATISTICS by DATALOAD"	210

Figure 75. DBSPACE Scan Being Performed - Actions

There are basically four approaches to addressing "DBSPACE Scan Being Performed" problems:

1. Try to Avoid DBSPACE Scans

For data manipulation operations, you should try to avoid DBSPACE scans when they are not really necessary. Usually this means proper indexing or some type of data reorganization. See the actions for the problems listed in Figure 75 under "For other possible DBSPACE scan problems."

2. Minimize the impact of the scans

If DBSPACE scans cannot be avoided (as in the case for CREATE INDEX, DROP TABLE and UPDATE STATISTICS), and the REORGANIZE INDEX command is not appropriate, then the next thing you might consider is minimizing the impact of a scan when it is done. That is, minimize the number of I/O's needed to do the scan. This could be done by:

a. Minimizing freespace allocations

By using a low freespace setting for the DBSPACE, you maximize the number of rows that fit on a page. This will minimize the number of pages needed to hold all the rows.

b. Using the most space efficient data types

Some data types are more efficient than others. VARCHAR might be considered instead of CHAR for character data that has many values

shorter than the maximum. Use SMALLINT instead of INTEGER where possible. For information about the space used by the various data types, see "Estimating Storage for a Table" in the *Database Administration* manual.

c. Correctly sequencing columns for storage efficiency

The following left-to-right arrangement of table columns may reduce the amount of space required to contain a table:

- Fixed-length columns for which NOT NULL is specified.
- Variable-length columns for which NOT NULL is specified.
- Variable-length columns with nulls allowed.
- Fixed-length columns with nulls allowed.

This technique takes advantage of the fact that if a row is inserted with the nullable columns being implicitly set to null (that is, the columns are not explicitly stated in the INSERT statement) and the nullable columns are the rightmost columns in the table definition, then no storage is allocated for the nullable columns explicitly stated (see Figure 76 on page 145). This method of reducing the space required by a table is only recommended in the case of "write-once" data, where table data is loaded and never changed.

The following factors should be considered before a table is organized in this fashion:

- 1) If the null data is updated, the storage required for the row containing the data will be larger than was initially allocated. This can cause data fragmentation, where table data is spread across multiple data pages, resulting in poorer performance of data manipulation statements.
- 2) If the DBS utility is used to unload the table or DBSPACE, the reload processing will result in more storage being required to contain the same table. This is because the utility always explicitly specifies all columns being reloaded, even when they contain null values.

In all cases where data rows will be updated, or where the DBS utility unload facility is required, tables should be organized so that the last column is not nullable or you should explicitly list all columns when data is inserted.

```
CREATE TABLE T1 (A SMALLINT NOT NULL, B VARCHAR(10) NOT NULL,
  C VARCHAR(10), D SMALLINT)
```

```
INSERT INTO T1 (A,B) VALUES(1,'A')
```

```
INSERT INTO T1 (A,B,C) VALUES(2,'B',NULL)
```

```
INSERT INTO T1 (A,B,C,D) VALUES(3,'C','D',NULL)
```

PAGE HEADER				ROW HDR1		0001	01	A	ROW HDR2		0002	01	B
01	FF	ROW HDR3	0003	01	C	02	00	D	FF			
											3	2	1

Note: As each row is inserted, a 6 byte row header and a 2 byte page offset field (at the end of the data page) are allocated. Row 1 consists of the 6 byte header, 2 bytes for the small integer column, 1 byte for the length of the VARCHAR column and one byte for the data that was entered. The columns that were not specified take up NO space. The second row is like the first but because the third column was explicitly set to null it takes up 2 additional bytes (one for the length of the field (including the null byte) and one for the null byte). When the third row is inserted all columns are specified explicitly. The third field in that row has one byte for the length, one byte for the null byte and one byte for the data entered. The fourth field, though null, takes up 3 bytes because it is a fixed length field. It uses 1 byte for the null byte and 2 bytes containing unknown data.

Figure 76. Internal View of Data Base Page

3. Create INDEX *before* loading the table(s)

Normally you want to create indexes *after* loading the table. However, for tables that have a small number of large rows (such as tables with LONG VARCHAR columns), it may be more efficient to create the index *before* loading. The time lost doing index maintenance during load in such cases may be less of a problem than the time lost scanning the active pages after the load.

4. Run Offending Jobs Off-hours

If all else fails, you should consider running the applications or commands that cause the scans when the impact to other users is the least. That is, run the offending jobs during non-peak hours.

Deadlocks

Problem Description:

When two update applications run concurrently, deadlock occurs and one application is rolled back. Note that in this context the SQLPREP preprocessor is considered to be an update application; it updates the catalog tables.

Possible Actions:

ACTION	PAGE
For High I/O problems, see: - "I/O Related Performance Problems"	113
SET ISQL AUTOCOMMIT ON see also - "Locks Held for Long Duration"	180
Revise application logic	Below
Schedule preprocessing jobs which relate to a common set of tables in sequence.	Below
For other lock-related problems, see: - "Locking Related Performance Problems"	114

Figure 77. Deadlocks - Actions

Deadlocks can arise for a variety of reasons. Four causes are listed here. These are not the only causes.

1. Excessive number of DASD I/Os

When a DASD I/O is required, that user's work (agent) is suspended and another user's agent may be dispatched. Often, when an agent is suspended waiting for DASD I/O, it holds SHORT locks, (locks which would be released at the end of the DBSS call). The longer duration that such locks are held, the greater the likelihood of lock conflict and hence of deadlock.

2. ISQL AUTOCOMMIT OFF

This is one instance of the more general case of holding locks for an excessive duration. If two ISQL users are both updating the same table with AUTOCOMMIT OFF, and also issuing occasional SELECT statements interspersed with the UPDATE/DELETE/INSERT statements, then there is a high likelihood of deadlock.

3. Application logic

The classical example of deadlock due to application logic is where one user tries to update row A followed by row B while another user tries to update row B followed by row A. If both users succeed in performing their respective first updates before either commences their second, deadlock occurs.

However, deadlock can also occur where an application does:

- a. INSERT a row
- b. SELECT or FETCH the row just inserted

when the table has a nonunique index.

If two users U and V say run this application, and U successfully inserts a row with a particular key, and then V attempts to insert a different row but

having the same key value, and then U attempts to SELECT or FETCH the row he inserted, deadlock occurs.

4. Concurrent preprocessing of programs which all access a common set of tables

If several users are concurrently preprocessing programs which all access a common set of tables and if those programs are large enough that the preprocessing step takes an appreciable amount of time, to the extent that several preprocessing tasks are in progress concurrently, then there is a likelihood of deadlock involving updates to the Catalog tables which relate to package usage (SYSUSAGE).

DRDA Protocol Used to Access an SQL/DS Database

Problem Description:

When the application server is an SQL/DS server and the application requester is an SQL/DS user machine, the DRDA protocol should only be used in specific cases where the extra processing involved is not a concern (for example, prototyping).

There is additional overhead when the DRDA protocol is used. The extra overhead is primarily caused by the generation and parsing of application requests and replies from SQL/DS internal format to DRDA DDM/FD:OCA format, and DRDA DDM/FD:OCA format to SQL/DS internal format.

DRDA Usage

If your problem is occurring during distributed processing, it may be unrelated to the performance of the database or application. It may be a communication subsystem performance problem. Refer to the *Distributed Relational Database Problem Determination Guide* for additional information on problem diagnosis.

ECMODE ON for Accounting

Problem Description:

When running in 370 mode, it is preferable that ECMODE be set OFF when using the SQL/DS accounting facility. ECMODE OFF uses less CPU than ECMODE ON. Details on using the SQL/DS accounting facility are detailed in the *System Administration* manual. ECMODE does not apply in XA or XC mode.

Excessive I/Os on INSERT

Problem Description:

When an application issues the SQL INSERT statement to insert a row or rows into a table, there is a high number of DASD I/Os, on the DBSPACE DASD and/or Directory DASD. "High" here means "significantly higher than normally occurs."

If you can be sure that the DASD I/Os are caused by the INSERT statement and not by other activity, then it is likely that the database manager is unable to find free space to insert each row in its "preferred" page, and is searching through the DBSPACE to determine where the insertion will occur.

The "preferred" page is determined as follows:

- If there is an index, then it is the page containing the first row whose key is greater than or equal to the key of the row to be inserted. If there is more than one index, this is determined by the "clustering" index, i.e. the one which has CLUSTER = 'F' or 'W' in SYSINDEXES. SQL/DS assigns whichever index was created first to be the "clustering" index. When the "clustering" index is also highly clustered (it has a high CLUSTERRATIO, which can be achieved by loading the table in clustering sequence), then the database manager will keep that index highly clustered when new rows are inserted.

You can see if there is a "clustering" index on the table by executing the following SQL statement:

```
SELECT * FROM SYSTEM.SYSINDEXES  
WHERE TNAME=table_name AND CREATOR=table_creator  
AND (CLUSTER='F' OR CLUSTER = 'W')
```

Either one or no indexes will match this query.

If there is an index matching this query, then you can check its precise degree of clustering from the CLUSTERRATIO column. Refer to "Need a Highly Clustered Index" on page 187 for a discussion on CLUSTERRATIO. Also, if the entry for this index has CLUSTER = 'F' then this is a rough guide that the index is well clustered, whereas if the entry has CLUSTER = 'W' then this is a rough guide that the index is poorly clustered.

If there is no index matching this query, then you have no clustering index.

- If there is no index, then it is the page identified by CLUSTERROW in the table's entry in SYSCATALOG. (This is known as the "default insert rule"). However, the value in CLUSTERROW is used only on the first INSERT of a Logical Unit of Work; for all subsequent INSERTS in the same Logical Unit of Work, the "preferred" page is the one used for the previous INSERT.

In the case where there is at least one index, the cause of the high DASD I/Os is a lack of free space in the vicinity of the "preferred" page.

In the case where there is no index, the cause of the high DASD I/Os is either that the value of CLUSTERROW is out of date (many rows have been inserted since statistics were last updated) or that there are one or more other tables sharing the same DBSPACE and occupying the space starting at the page pointed to by CLUSTERROW.

Possible Actions:

ACTION	PAGE
If there is an index, Re-organize the table with more free space	Below
If there is no index, Re-organize the table in a key sequence and with sufficient free space. Then create a clustering index	Below
If there is no index, move other tables to another DBSPACE and UPDATE STATISTICS	Below
For other High I/O problems, see: - "I/O Related Performance Problems"	113

Figure 78. High I/Os on INSERT - Actions

1. Re-organize the Table with more free space

If the intended clustering index is the first created index, re-cluster the table by using the DBS Utility to UNLOAD and then RELOAD the table. This procedure is described in the *Performance Tuning Handbook*.

If you anticipate a large amount of additional INSERT activity against the table, consider increasing the amount of free space that is reserved on each page as this will increase the amount of time that the index will retain its clustering. Free space is determined by the PCTFREE parameter on the ALTER DBSPACE statement. Set PCTFREE to the desired value just prior to doing the RELOAD and then set it to a low value after the RELOAD has completed.

2. Create a Clustering Index

If you wish to keep the rows of the table in some logical sequence as rows are inserted, then create an index on that sequence. If the table is not already ordered in that sequence, then you will first need to re-organize it. This can be done by creating the new index, UNLOADING and RELOADING, as described in the *Performance Tuning Handbook*.

3. Move other tables to another DBSPACE and UPDATE STATISTICS

If you don't want newly inserted rows to be inserted in any particular logical sequence, then you should arrange that rows from other tables in the same DBSPACE are not inserted into pages beyond the start of the table under discussion. The easiest way to ensure this is not to have any other tables in the same DBSPACE. In addition, you need to update statistics after every so many rows have been inserted.

If the problem is neither lack of free space nor out of date statistics nor interference from other tables in the same DBSPACE, then you should go back to the index on "high I/O" problems and look for another possible cause.

Excessive Locking in User Data

Problem Description:

The database manager locks all data it has to read in order to satisfy a user's request. In particular, more data may be locked than what is specifically identified by the request (in the WHERE clause). As a result, you may experience

"Excessive Locking in User Data" due to the amount of data the database manager had to search in order to satisfy the request.

In many cases, the amount of locking will vary widely depending on which access path is used to find the data. For example, a DBSPACE scan will typically lock more data than an index scan. See "Locking Concepts" on page 55 for more information on locking done by scans.

This excessive locking may not be necessary for the application and in many cases can be avoided. Excessive locking in user data can have several kinds of symptoms, but the most common will be **lock waits**. That is, if an application or request is locking more than it logically needs to, it will probably encounter more lock waits or cause others to encounter lock waits.

Other symptoms that might be experienced are many locks or lock escalations. A large index scan in a DBSPACE with row level locking will generate a lot of lock requests. These can be seen by observing SHOW LOCK GRAPH userid or SHOW LOCK GRAPH agent-number output. These statements show how much locking an application or user is doing. If a user is doing a lot of locking, some of the locks may get escalated.

Escalations can be detected using the COUNTER ESCALATE and COUNTER LOCKLMT statements. COUNTER ESCALATE gives you the count of *successful* escalations, and COUNTER LOCKLMT gives you the count of *unsuccessful* escalations. These tells you if escalations are occurring on your system. You can detect if a particular user is experiencing escalations by looking for S or X locks at the DBSPACE level in the output of SHOW LOCK USER or SHOW LOCK AGENT.

Possible Actions:

ACTION	PAGE
Decrease Lock Level	Below
Increase Lock Level	Below
Index to Avoid DBSPACE Scans	Below
Reorganize DBSPACE to Avoid DBSPACE Scans	Below
Use Cursor Stability Isolation Level	Below
Use Multiple LUWs to Avoid Lock Contention	Below
For other problems with similar symptoms, see: - "NLRB Parameters Too Small" - "Lock Level Too Low" - "Lock Level Too High"	191 179 178
For other lock wait problems, see: - "Locking Related Performance Problems"	114

Figure 79. Excessive Locking in User Data - Actions

There are basically three ways of approaching the "Excessive Locking in User Data" problem in applications:

1. Change the locking level for the data

Changing the locking level on the data being accessed can reduce some of the excessive locking. This can be true of increasing locking level, as well as decreasing locking level.

- Decreasing Locking Level

By reducing the locking level on data from DBSPACE to page, or page to row level locking, you can reduce the amount of data locked by any one row access. However, this probably won't help if your application accesses a lot of rows.

- Increasing Locking Level

Increasing the locking level on the data can help if you are experiencing lock escalations. By increasing row locking to page locking, you might avoid escalations to DBSPACE locks.

If you are already using page locking in the DBSPACE where the locks are being escalated, you might be able to avoid the escalations by using the SQL LOCK TABLE statement. This will be effective if your access to the table is *read only* (that is, SHARE mode).

2. Reorganize data to obtain a better access path

In cases where the excessive locking can be blamed on the use of an inefficient access path to the data, a data reorganization may be called for. That is, locking might be significantly reduced by organizing the data such that a more efficient access path is used.

Generally, this means avoiding DBSPACE scans. A DBSPACE scan will lock (SHARE or EXCLUSIVE, depending on the SQL statement) the entire table. This, of course, may lock a lot more data than intended.

- Indexing to avoid DBSPACE scans

For requests that access a relatively few number of rows (compared to the total number of rows in the table), indexing can be the most effective way to avoid DBSPACE scans.

- Reorganizing DBSPACES to avoid DBSPACE scans

In some cases, indexing alone will not be enough to avoid a DBSPACE scan. If a large number of rows are being accessed, a DBSPACE scan may be chosen over an index. There are organization techniques that can be used to favor an index.

Specifically, if the table being scanned is not too large, you can cause the database manager to "favor" index scans by storing the table in a large DBSPACE with other tables, such that PCTPAGES for the table is a small value (such as 5 or 10%). When PCTPAGES is very small, the estimated I/O's for the DBSPACE scan will look less favorable when compared to the estimated I/O's using an index.

3. Release locks earlier or more frequently

If no way can be found to avoid the excessive locking that is being done, then the next alternative is to try to minimize the locking by releasing the locks as soon as possible. There are two ways this might be done:

- Use of Cursor Stability Isolation Level

The use of the cursor stability isolation level will, in most cases, release SHARE locks before the end of an LUW. If repeatable read capability is not required, the use of cursor stability should be considered.

- Use of Multiple LUWs

Another way of releasing locks earlier is to do COMMIT WORK statements more frequently.

- Use of AUTOCOMMIT ON

For ISQL usage, one way of releasing locks as early as possible is through the use of AUTOCOMMIT ON.

If AUTOCOMMIT OFF is to be used, it is recommended that it be used within routines. This is to avoid long delays while users dynamically respond.

If you are experiencing lock escalations, then you should also review "NLRB Parameters Too Small" on page 191 and "Lock Level Too Low" on page 179. These problems are the primary causes of lock escalations.

If you are *not* experiencing lock escalations, then you might want to consider "Lock Level Too High" on page 178. This problem addresses "excessive locking" where the excess is due to applications that request locks larger than they really need.

Frequent Checkpoints caused by SOSLEVEL

Problem Description:

The percentage of free pages in a storage pool is very close to the SOSLEVEL (short on storage level) defined at initialization. When DBSPACES in the storage pool are updated, SOSLEVEL is reached and a checkpoint is triggered. This can happen repeatedly until the checkpoint no longer frees enough pages to bring the storage pool above the SOSLEVEL. This problem can cause multiple checkpoints in a short period of time, especially during a dataload or an index creation which causes the SOSLEVEL to be reached. It can cause checkpoints to become so frequent that no other work can get done on the application server.

Possible Actions:

ACTION	PAGE
Add a DBEXTENT to the Storage Pool	Below

Figure 80. Frequent Checkpoints caused by SOSLEVEL - Actions

A DBEXTENT could be added to the storage pool to prevent SOSLEVEL from being reached. If this is temporarily inconvenient or impossible, you could decrease SOSLEVEL. However, you should make more space available as soon as possible.

Hot Spot in the Catalog Tables

Problem Description:

"Hot Spot in the Catalog Tables" is a condition that occurs when multiple users (or applications) are accessing the same catalog information in conflicting modes (multiple updaters or multiple readers and one updater). Since the catalogs are heavily used by the database manager and store a wealth of information, conflicts may not be uncommon. The conflicts can also be quite subtle due to the "where used" information kept and maintained. If you have a lock wait problem but not very many locks are being obtained, then it may be a hot spot in the catalog tables.

A hot spot in the catalog tables is a relatively small portion of the catalogs that is frequently referenced by multiple users. Since the catalog DBSPACE is defined with row level locking, a hot spot would typically be a small number of rows, or a hashed key value.

As you might expect, a hot spot in the catalog tables can have a much greater impact than locking on user data. Since the data being referenced by users is concentrated in the catalogs, almost any SQL work can be impacted.

Read access to the catalog tables is unavoidable. The database manager is designed to make heavy use of the catalogs. Update access, however, is something that should be exercised with discretion. The key to understanding the "Hot Spot in the Catalog Tables" problem is knowledge of when and where SQL/DS functions obtain EXCLUSIVE locks in the catalog tables. This is summarized in Figure 81:

FUNCTION	CATALOG	ROWS LOCKED
LOADING (with SET UPDATE STATISTICS ON)	SYSCATALOG SYSCOLSTATS SYSCOLUMNNS SYSDBSPACES SYSFIELDS SYSINDEXES	for table(s) being loaded. for column statistics. for certain columns. for DBSPACE being loaded. for field procedures on columns. for indexes on tables.
PREPROCESSOR	SYSACCESS SYSPROGAUTH SYSUSAGE	to register program. for owner of program. for tables, indexes, views and DBSPACEs used.
SQL AUTH - GRANT - REVOKE	SYSACCESS SYSCOLAUTH SYSPROGAUTH SYSTABAUTH SYSUSERAUTH	if package invalidated. for column update authority on individual columns. if RUN authority. for tables authorizations. for special authorities.
SQL DDL - ACQUIRE - ALTER - CREATE - DROP	SYSDBSPACES SYSCATALOG SYSCOLUMNNS SYSINDEXES SYSVIEWS SYSTABAUTH SYSCOLAUTH SYSUSAGE SYSACCESS SYSKEYS SYSKEYCOLS	for the DBSPACE acquired or altered. on create, for table or view created. on drop, for table or view dropped. to verify CCSID column attributes on a table. on alter, for table altered/dropped. columns for table or view created or dropped. columns of dependent views dropped. on alter, row for column added. for index created/dropped. for view created/dropped. row(s) for grant. revoke "cascades." revoke of colauth on drop of view or table. for any dependency established or dropped. for program or view being created or dropped. for invalidated programs when dependent object dropped. on create/drop, for activate or deactivate primary key, foreign key or unique constraint on create/drop, for activate or deactivate primary key, foreign key or unique constraint

Figure 81 (Part 1 of 2). Exclusive Locking in Catalog Tables

FUNCTION	CATALOG	ROWS LOCKED
UPDATE STATISTICS	SYSDBSACES SYSCATALOG SYSCOLSTATS SYSCOLUMNS SYSFIELDS SYSINDEXES	for DBSPACE involved. for table(s) involved. for column statistics. for certain columns. for field procedures on columns. for indexes on tables.

Figure 81 (Part 2 of 2). Exclusive Locking in Catalog Tables

Notes on Figure 81:

1. Loading

Loading user tables does *not* do any exclusive locking in the catalog tables, unless the load is done with STATISTICS set on. If statistics are updated as part of the load, then the SYSCATALOG row (or rows) for the table (or tables) being loaded are locked exclusively to update the table statistics. Furthermore, NACTIVE in SYSDBSACES is updated for the DBSPACE(s) being loaded.

2. Preprocessing

Preprocessing does exclusive locking in SYSACCESS to record the package created. An exclusive lock is also obtained on a row (and keys) in SYSPROGAUTH to record the fact that the creator of the program is authorized to run the program.

Multiple exclusive locks will typically be obtained in SYSUSAGE to record the dependencies the program has on tables, views, indexes and DBSPACEs. If you are experiencing lock contention with preprocessor executions, one of the places you want to look is in SYSUSAGE. Concurrent preprocessing of programs that access the same data can result in conflicts in SYSUSAGE.

3. SQL Authorization (GRANT/REVOKE)

GRANT and REVOKE do exclusive locking in several tables, but they are typically short operations. Serious conflicts due to GRANT and REVOKE are not likely. However, they can get caught in a lock wait behind long running operations such as preprocessing. When this happens, other users (such as users of data manipulation statements) may see an impact as authorization checks are attempted.

Another consideration on authorization statements are "cascading REVOKE operations." If your installation makes heavy use of the GRANT OPTION, the exclusive locking done on REVOKE statements can be quite extensive, as the REVOKE is propagated to other users.

4. SQL DDL

Naturally, data definition statements do a lot of exclusive locking in the catalogs. Most data definition statements have relatively short execution times and should not be a problem. However, CREATE INDEX and DROP TABLE are exceptions. They can have rather long execution times.

Note: Adding or activating a primary key, foreign key or unique constraint requires a CREATE INDEX operation.

Another factor in data definition locking are the implicit operations that typically will occur. For example, on a CREATE TABLE operation, entries are made in SYSTABAUTH as well as SYSCATALOG and SYSCOLUMNS.

Perhaps the most extensive locking will occur on DROP statements. The "change propagation" that occurs on DROP statements can be quite extensive. For example, dropping a table will typically cause entries to be deleted from SYSACCESS, SYSCATALOG, SYSCOLAUTH, SYSCOLSTATS, SYSCOLUMNS, SYSINDEXES, SYSKEYCOLS, SYSKEYS, SYSTABAUTH, and SYSVIEWS.

5. UPDATE STATISTICS

The amount of exclusive locking done by UPDATE STATISTICS will vary depending on the options used and whether or not indexes exist. If an index exists, some amount of exclusive locking will be done in *all* the catalog tables shown in Figure 81. If the ALL option is specified, all the SYSCOLUMNS rows for the table(s) will be updated (locked exclusive). If the statistics are being updated for a whole DBSPACE, then multiple SYSCATALOG rows will be locked, as well as the corresponding rows in SYSINDEXES and SYSCOLUMNS.

Also note that UPDATE STATISTICS can be a long running operation. That is, locking done by UPDATE STATISTICS can be held for a long time.

As you can see, the potential for locking conflicts in the catalog DBSPACE is quite impressive. However, the situation is better than you might think. If you are careful about the use of long running operations that get exclusive locks in the catalog tables, you should not experience too many problems.

However, there are some considerations that you should make when using *any* of the functions that do exclusive locking in the catalog tables:

1. Avoid long running LUWs

Even if you are performing a "short" operation that does exclusive locking in the catalogs, the effect can be significant if the function is performed at the start of a long running LUW. For example, doing CREATE functions from ISQL with AUTOCOMMIT set OFF has the potential of holding up other users quite noticeably. The user doing the CREATE statements may see very good response time. But other users may get stuck in a lock wait until the "creator" commits the LUW.

2. Beware of Catalog Queries

While it is convenient to be able to query the catalog tables, discretion should be used. Queries on the catalogs acquire SHARE locks on the data queried, and can hold up the operations shown in Figure 81.

Use of the cursor stability isolation level or running such queries at discrete times during the day is recommended.

Possible Actions:

ACTION	PAGE
Run Offending Jobs Off-hours	Below
Re-design Offending Applications	Below
You should also investigate: - "Conflict in Catalog Key Locking"	134

Figure 82. Hot Spot in the Catalog Tables - Actions

Since you cannot change the implementation of the SQL/DS catalogs, your options for resolving a "Hot Spot in the Catalog Tables" problem are limited. There are basically three approaches that you can try:

1. Run Offending Jobs Off-hours

The most effective approach to addressing the problem is to reschedule the offending jobs. That is, you would identify the jobs that are causing the most lock waits in the catalog tables and run them at a time when they are less likely to conflict with other work.

Rescheduling of offending jobs is probably the most effective solution for conflicts due to:

- a. Preprocessing jobs
- b. SQL Authorization Statements
- c. SQL Data Definition Statements
- d. Large Catalog Queries

2. Re-design Offending Applications

If rescheduling of the offending jobs is not attractive for your situation, then you might be able to do some redesign of your applications such that conflicts are less of a problem. Some of the possible changes are quite easy to make and can be very effective. Consider the following possibilities:

a. Use Cursor Stability on Catalog Queries

Since catalog queries can block functions that need to update the catalogs, the use of the cursor stability isolation level is recommended for user queries on the catalogs.

b. Load Jobs

Running DATALOAD jobs with STATISTICS set OFF, and postponing the updating of statistics (via an explicit UPDATE STATISTICS) to a later time is more costly due to the DBSPACE scan.

Catalog statistics are automatically accumulated during the DATALOAD (and RELOAD) command when UPDATE STATISTICS is set ON. This avoids the DBSPACE scan performed by an explicit UPDATE STATISTICS.

c. UPDATE STATISTICS

You might consider restricting the statistics you update during peak hours. That is, avoid the DBSPACE or ALL options during peak hours.

Note: Similar symptoms occur on adjacent key locking or key hash conflicts in the catalog tables. See "Conflict in Catalog Key Locking" on page 134 for more information.

Hot Spot in User Tables

Problem Description:

"Hot Spot in User Tables" is a condition that can occur when applications are locking at a reasonable level (row or page), but they conflict because they are going after the *same* data (row or page).

If response time is long or erratic in multiple user mode but satisfactory when only one user is connected to an application server, then it is probably a lock wait problem. If you have a lock wait problem but not very many locks are being obtained, then it may be a hot spot in user tables.

A hot spot in user tables is a relatively small portion of a table or DBSPACE, or a small table, that is frequently referenced by multiple users. A hot spot can be a data page, a small number of rows, or a hashed key value.

A hot spot in user tables will behave similar to a high lock level, but it can occur with page or row level locking. Since the data being referenced by users is concentrated in one spot in the database, locking a "popular" page, row or key can be equivalent to locking the whole DBSPACE.

A hot data page is most likely to show up with page level locking. If you are already using row level locking, then it may be a hot set of rows or a hot key hash.

If the table being accessed is a small table (low ROWCOUNT), then it may be a hot spot problem. However, a hot spot can also occur on large tables (high ROWCOUNT), if the table has a small number of very popular rows, or a small, very popular key range.

If many different rows and pages are being accessed (no hot rows or pages), then the problem could be a hot key range, and you need to investigate the indexing on the tables in question. EXCLUSIVE locks on keys or index pages are obtained on INSERT and DELETE operations. They are also obtained on UPDATE operations when a column being updated is part of the index key. If ROWCOUNT divided by FULLKEYCOUNT is a large value, then the problem could very well be locking on the index keys or index pages.

Note: Similar symptoms occur on adjacent key locking or key hash conflicts. See "Adjacent Key Locking in User Data" on page 115 and "Conflict on Key Hash in User Data" on page 136 for more information.

Possible Actions:

ACTION	PAGE
Decrease Lock Level	Below
Change Key Structures	Below
Use Redundant Data	Below
For problems with similar symptoms, see: - "Adjacent Key Locking in User Data"	115
- "Conflict on Key Hash in User Data"	136
For <i>catalog</i> locking problems, see: - "Hot Spot in the Catalog Tables"	153
- "Conflict in Catalog Key Locking"	134
For other lock wait problems, see: - "Locking Related Performance Problems"	114

Figure 83. Hot Spot in User Tables - Actions

There are basically three approaches to resolving a "Hot Spot in User Tables" problem:

1. Decrease Lock Level

If the DBSPACE is not already defined to have row level locking, altering the DBSPACE to row level locking is probably the simplest action to take. However, this may not be appropriate if the users and applications access a lot of rows. Using row level locking when many rows are accessed can result in lock escalations.

2. Re-design index keys

If a "hot key range" is indicated, you may want to re-design the indexes such that FULLKEYCOUNT is closer to ROWCOUNT. If the current indexes are needed, then it may be sufficient to add columns to the index definition to achieve a higher FULLKEYCOUNT.

3. Re-design tables

If the above solutions do not work, consider using redundant data. Specifically, if update access to the data is conflicting with read access to the data, separate copies of the data may be needed, one for the updaters for dynamic maintenance and the other for readers. The read only copy would have to be periodically refreshed from the updater's copy.

You might also want to investigate the other problems that have similar symptoms. In particular, if the locking conflicts are in the indexes, see "Adjacent Key Locking in User Data" on page 115 and "Conflict on Key Hash In User Data" on page 136.

Furthermore, if the locking conflicts are in the *catalog* DBSPACE (DBSPACE 1), then you should be investigating "Hot Spot in the Catalog Tables" on page 153 and "Conflict in Catalog Key Locking" on page 134.

I/O Capacity Exceeded

Problem Description: The database is not spread across enough channels and/or devices. This condition is verified by examining the reports generated from VM monitor data. For VM/SP and the VM/ESA 370 Feature, the *VM/370 Performance Monitor Analysis Program* (VMMAP) generates reports called "Channel Activity Summary" and "Disk and Tape I/O Summary" that should be examined. For VM/XA and VM/ESA ESA Feature, the *VM Performance Reporting Facility* (VMPRF) generates reports called "Channel Busy" and "DASD By Activity" that should be examined.

Insufficient channels is indicated if the utilization of each channel supporting the database exceeds 30 percent. If some are high and some are low, see "I/O Not Balanced" on page 161.

Insufficient DASD actuators is indicated if the utilization of each device supporting the database exceeds 60 percent. If some are high and some are low, see "I/O Not Balanced" on page 161.

Possible Actions:

ACTION	PAGE
Increase the Number of Buffers	Below
Perform Tuning to Achieve More Efficient Access	Below
Identify the Usage that Generates the Most I/O	Below
Eliminate Contention due to non-SQL/DS I/O	Below
Add More Channels/Devices	Below
If some utilizations are low, see: "I/O Not Balanced"	161
For mostly Read Only data, see: "Data Not Cached"	139

Figure 84. I/O Capacity Exceeded - Actions

1. Increase the Number of Buffers

Consider increasing the size of the buffer pools. This is a good first step since it is easy to try and will sometimes cause a large reduction in database I/O. See "Buffer Pool Too Small" on page 128 for a discussion of the factors you should consider before doing so.

2. Perform Tuning to Achieve More Efficient Access

Review how the database manager is being used in your installation to see if there are tuning steps that can be taken that would allow the requested data to be accessed more efficiently. See "I/O Related Performance Problems" on page 113.

3. Identify the Usage that Generates the Most I/O

See if most of the database I/O is being generated by one or two applications. The count of looks in the page buffer provided in the SQL/DS user accounting records can be useful for this purpose. If so, see if this work can be done more efficiently or at a different time.

4. Eliminate Contention due to non-SQL/DS I/O

Check to see if the problem is being caused by accesses to non-SQL/DS data that happens to be stored on the same channels and/or devices that contain the database. If this is the case, consider moving the non-SQL/DS data elsewhere.

5. Add More Channels/Devices

Increase the capacity of the I/O subsystem used to support the database. If there are insufficient channels, move some of the devices to one or more additional channels. If there are insufficient devices, move some of the data to one or more additional DASD actuators. You can do this either by moving tables/DBSPACES or by moving or copying DBEXTENTS. Moving tables/DBSPACES is discussed under "Maintaining your Database" in the *Database Administration* manual. Moving and copying DBEXTENTS is discussed in the *System Administration* manual.

I/O Not Balanced

Problem Description:

Database I/O activity is not evenly balanced across the channels and/or devices that are being used to support the database. This condition is verified by examining the reports generated from VM monitor data. For VM/SP and VM/ESA 370 Feature, the *VM/370 Performance Monitor Analysis Program (VMMAP)* generates reports called "Channel Activity Summary" and "Disk and Tape I/O Summary" that should be examined. For VM/XA and VM/ESA ESA Feature, the *VM Performance Reporting Facility (VMPRF)* generates reports called "Channel Busy" and "DASD By Activity" that should be examined.

Unbalanced channel usage is indicated if the utilization of one or more of the channels supporting the database exceeds 30 percent, while the utilizations of the remaining channels are lower. If they all exceed 30 percent, see "I/O Capacity Exceeded" on page 160.

Unbalanced device usage is indicated if the utilization of one or more of the DASD supporting the database exceeds 60 percent, while the utilizations of the remaining devices are lower. If they all exceed 60 percent, see "I/O Capacity Exceeded" on page 160.

Possible Actions:

ACTION	PAGE
Look for Ways to Reduce the I/O Rate	Below
Balance Channel Usage	Below
Balance Device Usage	Below
If all utilizations are high, see: "I/O Capacity Exceeded"	160

Figure 85. I/O Not Balanced - Actions

1. Look for Ways to Reduce the I/O Rate

Before trying to redistribute the data, consider looking for ways to reduce either the overall I/O rate or the I/O rate associated with the overloaded devices/channels. See "I/O Related Performance Problems" on page 113.

2. Balance Channel Usage

To balance channel usage, first study the VM MAP or VM PRF device usage report to determine what grouping of devices would result in a rough balance of I/O load across the available channels. Then either, physically move the device to another channel, or, copy its contents to a device on another channel. Copying the contents can be achieved by moving the DBEXTENT. For details on moving DBEXTENTS, see the *System Administration* manual.

3. Balance Device Usage

First determine what DBEXTENTS reside on each of the devices that show a utilization in excess of 60 percent.

Then, for each such device, determine how much each DBEXTENT contributes to the overall I/O load on that device. This can be done by collecting VM monitor data with the SEEK class enabled and reducing the data using VM MAP. The seek report provided by VM MAP will show the distribution of I/Os across the database minidisks. On VM/XA or VM/ESA ESA Feature systems, the *VM Performance Reporting Facility* (VM PRF) provides the same function as VM MAP.

Use this information to make and carry out a plan to add or copy one or more DBEXTENTS to, or delete from one or more DBEXTENTS to the low-usage devices. The procedure for doing this is described under "Managing Storage Pools" in the the *System Administration* manual.

Inaccurate Statistics

Problem Description:

If the statistics in the SQL/DS catalogs do not accurately reflect the actual characteristics of the data, the SQL/DS optimizer may choose an inefficient access path. This will not cause the SQL request to fail, but the response time experienced may be longer than expected. In fact, ISQL query users may notice strange query cost estimates for the queries. Queries with small query cost estimates may take a relatively long time.

Note: You can also have cases where the query cost estimate is quite large, but the query runs relatively fast.

Inaccurate statistics might exist if:

1. Statistics have never been generated ¹ for the data, or
2. The data has changed significantly since the statistics were last updated.²

¹ Statistics can be updated by the UPDATE STATISTICS statement, or, as a result of a DBSU DATALOAD or RELOAD command with UPDATE STATISTICS set ON.

² Some statistics are generated when the CREATE INDEX statement is processed.

If UPDATE STATISTICS has never been run on the table, the database manager will use default values for the statistics. For the table in question, these values may not come close to the statistics for the actual data. This, of course, can result in a very bad choice of access path.

A similar result can occur if the statistics in the catalog become out of date. This can occur if the table had a lot of INSERT, UPDATE and/or DELETE activity since the last time UPDATE STATISTICS was run.

Catalog statistics are relevant only to the SQL DML statements (and not data definition or authorization statements). Other SQL statements are not subjected to access path selection. Thus, you will experience the "Inaccurate Statistics" problem only on the DML statements. This is true of DML statements embedded in application programs, as well as those issued dynamically (through ISQL or the DBS Utility).

With inaccurate statistics, there will be little difference between response times in single user environments and multiple user environments. The statistics will be just as inaccurate for either case.

One symptom you might observe is that the query cost estimate (QCE) for queries on the table in question may appear to be unrealistic because the QCE is based on out-of-date statistics.

Another problem that has symptoms similar to "Inaccurate Statistics" is the "Bad Data Distribution" problem. With the "Bad Data Distribution" problem, the statistics can exist and be up-to-date, but are misleading because of peculiarities in the data. See "Bad Data Distribution" on page 122 for more information on this problem.

Possible Actions:

ACTION	PAGE
Update Data Statistics	Below
For other problems with similar symptoms, see: - "Bad Data Distribution"	122
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 86. Inaccurate Statistics - Actions

There is basically only one action that can be taken for the "Inaccurate Statistics" problem. That is to update the statistics for the problem table(s). However, there are variations that you may want to consider. You need to decide whether or not you need to update the statistics for only the table, or to update the statistics for *all* the tables in the DBSPACE. Furthermore, you may also want to specify the ALL option to update the statistics for all the columns in the table. The application must be reprocessed for the updated statistics to be used. See "Package Needs Re-preprocessing" on page 194 for more information.

If your statistics are up-to-date and you still have the same (or similar) symptoms, then you may have the "Bad Data Distribution" problem. You should review that problem (on page 122) next.

If the problem is neither inaccurate statistics nor bad data distributions, then you should go back to the index on "high I/O" problems and look for another possible cause.

Index Disqualified

Problem Description:

An index was either completely disqualified as a possible way to access the table (Case 1 below), or it was disqualified as a means of gaining selective access to the table (all remaining cases).

1. The index was disqualified as a possible way to access the table because one of its columns is being updated by this SQL statement, or by another SQL statement based upon cursor position provided by this SQL statement. However, the index is not disqualified if the SET clause of the UPDATE statement has the form "column = value," and there is a predicate in the WHERE clause which identifies a particular index key value. For example, the statement "UPDATE TABLE1 SET C1 = 125 WHERE C1 = 100" would be able to use an index on C1.
2. The index could not be used to selectively access the table because the data type of the value in the predicate could not be converted to the data type of the indexed column.

Note: Such a predicate is also not eligible as a Database Storage Subsystem (DBSS) search argument (SARG).

For the numeric data types, the conversions that can be done are summarized by the following diagram:

SMALLINT-->INTEGER-->DECIMAL-->FLOAT

A value's data type can be converted into any of the data types that lie to its right. For example, INTEGER can be converted into DECIMAL or FLOAT, but not SMALLINT.

If the data type of the column is CHAR(n) or GRAPHIC(n), that column is eligible for use with an index if the length of the predicate value is less than or equal to "n".

If the data type of the column is VARCHAR(n) or VARGRAPHIC(n), that column is eligible for use with an index if the predicate value is any character data type (fixed or variable, any length).

A join predicate (for example, column1=column2), is eligible only for use with an index if the data types of the two columns are identical (except for whether the columns support NULLS):

- In the case of CHAR(n), VARCHAR(n≤254), GRAPHIC(n), and VARGRAPHIC(n≤127), the lengths must match.
 - In the case of DECIMAL (m,n), precision and scale must both match.
3. An index was not used to selectively access the table because OR was used in the WHERE clause.

4. A multicolumn index was not used to selectively access the table because the selective column specified in the predicate was not first in the index key.
5. The database manager does not provide index support for the predicate as written. A predicate can generally be written in two or more equivalent ways. For example, you could write "BALANCE + 100 = 1000" or "BALANCE = 900." Only the second case is eligible for selective index access consideration, ("submitted key = value"). In the first case, a nonselective index scan is used.

Possible Actions:

ACTION	PAGE
Replace UPDATE with DELETE/INSERT Create Another Index that Excludes Updated Column Redesign so that the Updated Data is Not Indexed	Below
Use Compatible Data Types	Below
Use IN or UNION instead of OR	Below
Create Another Index with Selective Column First	Below
Write Index-Eligible Predicates	Below
See the following closely related problems: "Inefficient Search" "No Selective Index"	170 192
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 87. Index Disqualified - Actions

1. Index key column is being updated:

- Replace UPDATE with DELETE/INSERT

If you need to update the column corresponding to the most selective index into the table, it may be more efficient to delete and then reinsert the row instead of updating it. The reason for this is that an index on an updated column may be ineligible for use in accessing the table, whereas no such restriction applies to the INSERT and DELETE SQL statements.

The index will be eligible if the SET clause of the UPDATE statement has the form "column = value," and there is a predicate in the WHERE clause which identifies a particular index key value. For example, the statement "UPDATE TABLE1 SET C1 = 125 WHERE C1 = 100" would be able to use an index on C1.

- Create Another Index that Excludes Updated Column

Take a look at the WHERE clause of the UPDATE statement and see if selective access to the table could be achieved by creating another index on a different column.

In the case of a multicolumn index where the updated column is not the first column in the index key, reconsider whether the index has to include the updated column. If it doesn't, drop and re-create the index excluding the updated column. If it does, consider creating an additional index that is just on the first column. Such an index could be used to get efficient table access when the column that participates in the multicolumn index is being updated.

- Redesign so that the Updated Column is Not Indexed

Try to redesign the table in such a way that the data that has to be updated is in a different, unindexed column. For example, it may be the case that only the right-hand portion of the column is ever updated. In that case, it may be possible to split the column into two equivalent columns and index only the column derived from the left-most portion.

2. Use Compatible Data Types

When writing an SQL/DS application, make sure that the data type of each program variable is compatible with the data type of the column it is associated with. The index compatibility requirements are provided above in the *Problem Description*. For best performance, data types and lengths should match exactly.

3. Use IN or UNION instead of OR

Rewrite the SQL statement using IN or UNION, as applicable. See Point 2 on page 172 for further information.

4. Create Another Index with Selective Column First

Consider creating an index on the column that appears in the predicate. This is also a good time to rejustify the multicolumn index. Perhaps there only needs to be an index on the first column.

5. Write Index-Eligible Predicates

Write predicates in index-eligible form whenever possible. A predicate must be in one of the following forms to be eligible for use with an index to provide selective access to a table:

```
colname op value  
("op" is "<", "<=", "=", ">=", or ">")
```

```
colname1 op colname2  
("op" is "<", "<=", "=", ">=", or ">")
```

This case applies only to joins. Either colname1 or colname2 will be considered for use with an index, depending on the join sequence chosen by the optimizer.

```
colname BETWEEN value1 AND value2
```

```
colname IS NULL
```

```
colname IN (value1, value2, ...)
```

```
colname LIKE value  
(if value does not start with "_" or "%")
```

Index Maintenance

Problem Description:

You can expect execution times for data maintenance operations to be noticeably longer if the target table has several indexes defined on it. This is particularly true for DATALOAD, bulk INSERT or bulk DELETE operations. However, it is also possible with bulk UPDATE operations when the column(s) being updated occur in one or more indexes.

DATALOAD execution time can be expected to be much longer if the target table has one or more indexes defined on it. For each row inserted into the table, index maintenance has to be done for each index on that table. Consequently, the more indexes, the larger the effect.

Possible Actions:

ACTION	PAGE
Drop Some Indexes	Below
Create Indexes after Load	Below
DROP/Re-CREATE INDEX	Below
REORGANIZE INDEX	169
Increase the Number of Buffers	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 88. Index Maintenance - Actions

1. Drop Some Indexes

First, evaluate the indexes on the table and eliminate any indexes that are not of significant value. Most very nonunique indexes can be removed because they cannot provide very selective access to the table anyway. Indexes on heavily updated columns should also be avoided.

2. Create Indexes after Load

For DATALOAD or bulk INSERT activity, it is preferable to create indexes *after* rows are loaded/inserted. Two exceptions: 1) the number of rows being loaded is much smaller than the number of rows already in the table, and 2) the table contains a small number of very large rows (for example, 10,000 bytes).

3. DROP/Re-CREATE INDEX

The most common reason for wanting to do DATALOADs against an indexed target table is that new batches of data periodically need to be added to an existing table.

If the table is not too large, or the percentage of the dataload to the overall table small, it is faster to drop the indexes on that table, load the new data and then re-create the indexes. An advantage of this approach, is that when the index has been recreated, the index tree is balanced, however, packages will need to be reprocessed. Similarly, for bulk DELETE operations, it may be more efficient to DROP indexes before the DELETE, and recreate

them after the DELETE. This is certainly true when all the rows of the table are being deleted. The same approach can be used with bulk INSERT and DATALOADs, if the number of rows in the existing table is smaller than the number of rows being loaded/inserted.

4. Increase the Number of Buffers

If none of the above actions can be employed, consider increasing the size of the buffer pool. This will keep many of the index pages in virtual storage, thus minimizing database I/O to the indexes.

Index No Longer Highly Clustered

Problem Description:

A query involving a sequential scan of some significant number of rows (for example, more than 20) is being performed inefficiently, for example, by a DBSPACE scan, because the degree of clustering of a previously highly clustered index dropped so low that SQL/DS no longer considers it to be highly clustered. The database manager calculated the degree of clustering the last time UPDATE STATISTICS was explicitly run for this table, or when it was implicitly run while data was being loaded (using the DBSU DATALOAD with UPDATE STATISTICS SET ON, or the DBSU RELOAD command).

We say that an index is highly clustered if the sequence in which the table's rows are stored in pages in the database corresponds closely to the index key sequence. Whether or not an index is highly clustered does not have a significant effect on performance if the number of rows examined by scanning the table with that index is small (for example, less than 20). In that case a drop in the degree of clustering of an index is probably not the problem.

If a table does not have a highly clustered index, access to that table will be less efficient for sequential scans because the database manager must access the table using a DBSPACE scan or using an index scan via an unclustered (or only slightly clustered) index. In the case of a DBSPACE scan, all active pages in the DBSPACE are examined, not just those that contain rows of the desired table. In the case of an unclustered index scan, each row examined will often require another I/O to the database.

This problem can arise either because you never created a highly clustered index or because a highly clustered index became less clustered. The former problem is covered in "Need a Highly Clustered Index" on page 187; this section deals with the latter. Note however that the symptoms are the same.

