DB2® for VSE & VM

Performance Tuning Handbook

Version 6 Release 1

GC09-2669-00

DB2® for VSE & VM

Performance Tuning Handbook

Version 6 Release 1

GC09-2669-00

Note!

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

This book is also provided as an online book that can be viewed with the IBM® BookManager® READ and IBM Library Reader™ licensed programs.

| **First Edition (December 1998)**

| This edition, GC09-2669, applies to Release 6 of the IBM DATABASE 2® Server for VSE & VM Program 5648-A70, and to all
| subsequent releases of this product 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, comments may be addressed to: IBM Canada Ltd. Laboratory, Information Development, 2G/345/1150/TOR 1150 Eglinton Ave East, North York, Ontario, Canada M3C 1H7.

You can also send your comments by facsimile to (416) 448-6161 addressed to the attention of the RCF Coordinator. If you have
| access to Internet, you can send your comments electronically to **torrcf@ca.ibm.com**; IBMLink™, to **toribm(torrcf)**; IBM/PROFS®, to **torolab4(torrcf)**; IBMMAIL, to **ibmmail(caibmwt9)**; or through our home page at
| **http://www.software.ibm.com/data/db2/vse-vm/**

If you choose to respond through Internet, please include either your entire Internet 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.

Contents

Notices . . . ix
Trademarks . . . ix

About This Manual . . . xi
Who Should Use This Manual . . . xi
Organization . . . xi
Prerequisite Reading . . . xii
Syntax Notation Conventions . . . xii
SQL Reserved Words . . . xvi
Conventions Used for Highlighting Examples . . . xvii

| **Summary of Changes for DB2 Version 6 Release 1** . . . xix
| Enhancements, New Functions, and New Capabilities . . . xix
| DRDA® RUOW Application Requestor for VSE (Online) . . . xix
| Stored Procedures . . . xix
| TCP/IP Support for DB2 Server for VM . . . xx
| New Code Page and Euro Symbol Code Page Support . . . xx
| DataPropagator™ Capture . . . xx
| QMF for VM, QMF for VSE, and QMF for Windows® . . . xxi
| RDS Above the Line . . . xxi
| Combining of NLS Feature Installation Tapes with Base Product Installation
| Tape . . . xxi
| Control Center Feature . . . xxii
| Data Restore Feature . . . xxii
| DB2 REXX SQL Feature . . . xxii
| Reliability, Availability, and Serviceability Improvements . . . xxii
| Migration Considerations . . . xxii
| Library Enhancements . . . xxiii

Chapter 1. Improving Performance . . . 1
Elements of Performance . . . 1
Tuning Guidelines . . . 1
Performance Improvement Process . . . 2
How Much Can a System be Tuned? . . . 3
Workload . . . 3
Performance Indicators . . . 4
Establishing Performance Objectives . . . 4
Response Time . . . 4
Components of Response Time . . . 4
Throughput . . . 5
Availability . . . 5
A Less Formal Approach . . . 5
Monitoring Performance . . . 6
Creating a Monitoring Plan . . . 6
Monitoring Interval . . . 6
Cost of Monitoring . . . 7
Measurements . . . 7
Tools . . . 8
VM Tools . . . 8
VSE Tools . . . 8

CICS Tools . . . 8
DB2 Server for VSE & VM Tools . . . 9
Factors Affecting Performance . . . 10
Resources . . . 10
Processor . . . 10
Storage . . . 10
Overhead . . . 11
Concurrency . . . 11
Locking . . . 11
Recovery . . . 12
Choosing Between Tuning Trade-offs . . . 12

Chapter 2. Measuring Performance . . . 15
Operating System Measurements . . . 15
Processor (CPU) Load . . . 15
Real and Virtual Storage Load . . . 15
System Paging DASD Load . . . 15
Machine or Partition DASD I/O Load . . . 15
Individual Device Utilization . . . 15
Translating Performance Measurements to Indicators . . . 16
CP INDICATE USER and QUERY TIME Commands . . . 16
CICS Monitoring (CICSPARS for VSE) . . . 18
DB2 Server for VSE & VM Tools . . . 20
Physical Data Locations . . . 20
Disk Locations (VM) . . . 20
Data Set Placement (VSE) . . . 20
Initialization Parameters . . . 21
CIRD Transaction (CICS) . . . 22
COUNTER Operator Command . . . 23
DB2 Data Spaces Support . . . 25
SHOW Commands . . . 25
Storage . . . 25
Concurrency . . . 28
VSE SHOW CONNECT . . . 33
Locking . . . 36
Database catalog . . . 38
SYSTEM.SYSCATALOG . . . 38
SYSTEM.SYSCOLUMNS . . . 39
SYSTEM.SYSDBSPACES . . . 40
SYSTEM.SYSINDEXES . . . 41

Chapter 3. Managing Storage and Configuring the Operating System . . . 43
Real and Virtual Storage . . . 43
Virtual Addressing . . . 43
Pages . . . 43
Auxiliary Storage . . . 43
Storage Queues . . . 46
Address Space Size . . . 47
Storage Above 16MB (31 Bit Addressing) . . . 47
Saved Segments (VM Only) . . . 48
DB2 Data Spaces Support (VM/ESA only) . . . 48
Virtual Disk Support for VSE/ESA for Internal Dbspaces . . . 50
Virtual Disk Support for VM/ESA for Internal Dbspaces . . . 56
DASD Storage . . . 61

In VSE . . . 61
In VM . . . 61
Mapping of Dbspaces to DASD . . . 62
Logical To Physical Page Relationships . . . 62
Storage Pools . . . 62
Managing Storage Pool Space . . . 62
Short on Storage Cushion . . . 62
Types of Pages . . . 63
Number of Header Pages . . . 63
Proportion of Index to Data and Header Pages . . . 64
Free Space in Data Pages . . . 64
Free Space in Index Pages . . . 66
Shadow Pages . . . 66
Running out of Dbspace Pages . . . 68
Data Clustering . . . 70
Clustered Indexes . . . 70
The Clustering Index . . . 70
How Indexes Become Unclustered . . . 71
Identifying Unclustered Indexes . . . 71
Reorganizing Data . . . 73
Reorganizing a Single Table . . . 74
Reorganizing all the Tables in a Dbspace . . . 75
Changing the Clustering Index without Dropping Indexes . . . 76
Index Fragmentation . . . 76
Reorganizing Fragmented Indexes . . . 77
Invalid Indexes . . . 78
Transient Indexes . . . 78
Reorganizing an Invalid Index . . . 79
DASD Balancing . . . 79
Evenly Distributing Workload across Physical Volumes . . . 79
Moving Dbextents . . . 79
Reassigning Dbspaces . . . 80
General Considerations . . . 81
VM . . . 82
Fair Share Scheduling . . . 82
VSE . . . 83
Dispatching Priority . . . 83
Fast CCW Translation . . . 83
Virtual Addressability Extension (VAE) . . . 84
31 Bit Addressing . . . 84
Compile Partition Size . . . 84
CICS . . . 84
AMXT/MXT . . . 84
ISQL . . . 84
Transaction Name . . . 85
Number of Concurrent Users . . . 85
Temporary storage . . . 85
Auxiliary versus Main . . . 85
Guest Sharing with VSE under VM . . . 86
Distributed Configuration Considerations . . . 86
In a VM System . . . 86
In a VSE System . . . 86
Performance Implications . . . 86
Applications Planning . . . 87

Fetch and Insert Blocking 87
Hold File 87
Local Copy 87

Chapter 4. Configuring the Application Server and Requester 89
Database Manager Storage 89
Database I/O 89
Tuning Parameters 91
Performance Indicator 91
DB2 Data Spaces Support (VM/ESA only) 92
Using Virtual Disks 94
Package Cache 94
Tuning Parameters 95
Concurrency 95
Agents 95
Allocating Users to Agent Structures 96
Tuning Parameters (NCUSERS) 96
Performance Indicator 97
CICS 97
Tuning Parameter (CIRB) 97
Performance Indicator 97
Pseudo-Agents 97
Tuning Parameter 98
Privileged Remote DRDA User (VSE Server) 99
Performance Indicator (SHOW USERS) 99
Dispatching Agents 99
Prioritization 100
Fair Share Auditing 100
Tuning Parameter (DISPBIAS) 100
Performance Indicator 100
Startup Mode 100
Locking 100
Locking Contention 101
Locking Hierarchy 101
Lock Modes 101
Lock Duration 102
Lock Compatibility 102
Number of Concurrent Users 103
Minimum Lock Level 103
Indexes 104
Access Path 104
Logical Unit of Work 104
Isolation Level 104
Catalog Tables 106
Performance Indicator 106
Lock Escalation 107
Deadlock 108
Tuning Parameters 109
Performance Indicator 110
Recovery 110
Logical Units of Work 110
CMS Work Units (VM) 110
Checkpoints 110
What occurs during the Checkpoint Process? 111

When do Checkpoints Occur? . . . 111
Performance Implications . . . 111
Choosing the Checkpoint Interval . . . 111
Logging and Archiving . . . 113
Log . . . 113
Archive . . . 113
Dual Logs . . . 113
Choosing a Logmode . . . 114
Tuning Parameters . . . 115
Performance Indicator (SHOW LOG) . . . 116
Communications . . . 116
DRDA Performance Considerations (VM) . . . 116
PROTOCOL Performance Considerations . . . 116
Fetch and Insert Blocking . . . 118
Implementing Blocking . . . 118
Suppressed Blocking . . . 119
Synchronous Communications (VM) . . . 120
Considerations for ISQL and Adhoc Queries . . . 120
AUTOCOMMIT . . . 120
Isolation Levels . . . 121
Temporary Tables . . . 121
Views . . . 121
DBS Utility Considerations . . . 121
Automatic Statistics Collection . . . 121
Suppressing Automatic Statistics Collection . . . 122
Lock Escalation . . . 122
DATALOAD and RELOAD Locking Considerations . . . 122
SELECT, DATAUNLOAD, and UNLOAD Locking Considerations . . . 123
UNLOAD and RELOAD PACKAGE Considerations . . . 123

Chapter 5. Improving Data Access Performance . . . 125
Access Paths and Indexes . . . 125
Dbspace Scans . . . 126
Index Scans . . . 126
Nonselective Index Scans . . . 126
Selective Index Scans . . . 127
Index-Only Access Scans . . . 127
Clustering Index . . . 127
Examples of Index only Access . . . 127
Creating Indexes . . . 128
Unique Index with Key Matching Predicate(s) . . . 128
Indexes for Sorting . . . 128
Recommendations for Indexes . . . 129
Disadvantages of Indexes . . . 129
Placing Tables into Dbspaces . . . 129
Organizing Referential Structures . . . 130
Predicate Processing . . . 131
Column Attributes . . . 132
Key-matching Predicates . . . 132
Form of Key-matching Predicates . . . 132
Suitable Index for Key-matching Predicate . . . 133
Sargable and Residual Predicates . . . 134
Example . . . 134
Join Predicates . . . 134

Search Conditions and their Processing Characteristics 134
Filter Factors 140
Examples of Predicate Processing 140
Impact of CCSIDs on Sargability 141
Tuning Queries with Several Tables 142
Methods of Joining Two or More Tables 143
Nested Loop Join (Type 1) 143
Merge Scan Join (Type 2) 144
Choosing an Access Method 145
Nested Loop, with EMPLOYEE as Outer Table 145
Nested Loop, with PROJECT as Outer Table 145
Merge Scan, with EMPLOYEE as Outer Table 146
Merge Scan, with PROJECT as Outer Table 146
Multiple Joins 146
Keeping Database Statistics Current 147
Using Catalog Statistics 149
Modelling your Production System 149
A Warning about Updating Statistics 149
Determining the Cost of Access Methods 150
Processing Cost 150
I/O Cost 150
Using Explanation Tables to Evaluate Performance 151
Explain Processing 151
Using the EXPLAIN Statement 151
Using the EXPLAIN Option 153
Comparing Implicit and Explicit Explain Processing 153
Using Explanation Tables 154
Estimating Sizes of Responses 164
Using EXPLAIN for Database Design 164
Modifying Table Designs to Enhance Performance 165

Bibliography 167

Index 171

Notices

Trademarks

About This Manual

Who Should Use This Manual

This manual will help you analyze and tune the performance of the DB2® VSE & VM product in an IBM VM system or in VSE. It is designed for the person who designs or customizes any of the following:

- Operating systems that support the DB2 Server for VSE & VM product
- DB2 Server for VSE & VM application servers
- DB2 Server for VSE & VM databases
- DB2 Server for VSE & VM application programs

Organization

Before you can make effective judgements about how to tune the DB2 Server for VSE & VM product, you need to understand what happens inside each part of the product. This manual will help you understand:

- How each part of the DB2 Server for VSE & VM product works
- How each of those parts affects performance
- How to tune the performance of each part
- How to monitor how it is performing.

This manual does not provide diagnostic information. (For the symptoms of common performance problems and potential cures, refer to the *DB2 Server for VM Diagnosis Guide and Reference* or the *DB2 Server for VSE Diagnosis Guide and Reference* manuals.)

The chapters of this manual are arranged as follows:

Chapter 1, Improving Performance: This is an introduction to the subjects of performance design and tuning. It discusses the basic process including the development of goals, strategies and plans.

Chapter 2, Measuring Performance: This is an overview of the various tools available to measure the performance of the application server itself and as a part of the entire VM or VSE system.

Chapter 3, Managing Storage and Configuring the Operating System: This discusses how to effectively manage physical (DASD) and virtual storage. It also explains various operating system parameters and how to set them to optimize the performance of a system that includes a DB2 Server for VSE & VM application server.

Chapter 4, Configuring the Application Server and Requester: This explains the various subsystems in the application server and requester and how they can affect performance. It discusses how each initialization parameter governs how each subsystem operates, and where to look for performance indicators that describe how well each subsystem is performing.

Chapter 5, Improving Data Access Performance: This discusses how to improve performance by changing either how the data is accessed or by changing the structure of the data itself. The first method involves analyzing and rewriting SQL statements, while the second method involves reorganizing data, effectively managing indexes, and working with database statistics.

Prerequisite Reading

This manual assumes that you are familiar with at least one of the following two sets of IBM publications:

VM

- *DB2 Server for VM Application Programming*
- *DB2 Server for VM System Administration*
- *DB2 Server for VSE & VM Operation*
- *DB2 Server for VM Database Administration*
- *DB2 Server for VM Diagnosis Guide and Reference*
- *DB2 Server for VSE & VM SQL Reference*.

VSE

- *DB2 Server for VSE Application Programming*
- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE & VM Operation*
- *DB2 Server for VSE Database Administration*
- *DB2 Server for VSE Diagnosis Guide and Reference*
- *DB2 Server for VSE & VM SQL Reference*.

It also assumes you are familiar with IBM VM systems, CMS commands, and EXECs; or VSE, job control language, and CICS®.

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:

```
►►──SAVE──────────────────────────────────────────────────────────────►◄
```

  Others must be followed by one or more keywords or variables. For example:

```
►►──SET AUTOCOMMIT OFF────────────────────────────────────────────────►◄
```

- 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).

```
►►──DROP SYNONYM──synonym─────────────────────────────────────────────►◄
```

- 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:

```
►►──SHOW DBSPACE──integer─────────────────────────────────────────────►◄
```

  This command might appear as:

```
SHOW DBSPACE 1
```

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

```
►►──CREATE──┬────────┬──INDEX─────────────────────────────────────────►◄
            └─UNIQUE─┘
```

  This statement could appear as either:

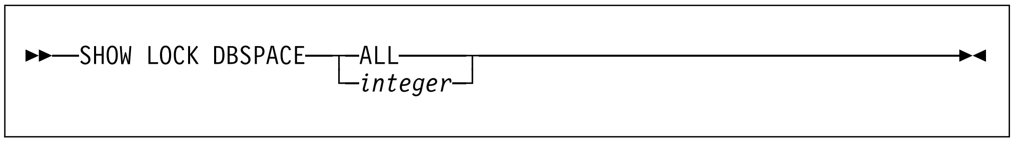
```
CREATE INDEX
```

  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:



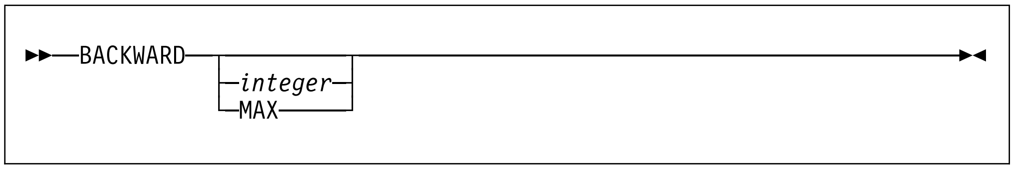
  Here, the command could be either:

```
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
```

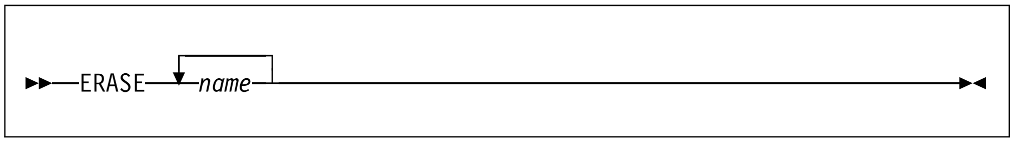
  or

```
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:

```
                   ┌─,─────────────────┐
►►──VALUES──(──▼──┬─constant──────────┬┴──)──────────────────────────►◄
                  ├─host_variable_list─┤
                  ├─NULL───────────────┤
                  └─special_register───┘
```

- 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.