Information as to whether or not a given index is highly clustered is maintained in the CLUSTERRATIO column of the row corresponding to that index in the SYSTEM.SYSINDEXES catalog table. For more information on CLUSTERRATIO refer to the following:

- discussion in "Need a Highly Clustered Index" on page 187
- the *Performance Tuning Handbook*.

You can determine the current clustering status of all the indexes on a table by issuing the following SQL statement:

```
SELECT INAME,ICREATOR,CLUSTERRATIO
FROM SYSTEM.SYSINDEXES
WHERE TNAME=table_name AND CREATOR=table_creator
```

The result will show you if the index you intended to be highly clustered is currently considered to be so.

Possible Actions:

ACTION	PAGE
Make the Index you want Highly Clustered the First Created Index	Below
Re-cluster the Table	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 89. Index No Longer highly clustered - Actions

A table can be loaded in any order you choose, but any given table loading has only one ordering associated with it. As a result, you can force only one of the table's indexes to be highly clustered. Occasionally, one (or more) of the other indexes on a table will also be highly clustered, but this is fortuitous and cannot be directly controlled.

1. Make the Index you want Highly Clustered the First Created Index

The index you intend to be highly clustered should always be the first index created on that table. This is because the database manager always tries to maintain clustering of the first created index. This index is known as the "clustering" index. Use the query shown in the problem description for Excessive I/Os on Insert on page 148 to make sure that the index you intended to be highly clustered is the first created index. If not, make it the first created index. The procedure for doing so is described in the *Performance Tuning Handbook*.

2. Re-cluster the Table

If the intended highly clustered index is the first created index, re-cluster the table by using the DBS Utility to UNLOAD and then RELOAD the table. This procedure is described in the *Performance Tuning Handbook*.

If you anticipate a large amount of additional INSERT activity against the table, consider increasing the amount of free space that is reserved on each page as this will increase the amount of time that the index will retain its high degree of clustering. Free space is determined by the PCTFREE parameter on the ALTER DBSPACE statement. Set PCTFREE to the desired value just prior to doing the RELOAD and then set it to a low value after the RELOAD has completed.

Indexes Are Fragmented

Problem Description:

Extensive modifications to a table have fragmented its indexes, resulting in increased I/O and associated processing overhead when those indexes are used. This condition should be suspected if there has been a large amount of data modification activity on the table since the time its indexes were created.

Keep in mind that any index can become fragmented, including the indexes on the system catalog tables. Catalog table INSERTs occur implicitly as part of

ALTER TABLE, CREATE INDEX, CREATE TABLE, CREATE VIEW, CREATE SYNONYM, GRANT, preprocessing a new program, and re-preprocessing a program. Catalog table DELETES occur implicitly as part of DROP INDEX, DROP PROGRAM, DROP TABLE, DROP VIEW, DROP SYNONYM, REVOKE, and re-preprocessing a program.

Use the SHOW DBSPACE operator command to determine whether or not a DBSPACE needs to be reorganized.

Possible Actions:

ACTION	PAGE
REORGANIZE INDEX	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 90. Indexes Are Fragmented - Actions

1. REORGANIZE INDEX

Reorganize the index using the DBSU REORGANIZE INDEX command. This command performs faster than a DROP/Re-CREATE because neither a DBSPACE scan nor a sort is performed.

An additional advantage of this feature is that packages do not need to be reprocessed.

In the case of the system catalog tables, use the catalog index reorganization utility SQLCIREO. For more information, see the *Database Administration* manual.

Inefficient Search

Problem Description:

1. The WHERE clause is not restrictive enough, resulting in a very large answer set.
2. The OR connector is used in the WHERE clause. When this is done, all predicates become ineligible for use in conjunction with an index to achieve selective access to the table.

Note: The above rule applies only to the outermost grouping of predicates. For example, all predicates in the following query are ineligible for use with an index:

```
SELECT * FROM PROJ_ACT
WHERE PROJNO='AD3100'
AND ACTNO=60 OR ACTNO=70
```

By way of contrast, in the following similar (but not equivalent) query "PROJNO = AD3100" is eligible for use with an index, while the other two predicates are not:

```
SELECT * FROM PROJ_ACT
WHERE PROJNO='AD3100'
AND (ACTNO=60 OR ACTNO=70)
```

3. Nullable expressions may cause quantified predicates to be processed in an inefficient way.
4. Even precision decimal can cause predicates to be processed in an inefficient way.
5. The SQL statement contains a subquery. This may force SQL/DS to satisfy the query in an inefficient way.
6. Host variables are used for values in one or more predicates in the WHERE clause. This problem applies only to precompiled applications.

If one or more of the predicate operators are BETWEEN or an inequality, the database manager may choose a suboptimal way to access the table. This is because the database manager normally uses linear interpolation to estimate how many of the table's rows will satisfy these types of predicates. If, however, the comparison value is a host variable, the actual value is not known at precompilation time so the database manager is forced to make a much less accurate estimate based on default rules. See "Range Predicate Used With Host Variables" on page 199.

7. The NOT modifier is used in conjunction with a predicate in the WHERE clause. When this is done, the predicate must be evaluated at the RDS level, resulting in increased processing overhead.
8. An indicator variable is used with a predicate in the WHERE or HAVING clause. When this is done, the predicate must be evaluated at the RDS level, resulting in increased processing overhead.

Possible Actions:

ACTION	PAGE
Use More ANDED Predicates	Below
Use IN or UNION	Below
Avoid Nullable Expressions in Quantified Predicates	Below
Avoid Even Precision Decimal in Predicates	Below
Formulate as a Join	Below
Use Dynamic Statements	Below
Use Negative Form of Predicate	Below
Avoid Indicator Variables on Predicates	Below
See the following closely related topics:	
"Insufficient Indexing"	174
"No Selective Index"	192
"Index Disqualified"	164
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 91. Inefficient Search - Actions

1. Use More ANDED Predicates
 Put in *all* valid predicates, even if they are not needed to give the desired answer set.

Reconsider what information you really need. Then see if you can further qualify the WHERE clause by ANDing it with additional predicates.

Instead of giving end users access to an entire table, provide them with a view on just the portion of the table that they need. This has the effect of adding the predicates provided in the view definition to the predicates that appear in the queries formulated by the users.

2. Use IN or UNION

Consider the following alternatives to the use of OR:

- If all predicates connected by OR are for the same column, substitute the equivalent IN predicate. The database manager will then be able to use an index on the IN column to selectively access just those rows that satisfy the values in the IN list. For example, instead of:

```
SELECT * FROM PROJ_ACT
WHERE PROJNO='AD3100'
AND (ACTNO=60 OR ACTNO=70)
```

write:

```
SELECT * FROM PROJ_ACT
WHERE PROJNO='AD3100'
AND ACTNO IN (60,70)
```

Note: When the predicate columns are the same, the IN formulation is generally the best, but the UNION approach (see below) is also applicable and may occasionally give better performance.

- If one or more of the predicates connected by OR are for a different column, consider rewriting the query as the UNION of two or more SELECTs. For example, instead of:

```
SELECT * FROM EMPLOYEES
WHERE JOB='CLERK'
OR NAME='JONES'
```

write:

```
SELECT * FROM EMPLOYEES WHERE JOB='CLERK'
UNION
SELECT * FROM EMPLOYEES WHERE NAME='JONES'
```

3. Avoid Nullable Expressions in Quantified Predicates

To avoid this case, consider either of the following:

- Define the column to be NOT NULL
- Rewrite the query to avoid the use of the nullable expression. For example, instead of:

```
C1 + 5 = 10
```

write:

```
C1 = 5
```

4. Avoid Even Precision Decimal in Predicates

Some application languages such as Assembler do not support even precision decimal. When such table columns are referenced in a predicate con-

taining a comparative host variable in these applications, the host variable must be declared with a precision one higher than the column. As a result, you get inefficient processing because the predicate is residual.

Redefine the table columns to odd numbered precision to avoid this situation.

5. Formulate as a Join

In general, correlated subqueries will perform better than subqueries without correlation. Furthermore, in many cases, a join can be used in place of either. Joins will typically outperform either correlated or uncorrelated subqueries.

For example, assume you want to find activity numbers and their descriptions for all projects (in the PROJ_ACT table). The natural way to express this in SQL is:

```
SELECT ACTNO, ACTDESC
FROM SQLDBA.ACTIVITY
WHERE ACTNO IN (SELECT ACTNO
                FROM SQLDBA.PROJ_ACT)
```

However, the correlated subquery version of this query would usually run faster:

```
SELECT ACTNO, ACTDESC
FROM SQLDBA.ACTIVITY A
WHERE ACTNO IN (SELECT ACTNO
                FROM SQLDBA.PROJ_ACT P
                WHERE P.ACTNO=A.ACTNO)
```

Furthermore, you would normally get the best performance by expressing the query using a join:

```
SELECT DISTINCT A.ACTNO, ACTDESC
FROM SQLDBA.ACTIVITY A,SQLDBA.PROJ_ACT P
WHERE P.ACTNO=A.ACTNO
```

6. Use Dynamic Statements

Instead of using host variables, execute the SQL statement dynamically with fixed values.

7. Use Negative Form of Predicate

Whenever possible, use the negative form of a predicate rather than negating it with the NOT modifier. For example:

- Use "colname < > or \neq = value" rather than "NOT colname = value."
- Use "colname < = value" rather than "NOT colname > value."
- Use "colname < value1 OR colname > value2" rather than "NOT BETWEEN value1 AND value2."

8. Avoid Indicator Variables on Predicates

Do not use an indicator variable with a predicate unless it is specifically necessary.

Inefficient SELECT List

Problem Description:

1. The data types of one or more program variables do not match the data types of the corresponding columns specified in the select list of a SELECT statement. As a result, the database manager has to do extra processing to perform the data conversions.
2. The select list includes more columns than are actually needed. As a result, extra processing must be done to return the additional data.

Possible Actions:

ACTION	PAGE
Use Same Data Types	Below
Select Only Required Columns	Below

Figure 92. Inefficient SELECT LIST - Actions

1. Use Same Data Types

To avoid data conversion processing, make sure that the data types of the program variables and the corresponding columns in the select list match.

2. Select Only Required Columns

Make sure that the select list includes only those columns that will actually be used. In particular, avoid overuse of the "SELECT *" notation.

Insufficient Indexing

Problem Description:

1. There are no indexes on the table. As a result, the database manager is forced to use a DBSPACE scan to access the table.
2. The absence of a suitable index forced an internal sort. This was required to eliminate duplicates (SELECT DISTINCT), satisfy an ORDER BY or GROUP BY, or support a join. When in doubt, the presence of an internal sort can be verified by executing EXPLAIN PLAN for the SQL statement in question. For more information on using the explanation tables, see the *Performance Tuning Handbook*.

Possible Actions:

ACTION	PAGE
Create Indexes	Below
Create Index on Sort Column(s)	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 93. Insufficient Indexing - Actions

1. Create Indexes

Create one or more appropriate indexes on the table. For recommendations on the number and placement of indexes, see the *Performance Tuning Handbook*.

2. Create Index on Sort Column(s)

Try creating an index on the column(s) being sorted. If this index is used to access the table, the rows returned will already be in sort order and the internal sort will be bypassed. Bear in mind, however, that if the database manager estimates that it can more efficiently access the table using some other index, it will use that index instead. In that case, an index on the sort column(s) will not affect performance.

This action is appropriate only if a large number of rows need to be sorted. The database manager does small sorts very quickly.

Invalid Entities Exist

Problem Description: Before any index operation, DBSS verifies that the index is valid. If there are invalid indexes, then this verification increases processing time.

Possible Actions:

ACTION	PAGE
Drop invalid indexes and recreate them.	Below

Figure 94. Invalid Entities Exist- Actions

1. Drop invalid indexes and recreate them.

Use the SHOW INVALID command to display the invalid indexes. For more information on invalid indexes, see "Invalid Indexes" on page 41. For more information on the SHOW INVALID command, see *Operation* manual.

You can also force an invalid index to be dropped and recreated by using the REORG statement.

Large Tables Share Same DBSPACE

Problem Description:

When large tables share the same DBSPACE, DBSPACE scans can be excessively long. That is, each of the large tables will use up a large number of pages in the DBSPACE. This will typically result in a large value for NACTIVE (active pages in the DBSPACE), and a small to medium value for PCTPAGES (the percentage of active pages occupied by any given table in the DBSPACE). A DBSPACE scan reads *all active* pages in the DBSPACE.

Note: The value used in path selection is calculated dynamically as

$$\frac{\text{SYSCATALOG.NPAGES}}{\text{SYSDBSPACES.NACTIVE}}$$

If the calculated value is small, SQL/DS path selection will tend to favor INDEX scans over DBSPACE scans. If the calculated value is medium to high, then SQL/DS path selection will tend to favor DBSPACE scans over index scans.

If a DBSPACE scan is used for one of the tables, all the pages for the other large table(s) will also be read. This, of course, means the scan will be doing a lot of useless I/O's for pages that do not contain rows of the table being searched.

For example, if you have a DBSPACE that contains a table that occupies 500 pages and another table that occupies 600 pages, and NACTIVE for the DBSPACE is 900, you may experience the "Large Tables Share Same DBSPACE" problem. If a DBSPACE scan is done on the 500 page table, 400 pages (900 minus 500) will be read with no rows found. Similarly, a DBSPACE scan of the 600 page table will result in 300 page reads with no rows found.

As you can see, the problem is most severe for large tables. However, you can also see the same basic problem if you have an unindexed table sharing a DBSPACE with a large table. For example, if you add a 2 page table to the DBSPACE in the previous example and do not index it, all accesses to the 2 page table will be via DBSPACE scans. Such accesses would result in 902 pages being read just to get 2 pages of rows!

Note, however, PCTPAGES for the 2 page table would be 0. If any index were created on the two page table, the index would be used on almost all accesses to the table.

When PCTPAGES is this small, the database manager will typically use an index rather than a DBSPACE scan to find pages with rows of the table.

In summary, you probably have the "Large Tables Share Same DBSPACE" problem if:

1. You have a large number of database I/O's,
2. The applications are doing DBSPACE scans,
3. You have a Large table in a DBSPACE with other large tables (or small tables that are not indexed), and
4. The problem persists in single user environments

Note: There are other problems that show similar symptoms (high I/O's with high CPU usage). See "I/O Related Performance Problems" on page 113 for the list of other possible "high I/O" problems.

The "Large Tables Share Same DBSPACE" problem can show up in the following situations:

1. On Data Manipulation Statements

Data manipulation statements (DELETE, INSERT, SELECT and UPDATE) can result in DBSPACE scans. In the case of INSERT, this would apply only to *format 2 INSERTs*.

2. On CREATE INDEX, DROP TABLE or UPDATE STATISTICS Statements

These statements are *always* done as DBSPACE scans.

3. DATALOAD Statements with SET UPDATE STATISTICS ON

DBSPACE scans are not performed during loading of data, when UPDATE STATISTICS is set ON. However the following are exceptions. Under these conditions, a DBSPACE scan *will* be performed.

- If data is being loaded into more than one table.

- If there are indexes on the table.
- If the table being loaded already contains data.

The above will require an explicit UPDATE STATISTICS following the DATALOAD, which *will* cause a DBSPACE scan.

4. Unloading (DATAUNLOAD or UNLOAD)

If whole (or large portions of) tables are unloaded, then the unload operation would typically be done as a DBSPACE scan.

Possible Actions:

ACTION	PAGE
Index tables	Below
Reorganize DBSPACES	Below
Redesign Applications to avoid DBSPACE scans	Below
For other problems that result in DBSPACE scans, see: - "DBSPACE Scan Being Performed" - "Need a Highly Clustered Index" - "Index No Longer Highly Clustered" - "Inaccurate Statistics"	141 187 168 162
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 95. Large Tables Share Same DBSPACE - Actions

There are basically three ways you can try to resolve the "Large Tables Share Same DBSPACE" problem:

1. Index the tables in the DBSPACE

In the first approach, the objective is to eliminate the problem by indexing tables such that DBSPACE scans never (or rarely) occur. Clearly they are going to occur on CREATE INDEX, DROP TABLE or UPDATE STATISTICS, but you might be able to avoid DBSPACE scans on your data manipulation statements through proper indexing. *Clustering indexes* are particularly important for avoiding DBSPACE scans.

This approach can be quite effective at eliminating DBSPACE scans for searches on the smaller tables in the DBSPACE. It is less likely to be effective if you have multiple large tables in the same DBSPACE. However, a lot depends on how the large tables are being accessed and whether or not the tables have a clustering index.

2. Reorganize the DBSPACES

In many cases, you will probably have to address the "Large Tables Share Same DBSPACE" problem by redefining where your tables are stored. That is, eliminate the problem by not storing large tables in the same DBSPACE.

3. Redesign Applications to avoid DBSPACE scans

DBSPACE scans can be avoided through index and DBSPACE reorganization. They can also be avoided during DATALOADs with SET UPDATE STA-

STATISTICS ON, with restrictions. For a list of restrictions see "DATALOAD Statements with SET UPDATE STATISTICS ON" under Problem Description.

If these do not sufficiently eliminate DBSPACE scans, then you might want to then consider the third approach. That is, redesign your applications to be more selective such that DBSPACE scans are less frequent.

In the first and third approaches, you would be trying to optimize performance within the constraint that large tables must share the same DBSPACE. The second approach more directly addresses the problem.

For other problems that could be causing high I/O's or DBSPACE scans, see "I/O Related Performance Problems" on page 113. You may want to specifically review the problems that are likely to result in DBSPACE scans (see list shown in Figure 75 on page 143).

Lock Level Too High

Problem Description:

"Lock Level Too High" is a condition that occurs when applications are locking more data than is necessary. Locking done by applications is determined by the lock level defined for the DBSPACES referenced (as defined by the LOCKMODE option on ACQUIRE or ALTER DBSPACE). However, the level of locking can also be overridden by the application through use of the SQL LOCK statement.

If response time is long or erratic in multiple user mode, but OK when only one user is connected to the application server, then it could be a LOCK WAIT problem.

If it is a lock wait problem, check to see what DBSPACE(s) the lock waits are occurring in and the level of locking being done in the DBSPACE(s). If the data being accessed is in a DBSPACE defined with DBSPACE locking level (LOCKMODE=S in SYSDBSPACES), then the lock level is probably too high.

Since SHARE locks conflict with EXCLUSIVE locks, and EXCLUSIVE locks conflict with all other locks, DBSPACE locking should not be used when multiple users are updating the data, or when only one user is doing frequent updates while other users are trying to read the data.

A similar problem can occur with LOCKMODE=P (page level locking). The locking level may be too high if LOCKMODE=P, but the number of pages occupied by the table (NPAGES in SYSCATALOG) is small. See "Hot Spot in User Tables" on page 158 for more information on this case.

If the DBSPACES being accessed are already defined with page or row level locking, then the problem could be lock escalations or adjacent key locking. For more information on these problems, refer to "NLRB Parameters Too Small" on page 191, "Lock Level Too Low" on page 179, "Excessive Locking in User Data" on page 149 or "Adjacent Key Locking in User Data" on page 115.

If the applications are using row level locking, then you may also want to investigate "Conflict on Key Hash in User Data" on page 136.

Possible Actions:

ACTION	PAGE
Decrease Lock Level	Below
If using PAGE or ROW locking, see also: - "Hot Spot in User Tables" - "Excessive Locking in User Data" - "Lock Level Too Low" - "Adjacent Key Locking in User Data"	158 149 179 115
If using ROW locking, see also - "Conflict on Key Hash in User Data"	136

Figure 96. Lock Level Too High - Actions

There is basically only one solution to the "Lock Level Too High" problem. If you currently are operating with DBSPACE level locking, use the ALTER DBSPACE statement to reduce the LOCKMODE to page (P) or row (T) level locking.

If you currently are operating with page or row level locking, continue the analysis of the problem with the problems listed in Figure 96.

Lock Level Too Low

Problem Description:

Generally speaking, a low lock level (row or page locking) will allow more users to access the same data (table) at the same time. However, if applications are accessing a lot of rows, then a lot of data will be locked anyway. It will just take more lock requests to get them. Thus, it is possible to access data with the *lock level too low*.

If the application performs satisfactorily in single user mode but has erratic or long response time in multiple user mode, then the problem could be a lock wait problem.

If your applications are experiencing lock waits *and* you are also experiencing lock escalations, then the problem may be that the locking level is too low. This will occur if the locking level is page or row locking, and the application(s) access a lot of rows.

Note: Rows accessed include rows inspected by the database manager to find the rows requested. The rows accessed are not necessarily limited to just those requested.

When applications access many rows, then page or row level locks can get escalated to DBSPACE locks. This escalation means that the applications are effectively running with DBSPACE locks. This is described more fully under "Excessive Locking in User Data" on page 149.

Note: If the applications are not accessing that many rows, then the escalations could be due to your NLRB initialization parameters being set too small (see "NLRB Parameters Too Small" on page 191).

Possible Actions:

ACTION	PAGE
Increase Lock Level	Below
Use LOCK Statement to Avoid Escalations	Below
Use Cursor Stability Isolation Level	Below
For other problems with similar symptoms, see: - "Excessive Locking in User Data" - "NLRB Parameters Too Small"	149 191
For other lock wait problems, see: - "Locking Related Performance Problems"	114

Figure 97. Lock Level Too Low - Actions

If the DBSPACE in which locks are escalated is defined with row level locking, then increasing the lock level to page level locking is the easiest corrective action you can take, and may be sufficient.

If the locking level is already defined to be page level locking, changing the locking level may help, but probably not much. If the DBSPACE is changed to DBSPACE locking from page locking, *all* applications on the DBSPACE will contend at the DBSPACE level.

Another approach to the problem is to reduce the number of locks held by an application at any one point in time by using the Cursor Stability Isolation Level in the applications that obtain many read locks. The use of Cursor Stability can drastically reduce the number of locks held by an application and the length of time SHARE locks are held.

For other reasons that might be causing lock escalations, see "Excessive Locking in User Data" on page 149 and "NLRB Parameters Too Small" on page 191.

For other possible lock wait problems, refer to "Locking Related Performance Problems" on page 114.

Locks Held for Long Duration

Problem Description:

Locks protect data and applications in multiple user, read/write environments. However, they also are a potential source of long response times due to long lock waits. Thus, locking facilities should not be used indiscriminately. The "Locks Held for Long Duration" problem is the condition that can occur when users or applications hold locks longer than necessary.

If you are experiencing long or erratic response times in Multiple User Mode, but the response time is OK when only one user is connected to the application server, then you could have a lock wait problem.

One of the types of lock wait problems that can occur is *locks being held for long duration*. This is when a user or an application obtains locks, but delays or postpones freeing them when it should.

The symptoms of the "Locks Held for Long Duration" problem are *lock waits* and possibly *link waits*. However, there are other "lock wait" problems that have the same symptoms.

Another condition to look for is *communication waits*. If users are waiting for locks held by a user in a long communication wait, then you probably have the "Locks Held for Long Duration" problem. A user in communication wait is not actively processing SQL requests, or using the data he/she has locked.

This problem is most likely to occur if you have the following types of activity on your system:

1. ISQL query usage

ISQL query usage is not necessarily a problem. However, if query users spend a lot of time in display mode (looking at query results), then you might experience the "Locks Held for Long Duration" problem. This is particularly true if the ISQL users run with isolation level set at "repeatable read" (ISOL=RR). While in ISQL display mode with ISOL=RR, ISQL will hold *all* the locks obtained to get the query result. The locks will not be freed until the user ENDS display mode.

2. ISQL usage in AUTOCOMMIT OFF mode

Use of the ISQL AUTOCOMMIT OFF function is another possible source of the "Locks Held for Long Duration" problem. With AUTOCOMMIT set OFF, all locks obtained between COMMIT (or ROLLBACK) WORK statements are held until the user explicitly enters COMMIT (or ROLLBACK) WORK.

3. Conversational Applications

A conversational application is an application that holds resources (such as SQL/DS locks) as it "converses" with the user. Such applications leave the system at the mercy of the user. Users that delay responding to such an application delay releasing of locks.

4. Long Batch jobs

Long batch jobs can create the problem, if they do not issue periodic COMMIT WORK statements.

5. Large Load jobs

Large load jobs can create the problem, if they do not use the COMMITCOUNT option of the DATALOAD command.

Possible Actions:

ACTION	PAGE
Use Multiple LUWs	Below
Use Cursor Stability Isolation Level	Below
Run Offending Jobs Off-Hours	Below
Use Redundant Data to Avoid Lock Contention	Below
For other problems with similar symptoms, see: - "Agents Being Held" - "Too Few Agents"	119 205
For other lock wait problems, see: - "Locking Related Performance Problems"	114

Figure 98. Locks Held for Long Duration - Actions

There are two basic approaches to resolving a "Locks Held for Long Duration" problem:

1. Free locks earlier

The first approach that is recommended is to see what can be done to have the offending applications or users free locks earlier. Two techniques that might be explored are:

- Use Multiple LUWs

By having the offending applications COMMIT WORK more frequently, you can reduce the amount of lock contention caused by the application.

- Use of Cursor Stability Isolation Level

By having the offending applications use the Cursor Stability isolation level, fewer locks are held until the end of the LUW. This may be sufficient to resolve the conflicts caused by the applications.

2. Isolate the Offending Work

The second fundamental approach to the problem is to try to isolate the offending applications. While multiple user read/write sharing is a desirable capability, it may not be practical in all cases. When an application cannot share data with other applications without causing undesirable contention problems, then the application should be isolated.

There are basically two ways that an application can be isolated:

- Reschedule Offending Jobs

By running the offending application at a time when other applications don't need the data, you can avoid contention while retaining the concept of common data. With this solution, you would be trading off application availability to minimize storage and data maintenance costs.

- Use of Redundant data to avoid lock contention

By running the offending application against a copy of the data, you can avoid the lock contention problems while retaining the ability to run the offending application on a more flexible (and presumably convenient) schedule. With this solution, you would be trading off storage and "real time data" for application function availability.

Before pursuing application isolation, you might want to investigate problems with similar symptoms. Specifically, you may want to investigate "Agents Being Held" on page 119 or "Too Few Agents" on page 205. This is particularly true if you are experiencing significant link wait conditions.

Logging during Load

Problem Description:

When loading a large amount of data with LOGMODE=Y, A or L, all rows inserted into the table(s) are also written to the log. In addition to consuming log space, the logging can noticeably affect the performance of the load operation. A high volume of logging will tend to aggravate checkpoint-related problems. See "CHKINTVL Too Small" on page 132 and "Long DBSS Calls Delaying Checkpoint" on page 184.

Possible Actions:

ACTION	PAGE
Load in Single User, NOLOG Mode	Below
Increase CHKINTVL	Below
Use Nonrecoverable DBSPACES	Below

Figure 99. Logging during Load - Actions

1. Load in Single User, NOLOG Mode

The most effective way to reduce the overhead of loading is to perform the load in single user mode with LOGMODE=N. See "Switching Log Modes" in *System Administration* manual for a discussion on how to do this. However, if you normally run with LOGMODE=A, data loaded with LOGMODE=N will not be recoverable from a DASD failure until after the first archive following the load operation. Therefore it is wise to use this solution when the load(s) can be done just before an normal archive operation is scheduled.

2. Increase CHKINTVL

Another action that can relieve logging-related overhead during a load operation is to set the CHKINTVL initialization parameter to a high value (for example, 200 or higher). The resulting reduction in checkpoint frequency will greatly reduce the amount of I/O and processing time associated with the checkpoint function. There are some trade-offs you should consider before increasing CHKINTVL. These are described in "CHKINTVL Too Small" on page 132.

3. Use Nonrecoverable DBSPACES

Yet another possible alternative is to use nonrecoverable DBSPACES for the data being loaded. This should be considered if the data can be recovered from an external source. Guidelines for the use of nonrecoverable DBSPACES are provided under "Nonrecoverable Storage Pools", in *System Administration* manual. A multiple user mode DATALOAD into a nonrecoverable DBSPACE will generally yield performance similar to loading in single user mode with LOGMODE=N.

Long DBSS Calls Delaying Checkpoint

Problem Description:

In order to perform a checkpoint, all execution in the DBSS must first be quiesced. To do this, SQL/DS first keeps any new requests from entering the DBSS, and then waits until all currently executing DBSS calls leave the DBSS. Checkpoint processing is then able to begin. Once the checkpoint has been completed, the processing of DBSS calls is resumed.

Nearly all SQL statements that can be issued by a user are ultimately implemented by one or more DBSS calls. Effectively, then, the response times of all users currently executing an SQL statement at the time checkpoint is initiated will be delayed by the time it takes all users currently executing in the DBSS to leave the DBSS plus the actual time to perform the checkpoint. This delay is normally a few seconds. However, if one of the currently executing DBSS calls is long-running, this delay can be much longer.

Most potentially long-running DBSS calls will make frequent checks to see if a checkpoint is pending, and if so, temporarily leave the DBSS. Therefore, most such calls should not cause undue delays at checkpoint time. The known exceptions to this are the DBSS calls that implement the SQL statements DROP TABLE and DROP INDEX, as well as the operator command SHOW DBSPACE.

This condition can be verified by issuing the SHOW ACTIVE operator command each time a long, unexpected delay is being experienced.

The long DBSS call problem is indicated if you often see "CHECKPOINT AGENT IS WAITING TO START CHECKPOINT" in the result. Then you will typically see one agent in I/O wait, while the remaining agents are either inactive, in communications wait, or in checkpoint wait. The one agent that is in I/O wait is the one that is holding up the checkpoint. You may be able to contact that user and see what was being executed at that time.

If you often see "CHECKPOINT AGENT IS PROCESSING A CHECKPOINT" in the result, delays are occurring because of the time it takes to actually do the checkpoint. This problem is addressed under "CHKINTVL Too Big" on page 131. If the results show that the checkpoint agent is inactive, checkpoint-related delays are not the problem.

Possible Actions:

ACTION	PAGE
Schedule Long-Running DROP TABLE, etc	Below
Set CHKINTVL As High As Possible	Below
If delays are due to checkpoint processing, see: "CHKINTVL Too Big"	131
If checkpoint is inactive during delays, see: "Sequential Processing"	200

Figure 100. Long DBSS Calls Delaying Checkpoint - Actions

1. Schedule Long-Running DROP TABLE, etc

Schedule the execution of long-running cases of DROP TABLE, DROP INDEX, and SHOW DBSPACE for periods when potential delays to other users will not pose a problem.

2. Set CHKINTVL As High As Possible

Set the CHKINTVL initialization parameter to as high a value as is practical. This will minimize the likelihood that a long-running DROP TABLE or DROP INDEX will coincide with a checkpoint. See "CHKINTVL Too Small" on page 132 for a discussion of the factors that you should consider before changing CHKINTVL.

Missing Search Condition

Problem Description:

A superior access strategy was overlooked during the preprocessing of a join due to a missing search condition.

The SQL/DS optimizer will automatically add to your query certain types of predicates, described below. However, there are other types of predicate which, if you omit them, will not be added by the optimizer.

It is important to explicitly state all predicates other than those described below when writing a join. Otherwise, the optimizer will sometimes choose an inferior access path, e.g. fail to consider a useful index or choose "inner" and "outer" tables of the join the wrong way around.

Predicates are automatically added by the optimizer when:

The query (or subquery) is a join, and there is an **equaljoin** condition relating two tables, that is of the form

A.COL1=B.COL2

and there are one or more local predicates on either of the join columns, of the form

A.COL1 <comparison_operator> <value>
or
B.COL2 <comparison_operator> <value>

then for each such local predicate, if it meets the following two conditions:

1. it is sargable
2. the query does not already contain a sargable predicate of the same kind on the other join column

then the optimizer will add the implied predicate on the other join column, of the form

B.COL2 <comparison_operator> <value>
or
A.COL1 <comparison_operator> <value>

respectively.

Consider the following example:

```
SELECT * FROM EMPLOYEE , DEPT
WHERE EMPLOYEE.DEPTNO = DEPT.DEPTNO
AND EMPLOYEE.DEPTNO = 288
```

The SQL/DS optimizer will automatically add the implied predicate

```
AND DEPT.DEPTNO = 288
```

and thus is able to consider a search strategy of accessing DEPT as the outer table, and EMPLOYEE as the inner table within DEPT, as well as the other way round. It will then choose whichever of these strategies is best.

However, in the following examples, the optimizer will not add any predicates :

```
SELECT * FROM EMPLOYEE , DEPT
WHERE EMPLOYEE.DEPTNO = DEPT.DEPTNO
AND EMPLOYEE.DEPTNO IN (288, 388, 488)
```

The optimizer will not add an additional predicate because the local predicate is not sargable. EMPLOYEE will become the outer table (the one first accessed) because this is the only table for which there is any selective access.

```
SELECT * FROM VEHICLES , BRIDGES
WHERE VEHICLES.WEIGHT < BRIDGES.MAXLOAD
AND VEHICLES.WEIGHT > 15
```

The optimizer will not add an additional predicate because the join condition is not an **equijoin**. VEHICLES will become the outer table (the one first accessed) because this is the only table for which there is any selective access.

Possible Actions:

ACTION	PAGE
Explicitly Write All Search Conditions other than those added by the Optimizer	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 101. Missing Search Condition - Action

When writing a join, always make any implicit data relationships explicit by writing additional search conditions other than those added by the optimizer. This will allow the optimizer to consider all possible alternatives.

Continuing the example shown in the problem description, we know that if VEHICLES.WEIGHT < BRIDGES.MAXLOAD AND VEHICLES.WEIGHT > 15 then it is also true that BRIDGES.MAXLOAD > 15. This implicit relationship must be stated explicitly in the form of an additional search condition for SQL/DS to be able to use it during the optimization process. Consequently, it is better to write the example join as follows:

```
SELECT * FROM VEHICLES , BRIDGES
WHERE VEHICLES.WEIGHT < BRIDGES.MAXLOAD
AND VEHICLES.WEIGHT > 15
AND BRIDGES.MAXLOAD > 15
```

Now the optimizer will consider both VEHICLES and BRIDGES as realistic candidates for being the outer table. Depending on the situation, this may result in a better access strategy and, hence, yield better performance when the join is executed.

Similarly, for the other example, you should explicitly state the predicate
AND DEPT.DEPTNO IN (288, 388, 488)

Need a Highly Clustered Index

Problem Description:

A query involving a sequential scan of some significant number of rows (more than 20) is being performed inefficiently by a DBSPACE scan, because there is no highly clustered index.

We say that an index is highly clustered if the sequence in which the table's rows are stored in pages in the database corresponds closely to the index key sequence. Whether or not an index is highly clustered does not have a significant effect on performance if the number of rows examined by scanning the table with that index is small (for example, less than 20). In that case lack of a Highly Clustered Index is probably not the problem.

If a table does not have a highly clustered index, access to that table will be less efficient for sequential scans because SQL/DS will have to access the table using a DBSPACE scan or using an index scan via an unclustered (or only slightly clustered) index. In the case of a DBSPACE scan, SQL/DS will have to examine all active pages in the DBSPACE, not just those that contain rows of the desired table. In the case of an unclustered index scan, each row examined will often require another I/O to the database.

This problem can arise either because you never created a highly clustered index or because a highly clustered index became less clustered. The latter problem is covered in "Index No Longer Highly Clustered" on page 168; this section deals with the former. Note however that the symptoms are the same.

Information as to whether or not a given index is highly clustered is maintained in the CLUSTERRATIO column of the row corresponding to that index in the SYSTEM.SYSINDEXES catalog table. This information is initially filled in when the index is created and is updated whenever an UPDATE STATISTICS statement is explicitly issued for the associated table, or during a DBSU DATALOAD with SET UPDATE STATISTICS ON. Some restrictions apply for the DATALOAD. For details see "Large Tables Share Same DBSPACE" on page 175. For more information on data clustering, see the *Performance Tuning Handbook*.

CLUSTERRATIO is a value in the range 0 to 10000. The higher the value, the more efficient will be an index scan on the table. The SQL/DS Optimizer considers CLUSTERRATIO when deciding whether to use a DBSPACE scan or index scan to scan a table, and to choose between indexes. There is no absolute value of CLUSTERRATIO which you can use as a criterion for whether the degree of clustering of an index is "good enough"; the criterion will vary from table to table and also depend on the queries you run. For example, if there are two similar tables with identical definitions, and each has a single index defined on the same column or set of columns, and if one table has many more rows than the other but both tables have the same number of "out-of-sequence" rows, then the CLUSTERRATIO of the larger table will be larger than that of the smaller table. However, for larger tables an index whose CLUSTERRATIO is less than 6000 is unlikely to give good performance when used for sequential scans, and a preferable value is 9500 or more. Also, you can create a totally clustered index (CLUSTERRATIO = 10000) by following the procedure below. Refer to "Index No

Longer Highly Clustered" on page 168 for actions needed to maintain a sufficient degree of clustering thereafter.

You can determine whether there are any highly clustered indexes on a table by issuing the following SQL statement:

```
SELECT INAME,ICREATOR,CLUSTERRATIO  
FROM SYSTEM.SYSINDEXES  
WHERE TNAME=table_name AND CREATOR=table_creator
```

This query tells you the CLUSTERRATIO of all indexes on the table.

- For very small tables, the value of CLUSTERRATIO is not too important, and the only important thing to check is that there is at least one index.
- For larger tables, you should check that the value of CLUSTERRATIO is sufficiently high, based on the discussion above and on your own rules of thumb.

Possible Actions:

ACTION	PAGE
Create a Highly Clustered Index	Below

Figure 102. Need a Highly Clustered Index - Actions

Most tables should be supplied with one or more indexes. The main exception to this would be very small tables for which optimum performance is not required. These should be put into one or more DBSPACES that do not contain any large tables.

For each indexed table, you should make one of these indexes highly clustered. This will usually be the "clustering" index. The process of creating a clustering index, and guidelines for deciding which indexes to make highly clustered, are described in "Clustering rows of a table on an Index" in the *Database Administration* manual. Guidelines for deciding which index to make the clustering index are provided under "Clustering Rows of a Table on an Index" in the *Database Administration* manual.

Need More CPU

Problem Description:

If you have a high CPU utilization and generally poor response times, it could mean that you are simply trying to do too much work on the CPU you have. That is, you may need a larger CPU. Obviously, before you conclude that this is the case, you want to make sure that your problem is not one of the other problems that result in high CPU usage. For more information on problems that may cause high CPU usage, see "CPU Related Performance Problems" on page 112.

There are, however, other symptoms that suggest that you might need a larger CPU. In particular, consider the following possible companion conditions:

1. Paging

If the paging rate is low on your system, then you are not using up CPU cycles doing paging. On the other hand, if your paging rate is high, then you might be able to "buy back" some CPU cycles by trying to reduce the paging.

2. Database I/O's

You will use fewer CPU cycles by working out of the buffer pools, rather than doing a lot of database I/O's. As a result, if you have a *low* buffer hit ratio, you might be able to "buy back" some CPU cycles by trying to improve your buffer hit ratio.

Note: A high CPU utilization on your system could be due to non-SQL applications. If you have a substantial amount of non-SQL work on your system, tuning your application server may not be of any great benefit.

Possible Actions:

ACTION	PAGE
Upgrade your CPU	Below
For other high CPU usage problems, see: - "CPU Related Performance Problems"	112

Figure 103. Need More CPU - Actions

If you cannot find another way to reduce the demand for CPU cycles, then your only alternative is to upgrade your CPU.

Before resorting to upgrading your CPU, you should review problems that can result in high CPU utilizations. Refer to "CPU Related Performance Problems" on page 112 for problems that cause high CPU usage.

Need More Real Storage

Problem Description:

If you have a significant paging rate, it could mean that you simply need more real storage to support the work on your system. That is, it may not be possible to support your workload with your current real storage configuration. Obviously, before you conclude that this is the case, you want to make sure that your problem is not one of the other problems that result in high paging rates. For more information on problems that may cause high paging rates, see "Storage Related Performance Problems" on page 114.

There are, however, other symptoms that suggest that you might need more real storage. In particular, consider the following possible companion conditions:

1. CPU Utilization

If the CPU Utilization is low on your system, and most of your workload is SQL work, then your database machine is not using up CPU cycles because it is spending too much time waiting on paging I/O. On the other hand, if your CPU utilization is high, then you might conclude that your database machine is still getting enough time on the CPU to get a reasonable amount of work done.

2. Database I/O's

You will use fewer CPU cycles by working out of the buffer pools, rather than doing a lot of database I/O's. On the other hand, a large buffer pool could be contributing to your paging problems. Thus, it may be more advisable to run with a smaller buffer pool and do more database I/O's, than do the paging. That is, a *high* buffer hit ratio would indicate that you might be able

to reduce paging by reducing the size of the buffer pools. See "Buffer Pool Too Big" on page 126 for more information on this case.

Note: A high paging rate on your system could be due to non-SQL applications. If you have a substantial amount of non-SQL work on your system, tuning your application server may not be of any great benefit.

Possible Actions:

ACTION	PAGE
Add more Real Storage	Below
For list of other paging problems, see: - "Storage Related Performance Problems"	114

Figure 104. Need More Real Storage - Actions

If you cannot find another way to reduce the demand for real storage, then your only alternative is to add more real storage.

NLRB Parameters Too Large

Problem Description:

The NLRB initialization parameters (NLRBS and NLRBU) control the number of *lock request blocks* that are allocated for use (in multiple user mode). The more you allocate, the less likely you are to experience lock escalation problems. However, LRBs are control blocks that consume storage. Thus, there is a practical limit to how many you can afford to have allocated. If you allocate too many, you may introduce paging problems.

With a large number of LRBs, you might cause a lot of page fault activity in the database machine. High paging can be very detrimental to the SQL/DS performance for *all* applications. See "Page Fault Serialization" on page 197 for more information on the effects of high paging rates.

Paging problems can be caused by many other problems. It really isn't likely that your NLRB initialization parameter settings are your problem. LRB control blocks each use only 24 bytes of storage. Thus, you would have to have them set extremely high for them to cause paging problems. Furthermore, just having them set high does not necessarily mean you will have a paging problem. You would also have to be *using* them. (Note: You can tell how many you are using through the SHOW LOCK MATRIX statement).

One way of considering this is that you would have to be using about 43000 LRBs in order to generate a demand for 1 million bytes of real storage. While this may seem ridiculous, it also means that 4300 LRBs will generate a demand for 100 thousand bytes of real storage. With row level locking, it is not that difficult to generate a demand for 4300 LRBs (and the corresponding need for 100 KB of real storage).

If you are experiencing paging problems, and you are using a large number of lock request blocks, you may have the "NLRB Parameters Too Large" problem. This will depend on whether or not you really *need* the LRBs you are using. You can determine this by checking the amount of *lock escalation* that is occurring on your system. You can check this by using COUNTER ESCALATE and COUNTER

LOCKLMT. If you are experiencing few or no lock escalations (ESCALATE + LOCKLMT), and none of them are failing (LOCKLMT), then you probably can afford to reduce the NLRB allocations.

Possible Actions:

ACTION	PAGE
Decrease NLRB Parameters	Below
For other problems with similar symptoms, see: - "Storage Related Performance Problems"	114

Figure 105. NLRB Parameters Too Large - Actions

If you suspect you have your NLRB parameters set too high, the obvious action to take is to decrease the NLRBU and/or NLRBS settings. This is easy enough to do, but not necessarily easy to do right. That is, you need to determine what settings can be done without causing other problems. You should use the SHOW LOCK MATRIX command to see how many lock request blocks are being used.

Before assuming that your problem is "NLRB Parameters Too Large," you should check the other potential paging problems (see "Storage Related Performance Problems" on page 114).

NLRB Parameters Too Small

Problem Description:

The NLRBU and NLRBS initialization parameters define how many lock request blocks are to be available per agent and for the whole system, respectively. When either one of these parameters is exceeded, locks held by the agent that causes the limits to be exceeded will be "escalated." Lock escalation "trades" many page or row (and key) level locks for a single DBSPACE level lock. In many cases, this does not cause a problem. However, in other cases conflicts arise when the database manager tries to obtain the DBSPACE lock.

If you are experiencing lock waits and Lock escalations, you may have the "NLRB Parameters Too Small" problem.

Other problems with similar symptoms include "Lock Level Too Low" on page 179, and "Excessive Locking in User Data" on page 149. If your users (or applications) are requesting a large number of locks, you may want to check for those problems as well.

Specific situations where you might expect to encounter the "NLRB Parameters Too Small" problem include:

1. Preprocessor Usage

Preprocessing of applications involves numerous catalog references. This frequently results in a large number of lock requests. The actual number of locks obtained will vary depending on the SQL statements in the programs being preprocessed.

Lock escalations due to preprocessing activity are particularly severe because most of the locks obtained are in the catalog DBSPACE (DBSPACE 1). Furthermore, preprocessing does updating to some of the catalog tables.

Thus, the escalation attempt is for an EXCLUSIVE lock on the catalog DBSPACE. This, of course, will conflict with most SQL operations.

2. High Loading/Unloading Usage

Loading or unloading can result in large numbers of locks being obtained. Bulk loading of rows into a DBSPACE with row or page level locking can result in numerous EXCLUSIVE lock requests. This, in turn, can result in a lock escalation for an EXCLUSIVE lock on the DBSPACE being loaded.

Unload operations are less likely to cause the LRB limits to be exceeded. But they can if they are accomplished through an index scan.

3. Large Query/Reports Activity

Like unload operations, queries and reports are less likely to cause the LRB limits to be exceeded. But they can if they are accomplished through an index scan.

Possible Actions:

ACTION	PAGE
Increase NLRB Parameters	Below
For other problems with similar symptoms, see: - "Excessive Locking in User Data" - "Lock Level Too Low"	149 179

Figure 106. NLRB Parameters Too Small - Actions

There is basically only one solution to the "NLRB Parameters Too Small" problem. That is to increase NLRBU and/or NLRBS.

If your NLRB parameters are already large and you do not have enough virtual storage to support the needed NLRB parameter values, you should then treat your problem as an "Excessive Locking in User Data" or "Lock Level Too Low" problem. See "Excessive Locking in User Data" on page 149 and "Lock Level Too Low" on page 179 for more information on these cases. Note however, solutions to these problems will not generally work if the problem is due to preprocessing activity.

No Selective Index

Problem Description:

No suitable index is available to support selective access to the table being referenced. As a result, every row in the table is accessed with one of the indexes on that table, with every row in the DBSPACE using a DBSPACE scan.

Possible Actions:

ACTION	PAGE
Create Index(es) on Selective Columns	Below
See the following closely related problems: "Insufficient Indexing" "Index Disqualified" "Inefficient Search"	174 164 170
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 107. No Selective Index - Actions

Take a look at each search condition associated with the SQL statement in question and determine whether or not it is eligible to be supported by an index. Then create an index on the column associated with each eligible search condition. When the SQL statement is preprocessed, the database manager will decide which index offers fastest access to the desired data.

This comprehensive indexing approach is most applicable when index maintenance overhead and index secondary storage requirements are not important factors. If they are, it would be better to just create one index. This should be placed on the column corresponding to the most selective search condition that is eligible for index support.

For more information on indexes and predicate processing, see the *Performance Tuning Handbook*.

One Database Machine Needs Too Much CPU

Problem Description:

In a multiprocessing environment, a virtual machine can be executing on just one processor at any given moment in time. Consequently, if the CPU demand rate of a database machine exceeds that of one of the processors, the result will be poor SQL/DS response times. This will be due to processor contention, even though there may be much unused processing capacity in the multiprocessing configuration as a whole.

This problem is indicated by the following combination of symptoms:

- A multiprocessing environment.
- SQL/DS response times are long, while the response times of non-SQL/DS requests are adequate.
- One database machine is using nearly one processor's worth of capacity.

To determine if this is the case, obtain VM monitor data for a representative ten minute interval when SQL/DS response times are long. Reduce this interval with the VM/370 Performance/Monitor Analysis Program (VMMAP) and then look at the "User Resource Utilization Summary". SQL/DS is using nearly one processor's worth of capacity if total CPU-seconds consumed by the SQL/DS database machine (including its use of CP services) is within 10 percent of the number of seconds in the measured interval. On VM/XA and

the VM/ESA ESA Feature systems, the *VM Performance Reporting Facility* (VMPRF) provides the same function as VMMAP.

Possible Actions:

ACTION	PAGE
Split Load across Two or More DB Machines	Below
Larger Processor	Below

Figure 108. One Database Machine Needs Too Much CPU - Actions

As a rule of thumb, try to keep the total CPU usage of any database machine less than 40 percent of total CPU usage.

1. In some cases, the best way to accomplish this is to split the SQL/DS usage across two or more database machines.
2. In other cases, the solution may be to move to a larger processor so that the same SQL/DS load uses a smaller percentage of the overall processing capacity. Other, non-SQL/DS work could be moved to that processor to use the processing capacity not needed by the database machine.

Package Needs Re-preprocessing

Problem Description:

A package can incorporate suboptimal access strategies if conditions in the database have changed significantly since the last time it was preprocessed. Probably the most dramatic example is the case where no indexes existed on the tables accessed by the application when it was preprocessed, but suitable indexes were created later.

Another example of when a package should be re-preprocessed is when the package was preprocessed with warning messages because one or more of the referenced base objects (for example, tables) did not exist in the database. If the missing base objects are later added to the database, such a package will execute. However, performance will be degraded for the SQL statements that reference those base objects since the database manager has to dynamically preprocess them every time they are executed.

Possible Actions:

ACTION	PAGE
Reprep or Invalidate Package	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 109. Package Needs Re-preprocessing - Actions

A package should be re-preprocessed whenever any of the following situations apply:

- A new index was added to one of the tables referenced by that package and there is some possibility that the package may be able to make use of it.
- The last time the package was preprocessed, warning messages were generated because one or more base objects did not exist, which now exist.
- The contents of one or more of the tables referenced by that package have changed significantly.
- The last time the package was preprocessed, the statistics for one or more of the tables it references were not up to date, but are now up to date.

There are three ways to re-preprocess a package. The preferred way is to use the DBS utility REBIND PACKAGE command. This command is not supported on a non-SQL/DS application server or if you are using the DRDA protocol. A second way is to run the appropriate preprocessor against the source code (recompilation is not necessary). The third way is to use the DBS utility UNLOAD PACKAGE command followed by a RELOAD PACKAGE command. This is faster than rerunning the preprocessor.

In any case, make sure that the statistics for all accessed tables are up to date.

You can use the following query to obtain a list of these tables:

```
SELECT BCREATOR,BNAME FROM SYSTEM.SYSUSAGE
WHERE BTYPE='R'
AND DNAME=package_name
AND DCREATOR=package_creator_name
```

Package Cache Too Big or Threshold Too High

Problem Description:

In general, you want the package cache size to be as large as possible for the best performance. That is, you want the size of the package cache and the threshold to be as high as possible. The larger the cache, the more packages that are loaded into storage, and the higher the threshold. The higher the threshold the more packages that remain loaded in storage at the end of the logical unit of work.

However, the package cache and threshold consume storage, and the imposed additional storage requirement can cause performance problems. The higher the value of NPACKAGE, NCUSERS and NPACKPCT, the more storage is required. If you set the values of these parameters high, and experience performance problems as a result, then the package cache is "too" large or the threshold is "too" high.

Thus, there is a practical limit to how much you can afford to have allocated. How large is "too large," depends on your system and the workload on your system. More specifically, "too large" can introduce paging problems. See "Page Fault Serialization" on page 197 for more information on the effects of high paging rates.

For more information on how these initialization parameters affect performance, see the *Performance Tuning Handbook*.

The size of the package cache is calculated by initialization parameters: NPACKAGE x NCUSERS. The number of agents established by the NCUSERS

parameter determines the level of concurrency. If you set the NCUSERS parameter too high, your users can experience paging and lock wait problems. See "Too Many Agents" on page 206 for more information on this case.

Possible Actions

ACTION	PAGE
Decrease Package Cache Size	Below
Decrease NPACKPCT	Below
Too Many Agents	206

Figure 110. Package Cache Too Large - Actions

1. Decrease package cache size.

To decrease the size of the package cache, decrease the value of the NPACKAGE initialization parameter. This decreases the amount of storage required for the cache.

2. Decrease NPACKPCT.

Performance can be improved by decreasing the threshold. When the threshold is decreased, the same number of packages are loaded into storage, but a lower number of packages remain in storage at the end of the logical unit of work. Decreasing the value of the NPACKPCT initialization parameter decreases the threshold.

For more information on performance improvements using the package cache, see the *Performance Tuning Handbook*.

Package Cache Too Small or Threshold Too Low

Problem Description:

The size of the package cache and size of the threshold determines how frequently loaded packages are released from storage. This has a direct effect on performance. If either the cache size is too small or the threshold is too low, sufficient packages are not kept in storage and performance problems can occur.

Possible Actions

ACTION	PAGE
Increase NPACKPCT	Below
Increase NPACKAGE	Below

Figure 111. Package Cache Too Small - Actions

1. Increase NPACKPCT.

Performance can be improved by increasing the threshold. When the threshold is increased, the same number of packages are loaded into storage, but a higher number of packages remain in storage at the end of the logical unit of work. Increasing the value of the NPACKPCT initialization parameter increases the threshold.

When increasing NPACKPCT, it is recommended to increase the value in large increments. For example, if NPACKPCT is set to 30, increase NPACKPCT to 50.

2. Increase NPACKAGE

Increasing the value of the NPACKAGE initialization parameter increases the size of the package cache. More packages are now able to be loaded into storage.

Increasing the size of the package cache also increases the threshold (threshold = package cache size x NPACKPCT / 100). Although the value of the NPACKPCT initialization parameter remains unchanged, the size of the threshold increases. As a result, not only is the cache larger, but more loaded packages remain in storage at the end of the logical unit of work.

This enlarges the cache, and increases the threshold.

For more information on performance improvements using the package cache, see the *Performance Tuning Handbook*.

Page Fault Serialization

Problem Description:

Whenever a page fault occurs in a database machine, each user request currently being processed must wait for that page fault to be resolved. This source of serialization normally has only a minor effect on response time, but can be important under the following worst case conditions:

- The system paging rate is high.
- A database machine supports a large number of concurrently active users.
- Page I/O is slow due to slow paging devices and/or high contention in the portion of the I/O subsystem used for paging.

Possible Actions:

ACTION	PAGE
Specify SET QDROP OFF USERS or SET QUICKDSP ON	Below
Split Load Across Two or More SQL Machines	Below
Add Real Storage	Below
Improve real storage usage efficiency	Below
Offload Some Work	Below
More/Faster Paging Devices	Below

Figure 112. Page Fault Serialization - Actions

1. Specify SET QDROP OFF USERS or SET QUICKDSP ON

Make sure that the database machine is being run with SET QDROP OFF USERS or SET QUICKDSP ON in effect. See "SET QDROP OFF USERS or SET QUICKDSP ON Not Used" on page 203.

2. Split Load Across Two or More SQL Machines

Split the database into two or more databases. This will spread the load across two or more database machines. This helps because page faults in one database machine do not serialize the users being serviced by a different database machine. If you do this, be sure to install the RDS and DBSS into saved segments so that just one copy is required. An explanation of how to do this is provided in the *System Administration* manual.

3. Add Real Storage

Reduce the paging rate by adding real storage.

4. Improve real storage usage efficiency

Reduce the paging rate by improving real storage usage efficiency (see "Storage Related Performance Problems" on page 114).

5. Offload Some Work

Off-load some SQL/DS or non-SQL/DS work onto another system.

6. More/Faster Paging Devices

Upgrade the paging subsystem to reduce the time required to do paging I/O. This solution is indicated if the paging rate is relatively low, but the time to handle each page fault is high. This condition can be determined from the device statistics provided by the VM/SP Performance/Monitor Program (VMMAP) or for VM/XA systems, the *VM Performance Reporting Facility* (VM/PRF).

Query Block Size Too Small

Problem Description: The query block size, as specified by the QryBlksize parameter on the SQLINIT exec, is too small. You can specify the query block size from 1 to 32 kilobytes. The default is 8 kilobytes. When using the SQL/DS-only protocol the query block size is always 8 kilobytes thus this problem only applies in a DRDA protocol environment.

The query block size determines the amount of data that is transferred with each transmission for SELECT and UNLOAD operations, when blocking is in effect. A small block size can result in a larger number of transmissions. This leads to more communication overhead and thus higher than expected response times.

According to the DRDA protocol, the application server will send back either a single row split over many blocks or multiple rows in one block (if blocking is being used). It is more efficient if the query block size is large enough to hold multiple rows and blocking is used.

Possible Actions: Increasing the query block size may improve your performance. Try rerunning the SQLINIT exec using the maximum query block size of 32 kilobytes. If this does not help, you may want to revert to a query block size of 8 kilobytes to conserve storage on the user machine.

Range Predicate Used with Host Variables

Problem Description:

Range predicates (BETWEEN, >, <, >=, =< and some forms of LIKE) require that the SQL/DS optimizer "guess" at the number of rows that fall within the range. When such predicates appear in programmed SQL statements, and the limits of the range are host variables, the optimizer does not have sufficient information to make an "informed" guess.

For example,

```
EXEC SQL DECLARE CURSOR C1 AS
      SELECT *
      FROM SQLDBA.ACTIVITY
      WHERE ACTNO BETWEEN :X AND :Y
```

presents such a problem to the optimizer. When the preprocessor is executed, the optimizer must estimate how many rows of the ACTIVITY table have a price that falls between :X and :Y. Of course at preprocessing time, the database manager doesn't know what :X and :Y are. As a result, the optimizer makes assumptions about how many rows will satisfy the "unknown" range.

The assumptions made by the optimizer in these cases are pessimistic. That is, the optimizer assumes a rather large percentage (25-33%) of the rows will qualify. This has the effect of "favoring" DBSPACE scans in many cases. It can also have the effect of "ignoring" an index in favor of another index. That is, you can have the "Range Predicate Used with Host Variables" problem even though you are not experiencing DBSPACE scans.

If you have such range predicates, and you are experiencing a large number of database I/O's, and you find on further investigation that the program is indeed doing DBSPACE scans, then you may have the "Range Predicate Used with Host Variables" problem. The primary symptom of the "Range Predicate Used with Host Variables" problem is high I/O's (with associated high CPU usage). There are many problems with the same symptoms (see "I/O Related Performance Problems" on page 113). Before concluding that you have the "Range Predicate Used with Host Variables" problem, you should consider at least four of these problems:

1. "Inaccurate Statistics" (see page 170.)
2. "Insufficient Indexing" (see page 174.)
3. "Inefficient Search" (see page 170.)
4. "Missing Search Condition" (see page 185.)

Note: This problem occurs only in precompiled applications. If your problem is with an ISQL or DBSU command, you do not have the "Range Predicate Used with Host Variables" problem.

Furthermore, it must be noted that the optimizer assumptions in these cases cause a problem only if they are indeed wrong. If the actual ranges defined at execution time are 25-33% of the table (or greater), the optimizer is, in fact, making the appropriate assumptions, and you are getting the most appropriate access path based on your selection criteria.

Possible Actions:

ACTION	PAGE
Use more Predicates	Below
Use Dynamic SQL Statements	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 113. Range Predicate Used With Host Variables - Actions

There are basically two approaches to resolving the "Range Predicate Used with Host Variables" problem:

1. Use more qualification in your WHERE-clause

The first approach is an attempt to add predicates to the WHERE-clause to give the optimizer other choices for selecting the access path. This is not always possible, but should be considered before taking more radical steps. For more information on this see "Inefficient Search" on page 170 and "Missing Search Condition" on page 185.

2. Use dynamic SQL statements

If you cannot get the optimizer to choose a more desirable access path through additional predicates, then you might try using dynamic SQL statements. That is, instead of "hard coding" the request in the application, construct the statement as a character string and use the PREPARE and EXECUTE statements. By doing this, you can substitute actual values (literals) in the statement, rather than using host variables. In this way, the optimizer evaluates possible access paths at execution time based on actual values for the range limits.

Note: One way of checking to see that dynamic SQL statements will help your situation is to try the statement on ISQL. The performance of a query issued from ISQL will be essentially the same as the same query coded as a dynamic SQL statement in your program.

Don't overlook other possible causes of high I/O problems. Host variables in range predicates will frequently result in poor access paths, but you should still consider other possible causes.

Sequential Processing

Problem Description:

This problem refers to problems that can arise when bulk sequential processing is done when the application server is running in multiple user mode. In general, bulk sequential processing can tend to dominate the database manager and use of its resources, resulting in high I/O activity, poor buffer hit ratios, and high CPU utilizations.

Bulk sequential processing refers to the following types of applications and application functions:

1. Loading and Unloading

2. Large Queries or Reports
3. Summary Queries or Reports on Large Tables
4. Large Sorts
 - a. ORDER BY
 - b. GROUP BY
 - c. Certain Joins
 - d. DISTINCT
 - e. UNION
5. Uncorrelated Subqueries
6. CREATE INDEX
7. DROP TABLE
8. UPDATE STATISTICS

The "Sequential Processing" problem will typically result in a high number of database I/O's. This is due to the sequential processing, itself, but also is due to *buffer pool flooding*. Buffer pool flooding occurs when an application accesses a lot of data and "floods" the buffer pool with its own data. Other applications will do more *actual* database I/O's than usual because the buffers are being used by the sequential application.

You may also see an increase in CPU utilization as a result of sequential processing on the system. This is partly due to the extra I/O activity, but is also due to the nature of sequential processing. Bulk sequential applications tend to use up a lot of CPU processing time without pausing. The only thing that slows them up are I/O waits (and sometimes lock waits).

Thus, a high number of database I/O's and increased CPU utilization could indicate you have the "Sequential Processing" problem. Your *buffer hit ratio* can further isolate the problem to the "Sequential Processing" problem. Sequential processing will give you a very poor buffer hit ratio. Sequential processing, by its nature, typically reads data once and does not read it again. As a result, very few references to data are satisfied out of the page buffer.

A problem that also results in high I/O's and a bad buffer hit ratio is the "Buffer Pool Too Small" (see page 128) problem. You may want to consider the size of your buffer pools as well. However, if users are doing any of the functions listed above, you probably have the "Sequential Processing" problem.

Another symptom of the "Sequential Processing" problem is *periodic high response times*. This is when users see inconsistent response times. Sometimes the response time is just fine, but other times (when the sequential applications are running), the response time is dramatically worse.

Possible Actions:

ACTION	PAGE
Increase DISPBIAS	Below
Index to Avoid I/O's	Below
Reorganize Data to Avoid I/O's	Below
Run Sequential Application Off-hours	Below
Redesign Offending Applications	Below
For another "bad buffer hit ratio" problem, see: - "Buffer Pool Too Small"	128
For other "periodic high response time" problems, see: - "Special Case Performance Problems"	115

Figure 114. Sequential Processing - Actions

1. Minimize Impact of Sequential Processing

The first approach is to try to minimize the impact of the sequential processing without making major changes to your basic data or application designs. There are basically two ways you can try to do this:

a. Increase DISPBIAS

The simplest action you can try is to increase the DISPBIAS initialization parameter. This parameter controls the processing priorities for applications. A higher DISPBIAS value favors short applications (LUWs) over long running applications. Since sequential processing applications are long running, a high DISPBIAS will *disfavor* the sequential processing.

b. Data Reorganization

The next action you might consider would be data reorganizations that minimize the resources required by the sequential processing. Reconsider your indexing and clustering of the tables used by the sequential application.

You might also reconsider how tables are stored. This includes DBSPACES used, the amount of free space reserved, and row lengths for the table.

2. Reschedule Sequential Processing

If the previous approaches do not alleviate the problem, then you should consider running the problem sequential processing applications at a time when they will be of least impact to other work. This may not be a very attractive alternative, but it may be more practical than the next approach (which involves redesign of the applications).

3. Redesign Sequential Processing Application

If none of the previous actions can be applied, you may be able to re-design your sequential processing application to be "less sequential." There is not much you can do for DROP TABLE or UPDATE STATISTICS, but with the introduction of the REORGANIZE INDEX command, there is now an alternative for DROP INDEX, loading and CREATE INDEX.

You may also be able to do something with sequential DML applications.

You also should consider whether or not your problem is the "Buffer Pool Too Small" problem. The symptoms are nearly identical to the "Sequential Processing" problem. However, with the "Buffer Pool Too Small" problem, response time will usually be bad all of the time (as opposed to periodically high).

You might also want to investigate other causes of the periodic high response time symptom. See "Special Case Performance Problems" on page 115 for the list of these possible problems.

Session Limit Exceeded

Problem Description:

When an application requester uses AVS to communicate with a remote application server, a connection is initiated. If this connection causes the established session limit to be exceeded, AVS pends the connection. This state is maintained indefinitely until another session becomes available. A session becomes available when an existing connection is disconnected or terminates.

To interrupt this condition and return to CMS, re-IPL (#CP I CMS).

SET QDROP OFF USERS or SET QUICKDSP ON Not Used

Problem Description:

QDROP refers to a set of functions that are performed by the VM/SP and VM/ESA 370 Feature operating systems every time a virtual machine becomes idle. QDROP processing is normal and desirable for user virtual machines, but is inappropriate for service machines such as SQL/DS database machines. Consequently, VM provides the "SET QDROP OFF" CP command that causes most of the normal QDROP processing to be bypassed for the specified virtual machine.

Failure to specify "SET QDROP userid OFF USERS" for an SQL/DS database machine will result in a significant amount of unnecessary paging and CPU overhead.

SET QDROP OFF is set on the database machine so that machine and user pages remain in main storage between execution cycles. This reduces paging and CPU usage.

With VSE Guest Sharing, SET QDROP userid OFF USERS is considered mandatory for good performance.

For VM/XA and the VM/ESA ESA Feature systems, the "SET QUICKDSP userid ON" CP command provides similar functions to the "SET QDROP userid OFF USERS" command (see "Database Machine Favored Too Little" on page 139).

Possible Actions:

ACTION	PAGE
Specify SET QDROP OFF USERS or SET QUICKDSP ON	Below
SQL/DS Machine Favored Too Little	139

Figure 115. SET QDROP OFF USERS or SET QUICKDSP ON Not Used - Actions

The VM operator should specify "SET QDROP userid OFF USERS" or "SET QUICKDSP userid ON" for each database machine that is run in multiple user mode. The best time to do this is right after each database machine is logged on. Alternatively for VM/XA and the VM/ESA ESA Feature, each database machine directory can have "OPTION QUICKDSP" specified, eliminating the need for operator intervention.

SQL/DS Code Not Shared

Problem Description:

SQL/DS components are not placed in saved segments. The resulting inefficient real storage usage caused by duplicate copies of the code increased paging. It also increased the time required to start up the application server and to perform SQL/DS-related initializations in the user machines.

Possible Actions:

ACTION	PAGE
Install SQL/DS Components in Saved Segments	Below

Figure 116. SQL/DS Components Not Shared - Actions

Any SQL/DS component that is frequently used by more than one user should be installed in a discontinuous saved segment (DCSS) if possible. ISQL, the Resource Adapter, RDS, DBSS, DSC, and the SQL/DS message repositories are eligible to be installed as saved segments. All other SQL/DS components are not. The Resource Adapter and ISQL (if used) should be installed as saved segments for all multiple user environments. The RDS and DBSS should also be installed as saved segments if the installation is running multiple database machines on the same processor.

It is advantageous to load SQL/DS components in saved segments even if paging is not a problem, as the time required to load the code is bypassed. If ISQL and the Resource Adapter are in DCSSs, the time to enter the ISQL environment is greatly reduced. If RDS and DBSS are in DCSSs, the time to initialize SQL/DS is greatly reduced. An explanation of how to install in saved segments is provided in the *System Administration* manual.

Storage Pool Full

Problem Description:

If an LUW is forced to rollback because of a storage pool being full, then checkpoints may be triggered frequently during rollback. This can significantly degrade performance. This is most likely to occur if the LUW is loading a significant amount of data into one storage pool.

Possible Actions:

ACTION	PAGE
Avoid loading large amounts of data in a single LUW	Below
Add a DBEXTENT to the storage pool	Below

Figure 117. Storage Pool Full - Actions

1. It is better to load the data using several LUWs. This will prevent a long running rollback, thus preventing frequent checkpoints.
2. Add a DBEXTENT to the storage pool and if necessary, after the DATALOAD, it can be deleted.

Synchronous APPC/VM Not Used

Problem Description:

Higher than necessary CPU usage is occurring because of asynchronous communication between user machines and the database machine. The synchronous communication protocol is *not* the default protocol. Synchronous communication performs better than asynchronous, because asynchronous requires more operating system overhead.

There are restrictions with the synchronous protocol. SQLHX or CANCEL cannot be used to cancel SQL statements, and the SQLQRY command cannot be used.

Synchronous APPC/VM is not supported by VM/XA SP.

Possible Actions:

ACTION	PAGE
Change the communication protocol to synchronous	Below

Figure 118. Synchronous APPC/VM Not Used - Actions

The SYNChronous parameter of the SQLINIT EXEC determines the communication protocol to be used. Invoke the SQLINIT EXEC using the SYNC(YES) option. For more details on the SQLINIT EXEC, see the *Database Administration* manual.

Too Few Agents

Problem Description:

The number of agents available to process SQL/DS requests from users is determined by the NCUSERS initialization parameter. When NCUSERS is set too low, there are more user requests than there are agents available to process them. When all available agents are already in use, any additional incoming requests are queued up until one of these agents becomes free. If NCUSERS is much lower than required, this queueing time can greatly increase the overall response time experienced by the end users.

If repetitive executions of the SHOW USERS operator command often show that one or more of users are waiting for an agent, insufficient agents is a likely problem.

If VSE Guest Sharing is used, the number of links specified by NOLINKS is reserved until the online support is terminated via CIRT. Increasing NOLINKS will tie up more agents and may have a significant negative impact on CMS users of the database.

Possible Actions:

ACTION	PAGE
Increase NCUSERS	Below
Decrease Link Holding Time	Below
Decrease Online Resource Adapter NOLINKS	Below

Figure 119. Too Few Agents - Actions

1. Increase NCUSERS

If the paging rate and CPU utilization on the system are good, then you probably can afford to increase the NCUSERS initialization parameter. Note that increasing NCUSERS may require you to allocate additional virtual storage to the SQL/DS database machine.

2. Decrease Link Holding Time

It is often the case that the available agents get used up because agents are being held for unnecessarily long periods of time. This most commonly arises when an SQL/DS application can (at least sometimes) be in-LUW over a terminal read. If such cases are identified and corrected, it may not be necessary to increase NCUSERS. See "Agents Being Held" on page 119. Reducing link holding time will not help free agents reserved via the CIRB transaction for VSE Guests.

3. Decrease Online Resource Adapter NOLINKS

If VSE Guest Sharing is used, the VSE online resource adapter reserves agents using CIRB NOLINKS. These NOLINKS are "permanently" assigned to the VSE Guest from the NCUSERS pool, and are only surrendered for use by VM users of the shared database machine via the CIRT transaction. If it is not practical to increase NCUSERS, decrease the number of links in CIRB assigned to VSE Guest users.

Too Many Agents

Problem Description:

The number of agents established by the NCUSERS initialization parameter of SQL/DS determines the level of *concurrency* on your system. Specifically, it defines how many LUWs can be active at the same time. If you set the NCUSERS parameter too small, your SQL users may experience delays due to "link waits" (See "Too Few Agents" for more information on this case). On the other hand, you might experience "Paging" or "Lock Wait" problems if you set NCUSERS too high. That is, you can run SQL/DS with **too many agents**.

Running with a large number of agents (high NCUSERS) exposes your system to three possible problems:

1. Paging

Each agent (active LUW) can require up to 250K of virtual storage. With a large number of agents, you might cause a lot of page fault activity in the database machine. High paging can be very detrimental to the SQL/DS performance for *all* applications. See "Page Fault Serialization" on page 197 for more information on the effects of high paging rates.

2. Lock Contention (Lock Waits)

Each agent (active LUW) also represents a certain demand for locks in the database. If you have a wide variety of applications that access different data, this may not be a problem. But if your applications (or users) share a lot of common data, then a large number of agents could cause lock contention problems. These will show up as lock waits. See "Locking Related Performance Problems" on page 114 for more information on the types of lock wait problems you might introduce by having too many agents.

3. Buffer Contention (High I/O's)

A third possible problem that can be caused by having too many agents is contention for space in the buffer pools. Agents do not actually execute at the same time. Agent requests are processed one at a time. When one agent goes into a wait (lock wait, I/O wait or communication wait), another agent's request is executed. In effect, the agents take turns at getting SQL/DS execution time.

With a large number of agents active at the same time, several other agents may do some processing before any one agent gets its next turn. As a result, data that it has read into the SQL/DS buffer pools may get forced out between turns. This means that each time an agent gets a turn, it has to re-establish its data in the SQL/DS buffer pools. This is referred to as **Buffer Pool Thrashing**.

If you are experiencing a buffer contention problem, this will show up as a high number of I/O's for many of your applications, and a poor *buffer hit ratio*. However, buffer contention and a poor buffer hit ratio could also be due to a buffer pool that is too small. See "Buffer Pool Too Small" on page 128 for more information on this case.

Obviously, there are many reasons why you might see paging, lock wait, or high I/O problems on your system. In such cases, the problem is not necessarily the "Too Many Agents" problem. However, if you do have a large number of agents, or you think you may have more agents than you really need, then you might consider decreasing NCUSERS.

One way of telling whether or not you have more agents than you need is consider the amount of time users are spending in link wait. This can be done using the SHOW USERS operator command. If you are not experiencing a lot of link wait conditions, then you may not need as many agents as you have allocated.

In general, it doesn't matter *what* SQL statements or applications you are running for the "Too Many Agents" problem to show up. That is, it doesn't matter if you are performing SQL DDL, SQL DML or SQL control functions. If lock contention or buffer contention is a problem, then the problem is probably too many SQL users (agents).

However, the paging problem could also be as a result of other "non-SQL" applications on the system. That is, other applications in other machines could be placing enough of a demand on virtual storage that you cannot afford to use as many SQL/DS agents as you are using.

Possible Actions:

ACTION	PAGE
Decrease NCUSERS	Below
For other paging problems, see: - "Storage Related Performance Problems"	114

Figure 120. Too Many Agents - Actions

If the problem is indeed that you are running with too many agents, then the solution is to reduce the number of agents. Basically, this means decreasing NCUSERS. However, this is slightly more involved than simply decreasing the setting for NCUSERS. You may also want to decrease the setting for MAXCONN by a similar amount. Refer to the System Administration manual.

Also, if NCUSERS are to be reduced, it may also be necessary to reduce the Online Resource Adapter NOLINKS (when VSE Guests are used). Otherwise, you may find that there will now be *too few agents* (See "Too Few Agents" on page 205 for more information on this case).

If you are not sure that you have too many agents, then you should also review the other problems that can cause paging. See "Storage Related Performance Problems" on page 114 for the list of possible paging problems.

Too Many Joins

Problem Description:

Although you can easily combine data from multiple tables, such activity is not without its costs. In particular, queries or subqueries that are joins of many tables may require a large number of database I/O's, and may require a large amount of CPU time.

If you have a query or subquery that takes a long time to execute, you could be asking the system to do too many joins. "Too Many Joins" could be your problem if the problem request causes a large number of database I/O's, or uses a lot of CPU time to execute. You should also observe no particular improvement when you run the request with no one else on the system. (That is, the single user response time is also poor).

If you suspect your problem could be the "Too Many Joins" problem, the best way to substantiate this is through the EXPLAIN PLAN function. If you EXPLAIN the query, you can determine the joins being done by inspecting the METHOD column in the resulting PLAN table entries. For information on using the EXPLAIN statement to determine the number of joins you have and how they are being done, see the *SQL Reference* manual.

Note: If the problem is with a Format 2 INSERT, then do the EXPLAIN PLAN analysis on the *subquery* of the INSERT. Similarly, if the problem is on a

DATAUNLOAD command, do the EXPLAIN PLAN analysis on the DATAUNLOAD subquery.

Before you conclude that you are doing too many joins, you may want to make sure that you don't have one of the other problems that would give you high I/O's. In particular, before you conclude that you have too many joins, you want to make sure you have given the database machine every opportunity to make your query perform well. If, for example, the EXPLAIN analysis of the query indicates that one or more of the joined tables is being accessed by a DBSPACE scan, you may want to consider adding indexes, rather than re-phrasing your query.

Note that the "Too Many Joins" problem can occur on a *subquery*, as well as a query. This means you can have this problem with a Format 2 INSERT as well as a SELECT statement. You can also see this problem if you have subqueries involving joins on UPDATE or DELETE statements. However, usually you will see this problem with your SELECT statements.

Possible Actions:

ACTION	PAGE
Index Tables to Avoid DB I/O's	Below
Reorganize Data to Avoid DB I/O's	Below
Use Redundant Data to Avoid DB I/O's	Below
Combining Data in the Application	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 121. Too Many Joins - Actions

There are basically three approaches you can take for the "Too Many Joins" problem:

1. Reorganize the Joined tables

In this case, you would be trying to reduce the number of database I/O's by finding a more effective organization. In its simplest form, this means making sure you have all the needed indexes. However, it may be necessary to reconsider how you have tables clustered or where (in what DBSPACES) you have them stored.

2. Use of Redundant Data to avoid DB I/O's

If the data reorganization techniques described above do not help, more radical action may be called for. In particular, you may have to resort to some forms of *redundant data*, in order to avoid the database I/O's. Some types of redundant data that might be considered include:

a. Stored Results

Rather than join tables to get all the column information desired, you may want to consider duplicating frequently used columns in the tables in which it is used. Updating of data can become more involved and you

will use more DASD space, but you can significantly reduce the number of joins required in most cases.

b. Table Splitting

You can also reduce the number of I/O's to do your joins by splitting tables, and thereby reducing the number of data pages required for each table. If you have tables with a large number of columns, this technique can be very effective at reducing the number of I/O's to do the joins. You would split the table based on column usage such that columns used together are stored together in the same table.

3. Combine the Data in the Application

Lastly, if all else fails, you will need to reconsider the implementation of your application. In certain cases, combining data in the application can be more efficient than having the database manager do it through join operations.

UPDATE STATISTICS by DATALOAD

Problem Description:

When UPDATE STATISTICS is set OFF, the DBS utility will issue an UPDATE STATISTICS statement following a DATALOAD. This can adversely affect performance.

When UPDATE STATISTICS is set ON, catalog statistics are automatically accumulated during DATALOAD. This avoids the cost of a DBSPACE scan.

First, when UPDATE STATISTICS is set OFF, a DBSPACE scan is performed. This can be quite time-consuming if the number of active data pages in that DBSPACE is large. Note that this can be the case even if the target table is small if other large tables also reside in the same DBSPACE. This can be quite expensive in terms of database I/O and CPU time. Second, the UPDATE STATISTICS updates the catalogs. This requires some exclusive locks in the catalogs, which may result in long lock waits.

Possible Actions:

ACTION	PAGE
Set UPDATE STATISTICS Off	Below
Set UPDATE STATISTICS On	Below
Put Each Large Table in its Own DBSPACE	Below

Figure 122. UPDATE STATISTICS by DATALOAD - Actions

1. Set UPDATE STATISTICS Off

Specify "SET UPDATE STATISTICS OFF". This is appropriate if there is not a large number of DATALOADs against the same target table, or if the DATALOADs are against different target tables. UPDATE STATISTICS could be executed after the last one, or on a periodic basis. It is also appropriate if you know the statistics are not going to change significantly. In that case you could postpone updating the statistics until more substantial changes have occurred.

However, this can be costly due to the DBSPACE scan required by the UPDATE STATISTICS statement.

2. Set UPDATE STATISTICS On

This is especially appropriate if there are a large number of DATALOADs against one target table.

3. Put Each Large Table in its Own DBSPACE

Each large table should be placed in its own DBSPACE. If desired, a number of small tables can be placed in the same DBSPACE, as the time to do a DBSPACE scan will still be short. It is best to provide each such table with an index so that a DBSPACE scan can be avoided for normal table accesses.

Very Nonunique Index Key Prefix

Problem Description:

Sometimes the first six or more characters of a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC column usually have the same value. For example, numbers stored in character form often have many leading (high order) zeros. These repeated characters can make it difficult to determine the best access path. This is especially true for table access based of a WHERE clause containing an inequality (for example, WHERE ACTNO < '0000000090').

Possible Actions:

ACTION	PAGE
Switch to Numeric Data Type	Below
Remove Repeating Characters	Below
Add a Redundant Column	Below
For other high CPU usage problems, see "CPU Related Performance Problems"	112
For other high I/O problems, see "I/O Related Performance Problems"	113

Figure 123. Very Nonunique Index Key Prefix - Actions

1. Switch to Numeric Data Type

If the column consists of numbers stored in character form, redefine the column using the appropriate numeric data type (generally INTEGER or DECIMAL). The specific steps involved are:

- a. Use the DBS utility's DATAUNLOAD facility to put the table's contents on an external file.
- b. Drop and recreate the table, changing the data type of the affected column(s).
- c. Use DATALOAD to reload the table. DATALOAD will handle the data type conversions.
- d. Recreate any indexes and views that were on the table and restore table authorizations.

2. Remove Repeating Characters

If the first several character positions are always the same, it may be feasible to redefine the column with a shorter length and exclude these characters. To do this, follow the steps shown above under "Switch to Numeric Data Type", changing the table-column identification for that column so as to exclude the repeating characters.

3. Add a Redundant Column

If the above suggestions are not applicable, consider adding an equivalent column that excludes the leading character positions. Drop the index on the original column and create an index on the new column. Finally, whenever you have a predicate that refers to the original column, add the equivalent predicate that refers to the new column.

Chapter 6. Recovering from DBSS Errors

Some SQL/DS abnormal ends are accompanied by message ARI0126E. This message, which is followed by diagnostic information, indicates that a failure occurred during DBSS database access (DBSPACE) processing. DBSS (Database Storage Subsystem) is the component that physically accesses the database.

If message ARI0126E is displayed, it may be possible to recover from the error and continue processing. If message ARI0126E is displayed:

1. Interpret the diagnostic information that follows message ARI0126E (see "Interpreting the Diagnostic Display").
2. You may need/be able to recover from the DBSS error (see "Filtered Log Recovery" on page 224)
3. Follow the instructions in Chapter 3, "Reporting Defects" on page 69 to report the defect to IBM.

If you did not receive message ARI0126E when the DBSS failure occurred, you cannot perform recovery as described in this chapter. You can be confident that the failure was not caused by bad data in DBSPACES or DBEXTENTS. In this case:

1. Perform the Directory Verify function (see Chapter 7, "Recovering from Directory Verify Errors" on page 243 for details)
2. Follow the instructions in Chapter 3, "Reporting Defects" on page 69 to report the defect to IBM.

Interpreting the Diagnostic Display

Figure 124 and Figure 125 on page 214 show examples of diagnostic displays that can follow message ARI0126E.

The action you should take to recover from the failure varies depending on whether message ARI0126E says FORWARD, ROLLBACK, UNDO, or REDO. Each action is described in topics starting with "Action to Take for FORWARD Processing Failures" on page 219. Before reading those descriptions, however, familiarize yourself with the meanings of the text and fields of the diagnostic display. The meanings of the text are given following Figure 125 on page 214, and the meanings of the fields are shown in Figure 126 on page 215.

Always save the diagnostic output.

```
ARI0126E DBSS TERMINATION DURING FORWARD PROCESSING
LUWID = 7B9                      USERID = SMITH
OPERATION = CREATE INDEX         DBSPACE = 10
TABLE-ID = -32765 (8003)        INDEX-ID = -32762 (8006)
PAGE-ADDRESS = 1EA000          PAGE-TYPE = INDEX
PAGE-NUMBER = 5A3
```

Figure 124. Example of Output Caused by a Forward Processing Failure

```
ARI0126E DBSS TERMINATION DURING ROLLBACK PROCESSING
LUWID = 7B9                      USERID = SMITH
DATE = 12-01-85                  TIME = 12:12:45
OPERATION = CREATE INDEX         DBSPACE = 10
TABLE-ID = -32765 (8003)         INDEX-ID = -32762 (8006)
PAGE-ADDRESS = 1EA000           PAGE-TYPE = INDEX
PAGE-NUMBER = 5A3
```

Except for ROLLBACK in the message text of ARI0126E, the displays for UNDO and REDO are identical to the display above.

Figure 125. Example of Output Cause by a Rollback Processing Failure

The following texts can be displayed for message ARI0126E:

DBSS TERMINATION DURING FORWARD PROCESSING

An error occurred while DBSS was performing an operation that was accessing data in the database (DBSPACE data). FORWARD indicates that the database manager was performing normal database activity on behalf of an SQL/DS application program or terminal user.

DBSS TERMINATION DURING ROLLBACK PROCESSING

An error occurred while the DBSS was trying to undo a database update previously made by the LUW identified by LUWID. ROLLBACK indicates that the database manager is performing log recovery for an LUW that failed because the application or terminal user or SQL/DS itself initiated the ROLLBACK WORK process. The error occurred while DBSS was accessing data in the database (DBSPACE data).

DBSS TERMINATION DURING UNDO PROCESSING

An error occurred during the DBSS Log Recovery phase of warm start or restore from archive. DBSS was trying to undo a database update previously made by the uncommitted LUW identified by LUWID. The error occurred while DBSS was accessing data in the database (DBSPACE data).

Note: This error can occur if a committed logical unit of work is rolled back via the ROLLBACK COMMITTED WORK command. See "Rolling Back Committed Work" on page 231.

DBSS TERMINATION DURING REDO PROCESSING

An error occurred during the DBSS Log Recovery phase of warm start or restore from archive. DBSS was trying to redo a database update previously made by the committed LUW identified by LUWID. The error occurred while DBSS was accessing data in the database (DBSPACE data).

These types of errors are caused by:

- Bad data in the database.
- Data in the database that cannot be handled by DBSS code.
- Bad log records.
- Log records that cannot be correctly processed by the SQL/DS code.

- Log record data that has somehow become unsynchronized with the data in the database. For example, the log record directs the DBSS to delete a row that does not exist in the database.

In any case, DBSS has encountered a "should not occur" condition and cannot complete its processing.

Field	Meaning
LUWID	Internal logical unit of work identifier. (The same identifier that is displayed in SHOW operator commands and in the output of a DBSS trace.) This identifier can be specified with the LUWID control keyword of the ROLLBACK COMMITTED WORK and BYPASS UNDO WORK commands (see "Filtered Log Recovery" on page 224).
USERID	SQL/DS userid. It identifies the user whose database access request failed in the DBSS. This userid can be specified with the USERID control keyword of the ROLLBACK COMMITTED WORK and BYPASS UNDO WORK commands (see "Filtered Log Recovery" on page 224).
DATE	Date that the log record was created in the form mm-dd-yy. This value could be used with the DATE control keyword of the ROLLBACK COMMITTED WORK command. This field is displayed only for ROLLBACK, UNDO, and REDO errors.
TIME	Time that the log record was created in the form hh:mm:ss. This value could be used with the TIME control keyword of the ROLLBACK COMMITTED WORK command. This field is displayed only for ROLLBACK, UNDO, and REDO errors.

Figure 126 (Part 1 of 5). Meanings of Fields in the Diagnostic Display

Field	Meaning
OPERATION	<p>DBSS operation that failed. Can be:</p> <ul style="list-style-type: none"> • DELETE - delete a row from a table • FETCH ROW - retrieve a row from a table (DBSS FETCH) • INSERT - insert a row into a table* • FETCH CURSOR - move the internal cursor to the next row in a table and retrieve the row (DBSS NEXT)* • OPEN CURSOR - open a cursor on a table* • UPDATE - update a row • FETCH SCR MCR ICR LCR - fetch a data control record. • NEXT SCR MCR ICR LCR - move the internal cursor to the next data control record. • CREATE TABLE - create a table (CINSERT MCR). • CREATE LIST - create a temporary list (CINSERT MCR) • ACQUIRE DBSPACE - acquire a DBSPACE (CINSERT SCR). • CREATE LINK - create a link (CINSERT LCR). Links are used internally. They have no external equivalent. • CREATE INDEX - create an index on a table (CINSERT ICR). • DROP TABLE - drop a table (CDELETE MCR). • DROP DBSPACE - drop a DBSPACE (CDELETE SCR). • DROP LINK - drop an internal link (CDELETE LCR). • DROP INDEX - drop a table index (CDELETE ICR). • ALTER TABLE - add a column to a table (CUPDATE MCR). • ALTER DBSPACE - change the lock size or PCTFREE of a DBSPACE (CUPDATE SCR). • ALTER INDEX - alter an index on a table. ALTER INDEX is caused by an ALTER DBSPACE command with the LOCK option specified. (Internally a CUPDATE ICR.) • SORT - DBSS sort. • UPDATE STATISTICS - either start or end UPDATE STATISTICS processing. Can be caused by either an UPDATE STATISTICS command or by a CREATE INDEX command. • ***** - due to an SQL/DS internal error, the DBSS operation cannot be determined. In this case TABLE-ID and INDEX-ID are not displayed. <p>* If the displayed DBSPACE value indicates INTERNAL, the table is actually a temporary list of rows.</p>

Figure 126 (Part 2 of 5). Meanings of Fields in the Diagnostic Display

Field	Meaning
DBSPACE	<p>The number of the DBSPACE. This number is the same number used in the DBSPACENO column of the SYSDBSPPACES catalog. If the number is followed by "INTERNAL", the DBSPACE is an INTERNAL DBSPACE. Otherwise, the DBSPACE is either PUBLIC or PRIVATE.</p> <p>If DBSPACE is followed by ***** , the DBSPACE value could not be determined due to an internal SQL/DS error. Because of this error, TABLE-ID and/or INDEX-ID may not be displayed.</p> <p>If the DBSPACE is PUBLIC or PRIVATE, and the operation is SORT, then the DBSPACE is the source for the sort. The sort itself takes place in INTERNAL DBSPACES. In addition to SORT, the following DBSS operations (see OPERATION value) can access INTERNAL DBSPACES:</p> <p style="padding-left: 40px;">INSERT FETCH CURSOR OPEN CURSOR FETCH SCR MCR NEXT SCR MCR CREATE LIST ACQUIRE DBSPACE</p>
TABLE-ID	<p>The internal table identifier in decimal followed by the same value in hexadecimal in parentheses. The DBSS internal table identifier, also known as the relation identifier (RID), is the same value that appears in the TABID column of the SYSCATALOG table. (If the table is a secondary table that is used to store LONG VAR data type fields, the RID corresponds to the LFDTABID column in SYSCATALOG.) In these cases, the DBSPACE number must match the DBSPACENO column value in SYSCATALOG. If the DBSPACE is a package DBSPACE (DBSPACE is SYS000n and n > 1), the RID corresponds to the TABID column in the SYSACCESS table and the displayed DBSPACE number corresponds to the DBSPACENO column.</p> <p>If the operation is not addressing a table at the time of the error, TABLE-ID is not displayed. TABLE-ID is not displayed for UPDATE STATISTICS or for INTERNAL DBSPACES.</p> <p>If ***** is displayed, an internal SQL/DS error caused an invalid table identifier.</p>

Figure 126 (Part 3 of 5). Meanings of Fields in the Diagnostic Display

Field	Meaning
INDEX-ID	<p>The internal index identifier (IID) in decimal, followed by the same value in hexadecimal in parentheses.</p> <p>You can use the displayed DBSPACE number and TABLE-ID identifier to determine the table name from SYSCATALOG. Then use the table name and INDEX-ID identifier to locate the index name in SYSINDEXES (submitting TNAME and IID and retrieving INAME and ICREATOR).</p> <p>INDEX-ID is displayed only for:</p> <ul style="list-style-type: none"> • Data manipulation and sort operations that have accessed at least one index page. • DROP INDEX, ALTER INDEX, and FETCH ICR. • CREATE INDEX and NEXT ICR (but only if a value is available). For CREATE INDEX, INDEX-ID is displayed only if an IID was assigned to the index by the time the failure occurred. <p>If ***** is displayed, an invalid index identifier was found. The invalid identifier is caused by an internal SQL/DS error.</p>
PAGE-ADDRESS ³	<p>The virtual storage address of the last DBSPACE page retrieved from the database by the DBSS operation. (This is the address of the DBSPACE page in the SQL/DS dump.)</p> <p>If the PAGE-ADDRESS value is "NONE," this DBSS operation has not retrieved any DBSPACE pages from the database. Note that NONE means no data was accessed in any DBSPACE page.</p> <p>If PAGE-ADDRESS is NONE, none of the remaining fields are displayed. (They do not apply.)</p>

Figure 126 (Part 4 of 5). Meanings of Fields in the Diagnostic Display

Field	Meaning
PAGE-TYPE ³	<p>Describes the type of DBSPACE page whose address is displayed by PAGE-ADDRESS above:</p> <p>EMPTY means that the page is a newly-allocated DBSPACE page that had not been formatted at the time of the failure. (After formatting, the page would have had one of the following types.)</p> <p>HEADER means the page is a header page for either a PUBLIC, PRIVATE, or INTERNAL DBSPACE. DBSPACE header pages contain DBSPACE data control records.</p> <p>DATA means that the page contains stored rows of tables.</p> <p>INDEX means that the page contains root, leaf, or non-leaf information for an index on a table in this DBSPACE.</p> <p>LIST means this page is an INTERNAL DBSPACE page that contains either the output of a DBSS sort operation or temporary data stored in a list by RDS.</p> <p>SORT means that the page is an INTERNAL DBSPACE page that contains the intermediate results of a DBSS sort. (Intermediate results are eventually merged into list pages for the final sort result.)</p> <p>***** means that an invalid page type was found. (See note below.)</p> <p>Note: For forward and rollback DBSS processing in multiple user mode, it is possible for the buffer containing the last retrieved DBSPACE to have been freed for reuse and to have been reused by another agent. If this occurs, its contents (and the contents of PAGE-TYPE) will not be meaningful. Further, the buffer could contain a log page, which would cause unpredictable values for PAGE-TYPE and PAGE-NUMBER.</p>
PAGE-NUMBER ³	<p>A hexadecimal number that represents the internal address that the DBSS uses to access the DBSPACE page whose address is displayed in PAGE-ADDRESS. If PAGE-TYPE is ***** , the PAGE-NUMBER number may not be valid.</p> <p>PAGE-NUMBER is not displayed if PAGE-TYPE is EMPTY.</p>

Figure 126 (Part 5 of 5). Meanings of Fields in the Diagnostic Display

Actions to take for each of the processing failure types are given under:

- FORWARD processing — "Action to Take for FORWARD Processing Failures."
- ROLLBACK processing — "Action to Take for ROLLBACK Processing Failures" on page 221.
- UNDO processing — "Action to Take for UNDO Processing Failures" on page 221.
- REDO processing — "Action to Take for REDO Processing Failures" on page 223.