```
    ┌─ASC──┐
►►──┼──────┼────────────────────────────────────────────────────►◄
    └─DESC─┘
```

- 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.

  elIn the following example, if you enter the optional keyword PCTFREE =, you also have to specify one of its associated optional parameters. If you do not enter PCTFREE =, the database manager will set it to the default value of 10.

```
    ┌─PCTFREE = 10──────┐
►►──┼───────────────────┼───────────────────────────────────────►◄
    └─PCTFREE = integer─┘
```

- 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:

```
              ┌─PRIVILEGES─┐
►►──REVOKE ALL──┴────────────┴──────────────────────────────────►◄
```

  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.

```
►►──┬────────────────────────────┬──┤ fieldproc_block ├──────────►◄
    └─NOT NULL──┬──────────────┬─┘
                ├─UNIQUE───────┤
                └─PRIMARY KEY──┘
```

fieldproc_block:

```
├──FIELDPROC──program_name──┬──────────────────────┬──────────────────────┤
                            │      ┌─,───────┐      │
                            └─(──▼─constant─┴──)─┘
```

SQL Reserved Words

The following words are reserved in the SQL language. They cannot be used in SQL statements except for their defined meaning in the SQL syntax or as host variables, preceded by a colon.

In particular, they cannot be used as names for tables, indexes, columns, views, or dbspaces unless they are enclosed in double quotation marks (").

ACQUIRE
ADD
ALL
ALTER
AND
ANY
AS
ASC
AVG

BETWEEN
BY

CHAR
CHARACTER
COLUMN
COMMENT
COMMIT
CONCAT
CONNECT
COUNT
CREATE
CURRENT

DBA
DBSPACE
DELETE
DESC
DISTINCT
DOUBLE
DROP

EXCLUSIVE
EXECUTE
EXISTS
EXPLAIN

FIELDPROC
FOR
FROM

GRANT
GRAPHIC
GROUP

HAVING

IDENTIFIED
IN
INDEX
INSERT
INTO
IS

LIKE
LOCK
LONG

MAX
MIN
MODE

NAMED
NHEADER
NOT
NULL

OF
ON
OPTION
OR
ORDER

PACKAGE
PAGE
PAGES
PCTFREE
PCTINDEX
PRIVATE
PRIVILEGES
PROGRAM
PUBLIC

RESOURCE
REVOKE
ROLLBACK
ROW
RUN

SCHEDULE
SELECT
SET
SHARE
SOME
STATISTICS
STORPOOL
SUM
SYNONYM

TABLE
TO

UNION
UNIQUE
UPDATE
USER

VALUES
VIEW

WHERE
WITH
WORK

Conventions Used for Highlighting Examples

Sample commands and messages are provided throughout this manual. While you will not see highlighting on your screen, it is included in this manual for emphasis:

- Commands are **`highlighted using bold type.`**
- Messages are not highlighted
- Important parts of some messages are emphasized with underlining.

For example:

```
set pool 1 seq
ARI0065I Operator command processing is complete.
show pool 1

POOL NO.  1:     NUMBER OF EXTENTS = 2  DS3 SEQ

EXTENT   TOTAL   NO. OF     NO. OF     NO. OF     %
 NO.     PAGES PAGES USED FREE PAGES RESV PAGES USED
   1       855         74       781              8
   2       855         47       808              5
TOTAL     1710        121      1589         20    7
ARI0065I Operator command processing is complete.
```

Summary of Changes for DB2 Version 6 Release 1

This is a summary of the technical changes to the DB2 Server for VSE & VM Version 6 Release 1 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 DB2 Server for VSE & VM product; see either the *DB2 Server for VSE & VM SQL Reference*, *DB2 Server for VM System Administration*, or the *DB2 Server for VSE System Administration* manuals for a discussion of incompatibilities. Version 6 Release 1 of the DB2 Server for VSE & VM database management system is intended to run on the Virtual Machine/Enterprise Systems Architecture (VM/ESA®) Version 2 Release 2 or later environment and on the Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA™) Version 2 Release 2 or later environment.

Enhancements, New Functions, and New Capabilities

DRDA® RUOW Application Requestor for VSE (Online)

DRDA Remote Unit of Work Application Requestor provides read and update capability in one location in a single unit of work.

This support provides CICS/VSE® online application programs with the ability to execute SQL statements to access and manipulate data managed by any remote application server that implements the DRDA architecture. Online application programs that access remote application servers need to be preprocessed to create a bind file and then bound (using CBND) to the remote application server. Online application programs that access a local application server are preprocessed as in previous releases.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE Database Administration*
- *DB2 Server for VSE Application Programming*
- *DB2 Server for VSE Installation*

Stored Procedures

The ability to use stored procedures provides distributed solutions that let more people access data faster.

A stored procedure is a user-written application program compiled and stored at the server. When the database is running in multiple user mode, local applications or remote DRDA applications can invoke the stored procedure. SQL statements are local to the server and issued by a stored procedure so they do not incur the high network costs of distributed statements. Instead, a single network send and receive operation is used to invoke a series of SQL statements contained in a stored procedure.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VM System Administration*
- *DB2 Server for VM Database Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE & VM Operation*

TCP/IP Support for DB2 Server for VM

TCP/IP support allows:

- VM applications to use SQLDS-private protocol to connect to VM databases over TCP/IP.
- VM applications to use DRDA protocol to connect to DB2 family databases (and any other database that supports DRDA connections) over TCP/IP.
- non-VM applications to use DRDA-protocol to access VM database over TCP/IP.

TCP/IP support for DB2 Server for VM integrated with the DB2 Server for VM application server means a system easier to configure and maintain.

The database manager will optionally secure TCP/IP connections using any external security manager that supports the RACROUTE interface.

New Code Page and Euro Symbol Code Page Support

The following CCSIDs are now supported:

- 1112: Latvian/Lithuanian
- 1122: Estonian
- 1123: Ukrainian
- 1130: Vietnamese
- 1132: Lao
- 1148: E-International
- 1140: E-English
- 1141: E-German
- 1144: E-Italian
- 1147: E-French

Additional support has been added for conversions from Unicode (UCS-2) to host CCSIDs.

For a complete list of CCSIDs supported refer to the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals.

DataPropagator™ Capture

DataPropagator Capture is part of the DB2 Family of DataPropagator products. DataPropagator Capture is updated for Version 6 Release 1 compatibility.

QMF for VM, QMF for VSE, and QMF for Windows®

IBM Query Management Facility (QMF™) is now an separately priced feature of DB2 Server for VSE & VM. QMF is a tightly integrated, powerful, and reliable tool that performs query and reporting for IBM's DB2 relational database Management System Family. It offers an easy-to-learn, interactive interface. Users with little or no data processing experience can easily retrieve, create, update, insert, or delete data that is stored in DB2.

QMF offers a total solution that includes accessing large amounts of data and sharing central repositories of queries and enterprise reports. It also allows you to implement tightly-controlled, distributed, or client-server solutions. In addition, you can use QMF to publish reports to the World Wide Web that you can view with your favorite web browser.

Using QMF, users can access a wide variety of data sources, including operational or warehouse data from many platforms: DB2 for VSE, VM, OS/390® and Windows. Via IBM Data Joiner, you can access non-relational data, such as IMS™ and VSAM, as well as data from other vendor platforms.

RDS Above the Line

The RDS component will load and execute above the 16 megabyte line. This support frees up approximately 1.5 megabytes of storage below the 16 megabyte line (or approximately 2.5 megabytes, if DRDA is installed) when compared to Version 5 Release 1. No installation or migration changes are required for this support to be used (except for the definition of VM Shared Segments and for users who execute the database server with AMODE(24)). If sufficient storage is available, the RDS component will be automatically loaded above the 16 megabyte line. When using VM Shared Segments, the RDS Segment should be defined above the 16 megabyte line.

VM users who wish to run the database server in 24-bit addressing mode (i.e. use the AMODE(24) parameter) **must** use a virtual storage size no greater than 16 megabytes. See the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* for release to release incompatibility information.

Combining of NLS Feature Installation Tapes with Base Product Installation Tape

All available NLS features for DB2 Server for VSE, DB2 Server for VM, Control Center for VSE and REXX SQL for VM have been combined with the respective base product installation tape. Customers interested in an NLS feature language will no longer need to order an additional NLS feature tape because all NLS languages will be available to all customers. In all cases, the default language as shipped is American English. The installation and migration processes have been changed to allow you to choose the default language. Refer to the *DB2 Server for VM Program Directory*, *DB2 Server for VSE Installation*, *DB2 for VSE Control Center Installation and Operations Guide*, and *DB2 REXX SQL for VM/ESA Installation* for the details of how these changes affect the installation process and how you can choose to have a different default language.

Control Center Feature

DB2 Server for VSE & VM Version 6 Release 1 enhances the new Control Center feature as follows:

For both VM/ESA and VSE/ESA:

- Access to the Query Management Facility (QMF)

For VM/ESA:

- Compatibility with DB2 Server for VM Version 6 Release 1 initialization parameters and operator commands
- Shared File System Support (SFS) in a VM/ESA environment
- CA-DYNAM/T Interface Support in a VM/ESA environment
- Data Restore Incremental Backup Support in a VM/ESA environment

For VSE/ESA:

- Control Center code installation on any library
- Ability to use while viewing a list of tables online
- Ability to create, reorganize, unload, reload, move and copy tables in batch mode
- Ability to update table statistics in batch mode
- Ability to drop tables online

Data Restore Feature

The Data Restore feature provides archiving and recovery functions in addition to those provided in DB2 for VSE & VM. Data Restore is enhanced in Version 6 Release 1 with incremental database archiving support. The support allows you to archive only the areas of the database that have been updated since the last database archive, instead of having to archive the entire database. This can provide significant savings for customers with large databases which are updated infrequently, or where only a small fraction of the database is updated frequently.

DB2 REXX SQL Feature

The DB2 REXX SQL feature provides a REXX interface for VM customers to allow SQL calls to be executed from REXX programs. The DB2 REXX SQL feature is updated for Version 6 Release 1 compatibility.

Reliability, Availability, and Serviceability Improvements

Migration Considerations

Migration is supported from SQL/DS™ Version 3 and DB2 Server for VSE & VM Version 5. Migration from SQL/DS Version 2 Release 2 or earlier releases is not supported. Refer to the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual for migration considerations.

Library Enhancements

Some general library enhancements include:

- The following books have been removed from the library:
  - *DB2 Server for VM Operation*
  - *DB2 Server for VSE Operation*
  - *DB2 Server for VM Interactive SQL Guide and Reference*
  - *DB2 Server for VSE Interactive SQL Guide and Reference*
  - *DB2 Server for VM Database Services Utility*
  - *DB2 Server for VSE Database Services Utility*
- The following books have been added to the library:
  - *DB2 Server for VSE & VM Operation*
  - *DB2 Server for VSE & VM Interactive SQL Guide and Reference*
  - *DB2 Server for VSE & VM Database Services Utility*

Refer to the new *DB2 Server for VSE & VM Overview* for a better understanding of the benefits DB2 Server for VSE & VM can provide.

Chapter 1. Improving Performance

Elements of Performance

Performance is the way a computer system behaves given a particular *workload*. It can be measured through the system's response time, throughput, and availability; and it is affected by:

- The resources available
- How well they are used and shared

In general, you should undertake performance tuning when you want to improve the cost-benefit ratio of your system. Specific goals would be:

- To process a larger or more demanding work load without increasing processing costs. (For example, without buying new hardware or using more processor time.)
- To obtain faster system response or higher throughput without increasing processing costs.
- To reduce processing costs without affecting service to your users.

Translating performance from technical terms to economic terms is difficult. Performance tuning certainly costs money (through people's time and through processor time), so before you undertake a tuning project, weigh its costs against its possible benefits. Some of these benefits are tangible, such as more efficient use of resources and the ability to add more users to the system, others such as greater user satisfaction because of quicker response time, are intangible. All of these benefits must be considered.

Tuning Guidelines

The following guidelines should help you develop an overall approach to performance tuning.

Remember the Law of Diminishing Returns: Your greatest performance benefits usually come from your initial efforts. Further changes generally produce smaller and smaller benefits and require more and more effort.

Do Not Tune Just for the Sake of Tuning: Tune to relieve identified constraints. If you tune resources that are not the primary cause of performance problems, this has little or no effect on response time until you have relieved the major constraints, and it can actually make subsequent tuning work more difficult. If there is any significant improvement potential, it lies in improving the performance of the resources that are major factors in the response time.

Consider the Whole System: You can never tune one parameter or system in isolation. Before you make any adjustments, consider how it will affect the system as a whole.

Change one Parameter at a Time: Do not change more than one performance tuning parameter at a time. Even if you are sure that all the changes will be beneficial, you will have no way of evaluating how much each change contributed. You also cannot effectively judge the trade-off you have made by changing each parameter. Every time you adjust a parameter to improve one area, you almost always affect at least one other area.

Measure and Reconfigure by Levels: For the same reasons that you should only change one parameter at a time, tune one level of your system at a time. You can use the following list as a guide:

- Hardware
- Operating System (VM or VSE)
- CICS (for VSE)
- Application Server and Requester
- Database
- SQL Statement
- Application Program

Check for Hardware and Software Problems: Some performance problems may be corrected by applying service, either to your hardware, through an engineering change (EC) or microcode assists, or to your software through a program temporary fix (PTF). Do not spend excessive time monitoring and tuning your system, when simply applying service may make it unnecessary.

Understand the Problem Before you Upgrade your Hardware: Even if it seems that additional storage or processor power could immediately improve performance, take the time to understand where your bottlenecks are. You may spend the money on additional DASD only to find that you do not have the processing power or the channels to exploit it.

Put Fallback Procedures in Place Before You Start Tuning: As noted earlier, some tuning can cause unexpected performance results. If this leads to poorer performance, it should be reversed and alternative tuning tried. If the former setup is saved in such a manner that it can be recalled, the backing out of the incorrect change becomes much simpler.

Performance Improvement Process

Use the following process to improve the performance of any system:

1. Establish performance indicators.
2. Define performance objectives.
3. Develop a performance monitoring plan.
4. Carry out the plan.
5. Analyze your measurements to determine whether you have met your objectives. If you have, consider reducing the number of measurements you make. Performance monitoring itself uses system resources. Otherwise continue with Step 6.
6. Determine the major constraints in the system.
7. Decide where you can afford to make trade-offs and which resources can bear an additional load. (Nearly all tuning involves trade-offs among system resources and the various elements of performance.)

8. Adjust the configuration of your system. If you think that it is feasible to change more than one tuning option, implement one at a time. If there are no options left at any level, you have reached the limits of your resources and need to upgrade your hardware.
9. Return to Step 4 above and continue to monitor your system.

Periodically, or after significant changes to your system or workload:

- Return to Step 1 above
- Reexamine your objectives and indicators
- Refine your monitoring and tuning strategy.

How Much Can a System be Tuned?

There are limits to how much you can improve the efficiency of a system. Consider how much time and money you should spend on improving system performance, and how much the spending of additional time and money will help the users of the system.

Your system may perform adequately without any tuning at all, but it probably will not perform to its potential. Unfortunately using the default tuning parameters is usually not a good solution. Each database is unique. As soon as you develop your own database, and applications to use it, investigate the tuning parameters available and learn how you can customize their settings to reflect your situation. In some circumstances, there will only be a small benefit from tuning a system, however in most, the benefit may be significant.

As your system approaches a performance bottleneck, it is more likely that tuning will be effective. If you are close to this and you increase the number of users on the system by, say, 10 percent, the response time is likely to rise by much more than 10 percent. However, there is a point beyond which tuning cannot help you. At that point, the only thing to do (other than adding new hardware) is to change your objectives.

Workload

When devising a strategy to improve performance, you need to consider the workload in two environments: test, and production. Ideally you should have access to both, but often tuning must be done without the benefit of a test system.

Test Workload: In a test environment you can use strictly defined workloads to model how changes to performance parameters may affect your production system. Consider modeling your production system with a small subset of transactions from it. By running a wide variety of SQL statements from different applications you can create a rough sketch of your system. While it may not perform exactly the way the production system will, it can help you discover unexpected effects before they occur in production.

Production Workload: In contrast, you probably do not have a great deal of control over the size and nature of the workload in your production environment—you can measure it, looking for maximums, minimums, averages, and variances over time, but it is almost impossible to accurately predict exactly what it will be. Instead look for trends that can help you predict future capacity requirements. For example, will you need to buy hardware or invest in additional performance tuning to support a rapidly growing workload.

Performance Indicators

The first rule of control engineering is:

If you cannot measure it, you cannot control it .

When you set your performance objectives, take a practical look at what you can measure. While you may want to establish a high-level throughput objective of, say, “55 transactions per second” there may not be an easy way to measure this. Instead, consider using an indicator that is readily available. For example, the BEGINLUW counter records how may logical units of work (LUWs) started during the last monitoring period. While this does not actually represent the number of transactions per second, it does act as a rough indicator of throughput.

Establishing Performance Objectives

How you define good performance depends on your particular needs and priorities. Performance objectives should be realistic, in line with your budget, understandable, and measurable.

Response Time

Response time represents the elapsed time between when a user submits an SQL request to a server (usually through an application program), and when the response arrives on the user's screen. It can also represent the elapsed time required to respond to an SQL request submitted from a batch application program.

The easiest response time objective to state is a maximum time, such as, “SQL queries will return in under 2 seconds.” However, response times can vary for many reasons. So, include acceptable tolerances in your targets. For example, “SQL queries will return in under 2 seconds 80% of the time.” This allows for unusual transactions that have exceptionally heavy processing or database access requirements.

Components of Response Time

Response time for any database transaction has three components. The SQL statement is generated in an application program; it travels through a network to an application server; and finally, the server generates a response, which is returned to the program through the network.

Application server response time represents the time it takes for the server to interpret your request and retrieve or update data. This can be affected not only by how well your database and SQL statements are designed, but also by how well the server's initialization parameters are tuned.

Network response time represents the communication delay between the application program and the application server. It also represents any delay between a user's terminal and the application program. This usually does not represent a large part of overall response time, unless your server and application program are physically separated by large distances.

Application program response time can often be the fastest part of the process, but do not overlook it. Some programs can take more time to process data than was required to retrieve this data from the server. For example, if you retrieve floating-point data that needs to be displayed in scientific notation, your program may take longer to perform the conversion than it took for the server to generate the answer set. Therefore, do not assume that if you double the speed of your server, your end users will see their response time cut in half.

| You can also use a stored procedure (user-written application program that is compiled and stored at the server) to eliminate many of the network send and receive operations, and thereby reduce network cost of distributed database access. For more information on stored procedures, see the *DB2 Server for VM Database Administration* manual.

Throughput

Throughput measures the amount of work processed over a period of time (refer to "Workload" on page 3). You can measure it either in a controlled test system or in a production system.

In a test system you are able to define representative workloads and measure how many of these transactions your system can complete per unit of time. For example, you can measure the number of transactions per second.

In a production environment, look for measurements that are effective as averages over time that will give you a rough indicator of throughput of your overall system, such as BEGINLUW. Also look at the throughput of your various subsystems — for example, the pages processed per second by your DASD I/O system.

Availability

Availability is a measure of the proportion of time a system or resource is ready when it is required.

It is usually measured in hours, weeks, or months. For example, you may want to set an objective of 8 hours downtime (time when your server is unavailable) per month, based on a 24-hour day. Downtime is not necessarily caused by a malfunction. You may need to shut the application server down to apply service or perform maintenance.

A Less Formal Approach

If you do not have enough time to set performance objectives and to monitor and tune in a comprehensive manner, you can address performance by listening to your users. Find out if they are having performance-related problems. You can usually locate the problem, or at least where to start, by asking a few simple questions. For example:

- What do you mean by "slow response"? Is it 10% slower than you expect it to be, or tens of times slower?
- When did you notice the problem? Is it recent or has it always been there?
- How many users are complaining? Is it just one or two individuals, or a whole group?
- If a whole group of users are experiencing difficulties, are they connected to the same terminal controller?

- Are the problems related to a specific transaction or application program?
- Do the problems appear during regular periods, such as lunch hour, or are they continuous?

Monitoring Performance

Performance monitoring can help you understand how the various parts of your overall computer system are working. There are two types:

Real time
You can monitor the immediate state of your system to solve problems such as locking contention or storage shortages.

Statistical
You can also monitor the performance of a system over a period of time to help you tune the parameters of the system or plan for future capacity requirements.

Creating a Monitoring Plan

You need to plan how you will monitor your system and how you will analyze the data that results. When you create your plan, do the following:

- Create a master schedule of monitoring. Large batch jobs or maintenance runs can cause peaks in activity. Coordinate monitoring with other operations so that they do not conflict with unusual peaks, unless that is what you want to monitor.
- Determine the kinds of analysis that you will perform and the tools that you will use. Document the data that you will extract from each monitoring tool. Some of these tools provide reports that help to organize data, but in addition you should create worksheets or utility programs to help you extract and organize the performance indicators specific to your system.
- Create a list of people who should review the results of your monitoring. These results should be summarized and shared with everyone involved with your system. Consider including application programmers, operators, and your end users.
- Determine standards and criteria for implementing changes in system parameters and workload. Describe how often you will permit changes, and outline a strategy to monitor their effects.

Monitoring Interval

An important factor affecting the accuracy of your performance measurements is the monitoring interval. Most useful performance values, whether measured directly or calculated from other measures, are averages over time.

If the interval that you use to calculate this average is **too long**, you may lose significant values. For example, you would not see a 10-minute peak in DASD paging load or a 10-minute drop in the effective use of the local buffers if you only look at your performance indicators once a day.

If the interval is **too short**, your results may not be statistically valid. For example, if one checkpoint occurred during one 30-minute interval, you could not confidently say that the database manager was performing two checkpoints per hour.

Cost of Monitoring

You need to weigh the benefits of making performance measurements against the additional overhead involved. While recording performance numbers every 10 seconds may give you an excellent picture of how your database manager is working, the additional load on the operating system may reduce your overall performance, or consume large amounts of DASD space.

Measurements

Performance measurements are relative: they tell how a system behaves for a particular workload. A system is considered to perform well if it can complete a particular workload faster than other systems or with fewer resources.

In a test system, you can control the workload by running the same tasks many times. During each iteration you can measure how fast your system completed the tasks and how much resource it used.

However, in a production system it is difficult to compare measurements taken at different times, because the workload is constantly changing. To obtain a performance measurement, you must compare the *average* performance of your system measured over a period of time to the workload it processed during that time. To make these comparisons, you need to calculate two types of relative measurements: load and performance.

Load is usually measured as a rate, in tasks per unit of time. These measurements help you determine the amount of work that the database manager or the operating system is performing during a period of time. High load values in some areas and low load in others may suggest a bottleneck in the system. Also, while similar load measurements do not guarantee that two workloads are comparable, different ones show that they are not comparable.

Performance can be measured as a percentage from 0% to 100%, where 100% is optimal. For practical reasons it is often calculated by comparing the number of successes compared to the number of attempts. For example, if the database manager looks in the local buffers for a page 100 times, and finds the page it is looking for 75 times, the local buffers are 75% effective. This measurement helps you estimate how effectively various components of the entire system are performing.

The percentage can be calculated in several ways:

$$\frac{\text{Success}}{\text{Attempt}} \times 100 = \frac{\text{Attempt - Failure}}{\text{Attempt}} \times 100 = \left[1 - \frac{\text{Failure}}{\text{Attempt}}\right] \times 100$$

You can also express a performance measurement as a *hit ratio*, with the following calculation:

$$\frac{\text{Attempt}}{\text{Failure}}$$

In this case, the higher the ratio the better the performance. The lowest value for a hit ratio is 1.

Use the formula that makes the most “sense” to you. Some formulas fit some measurements better or are easier to understand than others. Mathematically, they are all equivalent.

Tools

A wide range of tools for monitoring performance is available in both VM and in VSE. Each tool covers a particular area or a different level of the overall system.

VM Tools

The **CP Monitor** subsystem measures the performance of the VM operating system and its resources and the **VM/Performance Reporting Facility (VM/PRF)** product creates usage and historical reports from those measurements. You can control the amount and nature of the data collection, based on the analysis you want to do. To create reports from the collected data, you must either do some programming, or you can use VM/PRF to produce standard reports. This facility contains reports helpful in monitoring the overall DASD I/O performance of your database. The CP Monitor subsystem is included with the VM system. VM/PRF is available from IBM.

The **CP INDICATE USER and QUERY TIME** commands measure the resources consumed by your database virtual machine. Includes measurements of system paging use, database manager DASD I/O, and CPU load. Included with VM as a part of CP. (Refer to page 16.)

The **Real Time Monitor VM/ESA (RTM VM/ESA)** provides on-line performance monitoring. Data is typically gathered in short intervals, usually one to three minutes.

You can use this tool to capture system level data about your system and the database machine. It is available from IBM.

VSE Tools

The **VSE Interactive Interface** contains information about CPU use, system paging, active users, channel and device activity, storage layout, and system activity. Each is presented in a separate dialog. It is included with VSE. Refer to the *DB2 Server for VSE & VM Operation* manual.

VSAM LISTCAT provides information on the location of VSAM data sets. It is provided with VSE. (Refer to the *DB2 Server for VSE & VM Operation* manual.)

CICS Tools

The **CICS Monitoring Facility** measures the performance of CICS under VSE and **CICSPARS/VSE** creates historical reports. Both are available from IBM. (Refer to page 18.)

The **CIRD transaction** displays a snapshot of the links between CICS and your application server. It is provided with the DB2 Server for VSE & VM base product. (Refer to page 22.)

The **CICS statistics** facility gathers statistical data on CICS performance. It is provided with CICS. (Refer to the *CICS/VSE Performance Guide* manual.)

DB2 Server for VSE & VM Tools

As well as the tools described below, the *DB2 Family Solutions Directory* manual contains descriptions and ordering information for a wide variety of performance monitoring and tuning tools. These tools are available from a number of companies including IBM and are included under the section heading “Database Administration Tools.”

Whenever the application server starts, it displays how its **Initialization Parameters** are set. These parameters describe how the server has been configured. It is included with the DB2 Server for VSE & VM base product. (Refer to page 21.)

The **DB2 Server for VSE & VM system catalog** contains information about the dbspaces, tables, indexes, keys, packages, authorities, and other objects in the database. Much of the information is used by the database manager when it decides how to retrieve data from the database. It is included with the DB2 Server for VSE & VM base product. (Refer to page 38.)

The **SHOW operator commands** are available which display the status of the application server. For example, user activity, locking, log usage, and storage can all be monitored with these commands. It is included with the DB2 Server for VSE & VM base product. (Refer to page 25.)

The **COUNTER operator command** measures the performance of your application server by recording how often significant events occur in the database manager. These events relate to workload, locking, and database manager storage (buffer pools). It is included with the DB2 Server for VSE & VM base product. (Refer to page 23.)

IBM DB2 for VM Control Center automates DBA functions such as archiving, recovery, adding dbextents, deleting dbextents, adding dbspaces, startup, shutdown, startup parameter changing, dbspace reorganizations, catalog index reorganizations, and database monitoring. Any of these functions may be initiated immediately by an automated user (local or remote), or they may be scheduled to execute at any specified date and time, or repetitive execution interval. It is available from IBM and only for use with the DB2 Server for VSE & VM product running in an IBM VM system.

The **DB2 Server for VSE & VM accounting** facility records how much CPU time is consumed and how many buffer pool looks were done during the time that a user is signed onto the application server. The **DB2 Server for VSE & VM trace** facility records the sequence of events that occur in different components of the database manager (for example, you could trace the sequence of locks that lead up to a deadlock). While both these tools can be extremely useful in diagnosing performance problems, use them very sparingly. Both consume a great deal of system resources and can actually severely affect overall performance when they have been turned on. For more information on the accounting facility, refer to the *DB2 Server for VSE System Administration* or the *DB2 Server for VM System Administration* manuals. For more information on the trace facility, refer to the *DB2 Server for VSE & VM Operation* manual.

The **DB2 Server DSS SHOW TARGETWS operator command** measures the amount of main and expanded storage your database machine is currently using. It is included with the DB2 Server DSS Feature. (Refer to the *DB2 Data Spaces Support* manual.)

The **DB2 Server DSS COUNTER POOL operator command** measures the performance of individual storage pools, internal dbspaces, and the directory. It is included with the DB2 Server DSS Feature. (Refer to the *DB2 Data Spaces Support* manual.)

Factors Affecting Performance

Resources

Processor

The processor (sometimes referred to as the CPU) is generally the most expensive resource in a system. As such, they should be used as efficiently and fully as possible. In a highly-utilized, well-tuned system, the processor is in use at least 80% of the time. If yours is already above that level, you must either upgrade your processor or find a more efficient way to do the job. For example, rewrite your application program, or investigate the structure of your data or SQL statements. Refer to Chapter 5, “Improving Data Access Performance” on page 125.

Storage

Real and Virtual Storage: Your system's performance is directly affected by how well the database manager and your operating system share a common pool of storage between different processes.

For example, agent structures, buffer pools, locks, and packages all require storage. In general, the more storage allocated to a specific component, the faster it will perform (within limits). However, you can only allocate storage from the limited amount available in your database machine or partition. You need to trade-off the requirements of each component in order to balance the entire system.

For example, if DASD I/O is a performance bottleneck during regular operation and locking is not, consider using less storage for locks and more for the DASD buffer pools. For more information, refer to “Real and Virtual Storage” on page 43. (This is a good example of how performance issues interrelate. By increasing the number of buffers in the pool you decrease your DASD I/O during regular operation, but increase it during checkpoint processing. If checkpoint processing was a problem you have just made it worse. Refer to “Choosing the Checkpoint Interval” on page 111.)

The DASD I/O System: The database manager moves data to and from DASD as required. How efficiently it does that has a significant impact on the overall performance of your application server. How much real storage is available, the size of the buffer pools, and how often a checkpoint is performed all determine how often the database manager needs to move data between itself and DASD. You can also improve the performance of the DASD I/O system by using DASD caching, DB2 Data Spaces Support or Virtual Disk support. Refer to “Database Manager Storage” on page 89.

DASD Storage: How you manage DASD storage affects performance in four ways:

Dividing DASD
How you divide a limited amount of storage between indexes and data, and among dbspaces and among storage pools determines to a large degree how each will perform in different situations.

Wasting DASD
Wasted storage in itself may not affect the performance of the system that is using it, but it may represent a resource that could be used to improve performance elsewhere.

Distributing DASD I/O
How well you balance the demand for DASD I/O across multiple DASD devices, controllers and channels can affect how fast the database manager can retrieve information from DASD, refer to “DASD Balancing” on page 79.

Running out of DASD
While running out of storage can disrupt your users and you are forced to bring down the application server to add storage, just getting close can degrade performance. (If you reach the application server's short on storage level you trigger unnecessary SOSLEVEL checkpoints, refer to “Short on Storage Cushion” on page 62.)

For more information, refer to “DASD Storage” on page 61.

Overhead

Concurrency

The database manager uses agents and pseudo agents to allow concurrent use of its resources. It uses agent structures to divide processor time between multiple users and its own internal tasks, such as checkpoint processing and operator commands. The number of agents available, combined with how the agents are scheduled and dispatched can affect the overall performance of your system. For more information, refer to “Concurrency” on page 95.

Your operating system must also divide processor time among multiple applications (your application server being one). If the operating system favors your server and gives it more than its even share of time, your server may perform well, but at the expense of other applications. For VM, refer to “Fair Share Scheduling” on page 82. For VSE, refer to “Dispatching Priority” on page 83.

Locking

In multiple user mode (MUM), several agents may need to access the same data at the same time. This poses a problem if one agent tries to change data while another agent is still looking at it.

Consider two application programs, each trying to add ten dollars to the same account at the same time. Both programs read the account balance at the same time. They both see 100 dollars in the account. The first program updates the value in the account with 110 dollars, the second program does likewise. The problem is that when both programs are finished there is only 110 dollars in the account instead of 120.

To avoid this problem, the database manager can lock the account as soon as the first program looks at it and hold the lock until the program is finished updating the balance. The second program waits until the first is complete.

Performance Implications: Of course while locking protects your data, there is a performance cost. Not only can waiting for locks increase response time (locks can last to the end of a logical unit of work), but each lock requires additional storage and processing time. Refer to "Locking Contention" on page 101.

Also, because there are a set number of potential locks defined at initialization time, you may run out. You may need more than were originally defined. If this happens, locks will be *escalated,* (refer to "Lock Escalation" on page 107) a process that requires additional storage and processor time.

Deadlocks (refer to "Deadlock" on page 108) can also be a problem. While the database manager detects deadlocks before they occur, the more potential deadlock situations that you create the more resources are required to avoid them.

Recovery

Maintaining the integrity of your data means preventing its accidental or intentional destruction, alteration, or loss. If your data is ever affected, there are three systems to ensure that you can recover it.

Checkpoint Processing
A checkpoint ensures that any modifications to your database, which are temporarily stored in main storage, are written to DASD. This ensures that the integrity of your database is protected even if your application server crashes, refer to "Checkpoints" on page 110.

Logging
A log is a file maintained on DASD that records the old and new values each time a change is made in your database. If you lose any changes because of a system failure, you can use the log to undo or redo the changes and restore the data to its original state.

Archiving
A database archive is a copy of the entire database. A log archive is an archive, or series of archives of the log. In the case of a serious failure you can restore the database archive, and instruct the database manager to redo any of the changes recorded in the log archive.

For information on both logging and archiving, refer to "Logging and Archiving" on page 113.

Choosing Between Tuning Trade-offs

The art of tuning is finding and removing constraints. In most systems, performance is limited by a single constraint. However, removing that constraint, while improving performance, inevitably reveals a different constraint, and you often have to remove a series of constraints. Because tuning generally involves decreasing the load on one resource at the expense of increasing the load on a different resource, relieving one constraint always creates another. A system will always be constrained.

When you choose to remove a constraint, consider which resources can accept an additional load in the system without themselves becoming worse constraints. Tuning usually involves a variety of actions that can be taken, each with its own trade-off.

Chapter 2. Measuring Performance

This chapter discusses some basic performance measurements you need to make at the operating system level. It also includes descriptions of several basic measurement tools included with the DB2 Server for VSE & VM product.

Operating System Measurements

There are a wide variety of tools available to measure the performance of your operating system, some of which are included in "VSE Tools" on page 8, and "VM Tools" on page 8. When you look for performance measurements in those tools, focus on three questions. How well is the system performing as a whole? How well is your database machine or partition performing? How is the database machine or partition affecting the performance of other processes that are running at the same time? With that in mind, consider the following generic measurements:

Processor (CPU) Load

Measure the overall percent utilization of your processor (CPU), refer to "Processor" on page 10. You also need to measure the percentage of the total CPU time devoted to the database machine or partition, refer to "Concurrency" on page 11.

Real and Virtual Storage Load

Measure the number of virtual pages in your database machine or partition that have been allocated real storage. Break the real pages into main, and auxiliary pages (and in the case of VM, expanded storage pages). Refer to "Real and Virtual Storage" on page 43. Also compare the number of virtual pages that have been allocated above the 16MB line to those below, refer to "Storage Above 16MB (31 Bit Addressing)" on page 47.

System Paging DASD Load

Measure the rate of DASD I/O to and from auxiliary storage, refer to "Auxiliary Storage" on page 43. Pay special attention to I/O to and from system paging DASD. This is the slowest type of auxiliary storage and the largest drain on performance. Also, compare the overall system paging DASD load to that required by the database machine or partition.

Machine or Partition DASD I/O Load

Measure the rate of DASD I/O initiated by the database machine or the partition itself, refer to "Database I/O" on page 89. The database manager directs the operating system to write and read pages to and from its data, directory, log, and archive disks or datasets. These I/Os are independent of system paging DASD and are always measured separately.

Individual Device Utilization

This includes individual DASD volumes, channels, and controllers. Measure the percentage of time that these individual devices are busy. This is more effective than using a load measurement because it takes into account the capability of the device itself. Refer to "DASD Balancing" on page 79.

Translating Performance Measurements to Indicators

The following is a description of the CP INDICATE USER and QUERY TIME commands included with the VM operating systems. It serves as an example of how to extract performance counters and simple measurements and translate them into useful indicators.

CP INDICATE USER and QUERY TIME Commands

These two commands enables you to monitor the overall performance of your database machine. The most important indicators they provide are:

RES=*nnnn*
: Counts the number of virtual machine pages that are currently in main storage. Convert the number of pages into bytes by multiplying the value by 4096 (bytes per page).

READS=*nnnnnn*
: Counts the total number of pages moved from system paging DASD to main storage for a virtual machine since it was logged on. (Refer to “Auxiliary Storage” on page 43.)

WRITES=*nnnnnn*
: Counts the total number of pages moved from main storage to system paging DASD for a virtual machine since it was logged on. (Refer to “Auxiliary Storage” on page 43.)

CONNECT=*hh***:***mm***:***ss*
: Records the total elapsed time the virtual machine was logged on the system.

VIRTCPU=*mmm***:***ss*
: Records the total virtual machine processor time used since the virtual machine was logged on.

TOTCPU=*mmm***:***ss*
: Records the total virtual machine processor time plus the total CP processor time used (virtual plus overhead) since the virtual machine was logged on.

IO=*nnnnnn*
: Records the total number of I/O requests issued by the machine since it was logged on. This includes all I/Os started by the DASD I/O system, refer to “Database I/O” on page 89.

  Note: Several IUCV *BLOCKIO requests may be blocked together to form a single IO request. This count includes all the IO requests, it does not count each page or block moved.

  The IO value will not equal the DASDIO counter.

These commands are only really useful when you use them together. To issue QUERY TIME and INDICATE USER together, type the following from the operator console:

```
#CP QUERY TIME #CP INDICATE USER
```

Note: The # symbol is the default escape character. It may be different depending on how your system has been customized.

You also need to compare two consecutive commands. For example, consider the following two QUERY TIME and INDICATE USER commands:

```
#cp query time #cp indicate user
CP QUERY TIME
CP INDICATE USER
TIME IS 15:07:20 EST TUESDAY 02/14/97
CONNECT= 01:21:45 VIRTCPU= 000:06.28 TOTCPU= 000:09.86
USERID=SQLDBA   MACH=XC  STOR=0009M VIRT=V XSTORE=NONE
IPLSYS=CMS      DEVNUM=00031
PAGES: RES=001497 WS=001260 LOCK=000000 RESVD=000000
NPREF=000000 PREF=000000 READS=000130 WRITES=000018
CPU 00: CTIME=01:22 VTIME=000:06 TTIME=000:10 IO=007553
        RDR=000000 PRT=000738 PCH=000000
        VVECTIME=000:00 TVECTIME=000:00

#cp query time #cp indicate user
CP QUERY TIME
CP INDICATE USER
TIME IS 15:08:31 EST TUESDAY 02/14/97
CONNECT= 01:22:56 VIRTCPU= 000:07.50 TOTCPU= 000:11.82
USERID=SQLDBA   MACH=XC  STOR=0009M VIRT=V XSTORE=NONE
IPLSYS=CMS      DEVNUM=00031
PAGES: RES=001499 WS=001472 LOCK=000000 RESVD=000000
NPREF=000000 PREF=000000 READS=000135 WRITES=000022
CPU 00: CTIME=01:23 VTIME=000:08 TTIME=000:12 IO=009049
        RDR=000000 PRT=000974 PCH=000000
        VVECTIME=000:00 TVECTIME=000:00
```

The output shows that, during 71 seconds (CONNECT advanced from 01:21:45 to 01:22:56) the following occurred:

RES	The number of virtual pages in main storage increased by two (1499-1497).
READS	Five reads from system paging DASD (135-130)
WRITES	Four writes to system paging DASD (22-18)
VIRTCPU	1.22 seconds of virtual machine time were used (07.50-06.28)
TOTCPU	1.96 seconds of total CPU time were used (11.82-09.86)
IO	1496 I/O requests were issued (9049-7553)

There are four important values that you can calculate from these numbers:

Sampling Interval
ΔCONNECT. The change in elapsed time between CP QUERY TIME commands.

Main Storage Load
(RES+ (ΔRES/2))(4096)/(Total bytes of main storage). Indicates the average load on main storage.

System Paging DASD Load
(ΔREADS+ΔWRITES)/sampling interval. Indicates the average load on system paging DASD.

Total Processor (CPU) Load
(ΔTOTCPU/sampling interval)x100. Indicates the average percent of total CPU time your virtual machine is using. While this looks like an effective use measurement, it is really a measure of the load your database machine is placing on the CPU.

DASD I/O Load

ΔIO/sampling interval. Indicates the average load on the I/O system (tape and console I/O is also included, but not Paging or Spooling I/O).

For example, from the previous example:

- Sampling Interval: 71 seconds (01:21:45 to 01:22:56)
- An average of 1498 pages of main storage were used. This converts to 6135808 bytes or 5.85MB. If you knew, for example, that your processor has 32MB of main storage, you could calculate that the database machine was using almost 18.3% of it (5.85/32x100).
- System Paging DASD Load: 0.127/second ((4+5)/71)
- Total CPU Load: 2.76% ((1.96/71)x100)
- DASD I/O Load: 21.07/second (1496/71).

For more information, refer to the *VM/ESA: CP Command and Utility Reference* manual.

CICS Monitoring (CICSPARS for VSE)

This facility collects performance data during on-line processing for later off-line analysis. Monitoring data is recorded in the CICS journal data sets. This data can be formatted using the CICS Performance Analysis Reporting System (CICSPARS) field-developed program. The CICSPARS program is used with the VSE system for generalized performance analysis reporting (DOS/GPAR) to print analysis reports and summary reports of DB2 Server for VSE data as user clocks and counters.

CICSPARS collects performance class data for two general areas, link usage and call usage:

- Link usage data collected
  - Total number of link requests. This corresponds to the total number of logical units of work.
  - Total number of link requests resulting in a wait because all links are busy.
  - Total time waiting for links.
  - Total time holding links. This corresponds to the total time for all logical units of work.
- Call usage data collected
  - Number of calls to the database manager. This number can be greater than the total number of SQL statements issued by the application programs. This can occur because of the implicit connect support (for CICS users not required to provide user ID and password information to the database manager), the TPSP support, and the fact that a single SQL statement can result in multiple database manager calls. Multiple calls may occur when an SQL statement has a large amount of output.
  - Number of failing calls to the database manager. These are calls that result in negative SQLCODEs.
  - Total time waiting for database manager calls to process.

The CICS monitoring facility automatically associates all performance class data with the CICS transaction running at the time. This allows data reduction programs that process this information to construct a performance profile for any given transaction or call summarized by transaction type. With reference to the DB2 Server for VSE timings listed previously, the transaction profile shown in Figure 1 on page 19 can be created.

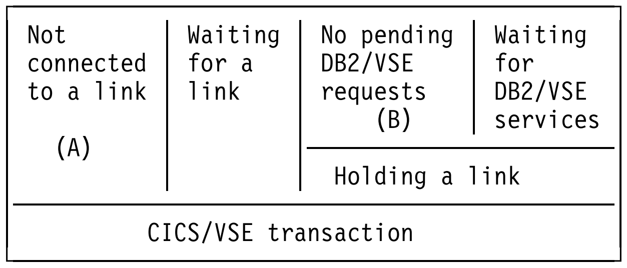


Figure 1. CICS Transaction Time Usage

In Figure 1, blocks (A) and (B) represent intervals during the lifetime of the transaction when other services within the CICS environment are being used. Because most of these other services are also represented in the performance class data, the use of these services can also be broken down, if required, in a manner similar to the breakdown shown for the database manager. Consequently, the database manager is integrated into a composite picture of each transaction's performance. This allows any transaction (or set of transactions) experiencing unacceptable response times to be investigated in a simple, systematic manner.

Before the CICS monitoring facility can be run, CICS must be set up to process the clocks and counters to be used and the journals used to record the data. For information on the entries required in various CICS tables, see the *DB2 Server for VSE Installation* manual.

After the CICS tables have been updated, the CICS monitoring facility can be started by using either the CICS CSTT transaction or the MONITOR=PER keyword of the CICS DFHSIT macro. These methods are also shown in the *DB2 Server for VSE Installation* manual.

Table 1 shows how to relate the DB2 Server for VSE clocks to the DFHMCT entries. The specification of the keyword ID maps to the clock definition. The specifications of the ID keyword must use the numeric values shown in Table 1.

Table 1. Relationship of CICS DFHMCT ID Keywords to Clocks

ID Keyword for CICS/VSE DFHMCT Entry	**Defines the Clock that Measures**

ID=(PP,16)
ID=(PP,17)
	Time waiting for a link

ID=(PP,18)
ID=(PP,19)
	Time holding a link

ID=(PP,20)
ID=(PP,21)
	Time for DB2 Server for VSE processing

The CICS DFHMCT entries also define the four DB2 Server for VSE counters. The argument for the ID keyword for these counters must be ID=(PP,22). The order of the four counters is:

- Counter 1, the number of link allocates
- Counter 2, the number of link waits
- Counter 3, the number of DB2 Server for VSE requests
- Counter 4, the number of DB2 Server for VSE errors.

DB2 Server for VSE & VM Tools

Physical Data Locations

Disk Locations (VM)

The file definitions that the application server uses to point to the directory, log, and dbextent disks appear in the start up message stream. For example:

```
sqlstart DB(SQLMACH1)
Ready; T=0.03/0.05 14:22:40
ARI0717I Start SQLSTART EXEC: 09/15/92 14:22:40 EDT.
ARI0663I FILEDEFS in effect are:
ARISQLLD DISK     ARISQLLD LOADLIB  Q1
BDISK    DISK     300
LOGDSK1  DISK     301
LOGDSK2  DISK     302
DDSK1    DISK     303
DDSK2    DISK     304
DDSK3    DISK     305
DDSK4    DISK     306
⋮
```

This database has its directory disk at virtual address 300, its log disks at 301 and 302, and its dbextents from 303 to 306. The physical minidisk locations are defined in the VM Directory. To find out the DASD type, volume identifier, and size of each disk, type: `#CP Q V DASD` from the operator console. For example:

```
#cp q v dasd
⋮
DASD 0300 3390 PA326B R/W      6 CYL  ON DASD  168B
DASD 0301 3390 PA3268 R/W      3 CYL  ON DASD  1688
DASD 0302 3390 PA3268 R/W      3 CYL  ON DASD  1688
DASD 0303 3390 PA326A R/W      5 CYL  ON DASD  168A
DASD 0304 3390 PA326A R/W      5 CYL  ON DASD  168A
DASD 0305 3390 PA326A R/W      2 CYL  ON DASD  168A
DASD 0306 3390 PA3269 R/W      2 CYL  ON DASD  1689
⋮
```

Data Set Placement (VSE)

In DB2 Server for VSE, dbextents are defined as VSAM datasets. To find out their dataset names, look in the database identification procedure for your server (shipped as an example procedure ARIS41DB), which is executed just before the ARISQLDS start up job step in the start up job stream. (Procedure ARIS41DB is only an example. The database identification procedure for your server may have a different name and point to different disks.)

```
// DLBL BDISK,'SQL.BDISK.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL LOGDSK1,'SQL.LOGDSK1.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL LOGDSK2,'SQL.LOGDSK2.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL DDSK1,'SQL.DDSK1.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL DDSK2,'SQL.DDSK2.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL DDSK3,'SQL.DDSK3.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL DDSK4,'SQL.DDSK4.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL DDSK5,'SQL.DDSK5.DBASE.DB',,VSAM,CAT=SQLCAT
// DLBL DDSK6,'SQL.DDSK6.DBASE.DB',,VSAM,CAT=SQLCAT
```

You can find the size and location of the datasets either by using the Access Method Services (IDCAMS) utility (part of VSAM LISTCAT), or through the VSE interactive interface. Both are documented in the *DB2 Server for VSE & VM Operation* manual.

Initialization Parameters

When you initialize the application server, important information is presented on the operator console:

```
sqlstart DB(SQLMACH1)
⋮
ARI0020I Virtual machine addressing mode = 31
         Virtual machine storage size = 24576KB
⋮
ARI0015I SYNCPNT parameter value is Y.
⋮
ARI0016I TRACEBUF parameter value is 0.
ARI0016I CHKINTVL parameter value is 150.
ARI0016I NCSCANS parameter value is 30.
ARI0016I NCUSERS parameter value is 5.
ARI0016I NDIRBUF parameter value is 100.
ARI0016I NLRBS parameter value is 2520.
ARI0016I NLRBU parameter value is 1000.
ARI0016I NPACKAGE parameter value is 10.
ARI0016I NPACKPCT parameter value is 30.
ARI0016I NPAGBUF parameter value is 100.
ARI0016I SLOGCUSH parameter value is 90.
ARI0016I SOSLEVEL parameter value is 10.
ARI0016I DISPBIAS parameter value is 7.
ARI0016I LTIMEOUT parameter value is 0.
ARI0283I Log analysis is complete.
ARI0282I LUW UNDO is completed.
ARI0281I LUW REDO is completed.
ARI0143I The application server has been initialized
         with the following values:
         CHARNAME = INTERNATIONAL, DBCS = NO, CHARSUB = SBCS,
         CCSIDSBCS = 500, CCSIDMIXED = 0, CCSIDGRAPHIC = 0.
ARI0060I Database manager initialization complete.
ARI0045I Ready for operator communications.
```

You can find the value of some of these parameters from the console log, or from the start up options file in VM, or from an options member that is specified in the PARM list of the start up EXEC statement. You can also use the 'SHOW INITPARM' operator command to display most of the parameters when the server is running in multiple user mode. For more information, refer to the *DB2 Server for VSE & VM Operation* manual.

CIRD Transaction (CICS)

CIRD is a DB2 Server for VSE-supplied transaction, that lets you display a snapshot of the links between CICS and the application server. While it does not provide historical information, it can help you diagnose problems with individual transactions or get an immediate feel for the level of link contention between CICS and the server.

It contains the following information:

- Which users are waiting for a link
- Which ones are currently using a link to access the server
- Which ones are holding a link but not accessing the application server
- Which ones previously held a link, but currently do not.

For example:

```
DBDCCICS CONNECTED TO SERVER SQLDB1_NEWYORK_INV.
STATUS OF ONLINE DB2 FOR VSE APPLICATIONS:

TRANSACTIONS WAITING TO ESTABLISH A LINK TO THE APPLICATION
SERVER ARE:
 TASKNO TRANID TERMID USERID   USERDATA WAIT TIME
 ------ ------ ------ -------- -------- ---------
 000033 MKE2                   L222     00:01:32
 000025 INV    L224   JIM               00:08:32

TRANSACTIONS HOLDING A LINK AND NOW ACCESSING THE APPLICATION
SERVER ARE:
 TASKNO TRANID TERMID USERID   USERDATA TIME USED    TOTAL LUW
                                        FOR CURRENT  TIME
                                        ACCESS
 ------ ------ ------ -------- -------- ------------ ---------
 000019 CISQ          DEPT222  L199     00:01:32     00:03:48
 000037 INV    L209   TERRY             00:00:01     00:00:03

TRANSACTIONS HOLDING A LINK TO THE APPLICATION SERVER BUT NOT
USING ARE:
 TASKNO TRANID TERMID USERID   USERDATA TIME SINCE   TOTAL LUW
                                        LAST ACCESS  TIME
 ------ ------ ------ -------- -------- ------------ ---------
 000003 CISQ          WILLIAM  L210     00:07:01     00:10:56

TRANSACTIONS WHICH PREVIOUSLY ACCESSED THE APPLICATION SERVER
(NOT HOLDING LINK):
 TASKNO TRANID TERMID USERID   USERDATA TIME SINCE
                                        LAST ACCESS
 ------ ------ ------ -------- -------- ------------
 000003 MKE2          ROBERT   L210     00:20:04

TIME=14:28:23 DATE=04/30/93
```

For information on how to display CIRD transaction information, and detailed information on how to use and interpret CIRD display information, refer to the *DB2 Server for VSE System Administration* manual.

COUNTER Operator Command

While a detailed description of this operator command is included in the *DB2 Server for VSE & VM Operation* manual, this section includes an example of how to:

- Use the command with the RESET operator command
- Interpret the counter values
- Turn these values into performance indicators.

After resetting the counters (with the RESET operator command) and performing several queries, the `COUNTER *` command was issued:

```
reset *
Counters reset at  DATE='09-06-92'  TIME='14:27:00'
ARI0065I Operator command processing is complete.
counter *
Counter values at  DATE='09-06-92'  TIME='14:58:12'
 Calls to RDS                RDSCALL :  68
 Calls to DBSS               DBSSCALL:  139
 LUWs started                BEGINLUW:  58
 LUWs rolled back            ROLLBACK:  11
 System checkpoints taken    CHKPOINT:  1
 Maximum locks exceeded      LOCKLMT :  0
 Lock escalations            ESCALATE:  0
 Waits for lock              WAITLOCK:  4
 Deadlocks detected          DEADLCK :  1
 Looks in page buffer        LPAGBUFF:  298722
 DBSPACE page reads          PAGEREAD:  200134
 DBSPACE page writes         PAGWRITE:  97451
 Looks in directory buffer   LDIRBUFF:  5054
 Directory block reads       DIRREAD :  4014
 Directory block writes      DIRWRITE:  120
 Log page reads              LOGREAD :  2
 Log page writes             LOGWRITE:  40
 Total DASD reads            DASDREAD:  4524
 Total DASD writes           DASDWRIT:  49
 Total DASD I/O              DASDIO  :  3986
 Lock timeouts detected      LTIMEOUT:  2
 ARI0065I Operator command processing is complete.
```

There are several important values that you can calculate from the COUNTER command:

Sampling Interval
ΔTIME. The elapsed time between the RESET and the COUNTER command. For more information on sampling intervals, refer to “Monitoring Interval” on page 6.

LUW Load
BEGINLUW/Sampling Interval. This is the average rate at which the database manager receives logical units of work. It measures the average load on the database machine or partition. You can also use it as a relative measure of throughput (refer to “Throughput” on page 5).

Checkpoint Load
CHKPOINT/sampling interval. This is the average rate of checkpoints. Checkpoints are overhead; they represent an additional load on the database machine. For more information on checkpoint processing, refer to “Checkpoints” on page 110.

Deadlock Performance

(DEADLCK/BEGINLUW)x100. Indicates the percentage of time a logical unit of work is rolled back because of a potential deadlock. Ranges from 0 to 100% where 0% indicates that no LUWs were rolled back. While some potential deadlocks are a normal occurrence in any multiple user system, a value above 5% should be investigated. For more information, refer to “Deadlock” on page 108.

This information can also be expressed as the *deadlock hit ratio* (BEGINLUW/DEADLCK). Any value over 20 is usually acceptable.

Waitlock Performance

(WAITLOCK/RDSCALL)x100. Indicates the percentage of time a call to the relational data system had to wait because it needed a resource that was blocked by an incompatible lock held by another call. Ranges from 0 to 100% were 0% indicates no waits. While waits are a normal occurrence in any multiple user system, a value above 10% should be investigated. For more information, refer to “Locking Contention” on page 101.

Note: While this value gives you an indication of how often an agent waits, it does not indicate the length of each wait. Always listen to your users and look for agents that are *stuck* in a lock wait with the operator SHOW LOCK commands, refer to “Locking Contention” on page 36.

This information can also be expressed as the *waitlock hit ratio* (RDSCALL/WAITLOCK). Any value over 10 is usually acceptable.

Lock Request Block Performance

ESCALATE+LOCKLMT. Indicates the number of times that a logical unit of work reached the user (NLRBU) or system (NLRBS) lock limit. This value should be close to zero, which indicates that there was no shortage of lock request blocks during the monitoring interval. For more information, refer to “Lock Escalation” on page 107.

Local Buffers Effective Use

(1-PAGEREAD/LPAGBUFF)x100. Indicates the percentage of time the database manager found a page in the local buffers and did not need to retrieve it from DASD. Ranges from 0 to 100%, where 100% means that every page the database manager needed was in the local buffers. For more information on the local buffer pool, refer to “Database I/O” on page 89.

This information can also be expressed as the *local buffers hit ratio* (LPAGBUFF/PAGEREAD).

Directory Buffers Effective Use

(1-DIRREAD/LDIRBUFF)x100. Indicates the percentage of time the database manager found a page in the directory buffer pool and did not need to retrieve it from DASD. Ranges from 0 to 100%, where 100% means that every directory page the database manager needed was in the directory buffer pool. For more information on the directory buffer pool, refer to “Database I/O” on page 89.

This information can also be expressed as the *directory buffers hit ratio* (LDIRBUFF/DIRREAD).

For example, from the previous output:

- Sampling Interval: 1872 seconds (14:58:12-14:27:00)
- LUW Load: 3.54/second (58/1872)
- Checkpoint Load: 2/hour (1/1872*3600) This value is not statistically valid. You need to monitor checkpoints over a longer period of time for an accurate calculation.
- Deadlock Performance: 1.72% (1/58)X100. While this value is statistically questionable it is far enough below the recommended value of 5% that deadlock should not be considered a significant problem.
- Deadlock Hit Ratio: 58 (58/1). 58 is greater than 20 and should be acceptable.
- Waitlock Performance 5.88% (4/68)X100. Below 10% and not a problem.
- Lock Request Block Performance: 0
- Local Buffers Effective Use: 33% (1-200134/298722)x100. Check the type of transactions you are running. Unless you are performing update intensive transactions, this value should be much higher. Consider increasing the NPAGBUFF initialization parameter.
- Local Buffers Hit Ratio: 1.49 (298722/200134)
- Directory Buffers Effective Use: 20.58% (1-4014/5054)X100. This value is as bad as the local buffer use value. Consider increasing the NDIRBUF initialization parameter.
- Directory Buffers Hit Ratio: 1.26 (5054/4014).

DB2 Data Spaces Support

DB2 Data Spaces Support also includes additional counters that can help you monitor the performance of the DASD I/O systems. Each storage pool has its own set of four counters. There is also a set for internal dbspaces and a set for the directory. For a complete description of all these counters and how to use them, refer to the *DB2 Data Spaces Support* manual.

SHOW Commands

The SHOW commands, documented in the *DB2 Server for VSE & VM Operation* manual, provide information on how your application server is performing. This section includes examples of how to use these commands to understand how the server is managing: storage, concurrency, locking.

Storage

Available Storage Pool Space: The **SHOW DBEXTENT** command displays physical storage information about each storage pool defined. For example, consider a database with an SOSLEVEL of 10% (refer to “Short on Storage Cushion” on page 62):

```
show dbextent
POOL    TOTAL     NO. OF      NO. OF      NO. OF     %     NO. OF
 NO.    PAGES   PAGES USED  FREE PAGES  RESV PAGES  USED   EXTENTS  SOS
  1      741      730         11          20       99      1      *
  2      171      11          160         20       6       1
  3      114      56          58          20       49      1
  4      114      0           114         20       0       1
FREE   22500
```

The asterisk (*) under the SOS column indicates that storage pool number one is short on storage. The flag is set if the difference between the NO. OF PAGES USED and the TOTAL PAGES is less than the SOSLEVEL percentage times the TOTAL PAGES. In this case, pool one has 99% (730/741 X 100) of its pages used or 1% or its pages are free. Because this is less than the 10% SOSLEVEL, the flag is set.

Note: The NO. OF PAGES USED includes the number of shadow pages in the pool.

Proportion of Available Pages: The **SHOW DBSPACE** command shows the division of pages between header, data, and index pages in a dbspace. For example:

```
show dbspace 3
 TYPE      NUMBER      NUMBER OF       % FREE     NUMBER OF
OF PAGES  OF PAGES   OCCUPIED PAGES    SPACE     EMPTY PAGES
 HEADER        8          1 ( 12 %)      73 %           0
 DATA       1365        756 ( 55 %)      25 %         392
 INDEX       128         27 ( 27 %)      79 %          18
ARI0065I Operator command processing is complete.
```

(The SYSTEM.SYSDBSPACES catalog table contains additional information on dbspaces, refer to page 40.)

This example shows more than half of the data page space is occupied. It also shows a large number of empty data pages, which indicates that you may need to drop and reacquire (reorganize) this dbspace. For more information, refer to “Running out of Dbspace Pages” on page 68.

Note: This command performs a dbspace scan, which can take a significant amount of time and affect the performance of other users if the dbspace is large. Refer to “Dbspace Scans” on page 126.

Available Dbextent Storage: The **SHOW POOL** command displays physical storage information about each dbextent in a storage pool.

For example, consider a database with two storage pools and an SOSLEVEL of 10% (refer to “Short on Storage Cushion” on page 62):

```
show pool

POOL NO.  1:         NUMBER OF EXTENTS =  3

EXTENT  TOTAL    NO. OF       NO. OF      NO. OF     %
NO.     PAGES  PAGES USED    FREE PAGES  RESV PAGES  USED
  1      285     274           11                     96
  2      285      33          252                     11
  3      741       0          741                      0
TOTAL   1311     307          1004         20         23

POOL NO.  2:         NUMBER OF EXTENTS =  1        SHORT ON STORAGE

EXTENT  TOTAL    NO. OF       NO. OF      NO. OF     %
NO.     PAGES  PAGES USED    FREE PAGES  RESV PAGES  USED
  4      285     260           25                     91
TOTAL    285     260           25          20         91


FREE AREAS:  NUMBER OF DELETED EXTENTS =  0

EXTENT  TOTAL
 NO.    PAGES
 END    10350
TOTAL   10350

Maximum number of DBEXTENTs = 64
ARI0065I Operator command processing is complete.
```

This example shows that pool number two is short on storage (SOS). While dbextent number one has less storage available than dbextent number four, the pool it belongs to (pool one) is not short of storage because it still contains two dbextents, each containing a significant amount of storage.

Virtual Storage: The **SHOW STORAGE** command displays how much of the database machine or partition's virtual address space has been allocated to various storage queues, refer to “Storage Queues” on page 46. Two storage queues are created for:

- Each real agent
- The `OPERATOR` agent, used to process operator commands
- The `CHECKPT` agent, used to process checkpoints
- The `RECOVERY` agent, used to write archives
- An area called `PROTOTYPE` that is used for global storage blocks such as accounting records.

Storage that has been allocated to control blocks and programming structures that must reside below the 16MB line are indicated by a `B` in the `LOC` column. Storage that may be allocated anywhere above or below the line is indicated by an `A` in the `LOC` column.

The SHOW STORAGE command displays how much of the database machine or partition's address space has been `ALLOCATED` to each queue. It also displays how much of it is actually `IN USE` and how much is `FREE`. “Free space” is space that has been reserved by the queue but is not currently in use.

Space allocated to a specific real agent queue (including “free” space) is released at the end of a logical unit of work (LUW). (A minimum amount of space, 8KB, always remains allocated for each real agent queue.)

The HIGHSTOR column contains the maximum amount of storage allocated to a queue since the RESET HIGHSTOR operator command was last issued. One high storage entry exists for storage that is restricted to below the 16MB line and one exists for storage that can reside either above or below the line.

A USERID column is also included for each real agent. It indicates the user ID that held the real agent when the A, or B high storage value was set.

SUMMARY contains the total amount of virtual storage allocated to all the QUEUES. It also contains the total amount of virtual storage allocated to the entire database manager SYSTEM. These values include the storage allocated to the queues plus the storage used by the database manager itself for structures such as trace buffers.

```
show storage
Status of Storage at DATE='01-15-93'  TIME='11:57:28'

AGENT      LOC  ALLOCATED     IN USE       FREE   HIGHSTOR  USERID
OPERATOR    A        4096       1104       2992       4096
            B        4096          0       4096       4096
CHECKPT     A        2048          0       2048       2048
            B        2048          0       2048       2048
RECOVERY    A        4096          0       4096       4096
            B        4096          0       4096       4096
1           A       16384       1952      14432     144816  MARISSA
            B        8192          0       8192       8192
2           A       72776      67360       5416     136472  LAURA
            B        8192          0       8192       8192
3           A        8192       2912       5280     161200  ANDREW
            B        8192          0       8192       8192
4           A       16384       3936      12448     185016  DAVID

            B        8192          0       8192       8192
5           A       16384       2992      13392     241600  DANIEL
            B        8192          0       8192       8192
PROTOTYPE   A     1154768    1053240     101528    1218784
            B      307232     299136       8096     307232
SUMMARY
QUEUES      A     1295128    1133496     161632    1702992
            B      358432     299136      59296     358432
PACKAGES    A      245760     201000      44760     270048
SYSTEM      A     1295952                          1703816
            B      388936                           388936
ARI0065I Operator command processing is complete.
```

Concurrency

The **SHOW ACTIVE** command displays the status of active real agent structures. Agents are used by the database manager to divide processor time between multiple users and its own internal tasks, such as checkpoint processing and operator commands. For more information, refer to “Agents” on page 95.

Use this command to:

- Identify the current state of an active agent — for example, whether it is currently processing work or waiting, and if the latter, what its waiting for.
- How many agents are available, and how many are currently being used.

For example:

```
show active
Status of agents:
   Checkpoint agent is not active.
   User Agent:   1 User ID:  SMITH    is R/W APPL   7B4
      Agent is     processing and is in I/O           Wait.
   User Agent:   2 User ID:  MICHAEL  is R/O SUBS   7B9
      Agent is not processing and is in communication Wait.
   User Agent:   3 User ID:  JESSICA  IS R/O APPL   5A4
      Agent is processing an operator command.
   User Agent:   4 User ID:  TESTUSER IS R/W APPL   7BB
      Agent is     processing and is in I/O           Wait.
   User Agent:   5 User ID:  MACNIELL IS R/O APPL   7B9
      Agent is not processing and is in communication Wait.
   0    agent(s) not connected to an APPL or SUBSYS.
ARI0065I Operator command processing is complete.
```

This display shows that there are five real agents available and they are all currently being used (`0 agent(s) not connected`). It also shows that agents one, two, four, and five are all processing work but are waiting for either communications or an I/O operation. Agent three is currently processing an operator command.

The **SHOW USERS** command (VM systems only) displays the status of both pseudo and real agent structures. (For information on these structures, refer to "Agents" on page 95.) You can use it to see how many, if any, pseudo agents are waiting for real agent structures. For example, consider a database machine that owns seven disks: four dbextents, one directory disk, and two log disks (do not include the service or the production disk):

```
show users
Status of connected users:
  6  users are connected to the application server.
  3  Users are active.
     User ID: DAVE      SQL ID: SMITH       not processing
     User ID: POTTS     SQL ID: BRIAN       not processing
     User ID: TUNA      SQL ID: FISH
  2  Users are waiting.
     User ID: KIM       SQL ID: TASK115
     User ID: MICHAEL   SQL ID: MIKE2
  1  Users are inactive.
     User ID: KOHLMANN  SQL ID: PETER
  0  Agents are available.
 44  User connections are available.
ARI0065I Operator command processing is complete.
```

This example shows that:

- MAXCONN is set to 58 (44+6+7+1). 44 `connections` are available, six `users are connected`, seven disks are connected to the machine, and one for the connection to *IDENT. (Refer to "VM (MAXCONN)" on page 98.)
- NCUSERS is set to 3 (3+0=3). All three real agents are occupied. (Refer to "Tuning Parameters (NCUSERS)" on page 96.)
- Two users (KIM and MICHAEL) are waiting for real agents. Neither KIM nor MICHAEL will become active until DAVE, POTTS, or TUNA complete their current logical unit of work.

- One user (KOHLMANN) is inactive; it is neither waiting for a real agent nor is it processing work.
- User ID TUNA is currently processing work.

Not only does the **SHOW CONNECT** command display much of the information included with the SHOW ACTIVE and SHOW USERS operator commands, it also includes:

- Information that uniquely identifies DRDA* application requestors
- The timestamp when the current state began
- The timestamp when the user was connected to the application server
- The CPU time used since the user was connected to the server (only displayed if you start the server with accounting on).

For the VSE System

The VSE SHOW CONNECT statement contains several additional values. See "VSE SHOW CONNECT" on page 33.

You can use this information to force specific users to end their work and terminate their conversations with the server. Refer to the *DB2 Server for VSE & VM Operation* manual.