Action to Take for FORWARD Processing Failures

When your SQL/DS application server fails because of a DBSS forward processing failure, save the diagnostic output from the failure. Run the Directory Verify function (see Chapter 7, "Recovering from Directory Verify Errors" on page 243). If Directory Verify finds discrepancies, perform the recovery described under "Recovery Actions for an Inconsistency" on page 243. Other-

³ These fields (PAGE-ADDRESS, PAGE-TYPE, and PAGE-NUMBER) are primarily to assist IBM support personnel in problem determination.

wise, restart the application server and allow normal database access. If the problem persists, compare the output from all the failures and try to find one of the following patterns (there is another alternative described after step 4):

1. The DBSPACE numbers, the TABLE-ID values, and the INDEX-ID values in the diagnostic displays are always the same.

Drop and re-create the index. If this index is a primary key or unique key index, use the DEACTIVATE PRIMARY/UNIQUE KEY clause of the ALTER TABLE statement to drop the index, and then use the ACTIVATE clause to re-create the index. If the problem still persists, take the action in the next step (below).

2. The DBSPACE numbers and the TABLE-ID values in the diagnostic displays are always the same.

Issue the following SELECT statement to determine the name of the table that is causing the failure:

```
SELECT CREATOR, TNAME
FROM SYSTEM.SYSCATALOG
WHERE TABID=table-id-value
AND DBSPACENO=dbspace-no
```

Prevent access to the table. If a known set of users access the table, you can prevent access by "word of mouth." Otherwise, use DBA authority to revoke all access to the table. Also, deactivate the primary and foreign keys on the table. If the primary keys or referential constraints exist for this table you should use the DEACTIVATE ALL clause of the ALTER TABLE statement to restrict access to the table. Because the table creator and other users with DBA authority will still have access to the table, you'll just have to ask them not to access it.

After preventing access to the table, and the table's referential constraint, allow normal access to the rest of the database. Then follow the instructions in Chapter 3, "Reporting Defects" on page 69 to report the problem to IBM. Once the problem is corrected, allow normal access to the table.

3. The DBSPACE numbers in the diagnostic displays are always the same but the TABLE-ID values are variable.

In this case, prevent further access to the DBSPACE by using the DISABLE DBSPACE command. Then follow the instructions in Chapter 3, "Reporting Defects" on page 69 to report the problem to IBM. Once the cause of the problem is determined and corrected, you can allow access to the DBSPACE by using the ENABLE DBSPACE command. See "Filtered Log Recovery" on page 224 for instructions on disabling and enabling DBSPACEs. If the corrective action could have invalidated any of the referential constraints they should all be deactivated and reactivated.

4. Nothing seems to match in the diagnostic displays except the userid and the log(s).

In this case, ask the user to stop running the application that appears to be causing the failure. Then follow the instructions in Chapter 3, "Reporting Defects" on page 69 to report the problem to IBM.

Another alternative is to try restoring the database from the last archive copy. This often eliminates the bad data that is causing the failure. If your last database archive was quite recent, it may contain the bad data. If you are using log archiving, you may restore the database from the next-to-last archive copy and

subsequent log archives. If you are using database archiving without log archiving, you may restore the next-to-last archive copy, but this will lead to a back-level database.

Action to Take for ROLLBACK Processing Failures

When your SQL/DS application server fails because of a DBSS ROLLBACK processing failure, warm start it. There is a good chance that the failure will not recur during a warm start. If it does recur, the diagnostic display output will be the same except that the message will say UNDO instead of ROLLBACK.

For persistent ROLLBACK failures, follow the instructions for FORWARD processing failures (previous topic).

Action to Take for UNDO Processing Failures

UNDO processing can fail during either a warm start or a restore from archive. UNDO processing can occur on a restore only if the database archive was created while users were updating the database. The action you should take is different for each case, as the following two sections describe.

UNDO Processing Failure During a Warm Start

When your SQL/DS application server fails because of a DBSS UNDO processing failure during a warm start (STARTUP=W), do the following:

1. Restart with STARTUP=W, but bypass the UNDO processing for the logical unit of work that is causing the failure.

Bypassing the UNDO processing for the logical unit of work allows you to get the application server operational quickly. Recovery processing is only done for logical units of work that do not cause DBSS errors.

Before bypassing UNDO processing, however, be sure you are aware of the possible database integrity and recovery consequences. For a specific logical unit of work that you wish to bypass, use the internal identifier of the logical unit of work as input to the BYPASS UNDO WORK command. The LUWID value in the diagnostic display is that identifier. Invoking BYPASS UNDO WORK and the consequences of doing so are discussed under "Filtered Log Recovery" on page 224 and "Filtered Log Recovery and Referential Integrity" on page 237.

2. If you bypass one logical unit of work, but others fail, perform the same action as above. Instead of specifying a single logical unit of work identifier on the BYPASS UNDO WORK command, however, specify:
 - The DBSPACE number if all failures are in the same DBSPACE.
In this case, you should also prevent further access to the DBSPACE using the DISABLE DBSPACE command.
 - The USERID value if all the failures display the same userid.
 - Multiple logical unit of work identifiers (or the ALL option) if there are multiple UNDO failures and there is no matching pattern on the displayed DBSPACE number or USERID.

An alternative to bypassing the UNDO processing is to restore the database from the last archive copy and the log(s). You can do the restore even after you do a warm start with bypass processing. The UNDO processing failure is not likely to occur during a restore because restore processing does not UNDO work. Instead, the logical unit of work is bypassed when the log is being processed. Also, if the problem is caused by bad data in the database, the restore process is likely to eliminate the bad data. This is because the restore process will not redo any LUW that did not commit. Therefore, if an LUW partially committed, and in so doing committed bad data (then the system crashes during undo), the restore process would skip the LUW and not insert the bad data. The disadvantage to restoring the database is that it takes much longer than a restart using `BYPASS UNDO WORK` commands. A restart using `BYPASS UNDO WORK` commands takes about as long as a normal warm start.

UNDO Processing Failure During a Restore

When the SQL/DS application server fails because of a DBSS UNDO processing failure during an SQL/DS archive restore, do the following:

1. Restart the restore from archive process, but bypass the UNDO processing for the logical unit of work that is causing the failure. You restart the restore process via warm start (`STARTUP=W` and `LOGMODE=your-current-value`).

Bypassing the UNDO processing for the logical unit of work allows you to restore the database and get it operational quickly. The database manager does recovery processing for logical units of work that do not cause DBSS errors.

Before bypassing UNDO processing, however, be sure you are aware of the possible database integrity and recovery consequences. For a specific logical unit of work you wish to bypass, use the internal identifier of the logical unit of work as input to the `BYPASS UNDO WORK` command. The LUWID value in the diagnostic display is that identifier. Invoking `BYPASS UNDO WORK` and the consequences of doing so are discussed under "Filtered Log Recovery" on page 224 and "Filtered Log Recovery and Referential Integrity" on page 237.

2. If you bypass one logical unit of work, but others fail, perform the same action as above. Instead of specifying a single logical unit of work identifier on the `BYPASS UNDO WORK` command, however, specify:
 - The DBSPACE number if all failures are in the same DBSPACE.
In this case, you should also prevent further access to the DBSPACE using the `DISABLE DBSPACE` command.
 - The USERID value if all the failures display the same userid.
 - Multiple logical unit of work identifiers (or the ALL option) if there are multiple UNDO failures and there is no matching pattern on the displayed DBSPACE number or USERID.

Action to Take for REDO Processing Failures

REDO processing can fail during either a warm start or a restore from archive. The action you should take is different for each case, as the following two sections describe.

REDO Processing Failure During a Warm Start

When the SQL/DS application server fails because of a DBSS REDO processing failure during an SQL/DS warm start (STARTUP=W), do the following:

1. Restart the SQL/DS application server with STARTUP=W, but use the ROLLBACK COMMITTED WORK command for the logical unit of work that is causing the failure.

Rolling back the committed work for the logical unit of work circumvents the DBSS processing that caused the error. ROLLBACK COMMITTED WORK allows you to get the database operational quickly. The database manager does recovery processing for logical units of work that do not cause DBSS errors.

Before using ROLLBACK COMMITTED WORK, however, be sure you are aware of the possible database integrity and recovery consequences. For a specific logical unit of work that you want to roll back, use the internal identifier of the logical unit of work as input to the ROLLBACK COMMITTED WORK command. The LUWID value in the diagnostic display is that identifier. Invoking ROLLBACK COMMITTED WORK and the consequences of doing so are discussed under "Filtered Log Recovery" on page 224 and "Filtered Log Recovery and Referential Integrity" on page 237.

2. If you bypass one logical unit of work, but others fail, perform the same action as above. Instead of specifying a single logical unit of work identifier, however, specify:
 - The DBSPACE number if all failures are in the same DBSPACE.
In this case, you should also prevent further access to the DBSPACE using the DISABLE DBSPACE command.
 - The USERID value if all failures display the same userid value.
 - Multiple logical unit of work identifiers if there are multiple REDO failures and there is no matching pattern on the displayed DBSPACE number or USERID.

An alternative to rolling back the committed work is to restore the database from the last archive copy and the log(s). (If your last database archive was quite recent, it may contain the bad data. If you are using log archiving, you may restore the database from the next-to-last archive copy and subsequent log archives.) You can do the restore even after you do a warm start with ROLLBACK COMMITTED WORK processing. However, the logical unit(s) of work that were rolled back via the ROLLBACK COMMITTED WORK command will not be redone, and thus their database updates will be bypassed (lost). The REDO processing failure is not likely to occur during a restore if the failure was caused by corrupt or very unusual (in its stored format) data in the database. Also, if the problem is caused by bad data in the database, the restore process is likely to eliminate the bad data. The disadvantage to restoring the database is that it takes much longer than a warm start using ROLLBACK COMMITTED WORK

commands. A warm start using ROLLBACK COMMITTED WORK commands takes about as long as a normal warm start.

REDO Processing Failure During a Restore

When the SQL/DS application server fails because of a DBSS REDO processing failure during an SQL/DS archive restore, do the following:

1. Restart the restore from archive process, but use the ROLLBACK COMMITTED WORK command for the logical unit of work that is causing the failure. You restart the restore process via warm start (STARTUP=W and LOGMODE=your-current-value).

Rolling back the committed work for the logical unit of work circumvents the DBSS processing that caused the error. ROLLBACK COMMITTED WORK allows you to get the database operational quickly. The database manager does recovery processing for logical units of work that do not cause DBSS errors.

Before using ROLLBACK COMMITTED WORK, however, be sure you are aware of the possible database integrity and recovery consequences. Use the internal identifier of the logical unit of work that you want to roll back as input to the ROLLBACK COMMITTED WORK command. The LUWID value in the diagnostic display is that identifier. Invoking ROLLBACK COMMITTED WORK and the consequences of doing so are discussed under "Filtered Log Recovery," and "Filtered Log Recovery and Referential Integrity" on page 237.

2. If you bypass one logical unit of work, but others fail, perform the same action as above. Instead of specifying a single logical unit of work identifier, however, specify:
 - The DBSPACE number if all failures are in the same DBSPACE.
In this case, you should also prevent further access to the DBSPACE using the DISABLE DBSPACE command.
 - The USERID value if all failures display the same userid value.
 - Multiple logical unit of work identifiers if there are multiple REDO failures and there is no matching pattern on the displayed DBSPACE number or USERID.

Filtered Log Recovery

The process of invoking the application server to recover from a DBSS error is known as Filtered Log Recovery. This process is performed by invoking the application server with Extended Processing. The Filtered Log Recovery process can consist of steps which allow you to:

Bypass UNDO processing

Rollback work that was committed

Disable a DBSPACE

Enable a DBSPACE.

The following diagram illustrates the LUW recovery process. The checkpoint shown is referred to as the base checkpoint for recovery since it is the last checkpoint taken before the system failure. At this checkpoint the database

manager writes all database changes to DASD. The diagram describes the actions taken during warmstart recovery when Filtered Log Recovery is **not** used.

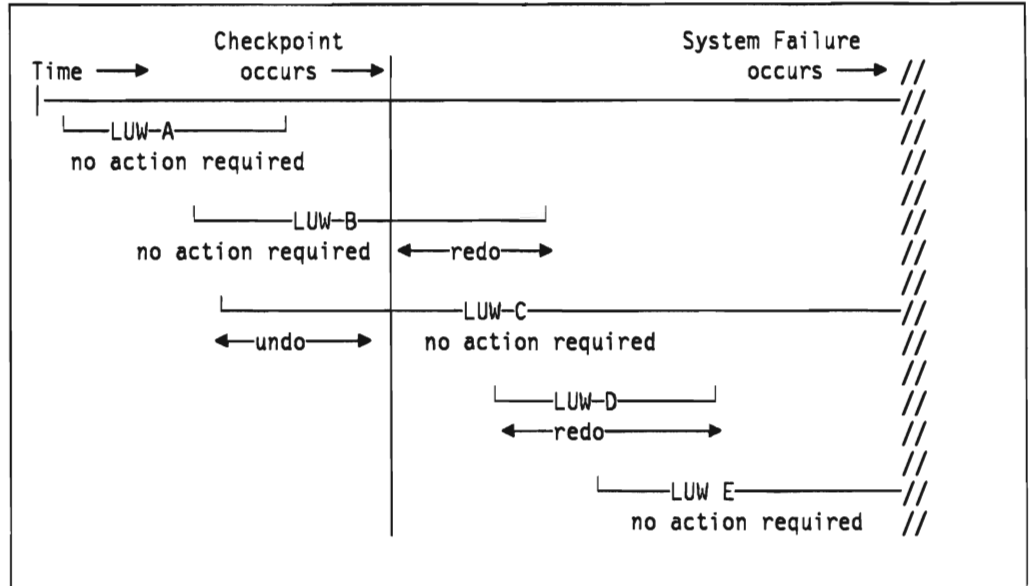


Figure 127. LUW Recovery Actions

Filtered Log Recovery allows you to do the following:

- LUW A (or completed LUWs in previous log archives) can be backed out using the ROLLBACK COMMITTED WORK command
- The committed updates from LUW B can be backed out using the ROLLBACK COMMITTED WORK command
- Recovery can be bypassed for LUW C with the BYPASS UNDO WORK command
- LUW D can be ignored using the ROLLBACK COMMITTED WORK command.

After performing Filtered Log Recovery, database integrity may be impacted in three ways:

1. Updates from uncommitted LUWs may be left in the database
2. Updates from LUWs may be based on LUWs that have been rolled back
3. Referential constraints may no longer hold true. You will have to reassert all relationships in the affected tables.

For these reasons, **call IBM support personnel for assistance before attempting to use Filtered Log Recovery.**

Extended Processing

The EXTEND input file commands will allow you to perform Filtered Log Recovery. To invoke the SQL/DS application server to recover from a DBSS error, specify the EXTEND=Y initialization parameter. EXTEND=Y specifies that you want it to use extended SQL/DS initialization to process the EXTEND input file. You can specify EXTEND=Y only if STARTUP is W or R or U. Otherwise, SQL/DS initialization ends. The default for EXTEND is N.

```
FILEDEF ARIEXTND DISK ZAP14 DATA A
SQLSTART DB(PROD) PARM(EXTEND=Y)

Contents of file ZAP14 DATA A:

BYPASS UNDO WORK WHERE
  USERID SCOTT
  DBSPACE 14 - 128
DISABLE DBSPACE 14

Commands
```

Figure 128. Example of Invoking your Application Server to Process EXTEND Input File

Figure 128 is an example of invoking your application server to process EXTEND input file commands. EXTEND=Y indicates that EXTEND input file commands are to be processed.

The CMS FILEDEF command identifies the EXTEND input file. The file you identify must have a fixed record length of 80 bytes. There are no other restrictions on the file. The ddname in the FILEDEF must be ARIEXTND, as shown. The commands in the EXTEND input file must all be in upper case.

You can specify the commands anywhere in columns 1 to 72 of the input records. The commands must be on lines by themselves.

Note: Comments are allowed. They must begin with an asterisk (*) in column 1. Blank lines are also permitted.

When EXTEND input file commands are read, they are displayed on the operator console.

The BYPASS UNDO WORK and ROLLBACK COMMITTED WORK commands have *control keywords* associated with them. A control keyword and its associated parameter strings must be entirely on a single separate line. They cannot be on the same line as the command.

Neither the commands nor the control keywords can span input records. Except for the "TIME" and "DATE" options in the ROLLBACK COMMITTED WORK command, a control keyword can be used only once per command. Both of these examples of the BYPASS UNDO WORK command are incorrect:

```
BYPASS UNDO WORK WHERE
  USERID JONES KROL SCOTT THOMPSON TRACY SMITH WALTERS HENRY
  COMO TOMS WALKER WARD SOLE HALL
```

(Error: USERID parameters span an input record.)

```
BYPASS UNDO WORK WHERE
  USERID JONES KROL SCOTT THOMPSON TRACY SMITH WALTERS HENRY
  USERID COMO TOMS WALKER WARD SOLE HALL
```

(Error: Duplicate control keywords used in one BYPASS UNDO WORK command.)

The correct way to bypass processing for those users is:

```
BYPASS UNDO WORK WHERE
  USERID JONES KROL SCOTT THOMPSON TRACY SMITH WALTERS HENRY
BYPASS UNDO WORK WHERE
  USERID COMO TOMS WALKER WARD SOLE HALL
```

Summary Messages: The database manager analyzes the log to determine which logical units of work will be affected by the BYPASS UNDO WORK and the ROLLBACK COMMITTED WORK commands. Summary messages are displayed for each logical unit of work affected. There are two types of summary messages:

1. Message ARI0212I identifies the logical unit of work that will be bypassed and/or rolled back because of the commands. For each of these logical units of work, **message ARI0237I displays the affected DBSPACES and the operations on these DBSPACES.** Message ARI0210I is displayed if no logical units of work will be bypassed or rolled back because of the commands.
2. Messages ARI0211I and ARI0214I are displayed if any logical units of work may be impacted by those logical units of work that will be bypassed or rolled back because of the commands. These impacted logical units of work may be updating objects that no longer exist (or exist in a different form) because of the bypassed or rolled back logical units of work. This could lead to a DBSS UNDO or REDO termination error (message ARI0126E). Alternatively, it could cause incorrect updates to be made to the data base. These database inconsistencies may cause problems later such as:
 - DBSS terminations during FORWARD processing (message ARI0126E).
 - "Should not occur" SQLCODE errors.
 - Incorrect output.

After displaying the summary messages, the database manager displays message ARI0213D. You should review the summary messages to ensure that your command input will do what you thought it would. You can then reply CONTINUE or CANCEL to message ARI0213D. A response of CONTINUE causes the records in the log for the displayed logical units of work to be modified before applying the logged changes. The changes made to the log will remain even if a failure occurs while applying the changes.

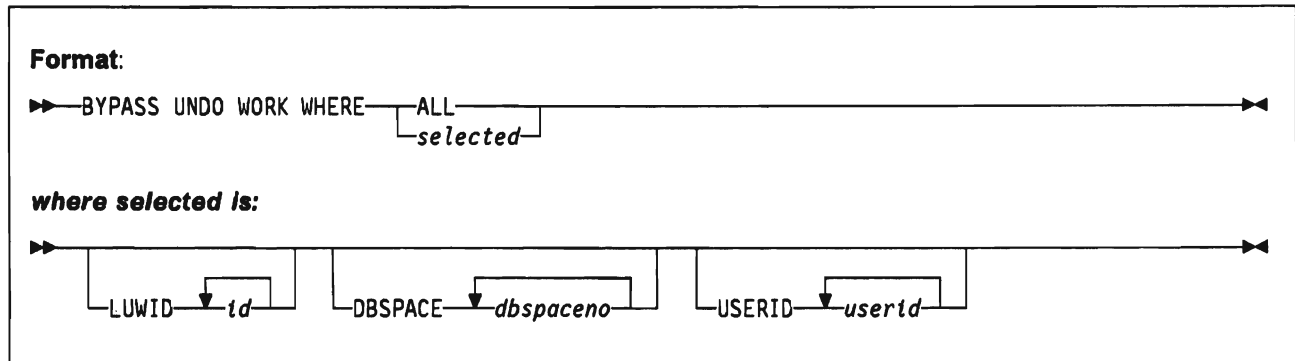
If you are restoring the database from both database and log archives (LOGMODE = L), you will get the summary messages and message ARI0213D for each log archive file restored. Because you can stop the application server before each log archive file is processed, you can submit different EXTEND input files for each log archive file restored.

Because summary messages are issued for each log archive (and you can change the EXTEND input file for each log archive), the database manager cannot detect impacted logical units of work in subsequent log archives. For example, if a committed logical unit of work is rolled back in the first log archive, it may impact logical units of work in subsequent log archives. However, no "impacted" logical units of work will be detected in subsequent log archives. This could lead to DBSS REDO termination errors (message ARI0126E). Alternatively, it could cause incorrect updates to be made to the database. These database inconsistencies may cause problems later such as:

- DBSS terminations during FORWARD processing (message ARI0126E).

- "Should not occur" SQLCODE errors.
- Incorrect output.

Bypassing an UNDO WORK Failure



You must indicate either ALL or at least one LUWID, DBSPACE, or USERID. You can specify a LUWID, DBSPACE, and USERID in any order, but each must be on a separate line. For a further explanation, see the examples listed on page 229.

During recovery processing, some logical units of work that are recorded in the log are "undone." The logical units of work that are undone during warm start are those that were not yet committed when the system failed. Logical units of work may also have to be undone when restoring from a database archive that was taken while users were accessing the database. When a DBSS error occurs during undo work processing, it is possible to make the database available again by using BYPASS UNDO WORK.

BYPASS UNDO WORK causes the database manager to bypass the undo processing for uncommitted logical units of work. Because the cause of the failure is bypassed, recovery processing can complete and the database can be available for use. The database is inconsistent, however. Some of the changes to the database from uncommitted logical units of work now exist in the database.

Use the diagnostic display information to determine what control keywords and parameters you should specify. The control keywords are:

LUWID *id* [*id2 id3 ...*]

identifies the logical unit of work for which undo work processing is to be bypassed. You can specify more than one logical unit of work identifier. Separate the identifiers with blanks. Use the diagnostic display to determine which logical units of work should be bypassed.

ALL

indicates that all logical units marked for undoing are to be bypassed. If you use ALL, you cannot specify other control keywords or another BYPASS UNDO WORK command with other control keywords.

DBSPACE *dbspaceno* [*dbspaceno1 dbspaceno2 ...*]

indicates that undo processing is to be bypassed for any logical unit of work that updates the specified DBSPACE(s). You can specify more than one DBSPACE. Separate the DBSPACE numbers with blanks.

You cannot specify DBSPACE 1, which contains the SQL/DS catalog. Bypassing DBSPACE 1 is not allowed because it contains the SQL/DS catalogs.

If a logical unit of work that is to be undone has updated the specified DBSPACE, the database manager bypasses undo processing for *all* changes recorded in the logical unit of work. That is, once a logical unit of work meets the bypass condition, undo processing is bypassed for *all* changes (including other affected DBSPACES) in that logical unit of work.

USERID *userid* [*userid1 userid2 ...*]

indicates that undo processing is to be bypassed for all logical units of work done by the specified user. More than one userid can be specified. Separate each userid with blanks.

In this example, undo processing is bypassed for all logical units of work that update either DBSPACE 14 or 47:

```
BYPASS UNDO WORK WHERE
      DBSPACE 14 47
```

When more than one control keyword (other than LUWID - see below) is specified, the parameters within each control keyword are ORed, while the control keywords themselves are ANDed. For undo processing to be bypassed in the following example, the logical unit of work must be either SCOTT's or THOMPSON's, and must have operated on either DBSPACE 22, 5, or 54.

```
BYPASS UNDO WORK WHERE
      DBSPACE 22 5 54
      USERID SCOTT THOMPSON
```

LUWID, however, is always ORed to the rest of the control keywords. In the following example, logical unit of work 7BF is always bypassed regardless of the userid associated with it or the DBSPACES it modifies. Additional logical units of work are bypassed only if they operate on DBSPACE 22, 5, or 54, *and* they have the userid SCOTT or THOMPSON.

```
BYPASS UNDO WORK WHERE
      DBSPACE 22 5 54
      LUWID 7BF
      USERID SCOTT THOMPSON
```

Regardless of what parameters you specify, it is always worthwhile to determine which DBSPACES have inconsistent data because of BYPASS UNDO work. A list of all log records that were bypassed is displayed. DBSPACES for which undo processing is bypassed will contain inconsistent data. You should consider disabling these DBSPACES so users don't update data based on inconsistent changes.

Sometimes you can bypass a logical unit of work based on your analysis of the diagnostic display and still get DBSS errors (message ARI0126E) on recovery. In this case you must, once again, try to detect a pattern and bypass the undo processing (UNDO failure) or redo processing (REDO failure) for the logical unit

of work that seems to be causing the failure. Often you'll find that the errors occur for undo or redo processing on a single DBSPACE. Bypassing undo or redo processing for that DBSPACE could allow the recovery to complete successfully. Remember to disable the affected DBSPACE. Otherwise, users would continue to update a DBSPACE that contains inconsistent data. If the affected DBSPACE(s) are not disabled, these database inconsistencies may cause problems later such as:

- DBSS terminations during FORWARD processing (message ARI0126E).
- "Should not occur" SQLCODE errors.
- Incorrect output.

The BYPASS UNDO WORK command leaves inconsistencies in the database. After you correct the problem that caused the DBSS failure, you have the following options to remove the inconsistencies if you are running with LOGMODE=L or LOGMODE=A:

1. If you have not taken a database archive since the BYPASS UNDO WORK, you can restore the database from the last archive.
2. If you have taken a database archive since the last BYPASS UNDO WORK and LOGMODE=L, you can restore the database from an earlier database archive and all subsequent log archives (if you have saved all the archive files).
3. You can choose to restore the database to a previous (to the BYPASS UNDO WORK) archive level. Refer to "Resetting the Database", in the *System Planning and Administration*.
4. You can manually undo the inconsistencies.
5. You can leave part of the database inconsistent indefinitely, perhaps using DISABLE DBSPACE.

With LOGMODE=Y, only the fourth and fifth options are available to you.

Example of Bypassing an UNDO WORK Failure:

The following is an example of using Filtered Log Recovery to bypass the undo processing for an uncommitted logical unit of work.

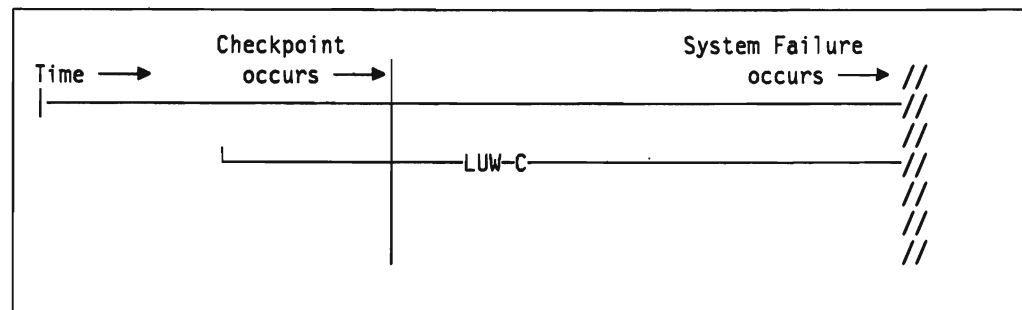


Figure 129. BYPASS UNDO WORK for LUW C

Consider the situation in Figure 129. The system has ended abnormally. Warmstart recovery should undo all changes made prior to the checkpoint by LUW C since it was not committed, but there is a bad spot on the log causing a

log record for LUW C to have an invalid DBSPACE number. The following message appears during warmstart:

```
ARI0126E DBSS termination during UNDO processing.  
LUWID = 103                USER ID = SQLDBA  
DATE = 06-22-92           TIME = 13:19:58  
OPERATION = INSERT        DBSPACE = -6169  
TABLE ID = -6169          (E7E7)  
PAGE ADDRESS = NONE
```

In this type of scenario, there is no possibility that any LUW has a dependency on LUW C, since LUW C never committed (and hence its changes were never accessible by other LUWs).

During restart, you can bypass the undo processing for LUW C with the following command:

```
BYPASS UNDO WORK WHERE  
LUWID 103
```

The result of the BYPASS UNDO WORK command is that some of the changes from the uncommitted LUW C remain in the database. The DBSPACES affected by LUW C should be disabled until the partial updates can be removed.

Rolling Back Committed Work

The ROLLBACK COMMITTED WORK command rolls back logical units of work that users or applications have already committed to the database. Before using ROLLBACK COMMITTED WORK review "Filtered Log Recovery and Referential Integrity" on page 237.

Normally, you would use ROLLBACK COMMITTED WORK when you get a DBSS error during redo processing. You can also use it to back out changes that a user had erroneously committed to the database.

When you use the ROLLBACK COMMITTED WORK command during a warm start, you can force the rollback of any logical unit of work that completed after the last checkpoint. In addition, for warm start:

- if LOGMODE = A or L, and
- you specify the DATE and/or TIME control keywords

you can force the rollback of any committed logical unit of work in the current log, except for any logical units of work which contain a DROP TABLE or DROP DBSPACE command.

When used with restore, you can do a ROLLBACK COMMITTED WORK for any logical unit of work that completed after the database archive was taken.

If you are trying to make the database available after a DBSS error, you should use the diagnostic display information to determine what control keywords and parameters you should specify.

If you want to back out user errors, you should get the userid associated with the erroneous logical unit of work, and the date and time it was run. Ideally, you should get the logical unit of work identifier, but few users know what identifiers are associated with their logical units of work. You should use ROLLBACK COMMITTED WORK for user errors only as a last resort.

When you roll back committed work, subsequent logical units of work are affected if they base their updates on changes made by that logical unit of work. Some of the subsequent logical units of work will fail during recovery processing (causing message ARI0126E to be displayed). This can happen if, for example, a row that it had updated no longer exists. Other logical units of work might not fail, but could yield database inconsistencies. These database inconsistencies may cause problems later such as:

- DBSS terminations during FORWARD processing (message ARI0126E).
- "Should not occur" SQLCODE errors.
- Incorrect output.

The following is an example illustrating one possible affect of the ROLLBACK COMMITTED WORK command on a subsequent logical unit of work.

Example: In this example, a unique index is affected by the ROLLBACK COMMITTED WORK command.

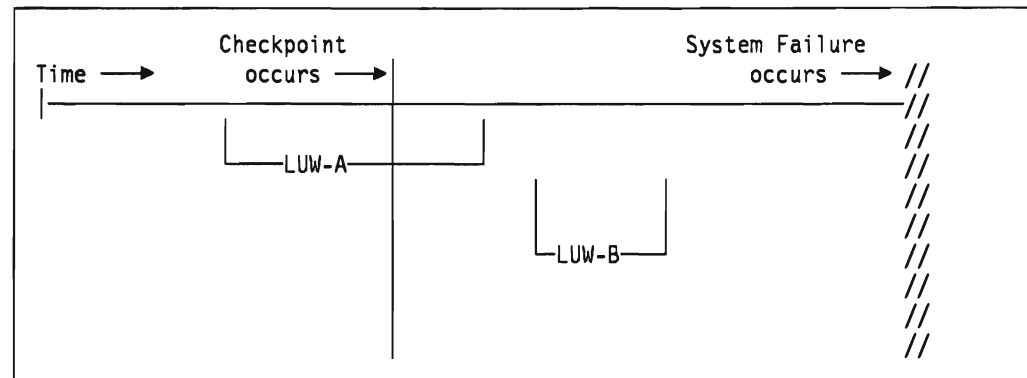


Figure 130. ROLLBACK COMMITTED WORK for LUW A

Consider the situation in Figure 130. LUW A (LUWID 1265) deletes rows from a table with a unique index. LUW B (LUWID 1268) then inserts new rows with the same keys. LUW B depends on LUW A. An attempt to ROLLBACK COMMITTED WORK for LUW A results in the following message during warmstart:

```
ARI0126E DBSS termination during REDO processing.  
LUWID = 1268                USER ID = SQLUSRA  
DATE = 06-22-92            TIME = 15:43:04  
OPERATION = INSERT         DBSPACE = 11  
TABLE ID = -32767 (8001)  
PAGE ADDRESS = 45F000      PAGE TYPE = INDEX  
PAGE NUMBER = 12C
```

This example shows what can happen when you perform the ROLLBACK COMMITTED WORK command for an LUW which affects the same table as a subsequent LUW. When attempting to insert a key into the unique index, the application server detected that the insert would violate the uniqueness of the index. A more insidious example would be where the second LUW could be redone successfully, but the answer is not what is expected.

During restart, you should rollback the committed work for LUW B as well with the following command:

```
ROLLBACK COMMITTED WORK WHERE  
LUWID 1268
```

The result of the ROLLBACK COMMITTED WORK commands is that both LUW A and LUW B are rolled back. You should disable the DBSPACES affected by these LUWs until the LUWs can be redone (if desired) and the referential integrity can be checked.

You are more likely to encounter this situation when restoring a database and its associated log archives, since the number of LUWs following the one being rolled back can be much larger.

The ROLLBACK COMMITTED WORK command causes records in the log to be modified. How these modified records will affect subsequent use of the log depends on the context in which ROLLBACK COMMITTED WORK is used. There are two cases: the command is used during a restore, or the command is used during a warmstart. In both cases, the distinction we need to make is whether the log dataset contains the current log, or a copy of an archived log.

1. Restore

While applying logged changes during a restore, we copy archived logs onto the log dataset (or datasets, for dual logging) so they can be manipulated by the SQL/DS application server. A ROLLBACK COMMITTED WORK command will modify the working copy of the log on the log dataset, but the archive copy is not affected. Therefore, in any subsequent restore using the same archived logs you will need to repeat the ROLLBACK COMMITTED WORK commands in order to achieve the same results.

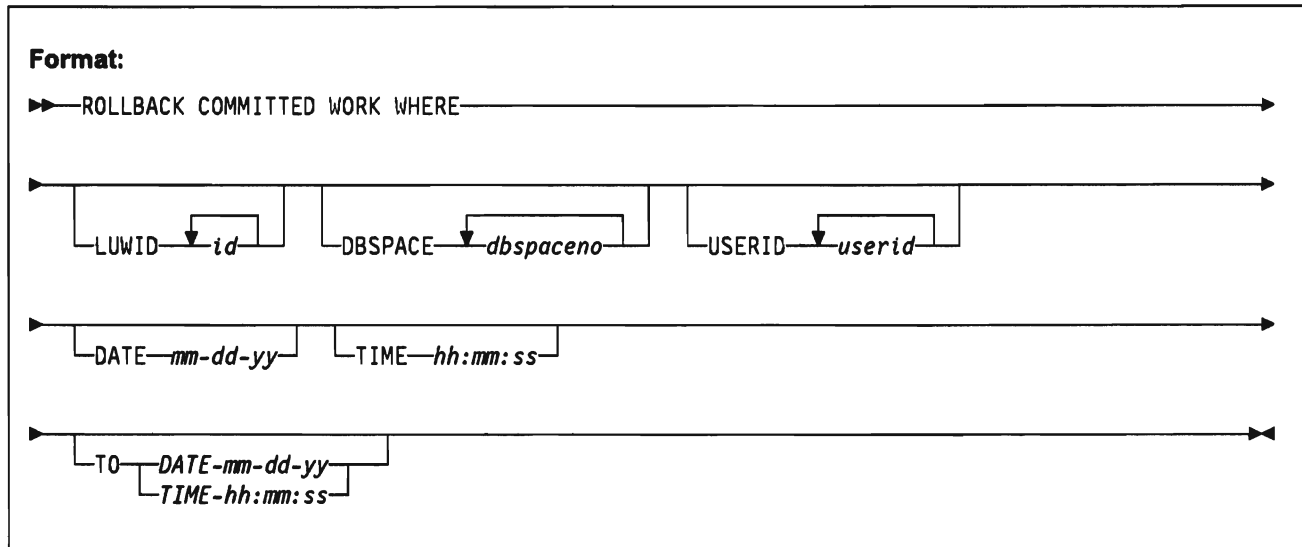
There is one exception to note involving the last log in the restore set. This is the current log that was archived at the beginning of the restore, and will be the last log you will apply (unless you end the restore prematurely). This log will become the current log once again at the end of the restore, and will not be part of the restore set. The changes made by ROLLBACK COMMITTED WORK are reflected in this current log, and will therefore be saved when the log is subsequently archived.

2. Warmstart

When performing a warmstart, the log which currently resides on the log dataset(s) is always the current log. The ROLLBACK COMMITTED WORK command modifies the records on this log, and these modified records will be saved when the log is subsequently archived. It will not be necessary to repeat any ROLLBACK COMMITTED WORK commands if this log is later used during a restore.

To summarize, the SQL/DS application server can manipulate only those log records which reside on the log dataset(s), and any changes made are permanent only when the resident log happens to be the current log rather than a copy of an archived log.

Once you have rolled back a committed logical unit of work, the only way to recommit it is to rerun the logical unit of work or by restoring the database from database and subsequent log archives (without using ROLLBACK COMMITTED WORK command).



You must specify at least one of the control keywords. If you choose more than one, they may appear in any order, but each must be on a separate line.

The control keywords are:

LUWID *id* [*id2 id3 ...*]

identifies the logical unit of work for which the committed work will be rolled back. You can specify more than one logical unit of work identifier. Separate the identifiers with blanks. Use the diagnostic display to determine which logical units of work should be rolled back.

DBSPACE *dbspaceno* [*dbspaceno1 dbspaceno2 ...*]

indicates that committed work is to be rolled back for any logical unit of work that updates the specified DBSPACE(s). You can specify more than one DBSPACE. Separate the DBSPACE numbers with blanks.

If a committed logical unit of work has updated the specified DBSPACE, the entire logical unit of work is rolled back, not just those changes that affect the DBSPACE.

USERID *userid* [*userid1 userid2 ...*]

indicates any committed logical unit of work for a given user is to be rolled back. More than one userid can be specified. Separate each userid with blanks.

DATE *mm-dd-yy*

Indicates the date from which the log is searched for the first committed logical unit of work. Normally, the starting date would be some date that work was done in the current log. If, however, you are restoring the database using log archiving, you can specify dates of logical units of work that were done in previous log archives. Each portion of the date must consist of two characters; that is, leading zeros must be added if necessary.

If the DATE control keyword is omitted, but a TIME control keyword is present, then the current date is used.

TIME *hh:mm:ss*

indicates the time from which the database manager is to search for the first committed logical unit of work that is to be rolled back. Each portion of the

time must consist of two characters; that is, leading zeros must be added if necessary.

If the TIME control keyword is omitted, but the DATE control keyword is present, 00:00:01 is used by default (12:00:01 A.M.).

TO

this allows you to bound the date and time. TO must follow a starting date and/or time, and must be followed by an ending date and/or time. See the examples below for clarification.

In this example, committed work is rolled back for all logical units of work that update either DBSPACE 14 or 47:

```
ROLLBACK COMMITTED WORK WHERE  
  DBSPACE 14 47
```

When more than one control keyword (other than LUWID - see below) is specified, the parameters within each control keyword are ORed, while the control keywords themselves are ANDed. For committed work to be rolled back in the following example, the logical unit of work must be either SCOTT's or THOMPSON's, and must have operated on either DBSPACE 22, 5, or 54.

```
ROLLBACK COMMITTED WORK WHERE  
  DBSPACE 22 5 54  
  USERID SCOTT THOMPSON
```

LUWID, however, is always ORed to the rest of the control keywords. In the following example, logical unit of work 7BF is always rolled back regardless of the userid associated with it or the DBSPACES it modifies. Additional logical units of work are rolled back only if they operate on DBSPACE 22, 5, or 54, and they have the userid SCOTT or THOMPSON.

```
ROLLBACK COMMITTED WORK WHERE  
  DBSPACE 22 5 54  
  LUWID 7BF  
  USERID SCOTT THOMPSON
```

In the next example, logical unit of work 801 is always rolled back regardless of when it occurred in the current log. All work started by SCOTT after 6:00 P.M. on 5-5-1985 is also rolled back.

```
ROLLBACK COMMITTED WORK WHERE  
  USERID SCOTT  
  DATE 05-05-85  
  TIME 18:00:00  
  LUWID 801
```

In this example, SCOTT's work is rolled back only if it started between 6:00 and 8:00 P.M. on 5-5-1985.

```
ROLLBACK COMMITTED WORK WHERE  
  USERID SCOTT  
  DATE 05-05-85  
  TIME 18:00:00  
  TO  
  DATE 05-05-85  
  TIME 20:00:00
```

DROP DBSPACE and DROP TABLE Considerations: The ROLLBACK COMMITTED WORK command can be used to roll back a logical unit of work containing a DROP TABLE or DROP DBSPACE command if that logical unit of work is failing during warmstart REDO recovery processing. It can also be used to roll back a logical unit of work containing a DROP TABLE or DROP DBSPACE command, if a database archive that still contains the table or DBSPACE is restored first. It is not intended to be used to back out changes that a user has erroneously committed to the database.

There are special considerations if a logical unit of work containing DROP DBSPACE and/or DROP TABLE commands is to be rolled back because of the manner in which SQL/DS processes DROPs. For performance and in order to minimize the size of the log, specific database changes are not logged for DROP DBSPACE and DROP TABLE (for example, the dropped rows are not logged). For each such command in a logical unit of work, SQL/DS schedules another logical unit of work that actually performs the DROP function. This logical unit of work executes after the COMMIT WORK process of the logical unit of work containing the DROP command completes. If the second logical unit of work completes successfully, the database manager writes a checkpoint to the log. As well, it is possible for a checkpoint to occur when storage is being released for a dropped table. You will not be able to use the ROLLBACK COMMITTED WORK command during a warmstart to successfully restore a table or DBSPACE which was dropped by accident if:

1. The second logical unit of work completes successfully, resulting in a checkpoint being written to the log
2. During processing of the DROP TABLE command, a checkpoint occurs while storage is being released.

There are two situations when you would be able to use the ROLLBACK COMMITTED WORK command to roll back the updates for a logical unit of work containing a DROP TABLE or DROP DBSPACE command.

1. If the database manager abended and the last checkpoint occurred before the beginning of the second logical unit of work that actually performed the drop. In this situation you could use the ROLLBACK COMMITTED WORK command during a warmstart of the SQL/DS application server.
2. If the following conditions apply:
 - a. The logical unit of work in question took place while LOGMODE was A or L, and
 - b. you restored the system from a database archive and its related log archives taken prior to this logical unit of work.

In other words, once a DROP TABLE or DBSPACE is committed, you cannot restore the table or DBSPACE unless you are able to restore from a database archive and use the ROLLBACK COMMITTED WORK command.

If a logical unit of work containing DROP DBSPACE and/or DROP TABLE commands is to be rolled back, the following rules apply:

1. The ROLLBACK COMMITTED WORK command must contain the DBSPACE control keyword. All of the DBSPACES affected by the DROP commands must be specified via the DBSPACE control keyword.

2. The ROLLBACK COMMITTED WORK command may also contain the LUWID control keyword, but must not contain any other control keywords. That is, the USERID, DATE, and TIME control keywords are not permitted. (This is because the ANDed options are not permitted in this case.)

If these rules are violated, the database manager displays message ARI0256E and ends after it displays all the summary messages.

Filtered Log Recovery and Referential Integrity

Users of Filtered Log Recovery must ensure that the integrity of the data is maintained after bypassing committed LUWs. A bypassed LUW may be inserting a parent row upon which later inserts in later LUWs depend. Therefore, after Filtered Log Recovery, the database may be in an inconsistent state even though the constraints all appear to be enforcing Referential Integrity.

It is the responsibility of the user to re-establish the integrity of the data by determining all tables in the affected DBSPACES and deactivating and then activating keys for those tables. Otherwise there is no guarantee of the integrity of the data:

The following scenario will be used as an example of how to re-assert Referential Integrity after Filtered Log Recovery.

Assume that the following sequence of events has taken place and each step is a separate LUW:

1. User ID JONES has created the table T1 in DBSPACE 14 and T2 in DBSPACE 15. Both DBSPACES are recoverable. T1 is the parent table and T2 is the dependent table.
2. User ID HALL inserts a row into the parent table T1.
3. User ID SMITH inserts a row into the dependent table T2.

Then, before a checkpoint is issued assume that a "failure" occurs due to a DASD I/O error.

To perform the Filtered Log Recovery to ROLLBACK the insert of a parent row, two things must be done. Firstly, identify to the database manager the EXTEND input file that contains recovery parameters. It can be identified in the following way:

```
filedef ariextnd disk rollback filtparm a
```

The ROLLBACK FILTPARM file causes the COMMITTED WORK which inserts the parent row to be bypassed. The command in the file looks like this:

```
ROLLBACK COMMITTED WORK WHERE  
USERID HALL
```

Secondly, invoke Filtered Log Recovery by starting the database with EXTEND=Y: as in Figure 131 on page 238.

The highlighted message in Figure 131 tells you in which DBSPACE the table is located. The operation on the table which was rolled back is also included. In this example it was one INSERT on a table in DBSPACE 14.

```
SQLSTART DB(SQLMACBS) PARM(DUMPTYPE=N,EXTEND=Y)
ARI0717I START SQLSTART EXEC: 07/26/88 16:08:22 EDT
ARI0663I FILEDEFS IN EFFECT ARE:
ARIEXTND DISK      ROLLBACK FILTPARM A1
ARISQLLD DISK      ARISQLLD LOADLIB Q1
BDISK   DISK      300
LOGDSK1 DISK      301
LOGDSK2 DISK      302
DDSK1   DISK      303
DDSK2   DISK      304
ARIUSRDD DISK      USERLIB  LOADLIB  *
ARIARCH  TAP1  SL  00001
ARITRAC  TAP2  SL  00001
ARILARC  TAP3  SL  00001
.
.
.
ARI0016I DISPBIAIS PARAMETER VALUE IS 7
ROLLBACK COMMITTED WORK WHERE
USERID HALL
ARI0283I LOG ANALYSIS COMPLETE
ARI0212I SUMMARY INFORMATION OF THE RECORDS TO BE ROLLED BACK :
        LUWID = C4C USER ID = HALL
        DATE = 07-26-88 TIME = 16:05:17
ARI0237I DBSPACE 14 <-----
ARI0237I 1 INSERTS
ARI0213D ENTER 'CONTINUE' TO CONTINUE SQL/DS INITIALIZATION, OR
        'CANCEL'  TO END THE INITIALIZATION.
continue
ARI0282I LUW UNDO COMPLETE
ARI0281I LUW REDO COMPLETE
ARI0134I DATABASE SQLMACBS HAS BEEN IDENTIFIED AS A
        GLOBAL RESOURCE.
ARI0060I SQL/DS INITIALIZATION COMPLETE
ARI0045I READY FOR OPERATOR COMMUNICATIONS
```

Figure 131. Invoke Filtered Log Recovery Example

Filtered Log Recovery processing can leave the database in an inconsistent state and the integrity of the data cannot be guaranteed. In this example, although the parent row in the parent table no longer exists on the database, the catalog does not reflect this unmatched state. Therefore, an active foreign key may contain a foreign key value without a matching primary key value.

After Filtered Log Recovery has completed you must recheck any referential constraints which may have been affected. The console might look as follows when determining which tables in the affected DBSPACE 14 have dependent foreign keys. The DBSPACE number was obtained from the highlighted statement in Figure 131.

```
ARI0870I ENTER: COMMAND FOLLOWED BY A SEMICOLON OR EXIT TO END
select creator,tname,dbspaceno from system.syscatalog
where dbspaceno = 14 and dependents > 0;
-----> SELECT CREATOR,TNAME,DBSPACENO FROM SYSTEM.SYSCATALOG
        WHERE DBSPACENO = 14 AND DEPENDENTS > 0;

CREATOR  TNAME                DBSPACENO
-----
SMITH    T1                        14
ARI0850I SQL SELECT PROCESSING SUCCESSFUL: ROWCOUNT=1
```

All active keys should be deactivated. The objective is to check that each foreign key value in the dependent table has a matching primary key value in the parent table. To do this you must first deactivate all active dependent foreign keys in the relationship. Deactivating a primary key will implicitly deactivate all active dependent foreign keys.

If the command

```
ALTER TABLE HALL.T1 DEACTIVATE PRIMARY KEY;
```

is entered the console will look like the following:

```
ARI0870I ENTER: COMMAND FOLLOWED BY A SEMICOLON OR EXIT TO END
alter table hall.t1 deactivate primary key;

-----> ALTER TABLE HALL.T1 DEACTIVATE PRIMARY KEY;
ARI0500I SQL PROCESSING WAS SUCCESSFUL.
ARI0505I SQLCODE = 0 ROWCOUNT = 0
```

To determine which foreign key, if any, does not have matching primary key values for its foreign key values enter the following command.

```
ALTER TABLE HALL.T1 ACTIVATE PRIMARY KEY;
```

By activating the primary key you are also activating all dependent foreign keys that were deactivated implicitly when the primary key was made inactive.

The console will look like the following:

```
ARI0870I ENTER: COMMAND FOLLOWED BY A SEMICOLON OR EXIT TO END
alter table hall.t1 activate primary key;

-----> ALTER TABLE HALL.T1 ACTIVATE PRIMARY KEY;
ARI0503E AN SQL ERROR HAS OCCURRED.
        THE FOREIGN KEY T1FOREIGN DEFINED FOR TABLE SMITH.T2
        CANNOT BE ACTIVATED. REASON CODE = 03
ARI0505I SQLCODE = -667 ROWCOUNT = 0
ARI0504I SQLERRP: ARIXIVR SQLERRD1: -810 SQLERRD2: 0
```

The SQLCODE = -667 and the reason code = 03 indicate that the operation failed because not every value in the foreign key is found in the primary key of its parent. For more information see the *Messages and Codes* manual.

A referential constraint has been discovered which is no longer valid. It is inactive and will remain inactive until every foreign key value has a matching primary key value. It can be fixed any of the following ways:

- Adding the appropriate row to the parent table
- Changing the appropriate row in the dependent table
- Moving the appropriate row from the dependent table to a special table.

See the *Database Administration* manual for more information on repairing rows which violate referential constraints.

Disabling a DBSPACE

Format:

```
▶▶—DISABLE DBSPACE—↓dbspaceno————▶▶
```

This command takes the specified DBSPACES "off-line." That is, it does not allow users to access the DBSPACE. Only *forward* processing is disabled. The specified DBSPACES will be disabled only if SQL/DS initialization completes. You cannot disable DBSPACE 1. Also, in order to maintain referential integrity, updates must not be performed on the tables which are related to the tables in the disabled DBSPACE.

Usually you would use this command to disable a DBSPACE that is causing a DBSS error, or that contains inconsistent data due to BYPASS UNDO WORK processing. This command is also useful, however, for disabling access to DBSPACES in which you have discovered user errors. Once you determine how to fix the user errors, you can use the ENABLE DBSPACE command to bring the DBSPACE back "online."

The DBSPACE remains disabled for all future invocations of SQL/DS until you issue an ENABLE DBSPACE or until you restore the database from a previous archive. When you restore a database from a previous archive, the only DBSPACES disabled are those that were disabled when the database archive was taken.

At startup, disabled DBSPACES are displayed in informative messages.

Enabling a DBSPACE

Format:

▶▶—ENABLE DBSPACE—*dbspaceno*—▶▶

This command enables a previously disabled DBSPACE. Users will be able to access the DBSPACE. You can specify more than one DBSPACE number. Separate the numbers with spaces. The specified DBSPACES will be enabled only if SQL/DS initialization completes.



Chapter 7. Recovering from Directory Verify Errors

Guidelines for Using Directory Verification

The Directory Verify function is invoked via the DVERIFY parameter of the SQLEND operator command.

You should use the Directory Verify function each time a database archive is taken—either SQL/DS archive or user archive. The Directory Verify function checks for inconsistencies in the directory, and thus allows you to avoid archiving an inconsistent directory.

In addition, you should periodically verify the directory on non-archive shutdowns. The interval depends on the frequency of the database archives and the volume of database updates. For example, Directory Verify should be invoked at shutdown after a significant amount of data has been loaded into the database or after some critical data has been updated. These intervening verifications can narrow the time period down as to when an inconsistency occurred and reduce the time to recover.

Directory Verify should also be invoked if there are persistent SQL/DS abnormal terminations in the DBSS. (These terminations would display message ARI040E which would identify a module whose first four characters are ARIY).

If Directory Verify does find discrepancies, you should:

- Follow the steps in Chapter 3, "Reporting Defects" on page 69 to report the defect to IBM, and
- Follow the procedures outlined below.

Recovery Actions for an Inconsistency

If an inconsistency is detected by the Directory Verify function, take these steps:

1. Restore the database from the last database archive. This will recover all database updates in single-thread mode. Frequently, single-thread recovery mode database updates will not re-create the discrepancy.

If your last database archive was quite recent, it may contain bad data if you did not request Directory Verify at that time. If you are using log archiving, you may want to restore the database from the next to last database archive and apply all the log archives.

2. If the restore from the database archive completes successfully, go to the next step. If the restore from the database archive fails, then take the recovery actions as indicated by the error messages displayed.
3. Shutdown your application server, specifying Directory Verify (issue SQLEND NORMAL DVERIFY).

If the inconsistency still exists, follow the procedures outlined below, according to the type of the DBSPACE(s) having the inconsistency.

System DBSPACES: System DBSPACES are those whose DBSPACE name is SYS000n. SYS0001 contains the catalog tables. The catalog tables are updated by SQL data definition statements (such as CREATE TABLE, DROP INDEX, ACQUIRE DBSPACE) and SQL authorization statements (such as GRANT and REVOKE). Preprocessing of applications also causes updates to the catalog tables. The other system DBSPACES (SYS0002 - SYS000n) contain the SQL/DS packages and views. These are updated when any applications are preprocessed or any views are created.

If LOGMODE = A,

1. Warm start the database (STARTUP=W) using the ROLLBACK COMMITTED WORK command (see "Rolling Back Committed Work" on page 231) to rollback all logical units of work that caused updates to the DBSPACE(s) having the inconsistency. You should use the DATE and TIME of the database archive you restored from and the DBSPACE number(s) as control keyword values.
2. Since the system DBSPACE(s) will contain back-level information, the rolled back work must be redone.
 - For SYS0001, redo all the rolled-back work that made catalog updates to bring the catalog tables up to date.
 - For other system DBSPACE(s), redo all the rolled-back preprocessing and CREATE VIEWS.

If LOGMODE = L,

1. Redo the restore of the database using the ROLLBACK COMMITTED WORK command (see "Rolling Back Committed Work" on page 231) to rollback all logical units of work that caused updates to the DBSPACE(s) having the inconsistency. Specify the DBSPACE number(s) as control keyword values.
2. Since the system DBSPACE(s) will contain back-level information, the rolled back work must be redone.
 - For SYS0001, redo all the rolled-back work that made catalog updates to bring the catalog tables up to date.
 - For other system DBSPACE(s), redo all the rolled-back preprocessing and CREATE VIEWS.

If you do not wish to use the ROLLBACK COMMITTED WORK commands, you can use this alternate procedure:

1. Invoke the COLDLOG function (via the SQLLOG EXEC), which removes all update activity from the log (see "Running the SQLLOG EXEC" in the *System Administration* manual).
2. Restore the database from the last database archive copy.
3. Redo all database update activity since the database archive.

PUBLIC and PRIVATE DBSPACE(s): If the inconsistency exists in PUBLIC or PRIVATE DBSPACE(s) (the DBSPACE name is not SYS000n), follow these procedures.

1. Unload each DBSPACE having an inconsistency using the DBS utility. If you are unable to unload the DBSPACE, do the next step, skip the rest and call your IBM support person.

Note: The DBSPACE should not be dropped. Dropping invokes the checkpoint process and, due to the inconsistency, your application server will most likely terminate abnormally.

2. Warm start the database (STARTUP=W) using the DISABLE DBSPACE command (see "Disabling a DBSPACE" on page 240) to disable all activity to the inconsistent DBSPACE(s).
3. Acquire a new DBSPACE(s) and load the contents of the faulty DBSPACE(s) using new table names.
4. If there were any views on the affected tables, re-create the views using the new table names.
5. Grant appropriate access authorizations to the newly created DBSPACE(s), tables, and views.
6. Update all applications with the new table, view, and DBSPACE names(s) and preprocess and recompile those applications.
7. Drop the old DBSPACE(s) after you are able to repair the data base with assistance from IBM Support Personnel.



Chapter 8. Problem Isolation and Handling

This chapter introduces how to isolate and handle system problems, and the tools that will help you to do so.

Problem handling refers to the activities involved in coping with problems on the system.

System Problems

SQL/DS Database Machine Problems

Abnormal termination occurs when the database manager detects an internal error, a resource limitation, a hardware error, a program check (or similar situation), or the operator issues an `SQLEND QUICK` command.

Facilities are provided for handling conditions that cause abnormal ends. The action taken depends on the environment and when the condition occurs. The action can be as simple as passing a return code to the application program, or as drastic as terminating the database machine. Figure 132 summarizes these actions.

Figure 132 (Page 1 of 2). SQL/DS Termination Action Summary—SUM and MUM

Component in Control When Error Occurs	Action Taken in Multiple User Mode	Action Taken in Single User Mode
Program Check When A Package is Being Processed	<p>FOR PROGRAM CHECK IN RANGE OF 8—12 and 15 WHERE THE PROGRAM CHECK RESULTS IN THE SELECT CLAUSE AND AN INDICATOR VARIABLE ASSOCIATED WITH THE CLAUSE: + 802 is returned in the SQLCODE, 0 in SQLERRD1 of the SQLCA. The indicator variable contains a -2 indicating a NULL program value returned as a result of a program check caused by user data. Processing continues.</p> <p>FOR PROGRAM CHECK IN RANGE OF 7—15 WHICH DOES NOT MEET THE ABOVE CONDITIONS: -802 is returned in the SQLCODE and -907 to -915 in the SQLERRD1 of the SQLCA. Processing continues</p> <p>FOR PROGRAM CHECK IN RANGE OF 1—6: -802 is returned in the SQLCODE and -901 to -906 in the SQLERRD1 of the SQLCA. A mini-dump is displayed on the database machine console. Processing continues.</p>	<p>Same as multiple user mode.</p> <p>Same as multiple user mode.</p> <p>Same as multiple user mode.</p>

Figure 132 (Page 2 of 2). SQL/DS Termination Action Summary—SUM and MUM

Component in Control When Error Occurs	Action Taken in Multiple User Mode	Action Taken in Single User Mode
RDS Component is Processing or there is a Non-Program Check in the Package	<p>Communication between user machine and the SQL/DS machine ends. The user machine must re-establish communication to continue.</p> <p>-933 is returned in the SQLCODE, 4 in the SQLERRD1, and 6 in the SQLERRD2 fields of the SQLCA.</p> <p>A mini-dump is displayed on the database machine console. The database manager then prints a virtual machine dump in accordance with the DUMPTYPE initialization parameter.</p> <p>Processing continues.</p>	<p>A mini-dump is displayed on the database machine console.</p> <p>A database machine dump is printed as specified by the DUMPTYPE parameter.</p> <p>Processing ends.</p>
All other SQL/DS Components in the SQL/DS Database Machine	<p>A mini-dump is displayed on the database machine console.</p> <p>A database machine dump is printed in accordance with the DUMPTYPE initialization parameter.</p> <p>Processing ends.</p>	Same as multiple user mode.

Notes:

1. PROGRAM CHECK refers to the Program Status Word interruption code bits 16 through 31. This value can range from 1 to 15.
2. A virtual machine dump does not occur if DUMPTYPE=N is specified.
3. The database manager routes partial virtual machine dumps to the virtual printer. Full virtual machine dumps are routed to the virtual reader. See "SQL/DS Virtual Machine Dump Processing" on page 249 for instructions to process dumps.
4. If DUMPTYPE=F, the virtual machine dump includes any SQL/DS code in discontinuous saved segments that the machine is using.
5. If an SQL/DS limit error (ARI0039E) or hardware error (ARI0041E) occurs, neither a virtual machine dump nor a mini-dump is taken.

The SQL/DS abnormal termination routines are invoked by VM whenever an abnormal termination (including an unrecoverable program check) occurs. These routines are also invoked internally whenever any SQL/DS System Error situation is detected. A First Failure Data Capture dump may also occur in this situation if the PROTOCOL=AUTO parameter was used when the SQLSTART EXEC was issued.

The last case of abnormal termination is the case when the SQL/DS operator issues an SQLEND QUICK command, which immediately terminates all LUWs in

progress. The LUWs are rolled back the next time the application server is started. This command may be issued anytime, even after a `SQLEND NORMAL` or `ARCHIVE` command. `SQLEND QUICK` is accompanied by the message:

```
ARI0043I SQL/DS return code is 508
```

The 508 is a warning return code implying that any subsequent command/EXEC that depends on the application server terminating normally might fail. An exception is the case where the operator issues a `SQLEND NORMAL`, and users have an IUCV or APPC/VM connection established with the database machine but do not have work in progress. In this case, a `SQLEND QUICK` would most likely not cause any problems.

Note: A CMS HX command also causes immediate termination. In this respect it is similar to `SQLEND QUICK`; however, it also prompts the SQL/DS operator as if a dump were desired (message ARI0044D). A reply of *1* or *YES* results in a dump, whereas a *0* or *NO* causes no dump.

In single user mode, if an application program establishes an abnormal end exit (for example, `ABNEXIT`), it overrides the SQL/DS `ABNEXIT`. In this situation, the database manager does not get control from CMS when an abnormal termination condition or program check arises. Users should follow the instructions under "CALL/RETURN Protocols for Application Programs in Single User Mode" in *System Administration*. That section discusses important considerations for user program abnormal end exits.

In installation exits that run on the database machine, any abnormal end exits must also abide by the instructions regarding abnormal end exits. For a discussion of these instructions, see the *System Administration*.

In multiple user mode, abnormal end exits in application programs are of no concern. In this case, they are operating in a different virtual machine. Thus, user exits do not override the SQL/DS exits.

SQL/DS Virtual Machine Dump Processing

The database manager routes partial virtual machine dumps to the virtual printer. This is a printable spool file containing a translated storage dump produced by the `CP DUMP` command. This file can be printed or received, and should be sent to IBM Service. The virtual machine console log file, if available, should also be sent to IBM Service.

The database manager routes full virtual machine dumps to the virtual reader. This is a special format spool file, with a spool class of "DMP." It cannot be printed or received. It must be processed by the `IPCSDUMP` or `DUMpload` command to create a CMS file.

If the dump was created on a VM/SP or VM/ESA 370 Feature system, use the `IPCSDUMP` command which is part of the Interactive Problem Control System (IPCS) component of the VM system.

If the dump was created on a VM/XA or VM/ESA ESA Feature system, use the `DUMpload` command which is part of the Dump Viewing Facility (DVF) component of the VM system.

When you have loaded the full dump into a CMS file, it can be viewed online or printed using `IPCS/DVF` facilities. The CMS file should be sent to IBM Service on

a tape created by the CMS TAPE or VMFPLC command. The virtual machine console log file, if available, should also be sent to IBM Service.

See your VM systems programmer for more information about the IPCS or DVF components and using them to process dumps.

Problem Isolation

SQL/DS Dumps

When the SQL/DS abnormal end routines are entered, a *mini-dump* is displayed. The SQL/DS mini-dump displays the following:

1. ABEND save area, which contains:

- PSW at time of failure
- Registers 0-15 at time of failure.

When a mini-dump is invoked internally by the database machine, the PSW points to the address following the BALR instruction (which branches to the SQL/DS system error routine) in the SQL/DS module that detected the error. The display of registers 0-15 shows their contents at the time of the BALR instruction of the SQL/DS module that detected the SQL/DS system error.

2. Abnormal end code if VM invoked the SQL/DS abnormal end routines.

3. PSW and registers. (Only the registers are displayed if the database manager invokes abnormal end internally.)

4. If determinable, the offset from the start of the program and the start of the module in which the failure occurred.

5. If determinable, the offset from the start of the program and the start of the module that called the failing module (for example, the BALR address).

6. 80 bytes of data around the failing address, normally starting 32 bytes before the failing address. This information is not displayed when the database manager internally invokes abnormal end.

7. If a potential wild branch may have occurred, the following information is also displayed:

- Address of the potentially failing BAL or BALR instruction.
- If determinable, the offset from the start of the program.
- If determinable, the offset from the start of the module.
- 80 bytes of storage around the BAL or BALR instruction.

This information is not displayed when the database manager internally invokes abnormal end.

8. Locations of the (global control blocks) DS2CVT, RDCVT and YRSSCVT.

9. Information about the users running and the programs they were executing at the time of the abend. This list may include residual userids and program names and may not necessarily be the current values. The information includes the addresses of each users DSCAREA, YTABLE1, RDAREA and DCE. The user executing at the time of the abend is also identified.

10. A symptom string that assists in uniquely identifying errors. There are three formats for symptom strings:

a. For system errors:

MS/ARI0040E – ARI0040E is the SQL/DS system error message.
PIDS/568810301 – the SQL/DS product identifier.
RIDS/nnnnnnnn – where nnnnnnnn is the name of the failing module.
PRCS/000000xx – where xx is the system error number or point.

b. For program checks:

AB/Sxxxx – where xxxx is the program check interrupt code.
PIDS/568810301 – the SQL/DS product identifier.
RIDS/nnnnnnnn – where nnnnnnnn is the name of the failing module.
ADRS/00nnnnnn – where nnnnnn is the offset into the failing module.

c. For abnormal ends other than program checks:

AB/yxxxx – if the abnormal end is a system abnormal end, y = S. If the abnormal end is a user abnormal end, y = U. xxxx is the VM abnormal end code.

PIDS/568810301 – the SQL/DS product identifier.

11. The module identifier may also include a PTF/APAR level identifier if service was done to that module.

In addition to the mini-dump, a virtual machine dump may also occur. The areas of the virtual machine to be dumped are determined by the DUMPTYPE parameter. (If DUMPTYPE=N was specified, the dump does not occur.)

If DUMPTYPE=F was specified, the entire database machine is dumped. In addition, if the database machine is using any SQL/DS code that resides in a discontinuous saved segment, the saved segment is also dumped.

If DUMPTYPE=P was specified, the database machine is dumped, excluding the SQL/DS DBSS/DSC program (ARISQLDS) and the RDS program (ARIXRDS). In single user mode, any load module name starting with "ARI" is excluded.

SQL/DS has CSECTs (object modules) for the DBSS/DSC program and the RDS program that contain the seven-character name of the CSECT and the storage address of each CSECT within that program. These CSECTs are called *LINKMAPs*. Whenever a dump for an abnormal end is taken, the database manager displays these LINKMAPs as part of the dump (including when DUMPTYPE=P).

When the abnormal end routines are invoked internally by the database manager, the mini-dump is preceded by SQL/DS error message ARI0040E, which is described in the *Messages and Codes* manual.

When the database manager ends due to a problem in the use of VM system services (such as CMSSTOR, NUCXLOAD, or IUCV), the VM return code is displayed as the reason code in message ARI0042I.

The ARI0043I message is always displayed when processing ends. A return code of 0 denotes that either the SQL/DS processing was successful, or that the application program did not pass back a return code in single user mode. If SQL/DS processing was not successful, the return code can be 504, 508, 512, 516, or 520. (Refer to the *Messages and Codes* manual for details.) Similarly, any return code from an application program operating in single user mode is also displayed when processing ends.

SQL/DS Link Maps and Access

The last module in the DBSS/DSC and RDS load modules contain the name and address of each module in the load module. When DUMPTYPE=P is specified (default), the link maps are the last two areas dumped. When DUMPTYPE=F is specified the link maps are at the end of each load module or phase.

Modules are ordered by their address (in the same order as the link edit book entries) allowing a given module to be located quickly through the address in register 15 in the save area chain.

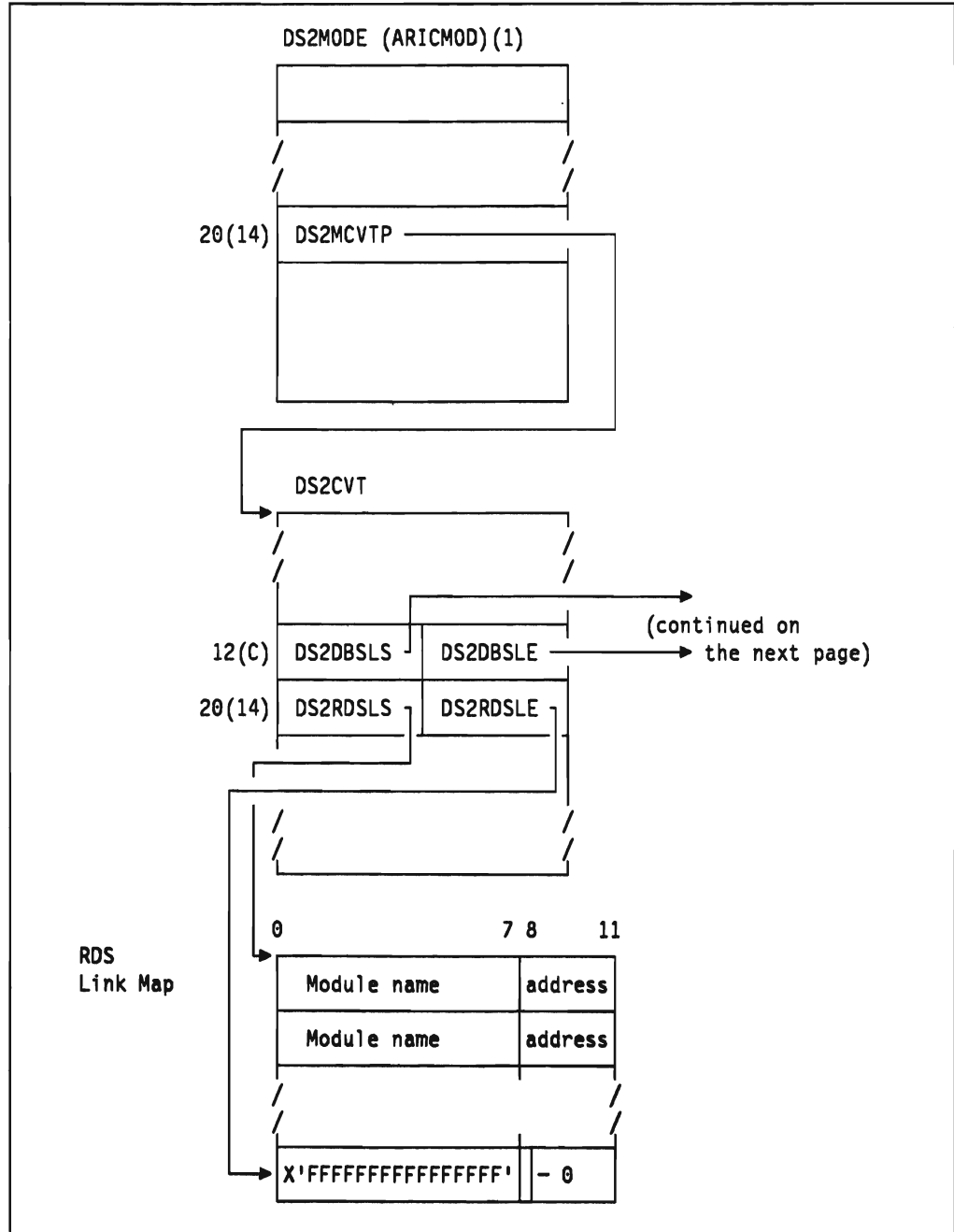


Figure 133 (Part 1 of 2). SQL/DS Link Maps and Access

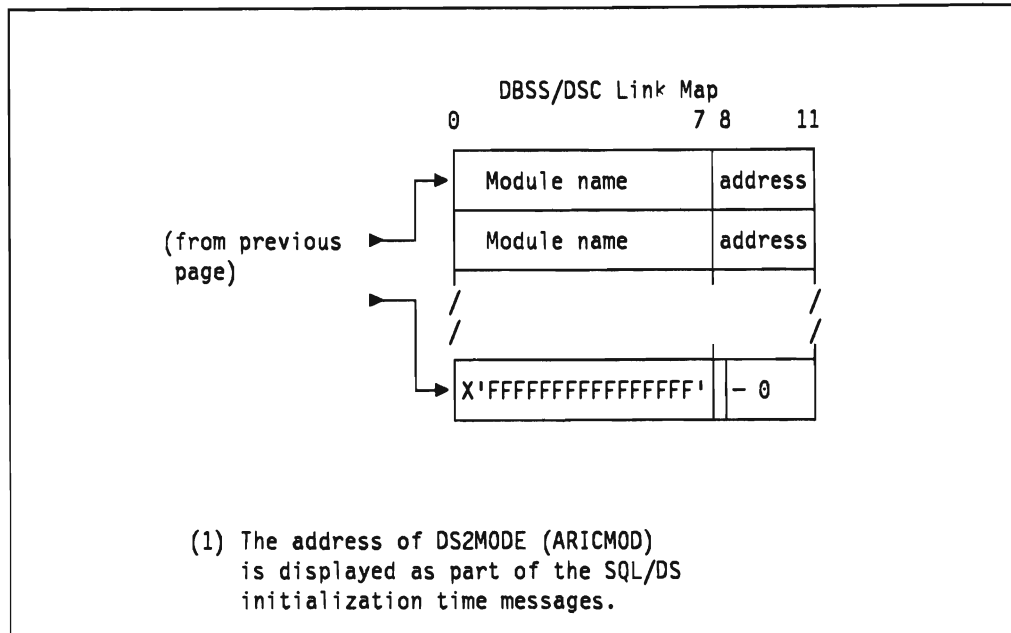


Figure 133 (Part 2 of 2). SQLIDS Link Maps and Access

Dump Navigation

Register 10 — Pointer to YTABLE1/RDAREA/RMLO (Bytes 0 — 7 contain YTABLE1-, RDAREA---, or RMLO---- as eyecatchers).

Register 13 — Pointer to save area (in stack) except for the Dispatcher, in which case it points to a DCE.

The register 9 save area slot in the DCE contains the pointer to the save area of the dispatcher's caller, either ARICDWT or ARICWAT).

Generally:

- Register 12 is used as a base register.
- Register 11 is used as a data register.
- Register 11 = Register 13.

Due to the structure of the SQL/DS system-dependent routine (ARISYSD), it is recommended that you use the registers displayed in the SQL/DS "mini-dump."

The save area is the start of the module's dynamic storage. Dynamic storage is found in the stack.

If register 10 points to a YTABLE1/RDAREA/RMLO, register 10 + X'0C' points to STACK.

Diagrams follow in Figure 134 on page 254 through to Figure 135 on page 254.

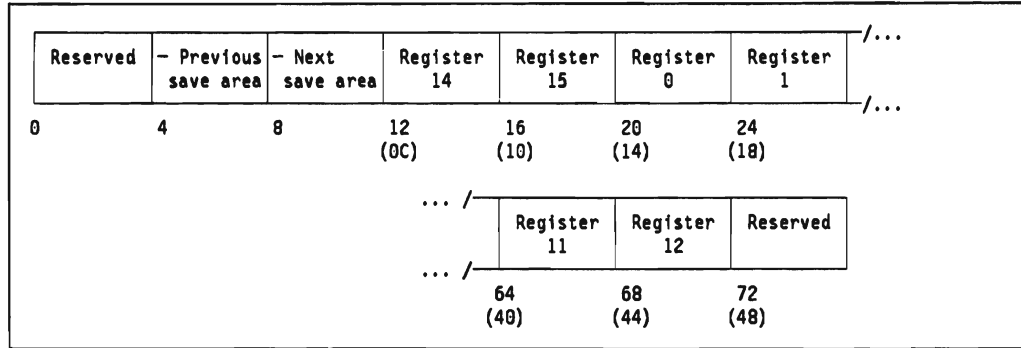


Figure 134. Register 13

Assume a program check occurred in Module C, but Module A is the source of the problem. The dynamic storage for Module A can be found by "back tracking" through the save area chain (Register 13 is a pointer to the Module C save area).

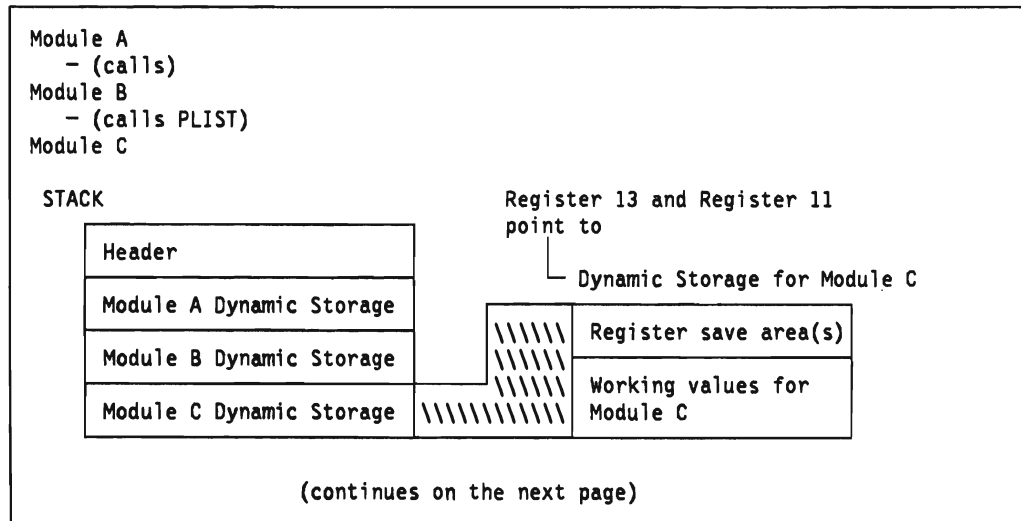


Figure 135 (Part 1 of 2). Dump Navigation Diagram

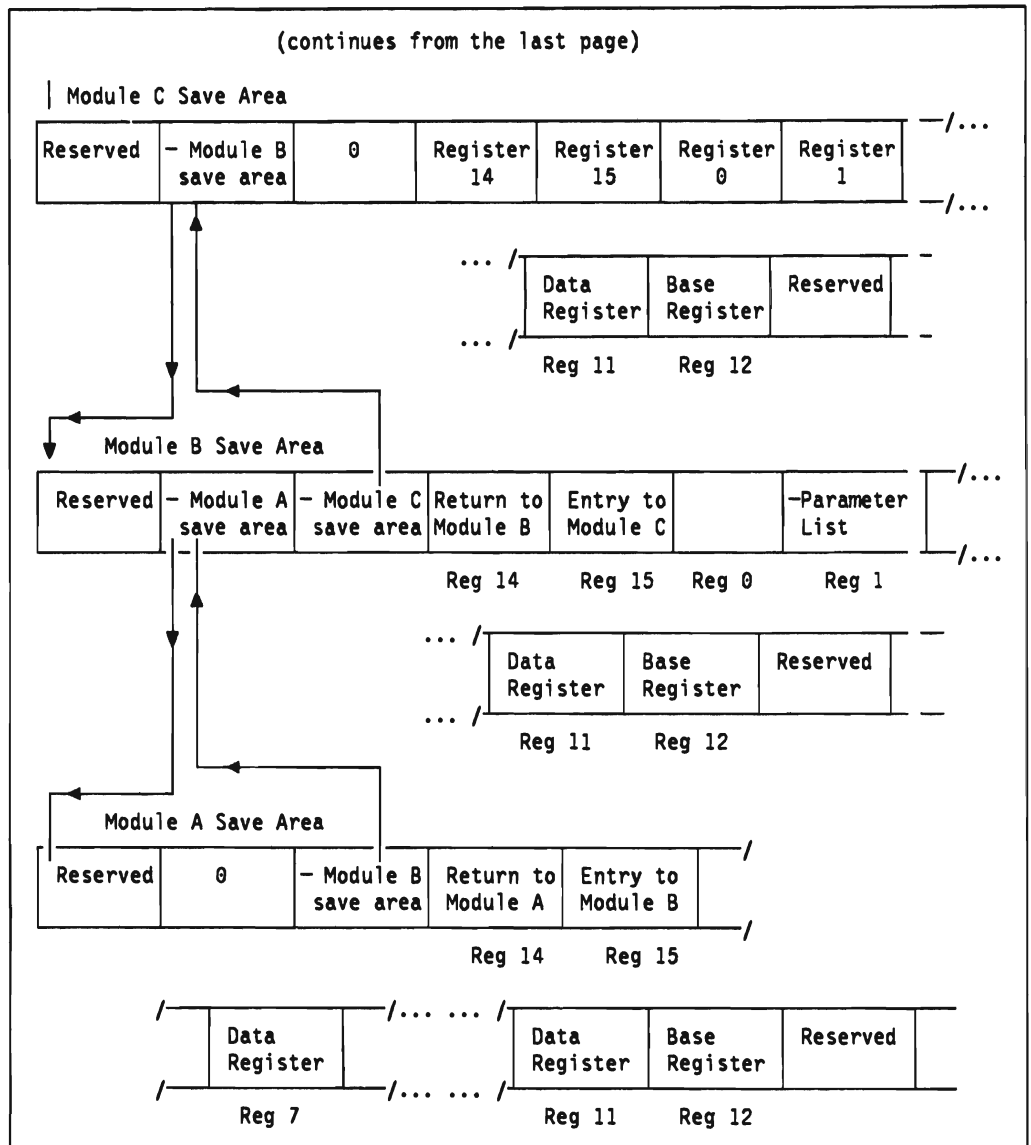


Figure 135 (Part 2 of 2). Dump Navigation Diagram

Storage Layout after Initialization

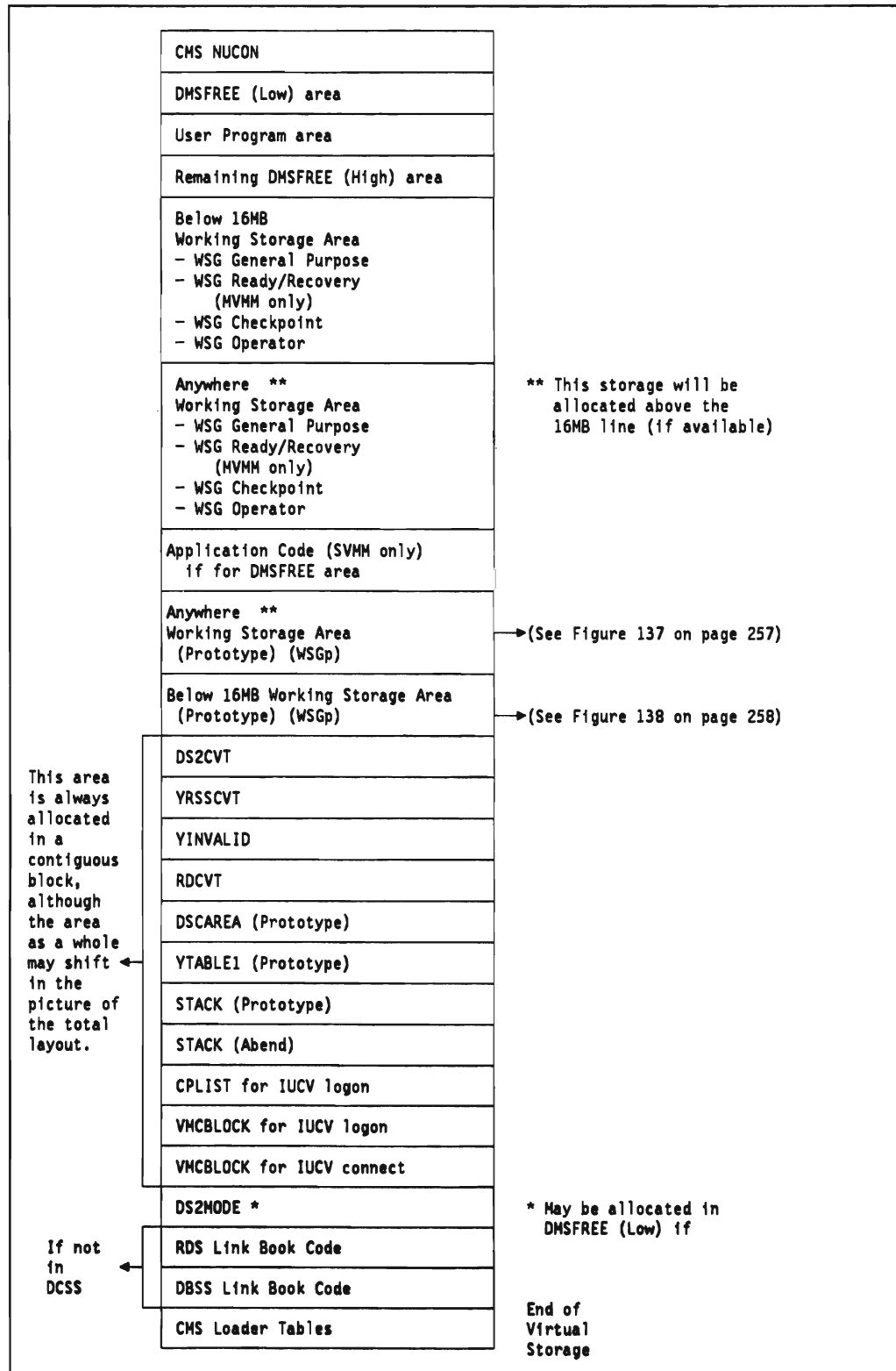


Figure 136. Storage Layout After Initialization

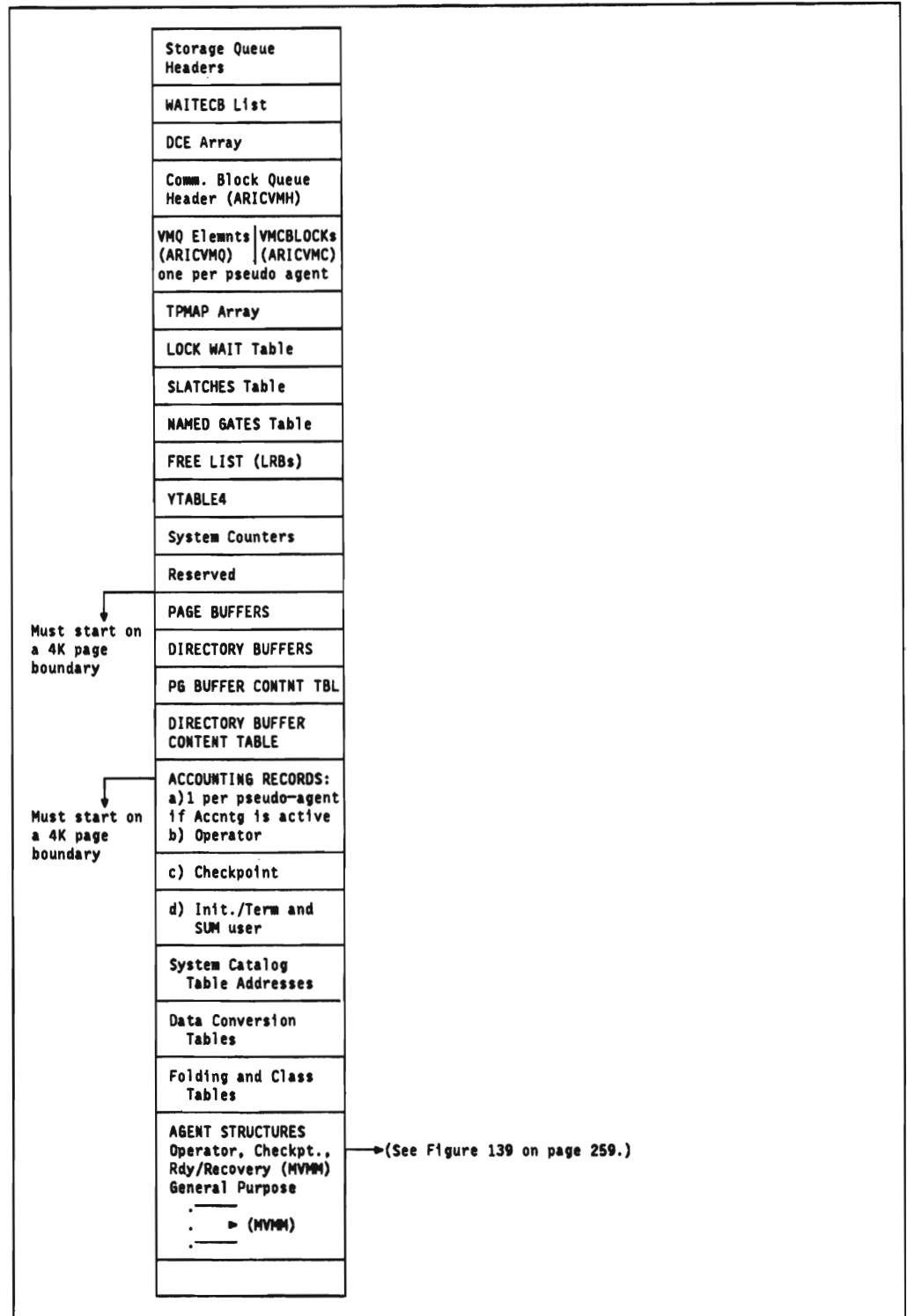


Figure 137. Anywhere Working Storage Area Layout After Initialization. This is an expansion of the Anywhere Working Storage Area in Figure 136.

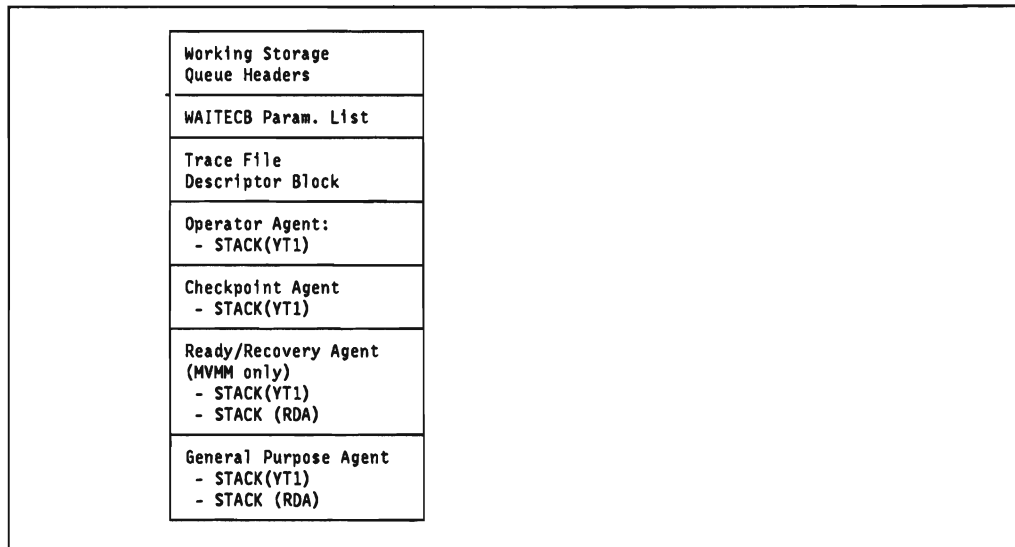


Figure 138. Below 16MB Working Storage Area Layout After Initialization. This is an expansion of the Below 16MB Working Storage Area in Figure 136 on page 256.

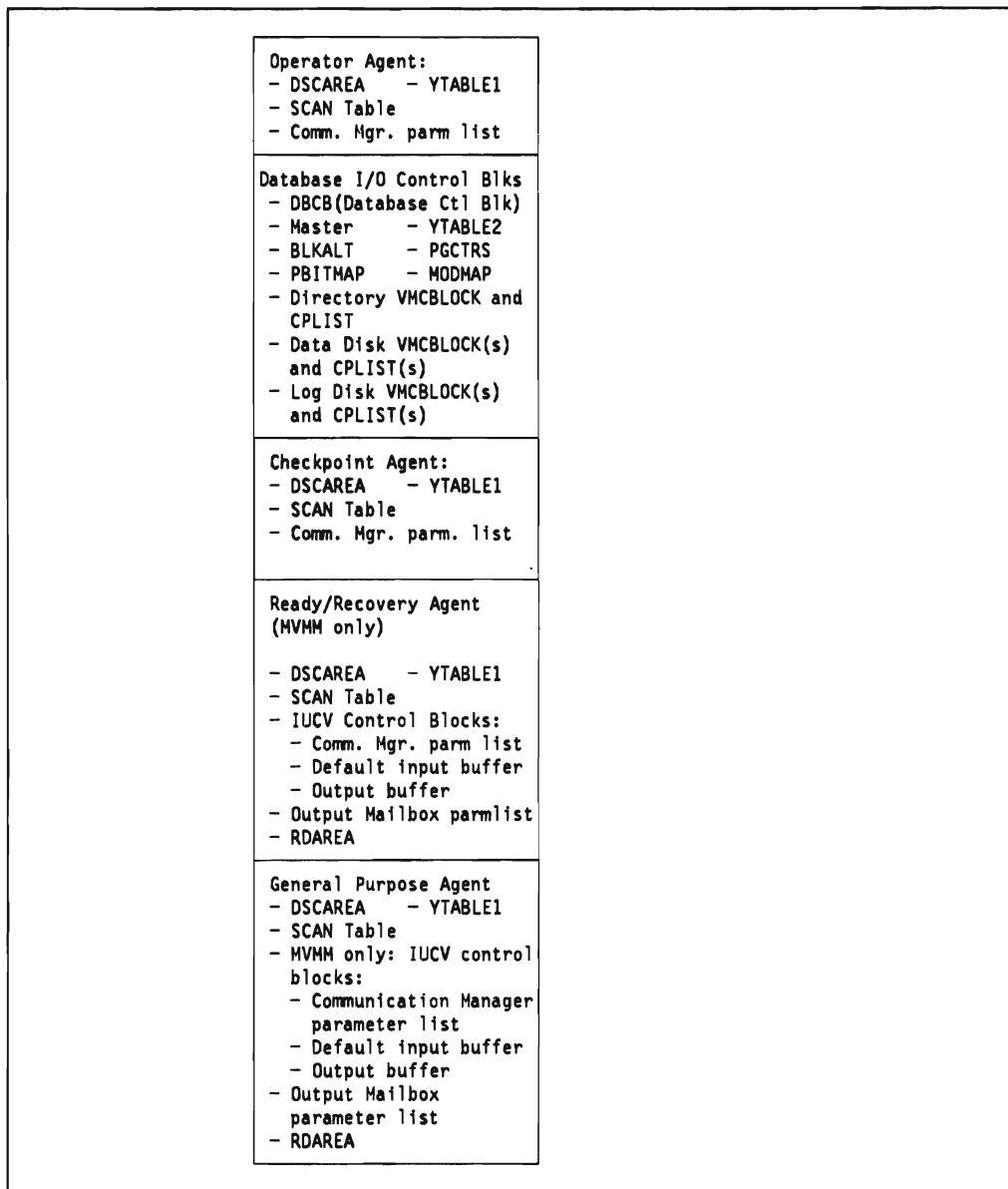


Figure 139. Agent Structures Storage Layout After Initialization. This is an expansion of the Agent Structures storage layout in Figure 137 on page 257.