```
show connect
Status of Connected Users                         1996-02-04  10:25:33
   Checkpoint agent is not active.
   User Agent:   1   User-ID: SHUM     SQL-ID: SHUM
      is R/W  APPL   7B1
      Agent is processing with LPAGEBUF=1032
                  State started: 1996-02-04  10:15:30
                  Conversation started: 1996-02-04  10:12:45
                  CPU time: 00:00:01
   User Agent:   2   User-ID: SQLUSRSS SQL-ID: SQLUSRSS
      is R/O  APPL  30BD
      Agent is not processing and is in communication  wait.
                  State started: 1996-02-04  09:48:28
                  Conversation started: 1996-02-04  09:48:00
                  CPU time: 00:00:02
                  LUWID: CAIBMOML.*IDENT.A532D460755B.0001
                  EXTNAM: SQLUSRSS.1
                  Requester: SQLDS/VM  V3.3.0   at TORVMLB4
   User Agent:   3   User-ID: PETERSON SQL-ID: PETERSON
      is R/O  APPL  3758
      Agent is processing and is in LOCK  wait.
                  State started: 1996-02-04  10:23:11
                  Conversation started: 1996-02-04  10:22:15
                  CPU time: 00:00:01
                  LUWID: CAIBMOML.STLLU.A5241A50FABD.0001
                  EXTNAM: PETERSON     .BATCH   .PETERSON.DSNESPRR
                  Requester: DB2        V2.3.0   at IBMREGRDBSTL0012
   User-ID: SWAGRMAN SQL-ID: SQLDBA
      User is waiting for an agent
                  State started: 1996-02-04  10:22:11
                  Conversation started: 1996-02-04  10:03:05
                  CPU time: 00:00:02
                  LUWID: IBMNET07.*IDENT.AB457DFF69BC.0001
                  EXTNAM: SWAGRMAN.1
                  Requester: SQLDS/VM   V3.3.0   at TOROLAB3
   User-ID: JAVIER   SQL-ID: JAVIER
      User is inactive
                  State started: 1996-02-04  10:02:11
                  Conversation started: 1996-02-04  09:27:49
                  CPU time: 00:00:03
                  LUWID: IBMNET07.*IDENT.AB457DFF6ABC.0001
                  EXTNAM: JAVIER.1
                  Requester: SQLDS/VM   V3.3.0   at TOROLAB
  3  Users are active.
  1  Users are waiting.
  1  Users are inactive.
  0  Agents are available.
  94  User connections are available.
ARI0065I Operator command processing is complete.
```

The current time is 10:25:33. There are three active users:

- Agent 1 (SHUM) has been processing for approximately ten minutes (10:25-10:15), and has used one second of CPU time since it connected to the server.
- Agent 2 (SQLUSRSS) has been in a communication wait for almost 37 minutes, and has used two seconds of CPU time.
- Agent 3 (PETERSON) has been in a lock wait for a little less than two minutes, and used one second of CPU time.

There are two other connected users. One is waiting for an agent, the other is inactive.

- User-ID SWAGRMAN has been waiting for an agent for a little over three minutes, and has used two seconds of CPU time.
- Finally, User-ID JAVIER has been inactive for over 23 minutes, and has used three seconds of CPU time.

You can also determine from the additional lines of information (LUWID, EXTNAM, Requester) that all the requestors, with the exception of Agent 1, are DRDA requestors.

The example below an agent is executing a stored procedure.

```
  show connect
  Status of Connected Users                  1997-09-30  08:56:42
  Checkpoint agent is not active.
  User Agent:   1   User-ID: SQLUSRKJ SQL-ID: SQLUSRKJ
     is R/O  APPL  1666
     Agent is not processing and is in communication  wait.
       State started: 1997-09-30  08:56:39
       Conversation started: 1997-09-30  08:56:12
       Protocol: SQLDS
       Package: SQLDBA.MAINPGM      Section: 4
       Procedure: PROC1             Modname:  MYPROC
       Procedure Package: SQLDBA.MYPROC       Section: 4
  User Agent:   2   User-ID: SQLUSRJR SQL-ID: SQLUSRJR
     is R/O  APPL  1667
     Agent is not processing and is waiting for a stored procedure
     server in group GROUP1
       State started: 1997-09-30  08:56:39
       Conversation started: 1997-09-30  08:56:12
       Protocol: SQLDS
       Package: SQLDBA.MAINPGM2     Section: 3
  User Agent:   3   User-ID: SQLUSRTH SQL-ID: SQLUSRTH
     is R/O  APPL  1668
     Agent is processing with LPAGEBUFF=1032
       State started: 1997-09-30  08:56:39
       Conversation started: 1997-09-30  08:56:12
       Protocol: SQLDS
       Package: SQLDBA.MAINPGM3     Section: 4
       Procedure: PROC3             Modname:  MYPROC3
    Procedure Package: SQLDBA.MYPROC3      Section: 2
  User Agent:   4   User-ID: SQLUSRJR SQL-ID: SQLUSRTL
     is R/O  APPL  1669
     Agent is not processing and is waiting for stored procedure
     PROC4 AUTHID SQLUSRTL to be started
       State started: 1997-09-30  08:58:00
       Conversation started: 1997-09-30  08:57:35
       Protocol: SQLDS
       Package: SQLDBA.MAINPGM4     Section: 3
  4  Users are active.
  0  Users are waiting.
  0  Users are inactive.
  2  Agents are available.
  2  User connections are available.

ARI0065I Operator command processing is complete.
```

In the example above:

- The information for User Agent 1 shows both Package and Procedure Package information, which means that the agent is currently running a stored procedure. In this example, Agent 1 is running the package SQLDBA.MAINPGM, and SQLDBA.MAINPGM has called the stored procedure executing the package SQLDBA.MYPROC. The agent is in communication wait, which in this case means that the stored procedure is processing, and the database manager is waiting for the procedure to end or to pass another SQL command.

- The information for User Agent 2 shows Package information, but no Procedure Package information. This means that the agent is running the main program, which in this example is SQLDBA.MAINPGM2. The agent is waiting for a stored procedure server in group GROUP1, which means that it has issued an SQL CALL statement, but no stored procedure servers are available in the group in which the procedure can run (GROUP1).
- The information for User Agent 3 shows both Package and Procedure Package information, which means that the agent is currently running a stored procedure. In this example, Agent 3 is running the package SQLDBA.MAINPGM3, and SQLDBA.MAINPGM3 has called the stored procedure executing the package SQLDBA.MYPROC3. The agent is processing, which means that the database manager is currently executing an SQL command that was passed to it by the stored procedure.

VSE SHOW CONNECT

If a CICS user requests that the operator terminate a CICS transaction containing DB2 Server for VSE statements, the operator should first force the associated DB2 Server for VSE agent by using the FORCE command before terminating the transaction. Prior to SQLDS Version 3 Release 5, the operator may not be able to determine which agent to force, because multiple agents may use the same DB2 Server for VSE user ID. Furthermore, if the agent is not forced before the transaction is terminated, the resource adapter may encounter an error and shut itself down. This would cause the links to DB2 Server for VSE through that particular resource adapter to be lost. In SQLDS Version 3 Release 4, only the CICS task number representing the AXE transaction is displayed, and only for remote (DRDA) users.

Operators can now identify which agent should be forced by displaying the CICS task number, the CICS terminal ID, and the RMID for all local CICS users as part of the output for the SHOW CONNECT command. This information will be displayed for both VSE and VM (Guest Sharing) users.

The additional information on the SHOW CONNECT command will enable the operator to identify which agent should be forced. This is accomplished by the following steps:

1. The user tells the operator to cancel the task associated with a particular terminal ID. Alternatively, they may ask that a specific task be terminated.
2. The operator then issues the SHOW CONNECT command to determine which agent is associated with either the task ID or the terminal ID that was specified by the user.
3. The operator can then force the correct agent and then terminate the CICS transaction.

The CICS task number, CICS terminal id, and RMID will be displayed for all local (VSE or VM Guest Sharing) CICS transactions whenever the agent is in work. The CICS terminal ID may contain a value of 'N/A' indicating that the terminal ID is not available, such as when a user issues queries through ISQL. The CICS task number, the CICS terminal id, and the RMID will not be displayed for batch users, or for agents whose work status is NIW (not in work). For remote (DRDA) users, only the CICS task number representing the AXE transaction will be displayed; the CICS terminal id and the RMID will not be displayed.

When a CICS transaction is using a release of the Resource Adapter prior to Version 3 Release 5, the CICS terminal id and the RMID are not available to the database server and `'N/A'` will be displayed.

Included below are sample outputs for each of the cases where additional information may be displayed.

SHOW CONNECT for CICS Transaction (Version 3 Release 5)

```
F4 004 User Agent: 1 User-ID: JOAO SQL-ID: JOAO
F4 004 is R/O APPL 12BCF
F4 004 Agent is processing and is in communication wait.
F4 004 State started: 1995-09-02 15:21:22
F4 004 Conversation started: 1995-09-02 15:21:22
F4 004 Task no.: 147 RMID: 32 Term. id: 077D
```

Figure 2. CICS Transaction (Version 3 Release 5 Requester)

SHOW CONNECT for ISQL Query

```
F4 004 User Agent: 1 User-ID: JOAO SQL-ID: JOAO
F4 004 is R/O APPL 12BCF
F4 004 Agent is processing and is in communication wait.
F4 004 State started: 1995-09-02 15:21:22

F4 004 Conversation started: 1995-09-02 15:21:22
F4 004 Task no.: 147 RMID: 32 Term. id: N/A
```

Figure 3. ISQL Query

SHOW CONNECT for Agent Not in Work

```
F4 004 User Agent: 2 User-ID: DBDCCICS SQL-ID: DBDCCICS
F4 004 is NIW SUBS

F4 004 Agent is not processing and is in communication wait.
F4 004 State started: 1995-09-03 15:19:57
F4 004 Conversation started: 1995-09-03 15:19:57
```

Figure 4. Agent Not in Work

SHOW CONNECT for Batch User

```
F4 004 User Agent: 2 User-ID: SQLDBA SQL-ID: SQLDBA
F4 004 is R/W APPL 18D9
F4 004 Agent is not processing and is in communication wait.
F4 004 State started: 1995-09-08 15:34:51
F4 004 Conversation started: 1995-09-08 15:34:41
```

Figure 5. Batch User

SHOW CONNECT for DRDA User

```
F4 004 User Agent: 2 User-ID: EDUARDA SQL-ID: EDUARDA
F4 004 is R/W APPL 12FC4
F4 004 Agent is processing and is in communication wait.
F4 004 State started: 1995-09-02 15:23:17
F4 004 Conversation started: 1995-09-02 15:23:15
F4 004 CPU time: 00:00:01
F4 004 LUWID: CAIBMOML.OECGW001.A6773D6F8611.0001
F4 004 EXTNAM: EDUARDA.1
F4 004 Requester: SQLDS/VM V3.5.0 at TOIVMLB6
F4 004 Rmtuser ID: 2
F4 004 LU name: OMPGW001
F4 004 Task no.: 0000134
```

Figure 6. DRDA User Accessing VSE Database

SHOW CONNECT for Guest Sharing

```
User Agent: 1 User-ID: VSEMCH10 SQL-ID: SQLDBA
is R/O SUBS 1796
Agent is not processing and is in communication wait.
State started: 1995-09-08 10:42:55
Conversation started: 1995-09-08 10:42:43
Task no.: 371 RMID: 12 Term. id: N/A
```

Figure 7. VSE Guest Sharing User to VM Database Using ISQL

SHOW CONNECT for VM User

```
User Agent: 1 User-ID: SQLUSRMR SQL-ID: SQLUSRMR
is R/O APPL 178F
Agent is not processing and is in communication wait.
State started: 1995-09-08 10:41:11
Conversation started: 1995-09-08 10:41:04
```

Figure 8. VM Requester Accessing VM Database

SHOW CONNECT for CICS Transaction (Version 3 Release 4)

```
F4 004 User Agent: 1 User-ID: JOAO SQL-ID: JOAO
F4 004 is R/O APPL 12BCF
F4 004 Agent is processing and is in communication wait.

F4 004 State started: 1995-09-02 15:21:22
F4 004 Conversation started: 1995-09-02 15:21:22
F4 004 Task no.: 147 RMID: N/A Term. id: N/A
```

Figure 9. CICS Transaction (Version 3 Release 4 Requester)

Locking

Locking Contention: The **SHOW LOCK** command can help you understand and resolve immediate locking contention problems. (For information on this area, refer to “Locking Contention” on page 101.) Consider the following situation:

1. The default lock level (PAGE) is in effect.
2. PETER, BRIAN, and LAURA all select the salary of MICHAEL THOMPSON in the EMPLOYEE table through ISQL. They are all granted a SHARE (S) lock on the pages that contain Michael's salary and the page that contains index keys used to retrieve it. They are also granted an INTENT SHARE (IS) lock on the table and dbspace that contain Michael's salary.
3. Instead of clearing her query, LAURA leaves its results on her screen. This places her in a communication wait.
4. PETER tries to add $1000 to THOMPSON's salary, but is placed in a lock wait. (While he is granted an UPDATE (U) lock on the data page, the EXCLUSIVE (X) lock he needs on that page is incompatible with BRIAN and LAURA's SHARE (S) lock.)
5. BRIAN tries to increase THOMPSON's salary by 5%, but is also placed in a lock wait. (The update lock he needs is incompatible with the update lock that PETER already holds.)

A **SHOW LOCK ACTIVE** command reveals that LAURA is in a communication wait, and BRAIN and PETER are in a lock wait. (You can also determine this with a SHOW ACTIVE command.)

```
show lock active
                 WAIT   TOTAL  LONG   WANTLOCK WANTLOCK
AGENT  USER      STATE  LOCKS  LOCKS  TYPE     DBSPACE
 C     CHECKPT   NIW    0      0
 1     BRIAN     LOCK   55     55     PAGE     7
 2     PETER     LOCK   55     55     PAGE     7
 3     LAURA     COMM   44     44
ARI0065I Operator command processing is complete.
```

A **SHOW LOCK MATRIX** reveals that BRIAN is waiting for PETER, and PETER is waiting for LAURA. (The number seven in the lock matrix indicates that the contention is in dbspace seven.)

```
show lock matrix
Lock Request Block (LRB) and Lock Status:
  NLRBS   IN USE   FREE   NLRBU  MAX USED BY LUW
 ------- ------- ------- ------- ---------------
    2520     213    2307    1000       386
          *** THE LOCKWAIT TABLE ***
   ENTRY = DBSPACE NUMBER ON WHICH THERE IS LOCK CONTENTION
   The presence of an entry shows
   the agent requesting the lock and
   the agent contending for or holding the lock.
 AGENT           AGENT CONTENDING FOR OR HOLDING THE LOCK
 REQUESTING
 LOCK
             1        2        3        4        5
             BRIAN    PETER    LAURA
 1   BRIAN   ........ 7        ........ ........ ........
 2   PETER   ........ ........ 7        ........ ........
 3   LAURA   ........ ........ ........ ........ ........
 4           ........ ........ ........ ........ ........
 5           ........ ........ ........ ........ ........

ARI0065I Operator command processing is complete.
```

A **SHOW LOCK GRAPH** of BRIAN clearly shows the chain of lock contention that has occurred. Until LAURA clears her screen and returns her SHARE (S) lock, neither PETER nor BRIAN can leave their lock wait.

```
show lock graph brian
LOCK         LOCK         WAIT LOCK DBSP  LOCK        REQ    REQ
REQUESTER    HOLDER       STAT TYPE NUMBR QUALIFIER   STATE  MODE DUR
1   BRIAN    2   PETER    LOCK PAGE 7     88          G WAIT U    LONG
2   PETER    3   LAURA    COMM PAGE 7     88          C WAIT X    LONG
ARI0065I Operator command processing is complete.
```

The **SHOW LOCK USER** command displays both how many locks of each type are held plus the number they are waiting for.

```
show lock user
                DBSPACE  LOCK                                      NUMBER
AGENT  USER     NUMBER   TYPE   IN  SIX IS  IX  S   U   X   Z   WAITERS
:
 1     BRIAN     7       DBSP   0   0   1   0   0   0   0   0      0
 1     BRIAN     7       IPAG   0   0   0   0   1   0   0   0      0
 1     BRIAN     7       TABL   0   0   1   0   0   0   0   0      0
:
 2     LAURA     7       DBSP   0   0   1   0   0   0   0   0      0
 2     LAURA     7       IPAG   0   0   0   0   1   0   0   0      0
 2     LAURA     7       PAGE   0   0   0   0   2   0   0   0      2
 2     LAURA     7       TABL   0   0   1   0   0   0   0   0      0
:
 3     PETER     7       DBSP   0   0   0   1   0   0   0   0      0
 3     PETER     7       IPAG   0   0   0   0   1   0   0   0      0
 3     PETER     7       PAGE   0   0   0   0   0   1   0   0      2
 3     PETER     7       TABL   0   0   0   1   0   0   0   0      0
ARI0065I Operator command processing is complete.
```

In this case, PETER has an UPDATE (U) lock on the page that holds THOMPSON's salary, but it cannot be promoted to an EXCLUSIVE (X) lock because it is incompatible with LAURA's SHARE (S) lock. BRIAN has a SHARE lock but cannot promote it to an UPDATE lock because it is incompatible with PETER's UPDATE lock.

Lock Escalation: The **SHOW LOCK MATRIX** command can help you understand and resolve lock escalation problems. (For information on this area, refer to "Lock Escalation" on page 107.) Consider the following situation:

```
show lock matrix
Lock Request Block (LRB) and Lock Status:
  NLRBS   IN USE   FREE   NLRBU  MAX USED BY LUW
 ------- ------- ------- ------- ---------------
    2520     213    2307    1000       386
⋮
```

The maximum number of lock request blocks that can be held by a single agent (NLRBU) is set to 1000. The number that can be held by all the agents (NLRBS) is set to 2520. 213 blocks are currently in use and 2307 are free. The maximum number of blocks held since the last lock escalation is 386.

Do not rely on this command alone. The `MAX USED BY LUW` may appear to be significantly lower than NLRBS, but remember `MAX USED` is reset to zero after every escalation. If the database manager is constantly escalating locks, you may be unlucky enough to only see the value immediately following an escalation. Make sure that you consult the ESCALATE and LOCKLMT counters as well. Refer to "Lock Request Block Performance" on page 24.

Database catalog

Information about the database is maintained in a set of tables called the catalog which are created during database generation. They describe tables, columns, indexes, keys, packages, authorities and other objects in the database. This section describes how to select information from various tables that contain performance information.

The catalog also holds statistical information on the data stored in the database. The database manager uses it to select an access path for each SQL request it processes. Refer to "Keeping Database Statistics Current" on page 147 for a description of each table that contains this information and what the values in each column means. Also refer to "Using Catalog Statistics" on page 149 for a discussion of how to model a large production database with a small test database by altering the values in the catalog.

SYSTEM.SYSCATALOG

The following SQL statement retrieves performance information from the SYSCATALOG table about all the tables in the `sample` dbspace:

```
SELECT tname, avgrowlen, rowcount, npages, noverflow
       FROM system.syscatalog
       WHERE dbspacename='sample'
       AND creator='sqldba'


TNAME              AVGROWLEN      ROWCOUNT        NPAGES     NOVERFLOW
------------------ ---------  ------------  ------------  ------------
ACTIVITY                  31            18             2             0
DEPARTMENT                39             9             1             0
EMP_ACT                   36            74             2             0
EMPLOYEE                  80            32             2             0
INVENTORY                 21            22             1             0
OPERATIONS                44            15             2             0
PROJ_ACT                  29            77             2             0
PROJECT                   64            20             1             0
PROJECTS                  43             5             1             0
QUOTATIONS                24            53             1             0
SUPPLIERS                 57            10             2             0
```

AVGROWLEN
The average length of all the rows in the table, measured in bytes, refer to “Free Space in Data Pages” on page 64.

ROWCOUNT and NOVERFLOW
ROWCOUNT is the total number of rows in the table. NOVERFLOW is the number of rows in the tables that have overflowed from their original page in storage to another page. This is caused by variable length rows expanding because of updates. As a rule of thumb, if NOVERFLOW is greater than 5% of ROWCOUNT, it is probably time to reorganize the table, refer to “Reorganizing a Single Table” on page 74. However, remember that there are no absolute rules in performance tuning. You have to balance the cost of reorganization against the performance impact of the overflow rows.

If you decide to reorganize the table because of this, you may also want to use the ALTER DBSPACE command to increase the PCTFREE value of the dbspace that contains the table, refer to “Free Space in Data Pages” on page 64.

NPAGES
An estimate of the number of pages on which rows of this table appear.

SYSTEM.SYSCOLUMNS

The following SQL statement retrieves performance information from the SYSCOLUMNS table about all columns in the `employee` table:

```
SELECT cname, coltype, length, nulls, ccsid
       FROM system.syscolumns
       WHERE tname = 'employee'
       AND creator='sqldba'



 CNAME              COLTYPE   LENGTH   NULLS         CCSID
 ------------------ --------  -------  -----  ------------
 BIRTHDATE          DATE               Y                 ?
 BONUS              DECIMAL   ( 9, 2)  Y                 ?
 COMM               DECIMAL   ( 9, 2)  Y                 ?
 EDLEVEL            SMALLINT           N                 ?
 EMPNO              CHAR            6  N               500
 FIRSTNME           VARCHAR        12  N               500
 HIREDATE           DATE               Y                 ?
 JOB                CHAR            8  Y               500
 LASTNAME           VARCHAR        15  N               500
 MIDINIT            CHAR            1  N               500
 PHONENO            CHAR            4  Y               500
 SALARY             DECIMAL   ( 9, 2)  Y                 ?
 SEX                CHAR            1  Y               500
 WORKDEPT           CHAR            3  Y               500
```

COLTYPE and LENGTH
The datatype and length of the column. It is important that the predicates in a WHERE clause have the same data type and length, refer to "Column Attributes" on page 132.

NULLS
Whether a column can contain NULL values affects how it is accessed. (Refer to page 127 and page 2 on page 137.)

CCSID
The coded character set identifier (CCSID) of the column. CCSIDs can affect whether a predicate becomes sargable or residual, refer to "Impact of CCSIDs on Sargability" on page 141.

SYSTEM.SYSDBSPACES

The following SQL statement retrieves performance information from the SYSDBSPACES table about the `subscriptions` dbspace:

```
SELECT dbspaceno, npages, nactive, pctindx, freepct, lockmode, pool
       FROM system.sysdbspaces
       WHERE dbspacename='subscriptions'
       AND creator='sqldba'


 DBSPACENO        NPAGES       NACTIVE  PCTINDX  FREEPCT  LOCKMODE    POOL
 ---------  ------------  ------------  -------  -------  --------  ------
        75        480819         18427       25        0  T              2
 * End of Result *** 1 Rows Displayed ***Cost Estimate is 1*********************
```

NPAGES
The number of logical 4KB (kilobyte) pages available in this dbspace. In this case there are 1.878GB (gigabytes) of storage in this dbspace, refer to "Dbspace Full" on page 68.

NACTIVE
The number of active pages in the dbspace. It represents the number of 4KB data pages that must be read during a dbspace scan. In this case, the database manager must scan almost 72MB of storage to complete a dbspace scan. For more information refer to “Dbspace Scans” on page 126.

PCTINDX
The percentage of pages to be reserved for index pages, refer to “Proportion of Index to Data and Header Pages” on page 64.

FREEPCT
The current percentage of space on each page that is kept free when data is inserted in the dbspace, refer to “Free Space in Data Pages” on page 64.

LOCKMODE
Indicates whether row (T), page (P), or dbspace (S) level locking is being used for this dbspace, refer to “Minimum Lock Level” on page 103.

POOL
The number of the storage pool where pages from this dbspace are stored. You can use the SHOW POOL operator command to display information about this pool, refer to page 26.

SYSTEM.SYSINDEXES

The following SQL statement retrieves performance information from the SYSINDEXES table about all the indexes in the `sample` dbspace:

```
 SELECT iname, cluster, clusterratio, lockmode, ipctfree, release
        FROM  system.sysindexes, system.syscatalog
        WHERE system.sysindexes.tname = system.syscatalog.tname
        AND dbspacename='sample'
        AND creator='sqldba'


INAME               CLUSTER  CLUSTERRATIO  LOCKMODE  IPCTFREE  RELEASE
------------------  -------  ------------  --------  --------  -------
PKEYB1PAIBMXWNCV    W                9375  P               10  6.1.0
PKEYB1PAIAXH1U6L    F               10000  P               10  6.1.0
MGRNOI              C               10000  P               10  6.1.0
PROJNOIN            W                9306  P               10  6.1.0
EMPNOIN             C               10000  P               10  6.1.0
PKEYB1PAIA5RUD1W    F               10000  P               10  6.1.0
WORKDEPTI           N                8667  P               10  6.1.0
INV1                F               10000  P               10  6.1.0
OPE1                W                9231  P               10  6.1.0
PKEYB1PAIBSXHRBH    F               10000  P               10  6.1.0
DEPTNOI             C               10000  P               10  6.1.0
PKEYB1PAIBEPN7Y2    F               10000  P               10  6.1.0
RESPEMPI            C               10000  P               10  6.1.0
PRO1                F               10000  P               10  6.1.0
QUO1                F               10000  P               10  6.1.0
SUP1                F               10000  P               10  6.1.0
```

You can also retrieve the same information for the indexes of a single table. In this case the `employee` table:

```
SELECT iname, cluster, clusterratio, lockmode, ipctfree, release
       FROM system.sysindexes
       WHERE tname='employee'
       AND creator='sqldba'