Major Control Blocks

Figure 140 on page 261 shows the major control blocks and their interconnections for an agent structure.

Note: DRRMSTR and ASPAREA are only allocated when you are using PROTOCOL=AUTO on the application server. They are only actually used when the application server is processing a DRDA protocol conversation.

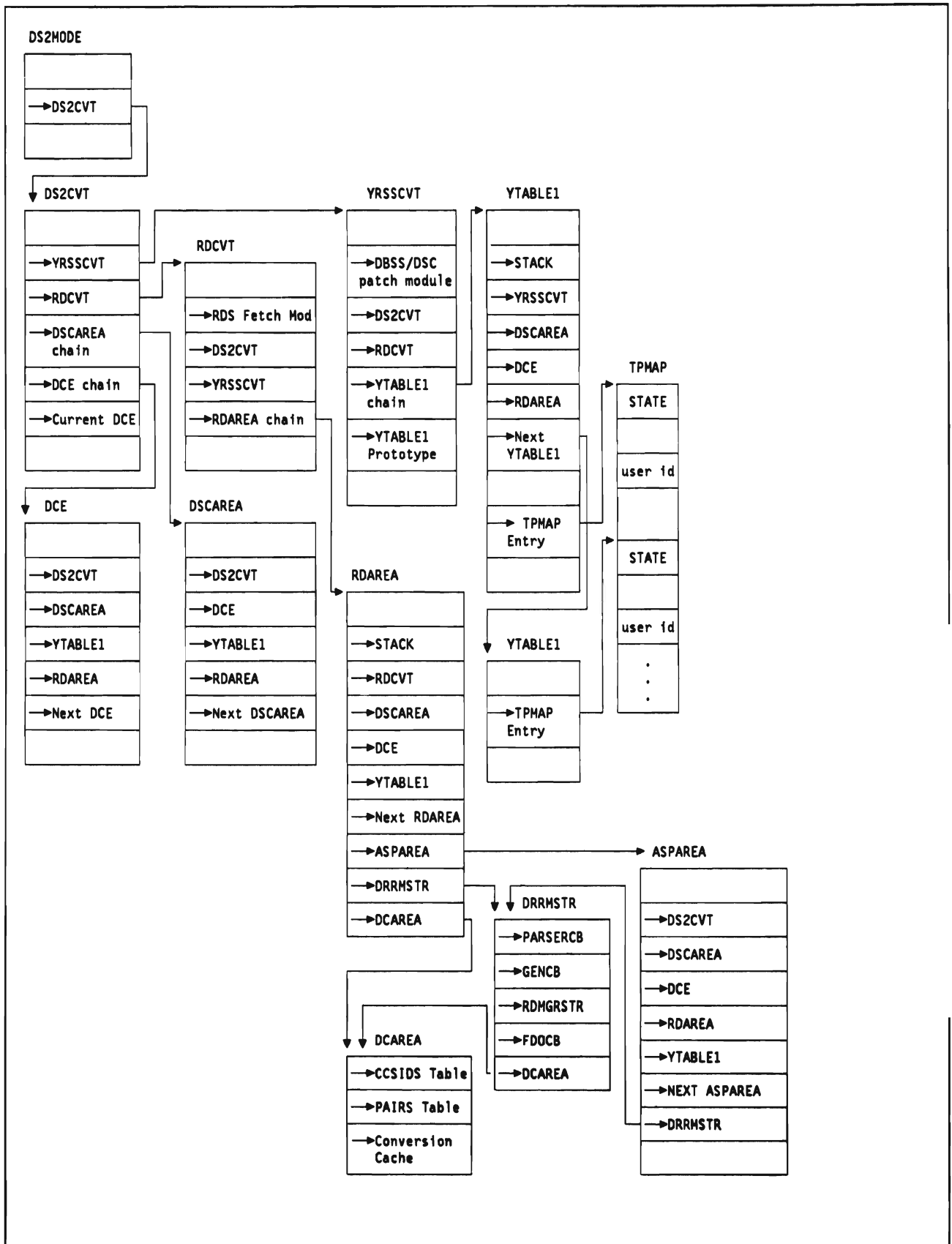


Figure 140. Major Application Server Control Blocks and Their Interconnections

Locating SQL/DS Statements Associated with a System Error

To find a package SECTION and associated SQL/DS application statements, the following process has been compiled. This process is illustrated in Figure 142 on page 264.

```
INPUT: DUMP MINIDUMP R10 --> YTABLE1
      R13 --> REGISTER SAVE AREA
```

```
YTABLE1: R10 +X'1C' --> RDAREA
```

```
RDAREA: +X'10' --> RDCVT
        +X'2C' --> index number into PROGS structure
                for this package
        +X'44' --> USERID running package
        +X'4C' --> 2 bytes = length of package name
                8 bytes = package name
        +X'58' --> package creator
```

```
RDCVT: +X'40' --> entry point of ARIXEBR .
        +X'68' --> PROGS structure
```

PROGS:

The PROGS structure consists of 1 entry for each PACKAGE loaded into storage. Each entry is X'28' bytes. To find the entry for N, where N is the index found at +X'2C' in the RDAREA.

$$A(\text{PROGS}) + (\text{X}'28' * (\text{N}-1))$$

For this PROGS entry:

```
+X'4' = package author
+X'C' = package name
+X'20' = address of package header
```

Now we need to find the address of the package section that was executing at the time of the failure. To do this we need two things:

```
R13 --> last save area
RDCVT +X'40' --> entry point of ARIXEBR
```

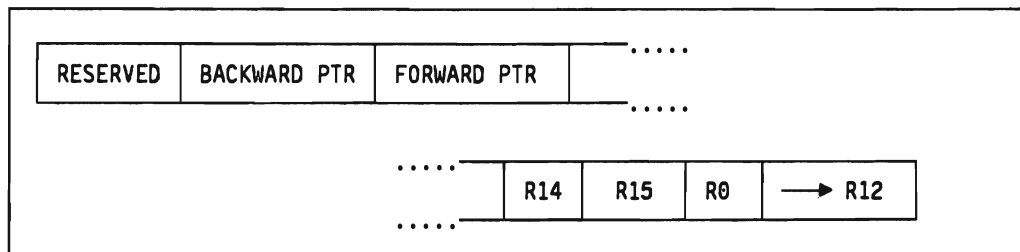


Figure 141. SAVE AREA Format

We now need to go backwards through the save areas to find the save area used when ARIXEBR was called (ARIXEBR passes control to the AUX). R15 in the save area is equal to the ARIXEBR entry found in the RDCVT +X'40'. When the ARIXEBR entry save area has been located, use the save area forward pointer (save area +8) to locate the save area where ARIXEBR called the package section.

In this save area, +X'10' points to the beginning of the section executing for this AUX. Subtract X'20' from the value found there. Call this value 'Section Location'.

Package Header:

+X'0' The halfword giving the number of sections for this package.
+X'98' The start of the section entry for the first section in this package. There is one entry for each section in the package. Each entry is X'1C' long. At X'18' into each entry is the address of where the section is located in storage.

Scan all section entries and find the entry whose value at offset X'18' matches the Section Location value calculated in the previous step. The first X'12' bytes of this entry are the CURSOR NAME. If the CURSOR NAME is blanks, you may want to determine the section number. The formula for this is:

$$(\text{ADDR}(\text{section entry}) - \text{ADDR}(\text{package header}) + \text{X}'98') / \text{X}'1\text{C}' = \text{SECTION \#}$$

You can now go to the application program prep listing and determine the call that was executing. The expansion of the SQL/DS calls in the listing produced by PREP has a RDIIN for each call. The RDIIN built for each call has the section number placed in RDISECT#. Look in the listing for the call in which the RDISECT# is equal to the SECTION # calculated in the previous step.

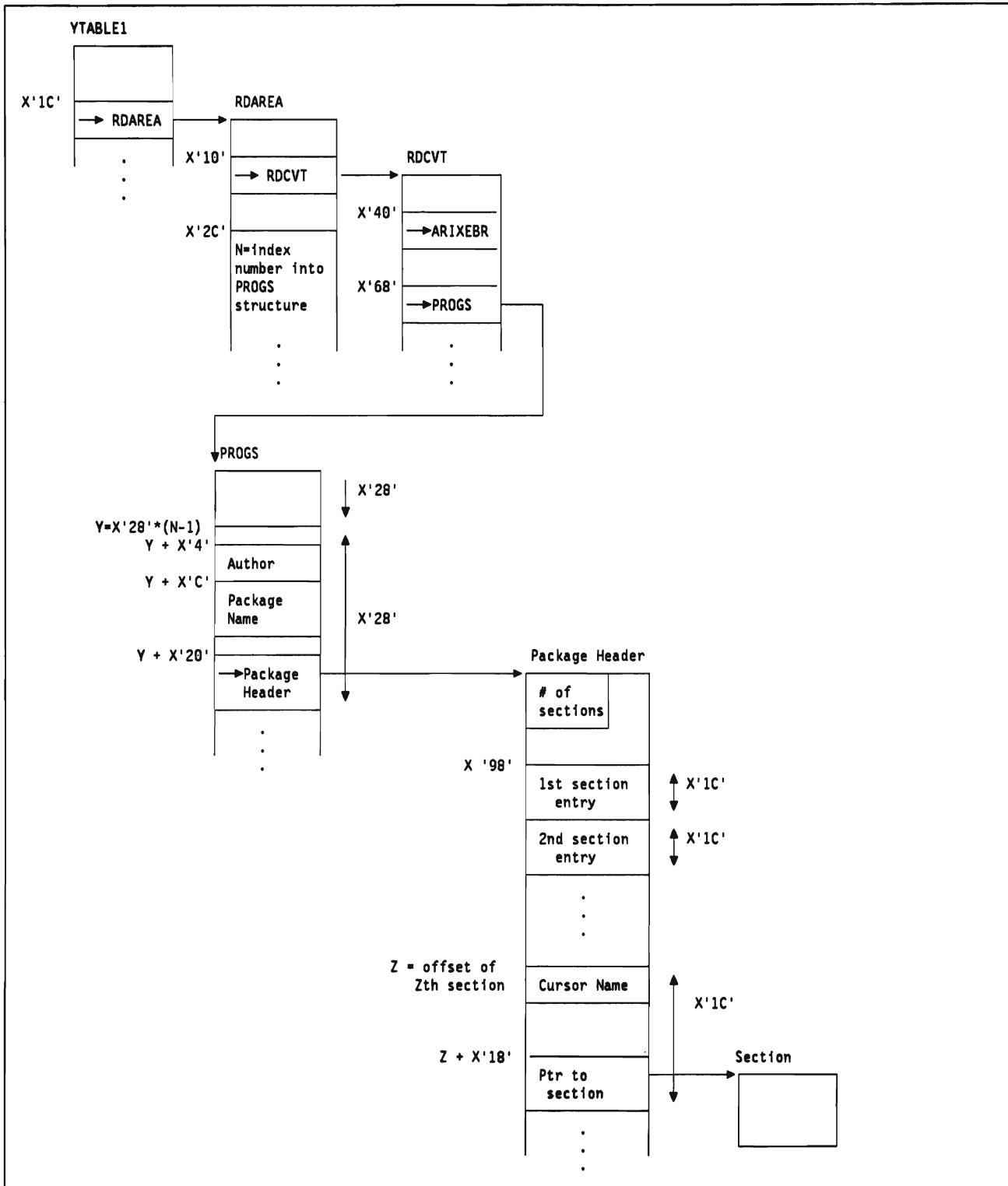


Figure 142. Locating SQL/DS Statements Associated with a System Error

DBSS OP Codes

The OP codes are located in field:

YT10PCPD (in YTABLE1) — It contains the current (or last) DBSS OPCODE.

OP CODE	MEANING
LUW Codes	
1	Begin work
2	Commit work
3	Save work state (for rollback)
4	Rollback work to save point
Lock Codes	
6	Lock a DBSPACE, table, or row
7	Unlock a DBSPACE, table, or row
Data Manipulation Codes	
8	Close a scan
10	Delete a row
12	Fetch a row
13	Insert a row
14	Get the next row
15	Open a scan
17	Update a row
DBSPACE Codes	
22	Acquire a non-permanent DBSPACE
Data Control Codes	
23	Get a control record
24	Get the next control record
25	Insert the control record
26	Delete the control record
27	Update the control record
Initialization/Termination Codes	
28	Initialize DBSS within this process
30	Terminate DBSS within this process
Additional DBSPACE Codes	
31	Release a non-permanent DBSPACE
Operator Command Code	
32	Process the remote operator command
More LUW Codes	
33	Retrieve in-doubt list
34	Prepare to commit LUW

|
Sort Code

| 35 Sort an object

|
More LUW Codes

| 36 Schedule user ID into DBSS and DSC

|
Update Statistics Codes

| 37 Start update statistics

| 41 End update statistics

Certain DBSS functions are executed without formal DBSI calls to the agent that executes that function. DBSS sets (in YT1OPCOD) special pseudo-OPCODEs to cover a number of these situations as follows:

OPCODE=99: DBSS is performing warm start as a result of the initialization parameter STARTUP=W.

OPCODE=98: DBSS is performing database generation and initialization as a result of the initialization parameter STARTUP=C.

OPCODE=97: DBSS is being initialized and is restoring the database from an archive tape as a result of the initialization parameter STARTUP=R.

OPCODE=96: DBSS is being initialized and is adding new DBSPACE(s) to the database as a result of the initialization parameter STARTUP=S.

OPCODE=95: DBSS is being initialized and is adding new DBEXTENT(s) to the database as a result of the initialization parameter STARTUP=E.

OPCODE=94: DBSS is being initialized and is redefining/formatting the log data set(s) (COLDLOG) as a result of the initialization parameter STARTUP=L.

OPCODE=93: DBSS is performing a SQL/DS checkpoint or checkpoint and archive in the checkpoint agent (agent 2).

OPCODE=92: DBSS is performing an asynchronous rollback or commit of an LUW. This can be caused by conditions such as deadlock and the FORCE operator command.

OPCODE=91: DBSS is executing (in the operator agent) a SQL/DS operator command from the SQL/DS operator.

Problem Isolation and The Trace Facility

Trace Facility

Trace In Storage

The SQL/DS trace facility can be used in problem determination for long-running tasks or to trace intermittent problems. In these cases, when large amounts of trace records are produced, the trace records should be directed to a designated wrap around memory area known as a trace buffer.

The trace buffer is a wrap around memory area holding variable length trace records. Execution of a trace point can produce one or more trace records. A typical trace point produces several hundred bytes of information.

For information on using the trace buffer, see the *Operation* manual.

Using Trace for Deadlocks

Trace can be used to determine the resources that are in conflict causing deadlocks. Refer to the *Operation* manual for the specifics on the TRACE command.

To invoke trace for the LOCK component specify the following:

1. Issue the TRACE ON operator command.
2. Specify the user IDs of the deadlock victims or all if they are the only ones running (message ARI0084D).
3. Specify DBSS to message ARI0087D.
4. Specify LOCK 2 to message ARI0090D.

Alternately, you can use the startup parameter TRACDBSS = 00020000000.

Now trace is active for the Lock component and the applications can be started. Refer to the *Operation* manual for the details on deactivating the trace facility.

Trace Sequences During Locking: At the start, a deadlock victim has DBSS OP Code=92. The entries to note at this stage are the agent trace point numbers, and their user IDs. For example,

Tracepoint 627 for this user ID is the wakeup of the agent to roll them back because of deadlock.

Tracepoint 626 occurs when an agent needs to wait for access to a gate.

For a situation that does not involve deadlocks, you would normally see the following sequence of trace points:

- Tracepoint 652 ARIYK41 — request a lock on a gate.
- Tracepoint 624 ARIYK12 — look for a deadlock cycle.
- Tracepoint 625 ARIYK12 — no deadlocks are detected.
- Tracepoint 626 ARIYK18 — wait for the lock to be freed.
- Tracepoint 627 ARIYK18 — wake up agent (either rollback, or lock freed).

For a deadlock situation, the wait is different. Trace points appear in the sequence as follows:

- Tracepoint 652 ARIYK41 — request a lock on a gate.
- Tracepoint 624 ARIYK12 — look for a deadlock cycle.
- A deadlock is found this time. Therefore, the next trace tracepoint is 628 ARIYK19 L_PROK=x, where x is the agent number that should be rolled back because of deadlock.
- Tracepoint 629 ARIYK19 — rollback scheduled.
- Tracepoint 625 ARIYK12 — deadlock situation handled.
- Tracepoint 652 involving one of these actions:
 - The agent calls ARIYK18 (trace point 626) to SUSPEND, waiting for the deadlock to be cleared. (that is, the other agent rolled back).
 - The agent in tracepoint 652 is rolled back.

The best approach in determining the deadlock resources is to locate in the trace the call to ARIYK12 that woke up (via a call to ARIYK19) the other user in conflict or a -1 return code from ARIYK41 where the request was denied because of deadlock. The call to ARIYK41 identifies one of the resources that caused the deadlock situation. From this point you must backtrack collecting the lock information from the trace output for the users in conflict. This requires interpreting the gatenames in the trace records for ARIYK41. From the gatename, it can be determined what DBSPACE, table, page, row or index caused the conflict.

Once the resources are known that caused the deadlock situation, then the applications must be reviewed to understand the processing being done. If the applications are doing Data Definition Language (DDL) statements (CREATE TABLE, etc.), then the names of the objects (table, index, DBSPACE) should be checked for similarities that might cause adjacent row or key conflicts in the system catalogs. If the applications are accessing the same tables(s), then the order of access should be checked.

By noting the trace point numbers and their user IDs, you can also look at resource contention. This involves finding areas in the trace where a call to ARIYK41 (trace point 652) is not immediately followed by a return from ARIYK41 (trace point (653)).

Trace Record Format for Lock Component Records:

Trace Record #652 for ARIYK41 Entry:

```
MOD_CALLED='ARIYK41' L_GNAME='0000000100008808'X L_HISMODE=6  
L_HISDUR=255 L_CONTROL=2
```

The following is a description of the keywords in the above record:

- L_GNAME is the gatename of the resource requested (See "How to Interpret GATENAME" on page 269).
- L_HISDUR is the duration for which the resource is held.
 - 1 = INSTANT
 - 2 = SHORT
 - 3 = MEDIUM
 - 255 = LONG

- **L_HISMODE** is the mode in which the resource is held.
 - 2 = INTENT SHARE
 - 3 = INTENT EXCLUSIVE
 - 4 = SHARE
 - 5 = UPDATE
 - 6 = SHARE with INTENT EXCLUSIVE
 - 7 = EXCLUSIVE
- **L_CONTROL** specifies whether or not to wait for the resource.
 - 1 = NO WAIT
 - 2 = WAIT

Trace Record #653 for ARIYK41 Exit:

MOD_RETURNED='ARIYK41 ' RETCODE=0

- **RETCODE** is the return code from ARIYK41 indicating the status of the request.
 - 7 = EXCLUSIVE MODE GRANTED
 - 6 = SHARE with INTENT EXCLUSIVE MODE GRANTED
 - 5 = UPDATE MODE GRANTED
 - 4 = SHARE MODE GRANTED
 - 3 = INTENT EXCLUSIVE MODE GRANTED
 - 2 = INTENT SHARE MODE GRANTED
 - 0 = TEST FAILED
 - 1 = DEADLOCK DETECTED

Trace Record #626 for ARIYK18 Entry:

MOD_CALLED='ARIYK18 ' L_WAITTYPE='02000000'x

- **L_WAITTYPE** is the type of wait that this user is to be placed in.
 - '02000000'x – WAITING FOR LOCK
 - '04000000'x – WAITING FOR CHECKPOINT TO RUN
 - '00200000'x – WAITING FOR I/O
 - '00400000'x – CHECKPOINT WAITING

How to Interpret GATENAME:

There are several subdivisions of GATENAME as follows:

1. The first byte describes the LOCK TYPE.

LOCK TYPE, in general, indicates level of locking. As is shown below, LOCK TYPE may indicate special case internal locks, like DATABASE lock.
2. The second byte is used for the ordinal number of the index (1 to 255) in cases of KEY-LEVEL locking when the index is unique.
3. Bytes 3 and 4 are the DBSPACE number. The value shown corresponds to the DBSPACENO column in the SYSDBSACES catalog table.
4. Bytes 5 – 8 identify the table or contain other information as shown below. When bytes 5 – 8 identify a table, the value displayed corresponds to the TABID (or LFTABID) column in the SYSCATALOG catalog table. If the DBSPACE is a package DBSPACE, the value corresponds to the TABID column in the SYSACCESS catalog table.

BYTE 1 — LOCK TYPE:

1. BIT 1, X'80'

- 1 for DBSPACE or TABLE lock.
- 0 for DATA or INDEX lock.

The lock is either PAGE level or INDEX KEY/ROW level.

2. BIT 2, X'40'

- 1 for PAGE lock.
- 0 for INDEX KEY/ROW lock.

3. BIT 3, X'20'

- 1 for INDEX KEY lock.
- 0 for ROW lock.

4. BIT 4, X'10'

- Special backup lock.

5. BIT 6, X'04'

- 1 for special system lock. If this bit is on then the lock is a fixed name. There is a fixed name for a DATABASE lock, and a fixed name for a BACKUP lock.

— GATENAME for the DATABASE lock is X'0400111111111111'.

The DATABASE lock is acquired INTENT EXCLUSIVE with LONG duration at the beginning of each LUW. When a log archive is initiated, the DATABASE lock is acquired EXCLUSIVE with SHORT duration while the checkpoint preceding the log archive is executed. This ensures that no LUWs are active (by waiting for them to end) when the begin log archive checkpoint is taken.

— GATENAME for the BACKUP lock is X'0400222222222222'.

This lock serializes the UNDO of a single update and is used by the ROLLBACK WORK process. The lock is held only for the length of time required to UNDO one log record. Contention for this resource should be brief, because no new locks are obtained during the UNDO process. A serious lockout occurs if the holder of this lock is in lock wait. This condition, if it occurs, is a system error.

- 0 for the general case.

6. BIT 7, X'02'

- 1 for INTERNAL DBSPACE. It is a system error if two users are ever contending for a INTERNAL DBSPACE.
- 0 for the general case.

7. BIT 5 and BIT 8 are reserved.

BYTE 2 — Internal Index ID if non-zero: This is the internal id of the locked index used for key locking of unique indexes.

BYTES 3 and 4 — DBSPACENO: This is the DBSPACENO that is being contended for. Based on what follows, either the entire DBSPACE is locked, or a part of it is locked. The DBSPACENO is displayed as a hexadecimal value, but in order to

associate this value with a DBSPACENO in SYSDBSACES, the value must be converted to decimal.

Identify the DBSPACE

The following example shows how to identify the DBSPACE:

Suppose that BYTES 3 and 4 have DBSPACENO=X'000A'.

The hexadecimal value translates to decimal 10.

The following SELECT statement identifies the DBSPACE:

```
SELECT DBSPACENO,DBSPACENAME FROM SYSTEM.SYSDBSACES
      WHERE DBSPACENO=10

DBSPACENO  DBSPACENAME
-----  -----
          10  COLLEGE
* END OF RESULT ***** 1 ROWS DISPLAYED *****
```

BYTES 5 through 8 – Resource in DBSPACE:

This field, x, identifies the resource in the DBSPACE that is being contended for. x varies as follows:

1. If BYTE 1 – LOCK TYPE is X'80', indicating DBSPACE or TABLE lock, then
 - If x = 0, there is no sub-resource in the DBSPACE that is being contended for. Thus, the gatename indicates the resource is the whole DBSPACE.
 - If x not = 0, x is the TABID. Thus, the gatename indicates that the resource is a TABLE in the specified DBSPACE.

The TABID is displayed as a hexadecimal value, but in order to associate this value with a TABID in SYSCATALOG, the value must be converted to decimal.

Identify the Table

The following example shows how to identify the table:

Suppose that BYTES 5 through 8 contain X'00008001'. The low order two bytes, X'8001' is a half-word and is negative.

The negative hexadecimal value translates to decimal -32767.

The following SELECT statement identifies the table:

```
SELECT TABID,TNAME FROM SYSTEM.SYSCATALOG -
      WHERE TABID=-32767 AND DBSPACENO=10

TABID  TNAME
-----  -----
-32767  STUDENTS
* END OF RESULT ***** 1 ROWS DISPLAYED ***
```

The following technique may be useful for the above translation of a negative half-word to a negative decimal value.

Subtract the half-word from X'FFFF', and add 1. For the example above we have:

```
X'FFFF'  
- X'8001'  
-----  
X'7FFE'  
+ X'0001'  
-----  
X'7FFF'
```

Translate the result to decimal. For the example above we have X'7FFF'
= DECIMAL 32767

Be sure to precede the result with a negative sign, thus using "...WHERE
TABID=-32767..." in your SELECT statement above.

2. If BYTE 1 – LOCK TYPE is X'60', indicating PAGE lock on INDEX, then x is the internal page identifier of the page containing the INDEX. Thus, the GATENAME indicates that the resource is a PAGE in the specified DBSPACE, and the PAGE is held to lock an INDEX KEY in the PAGE.
3. If BYTE 1 – LOCK TYPE is X'40', indicating PAGE lock on DATA, then x is the internal page identifier of the page containing the data. Thus, the GATENAME indicates that the resource is a PAGE in the specified DBSPACE, and the PAGE is held to lock a ROW in the PAGE.
4. If BYTE 1 – LOCK TYPE is X'20', indicating INDEX KEY lock, then x will differ for unique and non-unique indexes. The GATENAME indicates that the resource is an INDEX KEY in the specified DBSPACE. For unique indexes, x will be the TID (tuple identifier – the internal name for the pointer to a row) of the row associated with the INDEX KEY. In this case BYTE 2 will contain the internal index ID. For non-unique indexes, x is the internal hash value associated with the INDEX KEY. In this case BYTE 2 will contains zeros. (Because the INDEX KEY may be larger than four bytes, x is never the INDEX KEY itself; rather, the INDEX KEY is hashed down to four bytes. Therefore, it is possible for lock requests for two different INDEX KEYS in non-unique indexes in the same DBSPACE to cause contention.)
5. If BYTE 1 – LOCK TYPE is X'00', indicating ROW lock, x is the TID (tuple identifier – the internal name for the pointer to a ROW). Thus, the GATENAME indicates that the resource is a ROW in the specified DBSPACE.

Appendix A. RDIIN

This chapter documents product-sensitive programming interfaces.

Product-sensitive programming interface

The RDIIN is the control block passed to the application server by the application program. (RDICTYPE indicates the call type.) A sample invocation is illustrated.

Dec(Hex) RDIIN

(0)	RDICTYPE - (1)		RDIAUTHR - Creator of access module (2)					
8 (8)	RDIAUTHR (continued)		RDIPROGL - (3) Length of module name		RDIPROGN - module name (3)			
16(10)	RDIPROGN (continued)			RDISECT#	RDICLSSC			
24(18)	RDICODEP → error code structure (SQLCA)			RDIVPARM →(4)				
32(20)	RDIAUXPA →(5)			RDISQTIE → SQTIE				
40(28)	RDICNFLG			RDIVIND	RDIAIND	RDIEERROR	RDIDFLG	
	RDISPEC (6)	RDICALL (6)	RDIIWAIT (6)	RDIRELSE (6)	(7)	(8)	(9)	(10)
48(30)	RDIMBLN - Length of input Mailbox (11)			RDIRELNO (12)	RDICISL (13)	RDIDATE (14)	RDITIME (14)	
56(38)	RDIFDBCK → Feedback area in preprocessors for database options (15)			RDIEXTP → RDIEXT area (16)				
64(40)	RDIRESD							
	RDIFORCI (17)	RDIFORN (17)	RDIDBNTY (18)	RDIASREL (18)				

Figure 143. RDIIN Control Block

(1) This is a list of some of the RDICTYPE values:

- 30 - package call
- 35 - setup call
- 40 - describe call
- 45 - close call
- 50 - open call
- 120 - connect/schedule call
- 125 - recovery list call
- 130 - prepinit call
- 131 - CREATE PROGRAM call
- 132 - DROP statement call
- 135 - lookup call
- 140 - SQL call
- 145 - prepfinish call
- 155 - operator command call
- 160 - prepare-to-commit call
- 165 - set/reset exit call
- 166 - modify cancel call
- 170 - operator command continue call
- 175 - open call to unload/reload package
- 180 - package call to unload/reload package
- 185 - close call to unload/reload package
- 190 - set/change lang call
- 200 - reorganize index call
- 201 - rebind package call

(2) For statements with a user ID modifier on "IN program" clause or in CREATE PROGRAM, this is set to that value. When user ID is defaulted to in these cases, it is set to blanks. In other cases, the value is taken from PREP parameters or defaults.

(3) For statements with "IN program" clauses, these are set to that value. In other cases, the value is taken from PREP parameters or defaults.

(4) RDIVPARM points to:

- Input Name List for prep-init.
- Input or output SQLDA on EXECUTE, OPEN or FETCH.
- SQL statement on set-up call, SQL call, and EXEC immediate package call.
- User SQLDA on DESCRIBE.
- user ID on CONNECT.
- pointer to input RMRE on get recovery list or prepare-to-commit; pointer to input OCOMBLK on OPERATOR.
- I/O variable pointers on prep-init call. RDIIN pointer on look up calls (for feedback).
- user's buffer to return access module row information for unload program.
- package row to be inserted for reload program.

- (5) RDIAUXPA points to:
 - Input SQLDA on package call.
 - Password on CONNECT.
 - Output Name List on prep-init.
 - Name on lookup (search argument for comparison against PLSTCNAME of SLT).
 - Token/DCLLIST/caller's RDIIN on SQLCALL.
 - User SQLDA on DESCRIBE.
 - Pointer to output RMRE on get recovery list or prepare-to-commit; pointer to output OCOMBLK on operator.
 - Cursor name for statements with a host variable cursor-name.
 - Pointer to a pointer pair on a setup call
- (6) - See 'RDIIN Flags' on page 276.
- (7) - RDIVIND character:
 - 'S', 'I', or 'O' for package call.
 - 'I' on OPEN with input parameters.
 - 'A' on UNLOAD and RELOAD calls.
 - 'H' on if WITH HOLD specified on DECLARE ... CURSOR.
- (8) - RDIAIND character:
 - 'I' or ' ' for package call. 'I' on OPEN with input parameters.
- (9) - RDIERROR character:
 - 'E', 'W', 'B', or ' '. Tells Communication Manager to check for errors, warnings, or both.
- (10) - RDIFLG: Used for label support.
- (11) - See 'Mailbox contents' on page 276.
- (12) - RDIRELNO character:
 - SQL/DS release number in which the RDIIN was used/generated (by a preprocessor). ' ' or X'00' means Release 1, X'02' means Release 2, and so forth.
- (13) - Isolation Level: 'C' for cursor stability; 'R' for repeatable read.
- (14) - RDIDATE & RDITIME
 - 'I' for ISO
 - 'J' for JIS
 - 'U' for USA
 - 'E' for EUR
 - 'L' for LOCAL
 - ' ' for not defined
- (15) - RDIFDBCK - feedback pointer
- (16) - See RDIEXT on page 277.
- (17) - RDIFORCI 'Y' indicates force index requested. RDIFORNB 'Y' indicates force no blocking.
- (18) - RDIDBNTY indicates the data type of the dbname host variable.
 - RDIASREL indicates the release number of the SQL/DS application server.

RDIN Flags

OFFSET	FIELD NAME	BITS	MEANING
40(28)	RDISPEC (PREP Time only)		
	RDIPAMER	1...	Indicates error condition on package reload.
	RDIISCUR	.1..	Causes SQL/DS to turn ISCURSOR bit on in PPOPGNSTR. Set by caller on SQL call if prompted "DECLARE_CURSOR FOR SELECT..." (In Release 1, preprocessors initialized RDISPEC to X'40', which turned this bit on. Because this is used only at PREP time, there is no conflict.)
	RDIBLK	..1.	"BLOCK" preprocessor parameter specified. Apply blocking to the eligible sections in the access module.
	RDICHECK	...1	Tell the application server we are in check mode. (PREP INIT).
	RDIKEEP 1..	Tell the application server to keep current run authorizations during reprep or reload package or replace package on reload.
	RDIDESCR1..	Allow DESCRIBEs. Specified for CREATE PROGRAM.
	RDINew1.	New package on CREATE PROGRAM for OPEN package calls. If on, user wants to create a new package by way of reload. If off, user wants to replace an existing package or if none exists create a new package by way of reload.
	RDIMODFY1	Establish the access module as modifiable.
41(29)	RDICALL character:		'E'(COMMIT), 'R'(ROLLBACK), 'C'(CONNECT), 'S'(SET) or 'R'(RESET) on set/reset exit call, 'I' for implicit CONNECT on prep init call, or Reorganize index on a REORG call, 'B' for Extended EXECUTE or Extended DESCRIBE, 'D' for Extended Declare lookup call, 'F' for FETCH, 'P' for PUT, 'H' for CHANGE language, 'G' for GET language, 'T' for special operator commands (see Figure 145 on page 278 for specifics), or ' '. These are unique only within a particular call type.
42(2A)	RDIIWAIT character:		Always ' '. (Unused)
43(2B)	RDIRELSE character:		'R' or ' '.

Mailbox contents and lengths:

- Mailbox header
 - input header 88 bytes
 - output header 140 bytes

- RDIIN - 68 bytes
- SQLDA - 16 bytes + 44 bytes per variable
- Input Variables - Number of bytes required to store these variables
- Other values passed to SQL/DS (user id, etc.) - Number of bytes required to store the values.

RDIEXT: RDIEXT is a structure used to store additional information associated with an SQL request. It is used as an extension to RDIIN. This extension is required, and RDIEXTP *must* point to it.

Dec(Hex) RDIEXT

0 (0)	RDIEXTEC - eyecatcher "RDIEXT"	
8 (8)	RDIEXTL - length of RDIEXT	RDIEXTA - pointer declarations RDIDBNMP (1) - points to the database name
16(10)	RDICONSP - points to the consistency token	RDIBPOPT - points to the bind prep options
24(18)	RDIPIPD (2) - points to the connect indicator	Reserved
32(20)	Reserved	
40(28)	Reserved	
48(30)	Reserved	

Figure 144. RDIEXT Structure

- (1) If the database name host variable is declared as a VARCHAR field, RDIDBNMP points to a 2 byte length field, followed by the database name (in the CONNECT statement). If the host variable is declared as CHAR(n), RDIDBNMP points to the database name field which can range from CHAR(1) to CHAR(18).
- (2) RDIPIPD is a pointer to a CHAR(8) field representing a PIP data identifier (ARIEMQVQ). This is only for use with RDICTYPE=120, when connecting to perform operator commands.

OPERATOR COMMAND CALL (Commands Issued from ISQL):

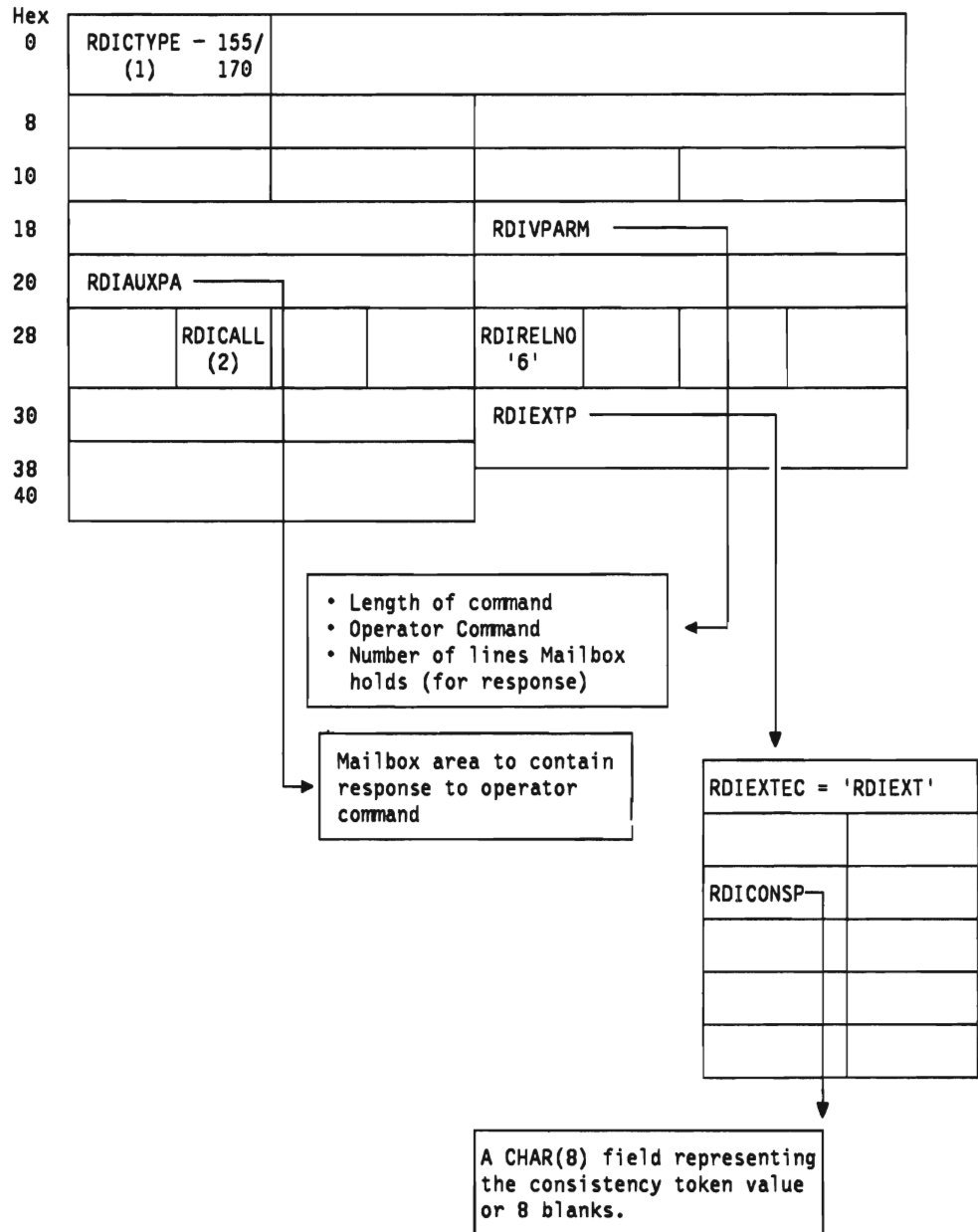


Figure 145. OPERATOR CALL (Commands Issued from ISQL)

- (1) RDICTYPE is 155 on first operator call and 170 on subsequent operator calls after the first one.
- (2) RDICALL is set to 'T' for a fixed format response (instead of formatted messages), for SHOW CONNECT, SHOW LOCK USERS, and SHOW LOCK WANTLOCK.

Special Operator Command Response Layouts:

The following are the layouts for the 80 byte records returned in the mailbox area when one of the commands SHOW CONNECT, SHOW LOCK USER or SHOW LOCK WANTLOCK is executed with an RDICALL value of 'T'. The offset values are shown in decimal format on the left-hand side of each layout.

0	record type 'CC'
2	VM user ID
10	SQL user ID
18	application requestor level (e.g. ARI03040)
26	application server name
44	state start time (370 TOD clock)
52	conversation start time (370 TOD clock)
60	CPU time if accounting indicator is 'Y' (fixed)
64	status
65	accounting indicator - 'Y' if on, otherwise 'N'
66	protocol indicator - X'00' for private flows, X'02' for DRDA
67	unused

Figure 146. SHOW CONNECT Common Record (CC). There is one record for each database connection.

0	record type 'CL'
2	SNA LUWID
37	DRDA External name (first 43 characters)

Figure 147. SHOW CONNECT LUWID Record (CL). This record may appear following a 'CC' record and represents additional information for the connection.

0	record type 'CA'
2	agent number
6	work status (0-NIW, 1-NEW, 2-R/O, 3-R/W)
7	subsystem or application (0-APPL, 1-SUBS)
8	agent processing status (1-not processing SQL operator command 2-processing SQL operator command 3-not processing and in wait 4-processing and in wait 5-waiting log archive checkpoint 6-processing LPAGEBUF RSCP)
9	wait status (4-communication, 5-lock, 6-checkpoint, 7-out of page, 8-out of block, 9-I/O, A-WITH LPAGE...)
10	filler
12	trandid (fixed)
16	resource consumption (fixed)
20	rollback/commit type (R-rollback, C-commit or blank)
21	rollback/commit status (3-active, 4-scheduled)
22	rollback/commit requestor (4-scheduled, 5-user, 6-system, 8-lock limit, 9-DBSS limit)
23	filler
24	package creator
32	package name
40	package section number (fixed)
42	unused

Figure 148. SHOW CONNECT Active Record (CA). This record may appear following either a 'CC' or a 'CL' record and represents additional information for the connection.

0	record type 'CP'
2	checkpoint agent status (1-not active, 2-waiting start log archive, 3-waiting start checkpoint archive, 4- waiting start checkpoint, 5-processing checkpoint archive, 6-processing checkpoint)
3	unused

Figure 149. SHOW CONNECT Checkpoint Record (CP). There is one record for the checkpoint agent status.

0	record type 'CI'
2	indoubt agent number
6	indoubt user ID
14	VM user ID
22	indoubt coordinator
30	indoubt adaptor (fixed)
32	transaction
36	terminal operator ID
40	terminal ID
44	unused

Figure 150. SHOW CONNECT Indoubt Record (CI). There is one record for each in-doubt logical unit of work.

0	record type 'W1'
2	agent number
5	SQL user ID
13	lock type
17	dbspace number
22	lock qualifier
33	request state
39	request mode
42	duration
47	VM user ID
55	database connection time (370 TOD clock)
63	unused

Figure 151. SHOW LOCK WANTLOCK Record (W1). There is one record for each agent in a lock wait.

0	record type 'L1'
2	filler
4	agent number
7	SQL user ID
15	dbspace number
20	lock type
24	filler
28	number of SIX modes (fixed)
30	number of IS modes (fixed)
32	number of IX modes (fixed)
34	number of S modes (fixed)
36	number of U modes (fixed)
38	number of X modes (fixed)
40	number of waiters (fixed)
42	VM user id
50	database connection time (370 TOD clock)
58	unused

Figure 152. SHOW LOCK USER Record (L1). There is one record for each database resource an agent has locked.

0	record type 'L2' or 'W2'
2	SNA LUWID
37	unused

Figure 153. SHOW LOCK Luwid Record (L2 or W2). This type of record may follow an L1 or W1 type record and represents additional information about the database connection of an agent holding or waiting for a resource.

_____ End of Product-sensitive programming interface _____

Appendix B. Catalog Updates and References

Authorization

This section lists SQL authorization functions and discusses how they reference and update the SQL/DS system catalog tables. For descriptions of these functions refer to the *SQL Reference* manual.

GRANT AUTHORITIES:

- Granting CONNECT authority.
 1. If the GRANT CONNECT is to change the issuer's password, check in SYSTEM.SYSUSERAUTH for a row where NAME equals the issuer and AUTHOR is blank. If it is found, update PASSWORD with the password specified in the GRANT CONNECT statement.
 2. If the GRANT CONNECT is for another user, check SYSTEM.SYSUSERAUTH for the issuer having DBA authority. If the issuer does not have DBA authority, the GRANT statement fails.

Check in SYSTEM.SYSUSERAUTH for a row with NAME equal to the user ID and update PASSWORD with the password specified. Otherwise, a row does not already exist, so insert a row into SYSTEM.SYSUSERAUTH for the user ID specified. Repeat the check and update or delete processing above for each user ID specified in the GRANT CONNECT statement.

- Granting RESOURCE, SCHEDULE or DBA authority.
 1. Check SYSTEM.SYSUSERAUTH for the issuer having DBA authority. If the issuer does not have DBA authority, the GRANT statement will fail.
 2. Check SYSTEM.SYSUSERAUTH to see whether the row already exists for the user ID(s) specified. If it is found, update the authority column specified, and update PASSWORD if a password was also specified. If no row exists for the user ID(s) specified, insert a row into SYSTEM.SYSUSERAUTH for each user ID, set each authority column to Y and, if a password was specified, fill in PASSWORD.

GRANT TABLE PRIVILEGES:

- Granting table privileges (except UPDATE on specific columns)
 1. Check SYSTEM.SYSTABAUTH to see whether the issuer has the GRANT OPTION on the privilege(s) specified for the table or view. If not, check in SYSTEM.SYSUSERAUTH to determine if the issuer has DBA authority and check in SYSTEM.SYSTABAUTH to determine whether the owner has the GRANT OPTION on the privilege(s) specified. If the issuer has the GRANT OPTION, or the issuer is a DBA and the owner has the GRANT OPTION, then insert a row into SYSTEM.SYSTABAUTH granting the privilege(s) on the table or view from the issuer (grantor) to the user ID(s) (grantee(s)) specified.
- Granting UPDATE privilege on specific columns
 1. Check SYSTEM.SYSCOLUMNS to see whether the column(s) exist for the table specified; if it doesn't exist, the GRANT UPDATE statement fails.

2. Check SYSTEM.SYSTABAUTH for the issuer having UPDATE authority with GRANT OPTION on the table or view. If the issuer has UPDATE authority with GRANT OPTION on only a subset of the table or view, check SYSTEM.SYSCOLAUTH to see whether the issuer has UPDATE authority for each column specified in the GRANT UPDATE statement. If so, insert a row into SYSTEM.SYSTABAUTH for the table or view and then insert a row into SYSTEM.SYSCOLAUTH for each column specified. The grantor is the issuer. The grantee(s) is/are the user ID(s).
3. Otherwise, check SYSTEM.SYSUSERAUTH to determine if the issuer has DBA authority and check SYSTEM.SYSTABAUTH to determine if the owner has the GRANT OPTION on the table or view. If the owner has UPDATE authority with GRANT OPTION on only a subset of the table or view, check SYSTEM.SYSCOLAUTH to see whether the owner has UPDATE authority for each column specified in the GRANT UPDATE statement. If so, insert a row into SYSTEM.SYSTABAUTH for the table or view and then insert a row into SYSTEM.SYSCOLAUTH for each column specified. The grantor is the issuer. The grantee(s) is/are the user ID(s).

- Granting ALL privileges

1. Check SYSTEM.SYSTABAUTH to see whether the issuer has GRANT OPTION on any privilege(s) for the table or view, including UPDATE on specific columns. If the issuer has at least one privilege with GRANT OPTION, insert a row into SYSTEM.SYSTABAUTH granting the privilege(s) on the table or view from the issuer (grantor) to the user ID(s) (grantee(s)) specified.

If the issuer has grant authority for UPDATE on specific columns but not for the overall UPDATE privilege on the table or view, check SYSTEM.SYSCOLAUTH to see which columns the issuer has GRANT OPTION on for the table or view. Insert a row into SYSTEM.SYSCOLAUTH for each column found. The grantor is the issuer. The grantee(s) is(are) the user ID(s).

2. Otherwise, check SYSTEM.SYSUSERAUTH to see if the issuer has DBA authority and check SYSTEM.SYSTABAUTH to see whether the owner has GRANT OPTION on any privilege(s) for the table or view, including UPDATE on specific columns. If the owner has at least one privilege with GRANT OPTION, insert a row into SYSTEM.SYSTABAUTH granting the privilege(s) on the table or view from the issuer (grantor) to the user ID(s) (grantee(s)) specified.

If the owner has grant authority for UPDATE on specific columns but not for the overall UPDATE privilege on the table or view, check SYSTEM.SYSCOLAUTH to see which columns the owner has GRANT OPTION on for the table or view. Insert a row into SYSTEM.SYSCOLAUTH for each column found. The grantor is the issuer. The grantee(s) is(are) the user ID(s).

GRANT RUN PRIVILEGE:

1. Check in SYSTEM.SYSACCESS for the program name specified. If not found, the GRANT RUN statement fails.
2. Check in SYSTEM.SYSPROGAUTH for issuer having GRANT OPTION on the program specified in the GRANT RUN statement. If he doesn't, check in SYSTEM.SYSUSERAUTH to see if the issuer has DBA authority and check in SYSTEM.SYSPROGAUTH to determine if the owner has the GRANT OPTION

on the program specified. If this is not the case, the GRANT RUN statement fails.

3. Otherwise, insert a row into SYSTEM.SYSPROGAUTH granting RUN authority from the issuer (grantor) to the user ID(s) (grantee(s)) specified.

REVOKE AUTHORITIES:

1. Check SYSTEM.SYSUSERAUTH for DBA authority.
2. Check in SYSTEM.SYSUSERAUTH for the user ID(s) specified in the REVOKE statement. If a row does not exist for each user ID, the statement fails.

Revoking CONNECT authority: Delete the row(s) found.

Revoking DBA authority: Check each user ID for DBA authority. If the user does not have DBA, the statement fails. Update each row found. RESOURCE and SCHEDULE authority, as well as DBA authority are removed.

Revoking RESOURCE or SCHEDULE authority: Check for non-DBA authority and for the authority to be revoked. If they don't exist, the statement fails. Update each row found, removing the authority specified.

REVOKE TABLE PRIVILEGES:

1. Check SYSTEM.SYSCATALOG for the table or VIEW name specified. If it doesn't exist, the REVOKE statement fails.
2. Check SYSTEM.SYSTABAUTH to see if the issuer (grantor) granted the privilege(s) to the user ID(s) (grantee(s)) specified; if not, the REVOKE table privilege statement fails.
3. Update the row(s) found, revoking the table privilege(s) specified. If there are no other privilege(s) remaining for the grantee, delete the applicable row from SYSTEM.SYSTABAUTH.

In the case of revoking UPDATE authority, all the associated rows in SYSTEM.SYSCOLAUTH are also deleted.

After each user ID has had the table privilege(s) revoked, the REVOKE TABLE statement also revokes the table privilege(s) from any users who cannot now maintain grantor/grantee chains from themselves to the creators of the base tables. As well, any package that exercises the table privileges will be marked invalid, if the creator of that package has had those privileges revoked.

REVOKE RUN PRIVILEGE:

1. Check SYSTEM.SYSACCESS for the program named specified; if it doesn't exist, the REVOKE RUN statement fails.
2. Check SYSTEM.SYSPROGAUTH to see whether the issuer (grantor) granted RUN authority to the user ID(s) (grantee(s)) specified; if not, the REVOKE RUN statement fails.
3. Delete the row(s) found from SYSTEM.SYSPROGAUTH.

If any revoked user ID has GRANT OPTION, the REVOKE RUN statement also revokes RUN authority from subordinate users who cannot maintain RUN authority through another path.

Interpretive Commands

This section lists SQL interpretive statements and how they reference and update the SQL/DS system catalog. The interpretive statements are all data definition SQL statements.

ACQUIRE DBSPACE:

1. PRIVATE DBSPACE
 - If the owner is specified in the statement, ensure that the owner is the connected user. Check SYSTEM.SYSUSERAUTH for RESOURCE authority.
 - If the owner is not the connected user, check SYSTEM.SYSUSERAUTH for DBA authority.
2. PUBLIC DBSPACE:
 - Check in SYSTEM.SYSUSERAUTH to see whether the connected user has DBA authority. The owner is PUBLIC.
3. Check SYSTEM.SYSDBSPACES to see whether the DBSPACE to be acquired already exists. If so, the ACQUIRE DBSPACE fails.
4. Scan SYSTEM.SYSDBSPACES for an available DBSPACE meeting the characteristic(s) specified in the ACQUIRE DBSPACE statement.
5. Update the available DBSPACE row found in SYSTEM.SYSDBSPACES with the attributes specified in the ACQUIRE DBSPACE statement.

ALTER DBSPACE:

1. If the owner of the DBSPACE is not the connected user, check SYSTEM.SYSUSERAUTH for DBA authority.
2. Scan SYSTEM.SYSDBSPACES for the DBSPACE specified to be updated. If the DBSPACE specified does not exist, the ALTER DBSPACE statement fails.
3. Update the row found in SYSTEM.SYSDBSPACES with the new attributes specified in the ALTER DBSPACE statement.
4. If the LOCK level is specified to be changed in the ALTER DBSPACE command, scan SYSTEM.SYSCATALOG to get the table name and the corresponding creator for all tables in the DBSPACE. Update the LOCKMODE in SYSTEM.SYSINDEXES for all entries with matching table name and creator.

ALTER TABLE:

1. Check SYSTEM.SYSCATALOG to see whether the table exists.
2. Check SYSTEM.SYSTABAUTH to see whether the currently connected user has ALTER privilege.

Further processing of the alter table statement depends on the clause that follows. There are nine such clauses:

- Add a column
 1. Update SYSTEM.SYSCATALOG to increment NCOLS by one.
 2. Insert a row into SYSTEM.SYSCOLUMNS for the new column. The insert fails if the column to be added already exists.

3. If a field procedure is specified then insert a row into SYSTEM.SYSFIELDS.
 4. If a field procedure is specified and uses a parameter list then insert row(s) into SYSTEM.SYSFPARMS.
- Add a primary key
 1. Check SYSTEM.SYSKEYS to see whether the key already exists.
 2. Create a unique index to enforce the primary key. See CREATE INDEX.
 3. Validate the primary key.
 4. Insert a row into SYSTEM.SYSKEYS to describe the primary key.
 5. Insert rows into SYSTEM.SYSKEYCOLS to describe each column that forms the primary key.
 6. Check SYSTEM.SYSUSAGE and update SYSTEM.SYSACCESS to invalidate packages with dependencies on this table.
 - Add a foreign key
 1. Check SYSTEM.SYSCATALOG to see whether the referenced table exists.
 2. Check SYSTEM.SYSCATALOG to see whether the referenced table is a real table.
 3. Check SYSTEM.SYSDBSPACES to determine the storage pool number of the parent table.
 4. Check SYSTEM.SYSTABAUTH to see whether the currently connected user has the REFERENCES privilege on the referenced table.
 5. Check SYSTEM.SYSKEYS to see whether the key already exists.
 6. Check SYSTEM.SYSKEYS to see whether the referenced table has an active primary key.
 7. Validate the foreign key.
 8. Insert a row into SYSTEM.SYSKEYS to describe the foreign key.
 9. Insert rows into SYSTEM.SYSKEYCOLS to describe each column that forms the foreign key.
 10. Check SYSTEM.SYSUSAGE and update SYSTEM.SYSACCESS to invalidate packages with a dependency on object table or the referenced table.
 11. Update SYSTEM.SYSCATALOG to increment the number of parents and dependents for the object table and the referenced table.
 - Add a unique constraint
 1. Check SYSTEM.SYSKEYS to see whether the key already exists.
 2. Create a unique index to enforce the unique constraint. See CREATE INDEX.
 3. Validate the unique constraint.
 4. Insert a row into SYSTEM.SYSKEYS to describe the unique constraint.
 5. Insert rows into SYSTEM.SYSKEYCOLS to describe each column that forms the unique constraint.

6. Check `SYSTEM.SYSUSAGE` and update `SYSTEM.SYSACCESS` to invalidate packages with dependencies on this table.
- Drop a primary key
 1. Check `SYSTEM.SYSKEYS` to see whether a primary key exists.
 2. Drop all the dependent foreign keys. See "Drop a foreign key" below.
 3. Drop the underlying unique index. See "DROP INDEX" on page 292.
 4. Check `SYSTEM.SYSUSAGE` and update `SYSTEM.SYSACCESS` to invalidate packages with a dependency on this table.
 5. Delete all the rows in `SYSTEM.SYSKEYCOLS` for this key.
 6. Delete the row in `SYSTEM.SYSKEYS` for this key.
 7. Update `SYSTEM.SYSCATALOG` to reset the number of dependents and inactive keys for this table.
 - Drop a foreign key
 1. Check `SYSTEM.SYSKEYS` to see whether the foreign key exists.
 2. Check `SYSTEM.SYSTABAUTH` to see whether the currently connected user has the `REFERENCES` privilege on the referenced table.
 3. Delete all the rows in `SYSTEM.SYSKEYCOLS` for this key.
 4. Delete the row in `SYSTEM.SYSKEYS` for this key.
 5. Check `SYSTEM.SYSUSAGE` and update `SYSTEM.SYSACCESS` to invalidate packages with dependencies on this table.
 6. Update `SYSTEM.SYSCATALOG` to reset the number of dependents and inactive keys for the object table and the referenced table.
 - Drop a unique constraint
 1. Check `SYSTEM.SYSKEYS` to see whether the unique constraint exists.
 2. Delete all the rows in `SYSTEM.SYSKEYCOLS` for this unique constraint.
 3. Delete the row in `SYSTEM.SYSKEYS` for this unique constraint.
 4. Drop the underlying unique index. See "DROP INDEX" on page 292.
 5. Update `SYSTEM.SYSCATALOG` to reset the number of inactive keys for this table.
 6. Check `SYSTEM.SYSUSAGE` and update `SYSTEM.SYSACCESS` to invalidate packages with dependencies on this table.
 - Activate a key or unique constraint
 1. Check `SYSTEM.SYSKEYS` to see whether the key or unique constraint exists.
 2. Check `SYSTEM.SYSTABAUTH` to see whether the currently connected user has the necessary `ALTER` and `REFERENCES` privileges for the tables involved.
 3. Revalidate the key or unique constraint.
 4. If you are dealing with a primary key, activate all dependently inactive foreign keys.
 5. Check `SYSTEM.SYSUSAGE` and update `SYSTEM.SYSACCESS` to invalidate packages with a dependency on the tables involved.

6. Update SYSTEM.SYSCATALOG to update the number of inactive keys for the tables involved.
- Deactivate a key or unique constraint
 1. Check SYSTEM.SYSKEYS to see whether the key or unique constraint exists.
 2. Check SYSTEM.SYSTABAUTH to see whether the currently connected user has the necessary ALTER and REFERENCES privileges for the tables involved.
 3. If you are dealing with a primary key, deactivate all active dependent foreign keys.
 4. Check SYSTEM.SYSUSAGE and update SYSTEM.SYSACCESS to invalidate packages with a dependency on the tables involved.
 5. Update SYSTEM.SYSCATALOG to update the number of inactive keys for the tables involved.

COMMENT ON:

1. If the table creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. If commenting on a table, update the REMARKS column in SYSTEM.SYSCATALOG for the table being commented on. Otherwise, it is comment on column(s) in which case the REMARKS column in SYSTEM.SYSCOLUMNS is updated for column(s) being commented on.

CREATE INDEX:

1. If the name specified for the table is in SYSTEM.SYSSYNONYMS, substitute for the synonym the CREATOR and TNAME values (creator.tname) for the matching row in SYSTEM.SYSSYNONYMS.
2. Check SYSTEM.SYSTABAUTH for INDEX authority on the table; if not, check SYSTEM.SYSUSERAUTH for DBA authority.
3. Check SYSTEM.SYSINDEXES to see whether the index name specified already exists. If so, CREATE INDEX fails.
4. Check SYSTEM.SYSCATALOG for table identifier and DBSPACE number.
5. Check SYSTEM.SYSDBSPACES for LOCKMODE.
6. Create and sort the new index keys. Statistics are generated automatically.
7. Insert into SYSTEM.SYSINDEXES a row for the new index created.
8. Update ROWCOUNT and NPAGES for the table in SYSTEM.SYSCATALOG.
9. Update NACTIVE for the DBSPACE in SYSTEM.SYSDBSPACES.
10. Update the index row in SYSTEM.SYSINDEXES with the new index statistics on CLUSTER, KEYLEN, FIRSTKEYCOUNT, FULLKEYCOUNT, NLEAF, NLEVELS and CLUSTERRATIO columns.
11. Update the first indexed column row in SYSTEM.SYSCOLUMNS with the new column statistics on COLCOUNT, HIGH2KEY, LOW2KEY, AVGCOLLEN, ORDERFIELD and COLINFO columns.
12. Insert a new row into SYSCOLSTATS if the first indexed column does not match the first indexed columns of other indexes in the table. Otherwise, the

row corresponding to this column is updated in SYSTEM.SYSCOLSTATS with the column statistics on VAL10, VAL50, VAL90, FREQ1VAL, FREQ1PCT, FREQ2VAL, and FREQ2PCT columns.

13. If this is the first index created on this table, the row for the table in SYSTEM.SYSCATALOG is updated by setting the column CLUSTERTYPE to I.

CREATE SYNONYM:

1. Check in SYSTEM.SYSSYNONYMS to see whether the creator and name exist. If so, the CREATE SYNONYM statement fails because a new synonym cannot be created on an existing synonym.
2. Check in SYSTEM.SYSCATALOG to see whether the synonym specified already exists as a table or view; if so, the CREATE SYNONYM fails.
3. Insert into SYSTEM.SYSSYNONYMS a row for the new synonym created; the insert fails if the synonym already exists.

CREATE TABLE:

1. If the table creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. Check SYSTEM.SYSCATALOG to see whether the table name specified in the CREATE TABLE statement already exists as a table or view. If so, the CREATE TABLE statement fails.
3. Check SYSTEM.SYSSYNONYMS to see whether a synonym already exists with the same name specified in the CREATE TABLE statement. If so, the CREATE TABLE statement fails.
4. Check SYSTEM.SYSDBSPACES for the DBSPACE specified. If no DBSPACE was specified, find a private DBSPACE belonging to the creator.
5. If the table is in a PRIVATE DBSPACE and the owner of the DBSPACE is different from the user of the statement, check in SYSTEM.SYSUSERAUTH for DBA authority.

If the table is in a PUBLIC DBSPACE and the DBSPACE name starts with SYS, check in SYSTEM.SYSUSERAUTH for DBA authority.

If the table is in a PUBLIC DBSPACE, check SYSTEM.SYSUSERAUTH for RESOURCE authority.
6. Update the row in SYSTEM.SYSDBSPACES corresponding to the DBSPACE in which the table is being created. Increment the corresponding field in the NTABS column.
7. Insert into SYSTEM.SYSCATALOG a row for the new table being created.
8. Insert into SYSTEM.SYSCOLUMNS a row for each column in the new table being created.
9. If a field procedure is specified then insert a row into SYSTEM.SYSFIELDS.
10. If a field procedure is specified and uses a parameter list then insert row(s) into SYSTEM.SYSFPARMS.
11. Insert into SYSTEM.SYSTABAUTH a new row for the table created. This row gives the table creator all the table authorities on that table with grant options.
12. If requested, create a primary key. See "ALTER TABLE" on page 286.

13. If requested, create foreign keys for this table. See "ALTER TABLE" on page 286.
14. If requested, create a unique constraint. See "ALTER TABLE" on page 286.

CREATE VIEW:

1. If the creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. Check in SYSTEM.SYSCATALOG to see whether the view specified exists as a table or view; if so, the CREATE VIEW fails.
3. Check in SYSTEM.SYSSYNONYMS to see whether the view specified exists as a synonym; if so, the CREATE VIEW fails.
4. Check in SYSTEM.SYSSYNONYMS to see whether any of the names specified in the view SELECT statement exists for the user; if so, substitute the synonym with the CREATOR and TNAME values (creator.tname).
5. Check in SYSTEM.SYSTABAUTH for SELECT authority on the table(s) or view(s) on which the view is being based; if it doesn't exist, check SYSTEM.SYSUSERAUTH for DBA authority.

- If the view is based on only one table or view (not a join), then DELETE, INSERT, and UPDATE authority is checked in SYSTEM.SYSTABAUTH. A row is inserted into SYSTEM.SYSTABAUTH for all the authorities that the creator has on the underlying table or view. If no authority is found, including SELECT authority, no row is inserted into SYSTEM.SYSTABAUTH.

If the creator has only UPDATE authority on selected columns on the underlying table or view, a row (or rows) is inserted in SYSTEM.SYSCOLAUTH for the associated new view column that corresponds to the row(s) found in SYSTEM.SYSCOLAUTH for the underlying table or view column.

Authorities on a table or view are also available if they are granted to PUBLIC. In this case, no row is made in SYSTEM.SYSTABAUTH for these authorities obtained by way of PUBLIC.

- If the view is a join of two or more tables, rows are inserted in SYSTEM.SYSTABAUTH. There is one row for each table or view in the join. Each row records only the SELECT authority (with or without grant option) that the creator has on that table or view.

Authorities on a table or view are also available if they are granted to PUBLIC. In this case, no row is made in SYSTEM.SYSTABAUTH for these authorities obtained through PUBLIC.

6. Insert into SYSTEM.SYSVIEWS a row for the view being created.
7. Insert into SYSTEM.SYSCATALOG a row for the view being created.
8. Insert into SYSTEM.SYSCOLUMNS a row for each column in the view being created.
9. Insert into SYSTEM.SYSUSAGE a row for each table or view in the view being created to record the dependency(s) of the view on the underlying table(s) or view(s).

10. Scan SYSTEM.SYSACCESS for a nonallocated row and update the row to record its binding to the newly created view. Store the newly created packages in the associate table for the row found.

DROP DBSPACE:

1. If the creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. Check SYSTEM.SYSDBSPACES to see whether the DBSPACE specified exists; if not, the DROP DBSPACE statement fails.
3. For each table in the DBSPACE being dropped, the following is done:
 - Drop the primary key, all foreign keys, and unique constraints for this table. See "ALTER TABLE" on page 286.
 - Process all dependencies on the table being dropped:
 - Delete all rows in SYSTEM.SYSUSAGES that refer to view(s) or program(s) dependent on this table.
 - If a view is dependent on the table being dropped, do the processing to drop the dependent view.
 - Update SYSTEM.SYSACCESS if a program is dependent on the table being dropped. Set the corresponding field in the VALID column to N.
 - Delete all rows for the table in SYSTEM.SYSCATALOG, SYSTEM.SYSTABAUTH, SYSTEM.SYSCOLAUTH, SYSTEM.SYSCOLUMNS, SYSTEM.SYSINDEXES, SYSTEM.SYSCOLSTATS, SYSTEM.SYSFIELDS and SYSTEM.SYSFPARMS.
4. Update the row in SYSTEM.SYSDBSPACES to indicate that the DBSPACE is unavailable until the end of the current logical unit of work. At COMMIT WORK time, the row in SYSTEM.SYSDBSPACES is updated to the status of available.
5. Insert a row in SYSTEM.SYSDROP for the DBSPACE being dropped. This row is processed at explicit COMMIT WORK time and storage is released.

DROP INDEX:

1. If the creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. Check SYSTEM.SYSINDEXES to see whether the index being dropped exists. If the index is a primary key index, it cannot be explicitly dropped.
3. Delete any rows in SYSTEM.SYSUSAGE that have a packages that depends on the index being dropped. Also update the row for the package in SYSTEM.SYSACCESS, setting the column VALID to N. This indicates that on the program's next invocation, it is preprocessed.
4. Delete the row in SYSTEM.SYSINDEXES corresponding to the index being dropped.
5. If the first indexed column hasn't been used as a first indexed column of any other index in this table, delete the row corresponding to this column from SYSTEM.SYSCOLSTATS. Also, update the first indexed row in SYSTEM.SYSCOLUMNS by setting the column COLINFO to N.

6. If this was a first index (CLUSTER = F or W), update the SYSTEM.SYSCATALOG row by setting the column CLUSTERTYPE to D.

DROP PACKAGE:

1. If the creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. Check in SYSTEM.SYSACCESS to see whether the program to be dropped exists; if not, the DROP PACKAGE statement fails; if so, delete the row found.
3. Delete all the rows in the package.
4. Update the row in SYSTEM.SYSACCESS for the program being dropped to change its status to available.
5. Delete all row(s) in SYSTEM.SYSPROGAUTH corresponding to the program being dropped.
6. Delete all row(s) in SYSTEM.SYSUSAGE referencing this program.

DROP SYNONYM:

1. If the creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. Delete from SYSTEM.SYSSYNONYMS the row for the synonym being dropped; the drop fails if the synonym specified doesn't exist.

DROP TABLE:

1. Check SYSTEM.SYSSYNONYMS for the name specified in the DROP TABLE statement being a synonym. If so, substitute the synonym with the CREATOR and TNAME values (creator.tname).
2. If the creator is specified and is not the connected user, check SYSTEM.SYSUSERAUTH for DBA authority.
3. Check SYSTEM.SYSCATALOG to ensure that the name specified is not the name of a view.
4. Drop the primary key, all foreign keys and unique constraints for this table. See "ALTER TABLE" on page 286.
5. Process all dependencies on the table being dropped:
 - Delete all rows in SYSTEM.SYSUSAGES that refer to views or programs dependent on this table.
 - If a view is dependent on the table being dropped, do the processing to drop the dependent view.
 - Update SYSTEM.SYSACCESS if a program is dependent on the table being dropped. Set the corresponding field in the VALID column to N.
6. Update the row in SYSTEM.SYSDBSPACES for the DBSPACE containing the table being dropped. Decrement the corresponding field in the NTABS column.
7. Delete all rows for the table in SYSTEM.SYSCATALOG, SYSTEM.SYSTABAUTH, SYSTEM.SYSCOLAUTH, SYSTEM.SYSCOLUMNS, SYSTEM.SYSINDEXES, SYSTEM.SYSCOLSTATS, SYSTEM.SYSFIELDS and SYSTEM.SYSFPARMS.

8. Insert a row in SYSTEM.SYSDROP for the table being dropped. This row is processed at explicit COMMIT WORK time and processed to release the storage for the table dropped.

DROP VIEW:

1. Check SYSTEM.SYSSYNONYMS for the name specified in the DROP VIEW statement being a synonym. If so, substitute the synonym with the CREATOR and TNAME values (creator.tname).
2. If the creator specified is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
3. Check SYSTEM.SYSCATALOG to see whether the view exists.
4. Delete rows in SYSTEM.SYSVIEWS corresponding to the view being dropped.
5. Delete all the rows in the package.
6. Update the row in SYSTEM.SYSACCESS for the view being dropped to change its status to available.
7. Delete all rows for the view in SYSTEM.SYSCATALOG, SYSTEM.SYSTABAUTH, SYSTEM.SYSCOLAUTH, and SYSTEM.SYSCOLUMNS.
8. Process all dependencies on the view being dropped:
 - Delete all rows in SYSTEM.SYSUSAGE that refer to views or programs dependent on this view.
 - If a view is dependent on the view being dropped, do the processing to drop the dependent view.
 - Update SYSTEM.SYSACCESS if a program is dependent on the view being dropped. Set the corresponding field in the VALID column to N.

LABEL ON:

1. If the table creator is specified and is not the connected user, check in SYSTEM.SYSUSERAUTH for DBA authority.
2. If labeling a table, update the row in SYSTEM.SYSCATALOG for the table being labeled; otherwise, it is label on column(s) in which case the row(s) in SYSTEM.SYSCOLUMNS for column(s) to be labeled are updated.

REORGANIZE INDEX:

1. If the user is not the owner of the index then check SYSTEM.SYSUSERAUTH for DBA authority.
2. Check SYSINDEXES to see if the index exists and is not a primary or unique key.
3. Check SYSTEM.SYSCATALOG for table identifier and DBSPACE number.
4. Reorganize the index. Statistics will be collected automatically.
5. Update the index row in SYSTEM.SYSINDEXES with the new IPCTFREE and RELEASE.
6. Update SYSTEM.SYSCATALOG with the new ROWCOUNT, and NPAGES.
7. Update NACTIVE for the DBSPACE in SYSTEM.SYSDBSPACES.

8. Update the index row in SYSTEM.SYSINDEXES with the new index statistics on CLUSTER, KEYLEN, FIRSTKEYCOUNT, FULLKEYCOUNT, NLEAF, NLEVELS and CLUSTERRATIO columns.
9. Update the first indexed column row in SYSTEM.SYSCOLUMNS with the new column statistics on COLCOUNT, HIGH2KEY, LOW2KEY, AVGCOLLEN, ORDERFIELD and COLINFO columns.
10. Update the first indexed column row in SYSTEM.SYSCOLSTATS with the new column statistics on VAL10, VAL50, VAL90, FREQ1VAL, FREQ1PCT, FREQ2VAL and FREQ2PCT columns. Insert a row with these values if it does not already exist.

UPDATE STATISTICS:

- If updating statistics on a DBSPACE:
 1. Check SYSTEM.SYSDBSPPACES to see whether the DBSPACE exists; if not, the UPDATE STATISTICS will fail.
 2. Find each table in the DBSPACE by searching SYSTEM.SYSCATALOG. For each table in the DBSPACE, the processing is explained in the update statistics for table.
- If updating statistics on a table:
 1. Check in SYSTEM.SYSCATALOG to see whether the table exists; if not, the UPDATE STATISTICS fails.
 2. Lock DBSPACE in share mode.
 3. Scan through each row of the table gathering statistics.
 4. Gather statistics on the first index on this table if it exists.
 5. Update the table row in SYSTEM.SYSCATALOG with the new table statistics on AVGWLEN, ROWCOUNT, NPAGES, and PCTPAGES columns.
 6. Update the DBSPACE row in SYSTEM.SYSDBSPPACES with the new DBSPACE statistics on the NACTIVE column.
 7. Update the index row(s) in SYSTEM.SYSINDEXES with the new index statistics on CLUSTER, KEYLEN, FIRSTKEYCOUNT, FULLKEYCOUNT, NLEAF, NLEVELS, and CLUSTERRATIO columns.
 8. Update the first indexed column row(s) in SYSTEM.SYSCOLUMNS with the new column statistics on COLCOUNT, HIGH2KEY, LOW2KEY, AVGCOLLEN, ORDERFIELD, and COLINFO columns.
 9. Update the first indexed column row in SYSTEM.SYSCOLSTATS with the new column statistics on VAL10, VAL50, VAL90, FREQ1VAL, FREQ1PCT, FREQ2VAL and FREQ2PCT columns.
 10. For every additional index on this table, gather index statistics and perform steps 7 and 8.

Column statistics are gathered only for the first column in an index on a table. If the ALL option is specified, statistics are approximated for the rest of the columns and their associated row(s) in SYSTEM.SYSCOLUMNS are updated.



Appendix C. SQL/DS Distributed Data Management (DDM) Command Support

This appendix describes the DDM commands and command parameters, command data objects, and reply data objects, that the SQL/DS product supports for DRDA level 1.

An application requester (AR) using the DRDA protocol to connect to an application server (AS) uses a subset of *Distributed Data Management (DDM)* as part of the underlying architecture of DRDA. DDM is a data management architecture used for data interchange among like or unlike systems and is independent of a particular system's hardware architecture and its operating system. DDM commands, parameters, objects, and messages actually define the structure and rules for the data interchange between the systems. DDM objects carry actual data items and are further described by Formatted Data: Object Content Architecture (FD:OCA) descriptors and data.

DDM commands can request the remote system to execute an SQL statement and return the result to the requester. For example, the DDM commands OPNQRY, CNTQRY, and CLSQRY perform functions similar to the SQL cursor statements OPEN, FETCH, and CLOSE.

For more information about how DRDA uses DDM, see *Distributed Data Library, Architecture Reference, SC26-4651*.

How to Read the Tables

The tables in this section show how the SQL/DS product supports each of the DDM codepoints and parameters defined by DRDA level 1. There are two types of tables:

- Commands and Command Data
- Reply Messages and Reply Data

Each DDM command can have the following object types associated with it:

- Parameters (instance variables)
- Command data objects
- Reply messages
- Reply data objects

These commands and associated object types are distinguished as follows throughout the tables:

- Names of commands and reply messages are in upper case and are in the top row of the table.
- Names of command objects are in upper case and, if present, are in the bottom rows of the table.
- Names of reply data objects are in mixed case (first letter capitalized).
- Parameter names are in lower case.

Command Tables

These tables are full page width. The AR column indicates how the SQL/DS application requester will handle the codepoint or parameter:

- Y** Will flow it to the application server.
- N** Will not flow it to the application server.

The AS column indicates how the SQL/DS application server will support the codepoint or parameter:

- Y** Will recognize and process it.
- S** Will allow the parameter, depending on its value. The supported values are defined after the table.
- I** Will ignore it if received.
- N** Will reject it.

Reply Tables

These tables are indented in this document to help distinguish them from the Command tables. Only a small subset of the reply messages are documented here. These are generally the messages that indicate that the command completed successfully. You might receive a reply message that is not documented here in an error situation.

The AS column in the Reply Tables indicates how the SQL/DS application server will handle the codepoint or parameter:

- Y** Will flow it to the application requester.
- N** Will not flow it to the application requester.

The AR column indicates how the SQL/DS application requester will support the codepoint or parameter:

- Y** Will recognize and process it.
- I** Will ignore it.

Figure 154 (Page 1 of 2). Definition of ACCRDB Command

DDM Codepoint	AR	AS	AR Values
ACCRDB Invoked as part of the handshaking process; see EXCSAT	Y	Y	
rdbnam (name of remote database). AR obtains value from SQLINIT's DBNAME option	Y	Y	
rdbaccl (access manager class).	Y	Y	
typdefnam (data type definition name).	Y	Y	'QTDSQL370'
typdefovr (data type definition override). Value is deduced from the SQLINIT CHARNAME option	Y	Y	
prdid (product specific id).	Y	Y	'ARI03040'
rdbalwupd (rdb to allow updates) < optional > .	N	Y	
prddta (product specific data) < optional > .	N	I	
sttdecdel (statement decimal delimiter) < optional > .	N	S	

Figure 154 (Page 2 of 2). Definition of ACCRDB Command

DDM Codepoint	AR	AS	AR Values
sttstrdel (statement string delimiter) < optional > .	N	S	
crrtkn (correlation token) < optional > .	N	Y	
trgdftrt (target default values return) < optional > .	Y	Y	TRUE

ACCRDB Parameters Clarification:

- PRTDTA
 - AS: will be ignored.
- STTDECDEL
 - AS: only decimal or package default will be supported
- STTSTRDEL
 - AS: only apostrophe or package default will be supported

Figure 155. Definition of ACCRDBRM reply message

DDM Codepoint	AR	AS	AS Values
ACCRDBRM Invoked as part of the handshaking process; see EXCSAT	Y	Y	
svrcod (severity code).	Y	Y	
prdid (product specific id).	Y	Y	'ARI03040'
typdefnam (data type definition name).	Y	Y	'QTDSQL370'
typdefovr (data type definition override). AS returns SYSOPTIONS' CCSIDS/M/G entries	Y	Y	
srvdgn (server diagnostic information) < optional > .	I	N	
rdbinttkn (RDB interrupt token) < optional > .	I	N	
crrtkn (correlation token) < optional > . AS will not return CRRTKN if the AR has sent one (eg. DB2*/CICS AR); otherwise AS will return the AR's LU6.2 LUWID. In the latter case, if the AR has no LU6.2 LUWID associated with its conversation (e.g. an SQL/DS AR running in VM/SP6), then the AS will generate one and return it in CRRTKN.	Y	Y	
pkgdfcst (package default character subtype) < optional > . AS does not use this value	Y	Y	
userid (user id at the target system) < optional > .	Y	Y	

Figure 156 (Page 1 of 3). Definition of BGNBND Command

DDM Codepoint	AR	AS	SQL/DS Prep Options
BGNBND The bind flow is initiated by the preprocessors, or by the DBSU RELOAD PACKAGE command, or by the SQL CREATE PACKAGE statement.	Y	Y	
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	

Figure 156 (Page 2 of 3). Definition of BGNBND Command			
DDM Codepoint	AR	AS	SQL/DS Prep Options
pkgnamct (package name and consistency token). rdbnam (AS database name). rdbcolid (rdb collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token).	Y	Y	
vrsnam (package version name) < optional > .	N	S	
bndchkexs (bind existence checking) < optional > .	Y	Y	NOEXIST / EXIST
pkgrplopt (package replacement option) < optional > .	Y	Y	REPLACE / NEW
pkgathopt (package authorization option) < optional > .	Y	Y	KEEP / REVOKE
sttstrdel (statement string delimiter) < optional > .	Y	S	SQLAPOST / SQLQUOTE
sttdecdel (statement decimal delimiter) < optional > .	Y	S	PERIOD / COMMA
sttdatfmt (date format of statement) < optional > .	Y	Y	DATE()
stttimfmt (time format of statement) < optional > .	Y	Y	TIME()
pkgisolvl (package isolation level).	Y	S	ISOL()
bndcrtctl (bind creation control) < optional > .	Y	Y	NOCHECK / CHECK / ERROR
bndexpopt (bind explain option) < optional > .	Y	Y	EXPLAIN()
pkgownid (package owner identifier) < optional > .	Y	S	OWNER()
rdbrlsopt (RDB release option) < optional > .	Y	I	RELEASE()
dftrdbcol (default RDB collection identifier) < optional > .	Y	S	QUALIFIER()
title (brief description of package) < optional > .	Y	S	LABEL()
qryblkctl (query block protocol control) < optional > .	Y	Y	SBLOCK / BLOCK / NOBLOCK
pkgdfcst (default character subtype) < optional > .	Y	Y	CHARSUB()
pkgdfcc (package default CCSID) < optional > .	Y	Y	CCSIDS, CCSIDM, CCSIDG
pkgrplvrs (replaced package version name) < optional > .	N	N	Parm not supported
decprc (decimal precision) < optional > .	N	N	

Figure 156 (Page 3 of 3). Definition of BGNBND Command

DDM Codepoint	AR	AS	SQL/DS Prep Options
BGNBND Parameters Clarification: <ul style="list-style-type: none"> • BNDEXPOPT <ul style="list-style-type: none"> – AS: do not explain SQL statements parameter will be supported • DFTRDBCOL <ul style="list-style-type: none"> – AS: parameter will be supported if it matches the <i>rdbcolid</i> specified in <i>pkgnamcsn</i> In SQL/DS product V3.4, it must be same as the AS authid. • PKGISOLVL <ul style="list-style-type: none"> – AR: <i>USER</i> defined option is not supported by the DRDA protocol. It will be mapped to <i>Cursor stability</i> instead. – AS: Isolation level will be mapped to <i>Cursor stability</i> or <i>Repeatable read</i> • PKGOWNID <ul style="list-style-type: none"> – AS: parameter will be supported if it matches the <i>rdbcolid</i> specified in <i>pkgnamcsn</i> In SQL/DS product V3.4, it must be same as the AS authid. • RDBCOLID <ul style="list-style-type: none"> – AS: in SQL/DS product V3.4, must be same as the AS authid. • RDBRISOPT <ul style="list-style-type: none"> – AS: will check for valid DRDA options • STTDECDEL <ul style="list-style-type: none"> – AS: only decimal will be supported • STTSTRDEL <ul style="list-style-type: none"> – AS: only apostrophe will be supported • TITLE <ul style="list-style-type: none"> – AS: length greater than thirty characters will be truncated. • VRSNAM <ul style="list-style-type: none"> – AS: null version name will be supported 			

Figure 157. Reply Objects for BGNBND command

DDM Codepoint	AR	AS	SQL State-ment
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	

Figure 158 (Page 1 of 2). Definition of BNDSQLSTT Command

DDM Codepoint	AR	AS	SQL Statement
BNDSQLSTT	Y	Y	Part of bind flow; see BGNBND and ENDBND
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgsn (package section number).	Y	Y	
sqlstnbr (source application statement number) < optional > .	N	I	

Figure 158 (Page 2 of 2). Definition of BNDSQLSTT Command

DDM Codepoint	AR	AS	SQL Statement
bndsttasm (bind statement assumptions) < optional >. AR makes assumption for all but the SELECT statements	Y	Y	
TYPDEFNAM (data type definition name) < optional >	N	Y	
TYPDEFOVR (TYPDEF override) < optional >	N	Y	
SQLSTT (SQL statement to be bound in the AS package). AR replaces the host variables within the SQL statement with ":H" as defined in DRDA. When the AS detects a syntax error, it will return ":H" instead of the original host variable name.	Y	Y	
SQLSTTVRB (description of each variable) < optional >. All host variable names and attributes are passed in this parm. The SQL/DS AR does not send the original names to the AS. The names sent are in the form of ":Hn" where "n" is the position of the host variable within the SQL statement. The SQL/DS application server will accept this parameter, but does not use the information that it contains.	Y	I	
BNDSQLSTT Parameters Clarification: <ul style="list-style-type: none"> • SQLSTTNBR <ul style="list-style-type: none"> – AS: parameter value will not be saved • SQLDIAGNAME <ul style="list-style-type: none"> – AR: length initialized to zero – AS: parameter value will be ignored 			

Figure 159. Reply Objects for BNDSQLSTT command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	

Figure 160. Definition of CLSQRY Command

DDM Codepoint	AR	AS	SQL Statement
CLSQRY	Y	Y	CLOSE
rdbnam (name of remote database as in ACCRDB) < optional >.	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgsn (package section number).	Y	Y	

Figure 161 (Page 1 of 2). Reply Objects for CLSQRY command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	

Figure 161 (Page 2 of 2). Reply Objects for CLSQRY command

DDM Codepoint	AR	AS	SQL Statement
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	

Figure 162. Definition of CNTQRY Command

DDM Codepoint	AR	AS	SQL Statement
CNTQRY	Y	Y	FETCH
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgsn (package section number).	Y	Y	
qryblksz (query block size).	Y	Y	

Figure 163. Reply Objects for CNTQRY command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	
Qrydta (query answer set data)	Y	Y	

Figure 164. Definition of DRPPKG Command

DDM Codepoint	AR	AS	SQL Statement
DRPPKG	Y	Y	DROP PACKAGE
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnam (package grouping name and identifier). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier).	Y	Y	
vrsnam (version name) < optional > .	N	S	
DRPPKG Parameters Clarification:			
<ul style="list-style-type: none"> • VRSNAM <ul style="list-style-type: none"> – AS: null version name will be supported 			

Figure 165 (Page 1 of 2). Reply Objects for DRPPKG command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	

Figure 165 (Page 2 of 2). Reply Objects for DRPPKG command

DDM Codepoint	AR	AS	SQL State- ment
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	

Figure 166. Definition of DSCRDBTBL Command

DDM Codepoint	AR	AS	SQL Statement
DSCRDBTBL	N	N	
rdbnam (name of remote database as in ACCRDB) < optional > .			
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgasn (package section number).			
SQLTBLNAM (SQL table name).			

Figure 167. Definition of DSCSQLSTT Command

DDM Codepoint	AR	AS	SQL Statement
DSCSQLSTT	Y	Y	DESCRIBE
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgasn (package section number).	Y	Y	

Figure 168. Reply Objects for DSCSQLSTT command

DDM Codepoint	AR	AS	SQL State- ment
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	
Sqlcard (SQLDA reply data)	Y	Y	

Figure 169 (Page 1 of 2). Definition of ENDBND Command

DDM Codepoint	AR	AS	SQL Statement
ENDBND	Y	Y	Part of bind flow; see BGNBND and BNDSQLSTT
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	

Figure 169 (Page 2 of 2). Definition of ENDBND Command

DDM Codepoint	AR	AS	SQL Statement
pkgnamct (package name and consistency token). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token).	Y	Y	
maxsectnr (maximum section number) < optional >.	Y	Y	

Figure 170. Reply Objects for ENDBND command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	

Figure 171. Definition of EXCSAT Command

DDM Codepoint	AR	AS	AR Values
EXCSAT When an implicit or explicit connect (via SQL CONNECT statement) occurs, a handshaking process is performed. The SQL/DS AR sends the EXCSAT command chained with the ACCRDB command to the AS, and expects the chained replies of EXCSATRD and ACCRDBRM in return.	Y	Y	
extnam (external name).	Y	Y	userid. CMS-work-unit-id (padded with trailing blanks)
mgrlvlls (manager level list).	Y	Y	four required 3's
spvnam (supervisor name) < optional >.	N	I	
srvcisnm (server class name).	Y	Y	'QSQLDS/VM'
srvnam (server name).	Y	Y	AR's RSCS-nodeid
srvrslv (server release level).	Y	Y	'ARI03040'

Figure 172 (Page 1 of 2). Definition of EXCSATRD Reply Object

DDM Codepoint	AR	AS	AS values
EXCSATRD Invoked as part of the handshaking process; see EXCSAT	Y	Y	
extnam (external name).	Y	Y	Requester's id at the AS
mgrlvlls (manager level list).	Y	Y	
srvcisnm (server class name).	Y	Y	'QSQLDS/VM'

Figure 172 (Page 2 of 2). Definition of EXCSATRD Reply Object

DDM Codepoint	AR	AS	AS values
srvnam (server name).	Y	Y	AS' RSCS nodeid
srvrslv (server release level).	Y	Y	'ARI03040'

Figure 173. Definition of EXCSQLIMM Command

DDM Codepoint	AR	AS	SQL Statement
EXCSQLIMM	Y	Y	EXECUTE IMMEDIATE
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgsn (package section number).	Y	Y	
TYPDEFNAM (data type definition name) < optional >	N	Y	
TYPDEFOVR (TYPDEF override) < optional >	N	Y	
SQLSTT (SQL statement - cannot contain host variables).	Y	Y	

Figure 174. Reply Objects for EXCSQLIMM command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	

Figure 175. Definition of EXCSQLSTT Command

DDM Codepoint	AR	AS	SQL Statement
EXCSQLSTT	Y	Y	EXECUTE
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgsn (package section number).	Y	Y	
outexp (output expected-specifies non-cursor SELECT) < optional > .	Y	Y	for non-cursor SELECT only
TYPDEFNAM (data type definition name) < optional >	N	Y	
TYPDEFOVR (TYPDEF override) < optional >	N	Y	
SQLDTA (SQL program variable data) < optional > .	Y	Y	

Figure 176. Reply Objects for EXCSQLSTT command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data)	Y	Y	
Sqlardt (SQL data reply data)	Y	Y	

Figure 177. Definition of INTRDBRQS Command

DDM Codepoint	AR	AS	SQL Statement
INTRDBRQS	N	N	
rdbnam (name of remote database as in ACCRDB) < optional > .			
rdbintkn (relational database interrupt token).			

Figure 178. Definition of OPNQRY Command

DDM Codepoint	AR	AS	SQL Statement
OPNQRY	Y	Y	OPEN
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgsn (package section number).	Y	Y	
qryblksz (query block size).	Y	Y	
qryblkctl (query block control) < optional > .	N	Y	
TYPDEFNAM (data type definition name) < optional >	N	Y	
TYPDEFOVR (TYPDEF override) < optional >	N	Y	
SQLDTA (input variable data) < optional > .	Y	Y	

Figure 179 (Page 1 of 2). Reply Message and Reply Objects for OPNQRY command

DDM Codepoint	AR	AS	SQL Statement
OPNQRYM (open query reply message)	Y	Y	
svrcod (severity code)	Y	Y	
qryprctyp (protocol type)	Y	Y	
sqlcsrhd (cursor hold flag) < optional >	Y	N	
srvdgn (server diagnostic information) < optional >	I	N	
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	

Figure 179 (Page 2 of 2). Reply Message and Reply Objects for OPNQRS command

DDM Codepoint	AR	AS	SQL Statement
Sqlcard (SQLCA reply data) < optional >	Y	Y	
Qrydisc (query answer set description) < optional >	Y	Y	
Qrydta (query answer set data) < optional >	Y	Y	

Figure 180. Definition of PRPSQLSTT Command

DDM Codepoint	AR	AS	SQL Statement
PRPSQLSTT	Y	Y	PREPARE
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	
pkgnamcsn (package name, consistency token and section #). rdbnam (AS database name). rdbcolid (RDB collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token). pkgsn (package section number).	Y	Y	
rtnsqlda (specifies if SQLDA should be returned) < optional > .	N	Y	USING clause
TYPDEFNAM (data type definition name) < optional >	N	Y	
TYPDEFOVR (TYPDEF override) < optional >	N	Y	
SQLSTT (SQL Statement)	Y	Y	
PRPSQLSTT Parameters Clarification:			
<ul style="list-style-type: none"> • RTNSQLDA <ul style="list-style-type: none"> - If requested by the AR, the PRPSQLSTT command is translated by the SQL/DS AS into a PREPARE call and a DESCRIBE ... USING BOTH call. 			

Figure 181. Reply Objects for PRPSQLSTT command

DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data) < optional >	Y	Y	
Sqlcard (SQLDA reply data) < optional >	Y	Y	

Figure 182. Definition of RDBCMM Command

DDM Codepoint	AR	AS	SQL Statement
RDBCMM	Y	Y	COMMIT < RELEASE >
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	

<i>Figure 183. Reply Objects for RDBCMM command</i>			
DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data) < optional >	Y	Y	

<i>Figure 184. Definition of RDBRLLBCK Command</i>			
DDM Codepoint	AR	AS	SQL Statement
RDBRLLBCK	Y	Y	ROLLBACK < RELEASE >
rdbnam (name of remote database as in ACCRDB) < optional > .	Y	Y	

<i>Figure 185. Reply Objects for RDBRLLBCK command</i>			
DDM Codepoint	AR	AS	SQL Statement
Typdefnam (data type definition name) < optional >	Y	N	
Typdefovr (TYPDEF override) < optional >	Y	N	
Sqlcard (SQLCA reply data) < optional >	Y	Y	

<i>Figure 186. Definition of REBIND Command</i>			
DDM Codepoint	AR	AS	SQL Statement
REBIND	N	N	
rdbnam (name of remote database as in ACCRDB) < optional > .			
pkgnamct (package name and consistency token). rdbnam (AS database name). rdbcolid (rdb collection identifier). pkgid (package identifier). pkgcnstkn (package consistency token).			
vrsnam (package version name) < optional > .			
pkgisolvl (package isolation level).			
bndexpopt (bind explain option) < optional > .			
bndownid (bind owner identification) < optional > .			
rdbrlsopt (RDB release option) < optional > .			
bndchkexs (bind existence checking) < optional > .			
dftrdbcol (default RDB collection identifier) < optional > .			

Glossary

access path. The path used to get to data specified in SQL statements. An access path can involve either an index, a sequential search, or a combination of both.

advanced program-to-program communication/virtual machine (APPC/VM). An application programming interface that uses the SNA LU 6.2 protocol to enable two application programs to communicate.

agent. A structure that the SQL/DS database manager allocates to a user to enable the processing of requests by an application server.

APPC/VM. See *advanced program-to-program communication/virtual machine*.

AUX. See *Package*.

checkpoint. A set of cleanup and recovery actions taken periodically by the database manager. Actions include writing a copy of the database to disk, recording information in the SQL/DS log, and freeing storage pool space.

commit. The operation that terminates a *unit of work* by releasing locks so that the database changes made by that unit of work can be perceived by other processes.

cursor stability. An isolation level that (1) prevents a row changed by a concurrently executing application process from being read until that row is committed by the application process and (2) ensures that the current row of every cursor is not changed by concurrently executing application processes. Under level CS, a row that is read and not updated during a unit of work and is no longer the current row of a cursor can be changed by concurrently executing application processes.

data pages. 4096-byte pages that contain the tables in a DBSPACE.

database archive. To copy the database (Directory and DBEXTENTS) to tape for the purpose of media recovery.

dbextent. The physical medium where database data is stored. Storage pools are composed of one or more dbextents.

dbspace. A logical allocation of space in a *storage pool* contained in a database. Contains one or more tables and their associated indexes.

dbspace scan. An access path for retrieving the results of an SQL statement by which all nonempty data pages of the DBSPACE are fetched.

directory. A list of identifiers that map corresponding items of data. For example, an SQL/DS directory maps dbspaces to addresses on a physical device.

directory block. A 512-byte record from the directory.

directory buffer. A storage area for directory blocks.

dispatcher. The component of SQL/DS that is responsible for allocating processor service to agents.

distributed data management (DDM). A protocol architecture that allows an application program to access data from a remote system. DDM, together with LU type 6.2, CDRA, and FD:OCA, provides the base for DRDA architecture.

FD:OCA. See *formatted data object content architecture*.

formatted data object content architecture (FD:OCA). An architected collection of constructs used to interchange formatted data. DDM, CDRA, LU type 6.2, and FD:OCA, provide the base for the remote unit of work architecture.

free class. An approximation of the free space on a data page.

free space. The number of bytes on a data page that are available for storing rows.

gatename. The SQL/DS NAME for a lockable object. It is a multipart name.

header pages. 4096-byte pages that contain the control row describing the tables and indexes in the DBSPACE.

***IDENT.** The VM system service that identifies the database as a LOCAL or GLOBAL resource.

index pages. 4096-byte pages that contain the indexes on the tables in a DBSPACE.

index scan. (1) An access path that uses an index to retrieve the results of an SQL statement. (2) To retrieve all the rows using an index.

IUCV. The VM Inter User Communication Vehicle (IUCV) function.

key. One or more columns identified as such in the description of a table, index, or referential constraint.

leaf page. An index page at the lowest level in the index tree structure; it contains key or TID pairs to the data in the table on which the index was created.

link. (1) A communication path between a user machine and SQL/DS machine. (2) An XPCC connection between an application partition and the SQL/DS partition; there is one link for each SQL/DS user agent.

lock escalation. A lock escalation occurs when the SQL/DS database manager increases the size of the data being locked. The database manager always escalates a lock to the dbspace level. (It does not increase locking from a row level to a page level.)

lock mode. The kind of lock that can be requested (and held). Possible lock modes are IS, IX, S, SIX, X, and U.

locking hierarchy. A prescribed sequence for locking objects such that conflicts can be recognized at their highest level.

locking protocol. Rules followed by the system to assure that actions taken on behalf of LUWs are protected against interference from other LUWs.

log archive. To copy the database log to tape or disk for the purpose of media recovery.

logical unit of work (LUW). A recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations.

LRB. Lock Request Block. A data area used to hold information about a lock request and define the characteristics of a lock on a data object.

LUW. See *logical unit of work (LUW)*.

mailbox. An area in which messages from one machine to another machine are constructed.

nonleaf page. An index page at the intermediate level in the index tree structure; it contains page numbers of pages in the next level in the index tree structure.

nonpreemptive. An environment in which agents willingly relinquish control of the processor; agents are not interrupted.

package. A control structure produced during program preparation that is used to execute SQL statements. Previously called *access module*.

PROGS. A table used by RDS to keep track of loaded packages.

pseudo agent. An SQL/DS structure that is allocated to a user upon establishing an IUCV or APPC/VM connection with the SQL/DS machine.

RDIIIN. The key input data area to RDS from an SQL/DS application program.

receive. The act of accepting a message from another machine.

repeatable read (RR). An isolation level that completely isolates an application process from all other concurrently executing application processes. Under level RR, (1) rows read during a unit of work cannot be changed by concurrently executing application processes until the unit of work is complete and (2) rows changed by concurrently executing application processes cannot be read until they are committed by that application process, and (3) phantom rows are prevented.

reply. The act of returning an answer to the virtual machine that sent the message.

request. A single SQL statement.

ROLLBACK WORK. The end of a logical unit of work that results in the backout of its changes to the database.

root page. The highest level in the index tree structure.

section. Portion of a package containing the processed form of the SQL/DS statement (PARSEDSECT, INTERPSECT, COMPILESECT, INDEFSECT).

section location table (SLT). Directory of sections and their corresponding statements in a package.

send. The act of sending a message from one machine to another machine.

storage pool. A specific set of available storage areas. These areas are used by the database administrator to control storage of the database. A storage pool contains one or more dbspaces.

TID. Tuple identifier. An internal identification of a row (tuple).

Bibliography

This bibliography lists publications that are referenced in this manual or that may be helpful.

SQL/DS Publications for VM

- *Managing the SQL/DS Environment for IBM VM Systems*, SH09-8109
- *SQL/DS Application Programming for IBM VM Systems*, SH09-8086
- *SQL/DS Database Administration for IBM VM Systems*, GH09-8083
- *SQL/DS Database Services Utility for IBM VM Systems*, SH09-8088
- *SQL/DS Diagnosis Guide and Reference for IBM VM Systems*, LH09-8081
- *SQL/DS General Information for IBM VM Systems*, GH09-8074
- *SQL/DS Installation for IBM VM Systems*, GH09-8078
- *SQL/DS Interactive SQL Guide and Reference for IBM VM Systems*, SH09-8085
- *SQL/DS Licensed Program Specifications for VM Systems and for VSE Systems*, GH09-8076
- *SQL/DS Master Index and Glossary for IBM VM Systems*, SH09-8089
- *SQL/DS Messages and Codes for IBM VM Systems*, SH09-8079
- *SQL/DS Operation for IBM VM Systems*, SH09-8080
- *SQL/DS Performance Tuning Handbook for IBM VM Systems and VSE*, SH09-8111
- *SQL/DS Quick Reference for IBM VM Systems and VSE*, SX09-1140
- *SQL/DS Solutions Directory*, GX09-1218

- *SQL/DS SQL Reference for IBM VM Systems and VSE*, SH09-8087
- *SQL/DS System Administration for IBM VM Systems*, GH09-8084

Virtual Machine/System Product (VMISP) Publications

- *VMISP Planning Guide and Reference*, SC19-6201

Distributed Relational Database Library

- *Application Programming Guide*, SC26-4773
- *Architecture Reference*, SC26-4651
- *Connectivity Guide*, SC26-4783
- *Executive Overview*, GC26-3195
- *Planning for Distributed Relational Database*, SC26-4650
- *Problem Determination Guide*, SC26-4782

Other Distributed Data Publications

- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Implementation Programmer's Guide*, SC21-9529
- *SAA Common Programming Interface Communications Reference*, SC26-4399
- *SAA Common Programming Interface Database Level 2 Reference*, SC26-4798
- *VM/Directory Maintenance Licensed Program Operation and Use Guide Release 4*, SC23-0437

VM Data Spaces Support Publications

- *VM Data Spaces Support*, SH09-8107

Miscellaneous Publications

- *IBM Dictionary of Computing.*, SC20-1699



Index

A

ABNEXIT 249
abnormal end user exits 249
abnormal termination 72, 247
access generation
 description 27
access module (see package)
access path 122, 162
 definition 311
 inefficient choices 122, 162
accounting
 ECMODE ON 147
ACQUIRE 108
 performance problem index 108
ACQUIRE DBSPACE
 how it works 286
adjacent index page locking
 possible lock wait problems 115
adjacent key locking
 definition 115
 example 135
 possible lock wait problems 115
 possible problems in catalogs 134
advanced program-to-program communication
 (APPC/VM) 20
advanced program-to-program communication/virtual
 machine (APPC/VM)
 definition 311
agent
 being held 119
 definition 311
 increasing 120
 index on related performance problems 111
 too many 206
agent handling 12, 13
 agent processing 17
 allocating users 13
 dispatcher components 14
 dispatching agents 16
 fair share auditing process 15
 finding deprived agents 15
 functional description 12
 prioritization of agents 14
 pseudo 13
 real 13
 setting fair share interval size 16
agent processing at end of an LUW 17
allocating users to agent structures 13
ALTER 108
 performance problem index 108
ALTER DBSPACE
 how it works 286

anatomy of a keyword string 69
 developing the first two 70
APPC/VM
 not resident, possible CPU usage problem 125
 synchronous 205
APPC/VM (see advanced program-to-program
 communication/virtual machine)
application design
 changing to avoid
 catalog conflicts 157
 DBSPACE scans 177
 I/O's 130, 171, 202, 210
 key lock contention 116
application function
 definition of 102
 performance problem index 106
application requester
 components 7
application server
 components 7
application server, SQL/DS
 components 8
archiving
 DBSS 52
 log 52
 user archive 52
ARI0126E message 213
 See also recovering from DBSS errors
arithmetic operator
 in syntax diagrams xiv
authorization 30
 checking 30
AUTOCOMMIT
 AUTOCOMMIT OFF
 implications on catalog locking 156
 possible link wait problems 119
 possible lock wait problems 181
 use in routines 121, 152
 AUTOCOMMIT ON
 use to avoid lock waits 152
 use to reduce link waits 121
AVS session limit
 exceeded 203

B

bad data distribution 122
basic index structure 39
batch
 possible link wait problems 120
 possible lock wait problems 181
BETWEEN, possible path selection problems 199
BLOCK I/O not resident, possible CPU usage
 problem 125

blocking suppression 125
books
 related 313
buffer hit ratio
 definition of 104
 large buffers 128
 possible CPU usage problem 189
 possible paging problem 189
 small buffers 128
 too many agents 207
buffer pool
 contention 207
 decreasing 127
 flooding 128, 201
 increasing 129, 160, 167
 management 39
 thrashing 104, 128, 207
 too big 126
 too small 128
bypassing UNDO WORK failure 228

C

cache (see data caching) 139
call processing 26
 access generation 26
 optimization 26
 parsing 26
 statement generation 26
cascading REVOKE
 locking done 155
catalog indexing, possible lock wait problems 134
catalog locking
 due to catalog queries 156
 due to naming conventions 134
 due to preprocessing 191
 exclusive locking in tables 153
 hot spot problems 153
 key conflicts 134
catalog statistics
 inaccurate values 162
 misleading values 122
 possible accuracy problems 162
 updating 163
channels
 adding more 160
 balancing usage 161
CHARNAME
 not set correctly 130
checkpoint
 "snapshot" of database 47
 and high logging volume on load 183
 being forced 130
 caused by SOSLEVEL 152
 CHKINTVL too big 131
 CHKINTVL too small 132
 definition 311
 delayed by DBSS calls 184

checkpoint (*continued*)
 frequent 152
 functional description 47
 interval too big 131
 interval too small 132
 need for in "soft" failure 47
CHKINTVL
 decreasing 132
 increasing 132, 183, 184
 too big 131
 too small 132
choose
 in syntax diagrams xv
CIRB 205
 too few agents 205
CIRT 205
 too few agents 205
CLOSE SCAN 60
clustering
 creating 188
 index 42
 need for 187
 to avoid DBSPACE scans 177
 to avoid I/O's 130, 202
CMS work unit support 134
COLCOUNT
 use in determining column selectivity 123
column selectivity, false sense of 123
column sequencing, for storage efficiency 144
combining columns 124
COMMENT ON
 how it works 289
commit
 definition 311
COMMIT WORK 111
 performance problem index 111
 to avoid lock waits 152, 182
 to reduce link waits 121
 use to avoid lock wait problems 117
COMMITCOUNT
 use to avoid link waits 121
communication concepts 17
 DRDA data streams 17
 mailbox functions 17
 resource adapter interaction 17
communication wait
 definition of 104
 holding agents 119
 performance problem index 111
 when locks held long time 180
COMMUNICATION WAIT indicator
 definition of 103
component identification 70
components of SQL/DS 6
concepts
 authorization 283
 catalog updates 283
 grant authorities 283

concepts (*continued*)

authorization (*continued*)
grant run privileges 284
grant table privileges 283
revoke authorities 285
revoke run privilege 285
revoke table privilege 285
interpretive commands 286
ACQUIRE DBSPACE 286
ALTER DBSPACE 286
COMMENT ON 289
CREATE INDEX 289
CREATE SYNONYM 290
CREATE TABLE 290
CREATE VIEW 291
DROP DBSPACE 292
DROP INDEX 292
DROP PACKAGE 293
DROP SYNONYM 293
DROP TABLE 293
DROP VIEW 294
how they work 286
LABEL ON 294
UPDATE STATISTICS 295
overview 6
concurrency level 206
consecutive keys, possible lock wait problems 115
consistently high response time
query block size too small 198
consistently high response time indicator
definition of 103
control blocks 280
conventions
syntax diagram notation xiv
conversational programming
possible link wait problems 119
possible lock wait problems 181
COUNTER
ESCALATE, uses of 150, 190
LOCKLMT, uses of 150, 190
LOGIO, use of 183
LPAGBUFF, in buffer hit ratio 126, 128
PAGEREAD, in buffer hit ratio 126, 128
CPU
one database machine needs too much 193
CREATE 108
performance problem index 108
CREATE INDEX 138
before loading 145
how it works 289
large sorts required 138
possible I/O problems 141, 175, 201
CREATE SYNONYM
how it works 290
CREATE TABLE
how it works 290

CREATE VIEW

how it works 291
cursor stability 56
definition 311
to avoid escalations 180
to avoid excessive locking 150
to avoid key locking problems 116
to avoid long lock waits 182
use on catalog queries 157

D

data areas
RDIEXT 277
RDIIN 273
data authorization
commands, definition of 104
performance problem index 108
data caching 139
data not cached 139
expanded storage 139
data conversion
possible CPU usage problems 174
data conversion (CONV)
component description 8
data definition 104, 108
language, definition of 104
performance problem index 108
data design
changing to avoid I/O's 124, 129
changing to avoid lock contention 117
use of efficient data types 143
data distribution, bad 122
data manipulation 104, 109
language, definition of 104
performance problem index 109
data page free-space management 33
Data System Control (DSC) 7
component description 7
overview 7
data type
use of same 165, 174
data utilities, performance problem index 110
database I/O's
versus paging I/O 126
database machine 139, 193
favored too little 139
needs too much CPU 193
database machine problems 247
Database Services Utility (DBSU) 7
description 7
Database Storage Subsystem (DBSS) 9
initialization 9
management 9
manipulation 9
OP Codes 265
overview 9
sort component 9

- dbextent
 - definition 311
- DBSPACE
 - definition 311
 - in deadlocks 268
 - scan via update statistics 210
- DBSPACE 1
 - possible key locking problems 134
 - possible locking problems 153, 191
- DBSPACE locking
 - decreasing 179
 - possible lock wait problems 178
 - through escalations 191
 - to avoid escalations 180
 - use of 180
- DBSPACE recovery 38
- DBSPACE scans
 - avoiding by indexing 151
 - avoiding by reorganization 151
 - avoiding by REORGANIZE INDEX 141
 - minimizing impact of 143
 - possible I/O problems 141, 175
 - possible lock wait problems 150
 - when range predicates used 199
 - with large tables 175
- DBSS (see Database Storage Subsystem)
- DBSS errors, recovering from
 - See recovering from DBSS errors
- DCSS, use to avoid paging 204
- DDL, definition of 104
- DDM (see distributed data management) 297
- deadlock
 - using trace facility for 267
- deadlocks 146
- default
 - in syntax diagrams xvi
- defect problems 69
- DELETE 60, 109
 - performance problem index 109
- deprived agents 15
 - finding 15
- developing the first keyword(s) 70
- deviations, functional 99
- devices
 - adding more 160
 - balancing usage 161
 - need more for paging 197
- diagnosing problems 247
- diagnosis
 - error codes
 - common user-related 96
 - rolled back due to deadlock (-911) 96
 - rolled back, excessive system wide lock requests (-912) 96
 - SQL command failed (-901) 95
 - system related 95
 - user-related 96
- diagnosis (*continued*)
 - functional deviations 99
 - functional problems 95
 - introduction to 1
 - of performance problems 101
- directory
 - definition 311
- directory inconsistencies
 - checking for 243
 - recovery actions 243
- directory verification 243
- directory, logs, DBEXTENTs, and I/O in DBSS 36
- disabling a DBSPACE 240
- dispatcher components 14
 - fair share auditing 14
 - locating a dispatchable agent 14
 - prioritization scheme 14
- DISPBIAS parameter
 - increasing 202
- disqualification, of indexes 164
- distributed data management
 - definition 311
- distributed data management (DDM) 297
- distributed relational database architecture
 - See DRDA protocol
- Distributed Relational Resource Manager (DRRM) 8
 - component description 8
 - overview 8
- DML, definition of 104
- document 81
- DRDA data streams 17
- DRDA protocol
 - APPC/VM communication protocol 24
 - communication concepts 17
 - overview of the SQL/DS RDBMS 11
 - possible performance problems 147, 198
 - SQL/DS components when using DRDA Protocol 6
- DROP 108
 - performance problem index 108
- DROP DBSPACE
 - how it works 292
- DROP INDEX
 - how it works 292
- DROP PACKAGE
 - how it works 293
- DROP SYNONYM
 - how it works 293
- DROP TABLE 141, 175, 201
 - how it works 293
 - possible I/O problems 141, 175, 201
- DROP VIEW
 - how it works 294
- DRRM, component overview 8
- dual logging 46
- dump 250
- dump navigation 253

dump processing 249
DVERIFY parameter of SLENL 243
dynamic SQL
 use for efficiency 171
 use with range predicates 200

E

ECMODE ON
 not needed for accounting 147
enabling a DBSPACE 241
errcds
 See no error codes, but problem exists
error codes
 -101 (command limitation exceeded) 96
 -204 (creator.table not found) 97
 -301 (input variable data type not compatible with
 column) 97
 -302 (input host variable too large) 98
 -305 (indicator variable is missing) 99
 -313 (mismatch between number of host
 variables) 99
 -901 (cannot use STORE CLOCK value) 95
 -911 (rolled back due to a deadlock) 96
 -912 (rolled back due to excessive lock
 requests) 96
 -915 (rolled back, excessive locks for this
 LUW) 96
 common user-related 96
 system related 95
ESCALATE counter
 uses of 150, 190
escalates indicator
 definition of 103
escalation
 definition of 104
 excessive locking 149
 low lock levels 179
 performance problem index 114
 small NLRB parameters 191
 when it is good 190
explicit termination of LUW 12
EXTEND input file commands 224

F

fair share auditing 15
 finding deprived agents 15
 interval sizes 18
 overview 15
FD:OCA (see formatted data object content architec-
 ture)
FETCH 80
fetch with cursor stability 99
fieldproc block
 in syntax diagrams xvii
filtered log recovery 224

filtered log recovery and referential integrity 237
finding deprived agents 15
first failure data capture 76
FIRSTKEYCOUNT
 use in determining index selectivity 123
formatted data object content architecture (FD:OCA)
 definition 311
forms for reporting an SQL/DS problem 89
 abnormal termination 89
 documentation problem 94
 incorrect or missing output 93
 message 90
 no response 91
 output missing or incorrect 93
 slow response 92
fragment of syntax
 in syntax diagrams xvii
fragmentation 43
fragmented index 169
free classes 34
free space
 lowering 130
 use to avoid DBSPACE scans 143
free space management 33, 34
 free classes 34
freeing log space 49
FULLKEYCOUNT
 use in determining index selectivity 123
 use to find hot keys 158
functional deviations 99
 fetch with cursor stability 99
 lockout with cursor stability 99
functional problems, diagnosis 95
 system-related error codes 95

G

glossary
 performance index headers 102
 performance indicator terms 102
 performance terms 103
GRANT 108
 performance problem index 108
grant authorities 283
GRANT option 155
 exclusive locking on REVOKE 155
grant run privileges (Authorization) 284
grant table privileges (Authorization) 283

H

handling of agents 12
hashing
 key, possible lock wait problems 115
hiding bad columns 124
high CPU usage
 bad statistics 122, 162
 CMS work unit support 134

high CPU usage (*continued*)

- CREATE INDEX 138
- DBSPACE scan 141
- DRDA protocol used to access an SQL/DS data-
base 147
- fragmented index 169
- in one DB machine 193
- index disqualification 164
- index maintenance 167
- inefficient search 170
- inefficient SELECT list 174
- insufficient indexing 174
- invalid entity 175
- invalid index 175
- joins 208
- large DBSPACE 175
- large sort 138
- missing search condition 185
- nonunique key prefix 211
- old package 194
- performance problem index 112
- possible problem 188
- QDROP ON 203
- range predicate 199
- small buffer 128
- small CHKINTVL 132
- unclustered index 168, 187
- VM code not resident 125

high CPU usage indicator
definition of 103

high I/O utilization
performance problem index 113
possible problems 160, 161

HIGH I/O UTILIZATION INDICATOR
definition of 103

high I/O's

- bad statistics 122, 162
- causing high CPU usage 188
- CREATE INDEX 138
- DBSPACE scans 141, 175
- fragmented index 169
- index disqualification 164
- index maintenance 167
- inefficient search 170
- insufficient indexing 174
- joins 208
- large DBSPACE 175
- large sort 138
- missing search condition 185
- nonunique key prefix 211
- old package 194
- performance problem index 113
- range predicate 199
- small buffer 128
- small CHKINTVL 132
- too many agents 208
- unclustered index 168, 187

high I/O's (*continued*)

- UPDATE STATISTICS 210
- high I/O's indicator
definition of 103
- high selectivity, false sense of 123
- host variable 199
 - in syntax diagrams xiv
 - possible path selection problems 199
- hot spot
 - in catalog tables 153
 - in user tables 158

I

I/O

- capacity exceeded 160
- index on performance problems 113
- not balanced 161
- reducing rate 161

IBM manuals, related 313

implicit termination of LUW 12

IN predicate
instead of OR 171
use in place of OR 165

inconsistent directory 243
checking for 243
recovery actions 243

incorrect or missing output 81

index

- clustering to avoid I/O's 169
- disqualification 164
- dropping 116
- fragmented 169
- invalid 41
- maintenance, possible I/O problems 167
- no longer clustered 168
- nonunique key prefix problems 211
- on performance problems
 - agent related 111
 - data authorization 108
 - data definition 108
 - data manipulation 109
 - data utilities 110
 - general 107
 - high CPU usage 112
 - I/O related 113
 - locking related 114
 - low CPU utilization 112
 - recovery control 111
 - special case 115
 - storage related 114
- on problems by performance symptom 111
- pages, possible lock wait problems 115
- re-creating 167, 170
- re-design to avoid hot keys 159

REORGANIZE INDEX 170

transient 41

index eligible predicate
 definition of 104
 use of 165
index key prefix, nonunique problems 211
index locking 63
index management 39
index management, fragmentation 43
index management, space management 40
index structure, basic 39
indexes 42
 clustering 42
indexing 39
 after loading 167
 concepts 39
 hiding bad columns 124
 insufficient 174
 need to cluster 187
 on catalog tables 135
 on sort columns 174
 possible I/O problems 138, 164, 168, 169, 187, 211
 smoothing keys 124
 to avoid DBSPACE scans 150, 177
 to avoid I/O's 165, 167, 169, 174, 188, 202, 209
 use of numeric data types 211
indicator
 COMM WAITS, definition of 103
 CONSISTENTLY HIGH RESPONSE TIME, definition of 103
 definition of 103
 ESCALATES, definition of 103
 HIGH CPU USAGE, definition of 103
 HIGH I/O UTILIZATION, definition of 103
 HIGH I/O, definition of 103
 LINK WAITS, definition of 103
 LOCK WAITS, definition of 103
 LOG I/O'S, definition of 103
 LOW CPU UTILIZATION, definition of 103
 LOW I/O UTILIZATION, definition of 103
 PAGING, definition of 103
 PERFORMANCE, definition of 102
 PERIODIC HIGH RESPONSE TIME, definition of 103
INSERT 60
 format 2 problems 209
 performance problem index 109
INSERT logic 42
 default logic 42
Inter-User Communication Vehicle (IUCV)
 protocol 18
interactive SQL (see ISQL) 7
 description 7
Interactive Structured Query Language (see ISQL)
interpretive commands, how they work 286
invalid entity 175
invalid index 40, 41, 175
invalidate package to improve performance 194

isolation level 56
isolation level, cursor stability (see cursor stability)
ISQL (Interactive Structured Query Language) 1
IUCV not resident, possible CPU usage problem 125

J

join
 too many 208
 use of 171
joining tables, in applications 210

K

key
 definition 311
key hashing
 hot key hashes 158
 possible lock wait problems 115, 136
key locking
 definition of 104
 hot keys 158
 in catalog tables 134, 153
 possible lock wait problems 115, 136
key sequence, possible lock wait problems 115
key structure
 changing to avoid hash conflicts 137
 changing to avoid hot keys 159
 changing to avoid lock conflicts 116
 making keys smaller 138
keyword
 additional 82
 in syntax diagrams xiv
keyword types 70
 abnormal termination 72
 component identification 70
 document 81
 first failure data capture 76
 incorrect or missing output 81
 message 74
 no response 80
 release level 71
 slow response 81
keyword, developing
 first two 70
 remaining 72
 abnormal termination 72
 document 81
 first failure data capture 76
 message 74
 no response 80
 output missing/incorrect 81
 slow response 81
 wait or loop 80

L

LABEL ON

how it works 294

large sort, required on CREATE INDEX 138

large tables 175, 177, 210

in same DBSPACE 175

putting in own DBSPACE 177, 210

layout of special operator command responses 279

layout of storage after initialization 256

link waits

causes of 180

definition of 104

due to held agents 119

indicator definition 103

performance problem index 111

ratio definition 104

when locks held long time 180

linkmap

access for SQL/DS 252

definition of 251

loading data

definition of 104

exclusive locking in catalogs 154

logging during 183

performance problem index 110

possible DBSPACE scan problems 145

possible I/O problems 200, 210

possible lock escalation problems 179, 192

possible lock wait problems 115, 136, 158, 178,
181, 210

locating and dispatching a dispatchable agent 16

locating SQL/DS statements 262

lock

held for long duration 180

lock concept 55

how it works 55

access to private DBSPACES 66

deadlock detection 65

isolation level cursor stability 56

isolation level repeatable read 56

lock compatibility 59

lock durations 59

lock escalation 65

lock modes 58

locking done by SQL/DS 60

locking hierarchy 57

isolation levels 56

lock escalation

definition 312

definition of 104

due to excessive locking 149

due to low lock levels 179

due to small NLRB parameters 191

performance problem index 114

when it is good 190

lock level

decreasing 116, 151, 159, 179

lock level (continued)

definition of 104

increasing 137, 151, 180

set too low 179

too high 178

lock request block (LRB) 190

possible paging problems 190

lock wait rate

definition 104

lock waits

definition of 104

due hot spots in data 158

due to adjacent key locking 115

due to DBSPACE scans 141, 175

due to excessive locking 149

due to held agents 119

due to held locks 180

due to high lock levels 178

due to key hash conflicts 136

due to key locking in catalogs 134

due to large DBSPACES 175

due to lock escalations 191

due to locking in catalogs 153

due to low lock levels 179

due to repeatable read usage 181

due to too many agents 207

due to UPDATE STATISTICS 210

indicator definition 103

performance problem index 114

locking

adjacent index page 115

adjacent key, example of 135

adjacent key, possible problems in catalogs 134

excessive 149

exclusive locking in tables 153

index on locking related problems 114

possible adjacent key problems 115

locking done by SQL/DS 60

locking on indexes 63

LOCKLMT counter

uses of 150, 190

LOCKMODE, possible lock wait problems 178, 179

lockout with cursor stability 99

log checkpoints 50

log I/O's

due to loading 183

performance problem index 113

LOG I/O'S indicator, definition of 103

log recovery, filtered 224

log space, freeing 49

logging

decreasing rate 132

during loads 183

running with 131

logging/recovery concepts 46

logical storage management 32

logical unit of work (LUW) 46
 avoiding short ones 131
 definition 312
 use of multiple 116, 121, 150, 182
LOGIO COUNTER, use of 183
LOGMODE=N, use for loading 183
long DBSS calls, possible problems 184
low CPU utilization indicator
 definition of 103
low CPU utilization, performance problem index 112
low I/O rate indicator
 definition of 103
low selectivity, false sense of 123
LPAGBUFF counter
 use in buffer hit ratio 126, 128
LRB (see lock request block) 190
LUW
 See logical unit of work (LUW)
LUW (see logical unit of work)
LUW concepts 11
 functional description 11
 logical unit of work 11
 LUW management 11
LUW recovery 48
LUW termination, implicit 12
LUW, explicit termination 12

M

mailbox functions 17
 resource adapter interaction 17
maintenance
 index 167
major control blocks 260
make BLOCK I/O resident 125
make IUCV resident 125
making key columns smaller 138
management of storage 32
manuals
 related manuals 313
mapping DBSPACES to DASD 36
materials for reporting a problem 85
materials to send to IBM 85
 environments 87
MAXCONN
 decreasing 208
 increasing 120
memory, when to add 127, 190
message 74
message ARI0128E 213
 See *a/so* recovering from DBSS errors
message handling, Resource Adapter 17
 relationship with DSC 17
 relationship with RDS 17
mini-dump 250
missing output 81
multiple database mode, to avoid paging effects 197

multiple LUWs
 to avoid excessive locking 150
 to avoid lock contention 182
 to reduce link waits 121
 to reduce lock waits 116
multiple queries, use of 171
multiple user mode 10
 DSC 10

N

NACTIVE
 large values 175
naming conventions, changing 136, 157
NCUSERS
 decreasing 129, 208
 increasing 120
 too high 206
NDIRBUF
 large enough 139
 too big 126
 too small 128
NEXT 60
NLRB parameters
 decreasing 191
 increasing 192
 too large 190
 too small 191
NLRBS
 increasing 192
 too large 190
 too small 191
NLRBU
 increasing 192
 too large 190
 too small 191
no error codes, but problem exists
 fetch with cursor stability 99
 lockout with cursor stability 99
no response 80
 wait or loop 80
NOLINKS 205, 206
 too few agents 205, 206
non-SQL work
 possible CPU usage problems 189
 possible paging problems 190, 207
non-unique index, key locking 136
nonrecoverable DBSPACE 183
 for loading without logging 183
nonunique index key prefix 211
 possible I/O problems 211
nonunique index, definition of 104
nonunique key prefix, definition of 104
NPAGBUF
 too big 126
 too small 128

O

- OP Codes 265
- OPEN SCAN 60
- operator commands
 - layouts of responses to special operator commands 279
- optimization 26
 - functional description 26
- optional
 - default parameter
 - in syntax diagrams xvi
 - keyword
 - in syntax diagrams xvi
- optional item
 - in syntax diagrams xv

P

- package 25, 194
 - concepts 25
 - definition 312
 - needs re-preprocessing 194
- page faults, serialization 197
- page locking
 - hot pages 158
 - possible lock escalation problems 179, 191
 - possible lock wait problems 115, 178
 - to avoid escalations 180
 - use of 151, 179, 180
- PAGEREAD counter
 - use in buffer hit ratio 126, 128
- paging
 - causing high CPU usage 188
 - due to code not shared 204
 - due to large buffers 126
 - due to large NLRB settings 190
 - due to QDROP ON 203
 - due to too many agents 207
 - effects of 197
 - performance problem index 114
 - possible CPU usage problem 188
 - possible problem 189
- paging devices, need for more 197
- paging faults, effects of 126
- paging indicator
 - definition of 103
- paging rate, high value 126, 189
- parentheses
 - in syntax diagrams xiv
- parsing 26
 - functional description 26
- PCTPAGES
 - high values 175
 - inaccurate values 163
 - low values 151
- performance analysis
 - chapter on diagnosing 101

- PERFORMANCE INDICATOR, definition of 102
- performance problem
 - diagnosing 101
 - index by
 - application function 106
 - data authorization functions 108
 - data definition functions 108
 - data manipulation functions 109
 - data utilities functions 110
 - recovery control functions 111
 - index on
 - agent related problems 111
 - CPU related problems 112
 - general problems 107
 - I/O related problems 113
 - locking related problems 114
 - special case problems 115
 - storage related problems 114
- performance symptom, index to problems 111
- periodic high response time
 - DRDA protocol used to access an SQL/DS database 147
 - DRDA usage 147
 - due to big CHKINTVL 131
 - due to forced checkpoints 130
 - due to logging during loads 183
 - due to long DBSS calls 184
 - due to sequential processing 200
 - due to small CHKINTVL 132
 - performance problem index 115
- periodic high response time indicator
 - definition of 103
- POSSIBLE PROBLEM, definition of 102
- predicates 186, 200
 - adding more 171, 186, 200
- PREP, locking problems 191
- preprocessor
 - definition of 104
 - description 7
 - exclusive locking in catalogs 154
 - performance problem index 110
 - possible locking problems 191
- prevention of log-record overlay 50
- prioritization of agents 14
 - overview 14
- problem handling 247
- problem isolation and the trace facility 267
- problems in the database machine 247
- procedures
 - abnormal termination 72
 - document 81
 - first failure data capture 76
 - incorrect or missing output 81
 - message 74
 - no response 80
 - slow response 81

processing dumps 249
pseudo-agent structure 13
pseudo-conversational programming
 use to reduce link waits 121
publications
 related 313
punctuation mark
 in syntax diagrams xiv

Q

QCE, definition of 104
QDROP OFF USERS
 not being used 203
 setting 203
query block size 198
 possible problem with size 198
query cost estimate
 definition of 104
 inaccurate results 122, 162
QUICKDSP ON 203

R

range predicates, possible path selection
 problems 199
RDIEXT (data area) 277
RDIIN (data area) 25, 273
 functional description 25
 RDIIN control block 25
re-preprocess
 package needs 194
 to improve performance 194
real agent structure 13
real storage 127, 190, 197
 when to add 127, 190, 197
recovering from DBSS errors 213–241
 bypassing UNDO WORK failure 228
 causes for errors 214
 committed work, rolling back 231
 disabling a DBSPACE 240
 enabling a DBSPACE 241
 examples of diagnostic displays 213
 extended processing 225
 forward processing failure action 219
 interpreting the diagnostic display 213
 invoking SQL/DS 224
 example to process EXTEND input file com-
 mands 228
 meanings of fields in display 215, 216, 217, 218,
 219
REDO processing failures 223
 during restore 224
 during warm start 223
ROLLBACK processing failure 221
rolling back committed work 231
UNDO processing failure 221
 during restore 222
 during warm start 221

recovery 224
 filtered log 224
recovery control commands, definition of 104
recovery control, performance problem index 111
recovery of DBSPACE 38
recovery, LUW 48
redundant data
 stored results 118
 table splitting 118
 to avoid
 I/O's 129, 209
 key lock contention 159
 lock contention 182
 lock waits on keys 116
 transaction tables 118
referential constraints, filtered log recovery 225
referential integrity, filtered log recovery 237
related publications 313
Relational Data System (RDS) 8
 executives 8
 functional description 8
 invocation 8
 loading and storing of packages 8
 optimizing 8
 overview 8
 parsing 8
reorganization
 to avoid DBSPACE scans 143, 150, 177
 to avoid I/O's 202, 209, 210
REORGANIZE INDEX
 indexes are fragmented 169
 to avoid DBSPACE Scan 141
 to avoid large sorts 138
repeat symbol
 in syntax diagrams xv
repeatable read 56
 locking problems with ISQL usage 181
repeatable read (RR)
 definition 312
reporting a problem 72
 environments 87
 forms for 89
 materials 85
reporting an SQL/DS program problem 85
reprep 30
 concepts 30
 invalid packages 30
repreprocessing 30
required item
 in syntax diagrams xiv
reserved pages 38
Resource Adapter 7
REVOKE 108
 performance problem index 108
revoke authorities 285
revoke run privilege (Authorization) 285

revoke table privilege (Authorization) 285
ROLLBACK WORK 111
 performance problem index 111
 to reduce link waits 121
 use to avoid lock wait problems 117
rolling back committed work 231
row locking
 hot rows 158
 in catalogs 134, 153
 possible lock escalation problems 179, 191
 possible lock wait problems 115, 136, 179
 use of 151, 159, 179
ROWCOUNT
 use in determining column selectivity 123
 use to find hot spots 158

S

saved segments, use to avoid paging 204
search
 inefficient 170
 missing condition 185
search condition
 missing 185
SELECT 109
 performance problem index 109
SELECT list, inefficient 174
selective index, definition of 104
selectivity, false sense of 123
sequential processing, possible I/O problems 200
serialization, on page faults 197
session limit
 exceeded 203
SET QDROP OFF USERS
 not being used 203
 specifying 203
 to avoid paging 197
SET QUICKDSP ON 203
SET STATISTICS OFF, on DATALOAD 210
setting fair share interval size 16
shadow pages 47
SHOW LOCK
 ACTIVE, use to find communication waits 119
 AGENT, uses of 150
 MATRIX, use to find LRBs needed 190
 USER, uses of 150
 WANTLOCK, to find key lock conflicts 135
single user mode 10
 DBSS 10
 RDS 10
 use for loading 183
slow response 81
smoothing keys 124
sort
 possible I/O problems 201
 required on CREATE INDEX 138
 spreading 138

Sort Concepts 44
Sorting 44
space management 40
specifying isolation levels 56
splitting
 tables 118, 210
 work among DB machines 194
spreading sorts 138
SQL/DS
 advanced program-to-program communication
 (APPC/VM) 20
 agent handling and communications 13
 code not shared 204
 in multiple user mode 10
 in single user mode 10
 Inter-User Communication Vehicle (IUCV)
 protocol 18
 machine favored too little 139
 statements associated with a system error 262
 structure and components 6
 termination 66
 trace facility 267
SQL/DS dumps 250
SQL/DS free space management 33
SQL/DS link maps and access 252
SQL/DS program problems 69
SQLHX 205
stack storage 31
statement generator
 description 27
statistics
 inaccurate values 162
 misleading values 122
 possible accuracy problems 162
 updating 163
 updating on a DBSPACE 295
storage component 36
 buffer pool management 39
 DBSPACE recovery 38
 directory, logs, DBEXTENTS, and I/O 36
 how it works 36
 mapping DBSPACES to DASD 36
 storage pools 37
storage layout after initialization 256
storage management concepts 31
storage management, logical 32
storage pool
 definition 312
 putting internal DBSPACES in 138
 relation to CHKINTVL 133
 shadow pages in 133
storage pools in DBSS 37
storage related performance problem index 114
storage services 31
 stack storage 31
 working storage 32

- stored results 118, 209
 - to avoid I/O's 209
- structure 13
 - pseudo-agent 13
 - real agent 13
- structure, basic index 39
- subqueries
 - possible I/O problems 201
 - possible problems 209
- symptom strings 250
- synchronous 205
 - SQLINIT parameter 205
 - SYNC(YES) 205
- syntax diagram
 - notation conventions xiv
- SYSACCESS
 - exclusive locking on 154
- SYSCATALOG
 - exclusive locking on 154
- SYSCOLAUTH
 - exclusive locking on 154
- SYSCOLUMNS
 - exclusive locking on 154
- SYSDBSPACES
 - exclusive locking on 154
- SYSINDEXES
 - exclusive locking on 154
- SYSPROGAUTH
 - exclusive locking on 154
- SYSTABAUTH
 - exclusive locking on 154
- system error SQL/DS statements 262
- system-related error codes 95
 - See *also* functional problems, diagnosis
- Systems Network Architecture (SNA)
 - AVS session limit exceeded 203
- SYSUSAGE
 - exclusive locking on 154
- SYSUSERAUTH
 - exclusive locking on 154
- SYSVIEWS
 - exclusive locking on 154

T

- table
 - large table problems 143, 175
 - locking 180
 - to avoid escalations 180
 - mapping to DBSPACES 130, 143
 - putting in own DBSPACE 210
 - small table problems 158
 - splitting 118, 210
 - transaction tables 118
- termination
 - abnormal 247
- terminology
 - performance analysis 102

- terminology and concepts 5
- trace facility 267
- trace facility for deadlocks 267
- transaction tables 118
- transient index 41
- type of keyword 70
 - abnormal termination 72
 - component identification 70
 - document 81
 - first failure data capture 76
 - incorrect or missing output 81
 - message 74
 - no response 80
 - release level 71
 - slow response 81

U

- UNION
 - instead of OR 171
 - use in place of OR 165
- unloading
 - definition of 104
 - performance problem index 110
 - possible I/O problems 141, 175, 200
 - possible lock escalation problems 179
 - possible lock wait problems 115, 136, 158, 178
- UPDATE 60, 109
 - performance problem index 109
- UPDATE STATISTICS
 - by DATALOAD 210
 - exclusive locking in catalogs 155
 - performance problem index 110
 - possible DBSPACE scan problems 141
 - possible I/O problems 175, 201
 - use of 163
- updating
 - data statistics 163
 - problems addressed 163
- use of multiple LUWs 116, 121, 150, 182
- user exits 249
- user-related error codes 96
 - command limitation exceeded (-101) 96
 - creator.table not found (-204) 97
 - indicator variable is missing (-305) 99
 - input host variable too large (-302) 98
 - input variable data type not compatible with column (-301) 97
 - mismatch between number of host variables (-313) 99
- using the directory verify function 243

V

- verify directory 243
 - using 243
- very nonunique index key prefix 211
 - possible I/O problems 211

very nonunique index, definition of 104
very nonunique key prefix 104
 definition 104

W

wait or loop 80
Work Unit Manager (WUM) 8
 component description 8
 overview 8
work unit support 134
working storage 32

Readers' Comments

**SQL/DS
Diagnosis Guide and Reference
for IBM VM Systems
Version 3 Release 4**

Publication No. LH09-8081-03

Please use this form only to identify publication errors or to request changes in this publication. Direct any requests for additional publications, technical questions about IBM systems, changes in programming support, and so on, to your IBM representative or to your nearest IBM branch office. You may use this form to communicate your comments about this publication, its organization, or subject matter **with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.**

- If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.
- If you would like a reply, check this box. Be sure to print your name and address below.

You can also send your comments by facsimile to (416) 448-6057 addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to torrcf@vnet.ibm.com; IBMLINK, to [toribm\(torrcf\)](mailto:toribm(torrcf)@ibmlink.com); IBM/PROFS, to [torolab4\(torrcf\)](mailto:torolab4(torrcf)@profs.ibm.com); IBMMAIL, to [ibmmail\(caibmwt9\)](mailto:ibmmail(caibmwt9)@ibm.com).

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

Page number(s): Comment(s):

_____	_____
Name	Address
_____	_____
Company or Organization	
_____	_____
Phone No.	



Fold and Tape

Please do not staple

Fold and Tape

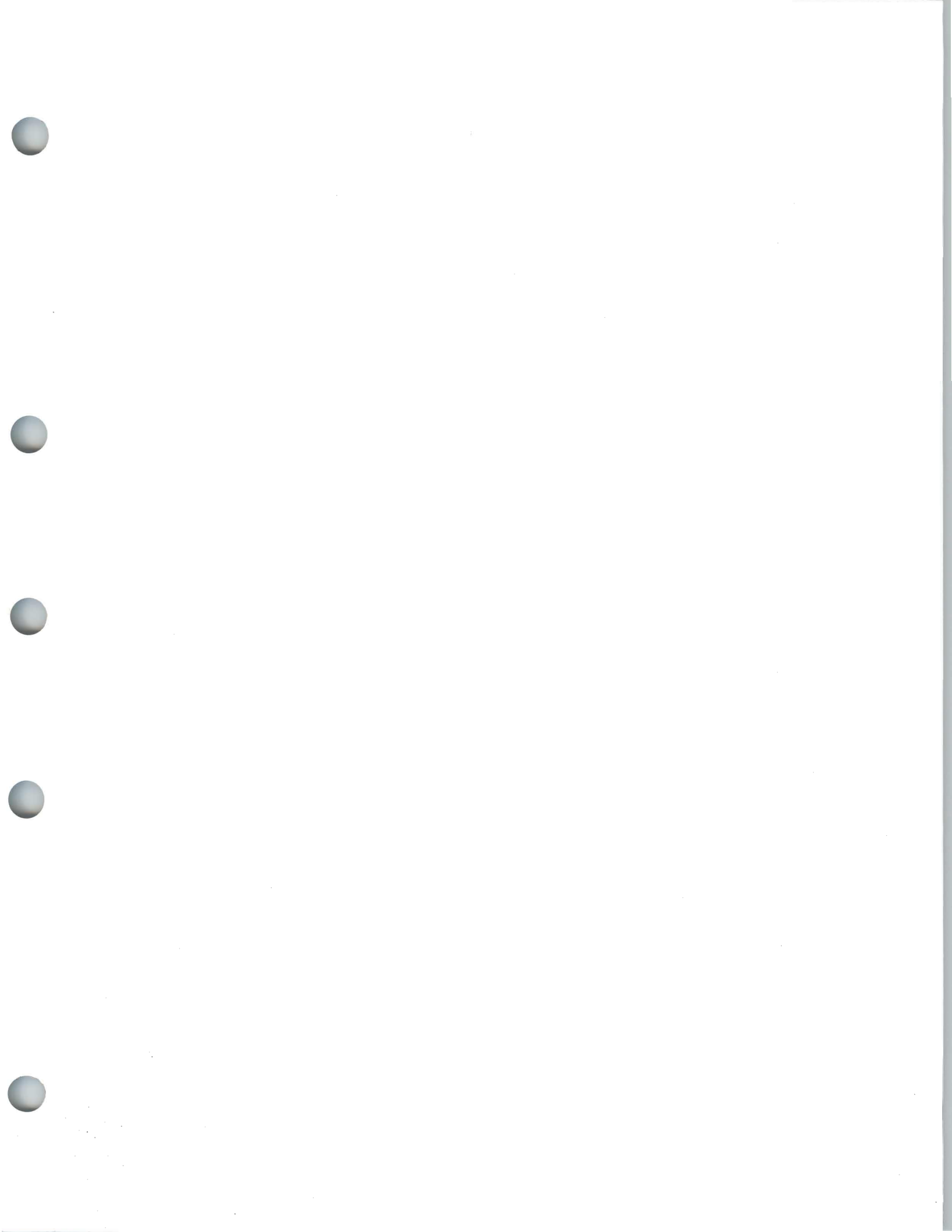
PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
21/986/844/TOR
844 DON MILLS ROAD
NORTH YORK ONTARIO CANADA M3C 1V7

Fold and Tape

Please do not staple

Fold and Tape





File Number: S370/4300-50
Program Number: 5688-103 8084

"Restricted Materials of IBM"
Licensed Materials - Property of IBM
LH09-8081-03 © Copyright IBM Corp. 1987, 1993

Printed in U.S.A.

LH09-8081-03