INAME              CLUSTER  CLUSTERRATIO  LOCKMODE  IPCTFREE  RELEASE
------------------ -------  ------------  --------  --------  -------
PKEYB1PAIA5RUD1W   F               10000  P               10  6.1.0
WORKDEPTI          N                8667  P               10  6.1.0
```

CLUSTER and CLUSTERRATIO

CLUSTER indicates whether the index is a clustering index, refer to "The Clustering Index" on page 70. You can also use it to get an idea of whether an index is clustered, refer to "Clustered Indexes" on page 70.

CLUSTERRATIO is updated when the index's statistics are updated (SYSTEM.SYSINDEXES catalog table). It indicates the percentage of time that the data pages are in a logical sequence in relation to the index. In this case, when the statistics for WORKDEPTI were last updated, the data pages it referred to were in a logical sequence 86.67% of the time.

For more information on how to interpret CLUSTER and CLUSTERRATIO, refer to "Identifying Unclustered Indexes" on page 71.

LOCKMODE

Indicates whether page (P) or row level locking (K) is being used on this index, refer to "Minimum Lock Level" on page 103.

IPCTFREE

The amount of free space reserved in the index for later insertions and updates, refer to "Free Space in Index Pages" on page 66.

RELEASE

The release of the DB2 Server for VSE & VM product that was installed when the index was created. If the index was created prior to Version 2 Release 2 (2.2) it should be dropped and recreated to take advantage of performance improvements incorporated into the index structure at that time.

Chapter 3. Managing Storage and Configuring the Operating System

Real and Virtual Storage

There are two types of storage: real and virtual.

Real Storage
Composed of main and auxiliary storage. Main storage is the fastest storage and it is where data and programs must be before the CPU can directly act upon them. Auxiliary storage comprises expanded storage, and system paging DASD. Data and programs reside in one of these two areas when there is no room in main storage.

Virtual Storage
Virtual storage is an addressable space image for the user from which instructions and data are mapped into real storage locations. The operating system uses real storage (main and auxiliary storage) to create virtual machines (in the case of VM) or partitions (in the case of VSE).

Virtual Addressing

In **VM**, each virtual machine has its own *virtual address space*, which is where you load and run programs. **VSE** supports multiple address spaces that can each contain several partitions.

Because these address spaces are *virtual*, the operating system does not dedicate a piece of main storage to each virtual machine or partition. You do not need to buy 8MB of main storage for each 8MB virtual machine or partition. Instead the operating system only uses main storage for those parts of virtual storage you need right now, or are likely to need in the near future.

Pages

These parts of virtual storage are divided into 4KB (4096 byte) blocks called *pages*. When a virtual machine or partition needs a page that it has not accessed before, the operating system retrieves the page from its location on DASD, and loads it into an empty page in main storage. (Before a page can be used, it must be in main storage.)

Auxiliary Storage

When the operating system runs out of free pages in main storage, it moves the least recently used ("oldest") page to auxiliary storage to create a free space for a new page.

The **VM** operating system uses two types of auxiliary storage: system paging DASD, and optional expanded storage. If your system has expanded storage, a page will be moved there first. If expanded storage is full, the least recently used page in expanded storage is moved to system paging DASD by way of main storage. When a virtual machine needs a page that it has previously used, the operating system moves it back to main storage from expanded storage or from system paging DASD, if it is not already in main storage.

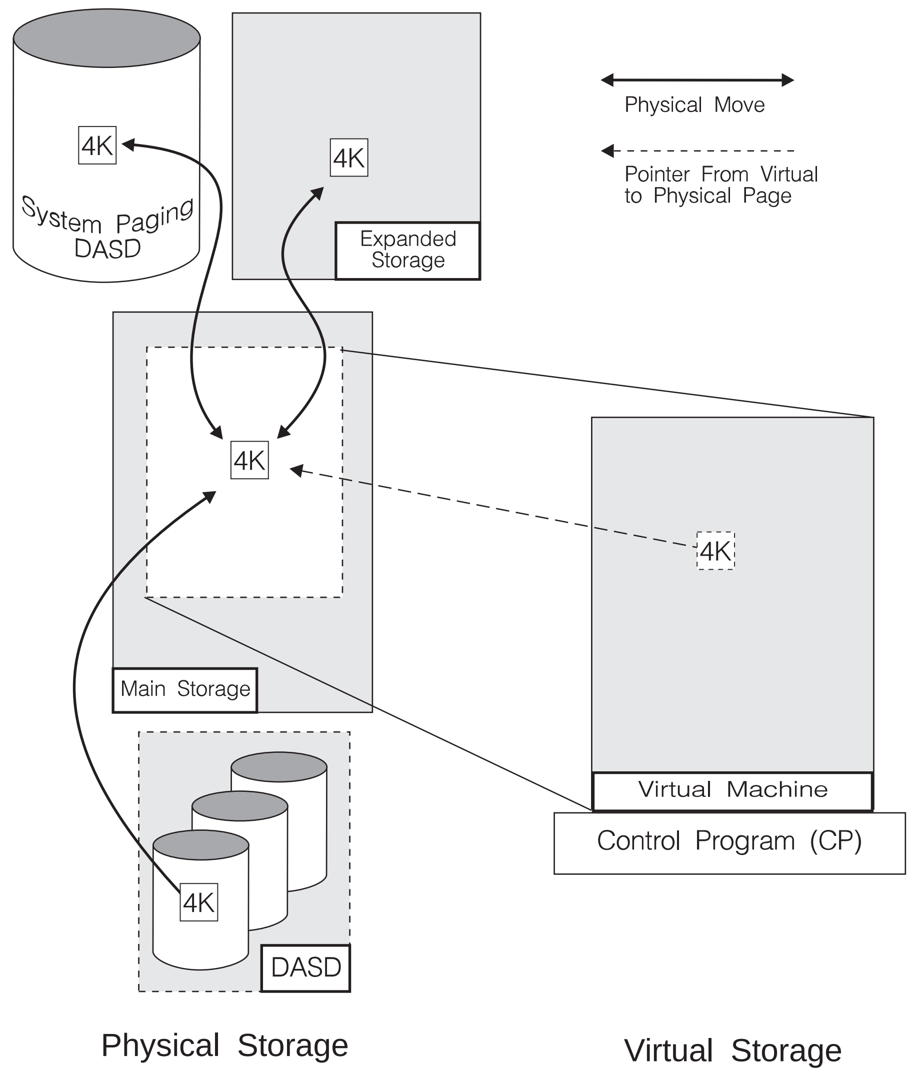


Figure 10. Standard Virtual Machine Storage

The **VSE** operating system only uses system paging DASD: it does not support expanded storage. When the operating system runs out of free pages in main storage, it moves the least recently used page directly to system paging DASD. When a partition needs a page that it has previously used, the operating system moves it back to main storage from system paging DASD, if it is not already in main storage.

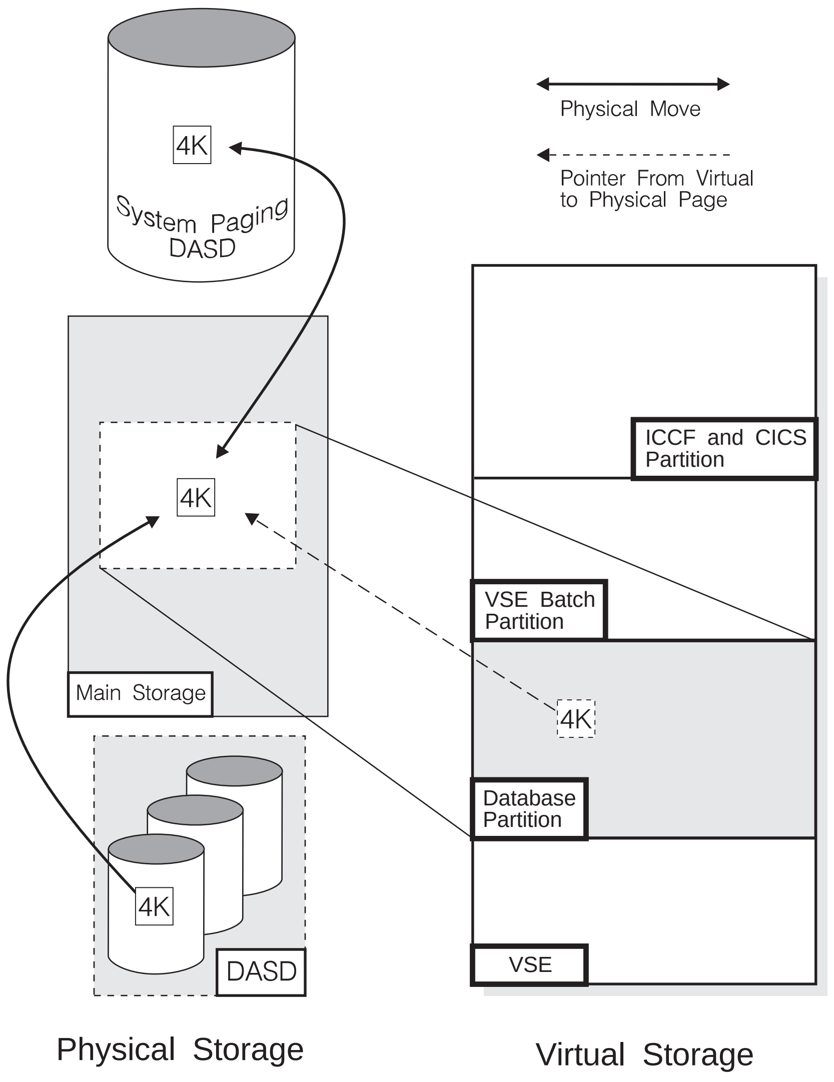


Figure 11. Standard VSE Partition Storage

This paging system accomplishes two things. First, it allows each virtual machine or partition to use much more storage than could be accommodated in main storage alone. Second, it keeps the most recently used pages in the storage devices that are the fastest to access. (The most recently used pages are the ones most likely to be used again in the near future.) Main and expanded storage are much faster than system paging DASD, and while expanded storage can be as fast as main storage, it is effectively slower because the operating system still needs to move the page into main storage before it can use it.

The Hidden Cost of System Paging DASD: Each time the database machine or partition (or the CICS partition) requires a page that the operating system cannot find in main storage, a page fault occurs. The entire database machine or partition, and therefore the entire database manager, must wait until the page is returned from auxiliary storage. Consider a system that requires an average of 50ms to return one page. So at 6 faults per second the database manager is idle 300 out of 1000ms, or almost one third of the time.

Note: This is **not** true for DASD I/O. A database machine does not wait for *BLOCKIO, nor does a database partition wait for VSAM. The database manager will dispatch another agent (unless you are running in single user mode) while it waits for the DASD I/O to complete. Refer to "Database I/O" on page 89.

Partition Deactivation (VSE): The TPBAL command (VSE system control statement) specifies the number of partitions that are eligible for deactivation. Whenever the TPIN macro (refer to the *IBM VSE/ESA System Macros Reference* manual) is executed, the number of partitions specified by the TPBAL command will be deactivated starting with the lowest priority partition and proceeding to the highest eligible partition (excluding the partition which executed the TPIN macro). All deactivated partitions are suspended (kept idle) until the TPOUT macro is executed. This can become a severe problem if, for example, the deactivated partition is an application server or a requester that is currently holding database locks. Whenever possible, ensure that the database partition, the CICS partition, or any DB2 Server for VSE batch partitions are **NOT** eligible for deactivation.

Storage Queues

A storage queue is a control structure that the database manager uses to share its virtual space between processes. Queues are created at startup for:

- Each real agent
- The operator
- For checkpoint processing
- Recovery
- Global storage blocks such as accounting records.

The database manager allocates virtual storage to the different processes as they require it (within limits). Once the storage has been allocated, it cannot be used by any other process until another process releases it.

Real agent queues release all but 8KB of their allocated storage at the end of a logical unit of work (LUW). Even if the real agent no longer requires the virtual storage, it may keep it until the end of its current LUW.

Virtual space may be allocated either above or below the 16MB virtual storage line (refer to "Storage Above 16MB (31 Bit Addressing)" on page 47.) However, because certain control blocks and program structures must always reside below the 16MB line, two storage queues are created for each process. One storage queue is for blocks and structures that must always reside below the line, the other queue is for blocks and structures that may reside either below or above the line.

When an agent requests virtual storage, the database manager decides whether the blocks and structures must reside below the 16MB line or whether they can reside either above or below the line. If they must reside below the line, space is allocated in the *below* or B queue. If they may reside above or below the line, space is allocated in the *anywhere* or A queue.

When the database manager looks for address space for blocks and structures in the B queue, it looks below the 16MB line. When the database manager looks for address space for blocks and structures in the A queue, it first tries to find free space above the 16MB line. If there is none, it will try to find free space below the line. If there is no free space below the line for either the A queue or for the B queue, releasing unused packages from storage. If it can no longer release packages, you will receive an error message and the database machine or partition may abend.

Address Space Size

While increasing the size of your virtual database machine or partition may increase your application server's capacity, it does not necessarily improve its performance. This is because the operating system has to supply enough fast real storage to make the virtual storage appear to be real storage.

However, if sufficient real storage is available, you can take advantage of additional virtual storage, by increasing:

- The number of concurrent users (NCUSERS). Refer to “Agents” on page 95.
- In VM, the number of pseudo agents (MAXCONN). Refer to “VM (MAXCONN)” on page 98.
- In VSE, increase the number of connections between the CICS partition and the database partition (CIRB transaction). Refer to “Tuning Parameter (CIRB)” on page 97. You can also increase the number of remote DRDA users (RMTUSERS). Refer to “VSE (RMTUSERS)” on page 99.
- The size of the buffers pools (NPAGBUF, NDIRBUF). Refer to “Database I/O” on page 89.
- The number of lock request blocks (NLRBU, NLRBS). Refer to “Lock Escalation” on page 107.
- The size of the package cache (NPACKAGE). Refer to “Package Cache” on page 94.

You need to monitor the real storage and the I/O in your system. If you do not have enough main (and in VM expanded) storage to support additional virtual storage, you may dramatically increase the load on your I/O subsystem and on your processor. In extreme cases this can lead to *thrashing*. (The processor and I/O subsystem spend most of their resources moving pages from main to auxiliary storage and have little or no resources left for practical work.)

Storage Above 16MB (31 Bit Addressing)

In **VM**, your database machine's address space can be larger than the normal 16MB limit, if you run it in either XA or XC mode. In **VSE**, you can use a database partition larger than the normal 16MB limit, if you are using VSE/ESA release 1.3 or later, and you are using supervisor mode ESA or VMESA.

(As mentioned in “Address Space Size,” anytime you increase your virtual storage, you can realize significant performance improvements by using storage above 16MB, but only if you have the resources to support it.)

Saved Segments (VM Only)

A *saved segment* is a range of pages of virtual storage you can define to hold data or reentrant code (programs), which can be shared by multiple virtual machines. For detailed information, on how to create saved segments and which DB2 Server for VM components can be loaded into them, refer to the *DB2 Server for VM System Administration* manual.

Loading frequently used DB2 Server for VM components in saved segments has several advantages:

- Because several users can access the same physical storage, real storage use is minimized.
- Using saved segments decreases the I/O rate and DASD paging space requirements, thereby improving virtual machine performance.
- Saved segments attached to a virtual machine can reside above its defined virtual storage. This allows the virtual machine to use its defined storage for other purposes.

For more information on saved segments, refer to the *DB2 Server for VM System Administration* manual.

DB2 Data Spaces Support (VM/ESA only)

By taking advantage of large virtual storage areas called data spaces, DB2 Data Spaces Support can dramatically improve the performance of your application server.

A program running in a machine's primary space can dynamically create additional address spaces for data, called *data spaces*. Like a virtual machine's primary address space, a data space is a virtual space with its pages in main storage, in expanded storage, and on DASD. However, unlike a primary space, you cannot run a program in a data space. Also, in DB2 Server DSS the VM paging system manages data space pages differently than virtual machine pages. This means that DB2 Server DSS data spaces do not use system paging DASD. (Internal dbspaces can use *unmapped* data spaces, which can use system paging DASD. Refer to the *DB2 Data Spaces Support* manual.)

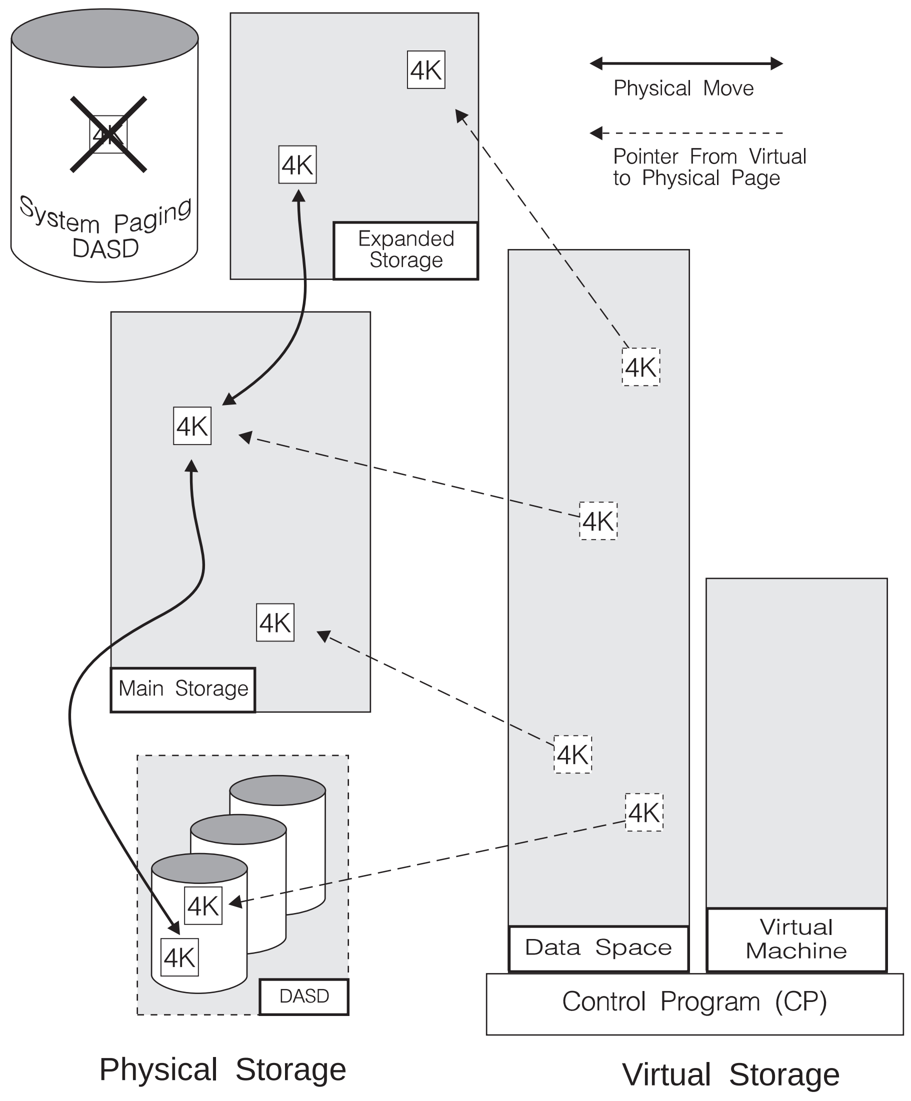


Figure 12. Data Spaces Storage

With DB2 Server DSS, if there is no longer any free space in main or expanded storage, the operating system will simply replace an old data space page in main or expanded storage with a new page. If the old page is needed again, it is reread from its original DASD source. If the old page was modified while it was in main storage, the operating system ensures that the modified page is written back to its original DASD source before it is replaced.

This expands a machine's effective virtual storage by providing additional addresses for data, thereby making room in the primary space for a larger database manager, refer to “Address Space Size” on page 47.

Also, the paging system used by DB2 Data Spaces Support can be much faster and more efficient than the standard DASD I/O system. For more information, refer to “DB2 Data Spaces Support (VM/ESA only)” on page 92.

For more information on data spaces, refer to the *DB2 Data Spaces Support* manual.

Virtual Disk Support for VSE/ESA for Internal Dbspaces

Your internal dbspaces can use a virtual disk to improve their performance. *Virtual Disk Support* lets you use a data space as a virtual disk. A virtual disk is much faster than a conventional disk because it uses main storage instead of DASD. The virtual disk appears to any program or job as just another disk, only faster.

However, virtual disk storage is temporary. Anything in a virtual disk is lost whenever the VSE operating system is restarted. For this reason, **DO NOT** use virtual disks for anything other than internal dbspaces. These dbspaces are only used as temporary workspace, so it does not matter if their contents are lost. The storage pool containing the virtual disk must NOT be used for any permanent dbspaces.

While the use of virtual disks is limited to internal dbspaces, they can improve the performance of index creation, joins, sorts, and other operations that require temporary workspace.

Remember, as mentioned in “Address Space Size” on page 47, anytime you increase your use of virtual storage, you can realize significant performance improvements, but only if you have the real storage to support it.

Note: It is recommended that database generation be done with real minidisks only. If you decide to generate a database which uses a virtual disk, ensure that the virtual disk is used in a pool containing only internal dbspaces.

Using Virtual Disks with Internal Dbspaces: To use a virtual disk with internal dbspaces, you must:

1. Take an archive of your database, before making any changes to it.
2. Modify the IPL procedure and the background initialization procedure to create a virtual disk.
3. Define a VSAM user catalog and dbextent on the virtual disk.
4. Add a label for the dbextent in the cataloged procedure and add the dbextent to a **NEW** storage pool that will contain only internal dbspaces.
5. Move some of the dbextents from the original pool that contained the internal dbspaces into the new pool or define additional physical dbextents to be added to the new pool.
6. Add internal dbspaces to the new storage pool.
7. Backup the VSAM user catalog defined on the virtual disk so that it can be restored whenever the VSE system is restarted.
8. Modify the application server startup job to restore the VSAM user catalog if the VSE system has been restarted since the VSAM user catalog and dbextent were created.
9. Archive your database after making the above changes so that you have an archive that reflects these changes.

A detailed example of how to complete these steps appears below. For more information on virtual disks in VSE/ESA, refer to the *IBM VSE/ESA Planning* manual and the *IBM VSE/ESA Extended Addressability* manual.

Note: All of the following steps should be read before executing any of them.

Step 1, Archive Your Database: This will be needed if problems arise during the setup for using a virtual disk and you need to restore your database to its previous state without any virtual disk.

Step 2, Modify IPL Procedure: Modify the IPL procedure to do the following:

- Include ADD statements for the virtual disk addresses
- Increase VSIZE and page data set allocation to accommodate the new virtual disk.

For example, consider a system where a 20MB virtual disk is added to a VSE system with VSIZE=75MB:

```
⋮
009,$$A$SUPX,VSIZE=95M,VIO=576K,VPOOL=194K,LOG
⋮
ADD 900:906,FBAV
⋮
DPD VOLID=DOSRES,CYL=209,NCYL=100,TYPE=N,DSF=N
DPD VOLID=DOSRES,CYL=398,NCYL=8,TYPE=N,DSF=N
DPD VOLID=DOSRES,CYL=410,NCYL=29,TYPE=N,DSF=N
⋮
DLA NAME=AREA1,VOLID=DOSRES,CYL=60,NCYL=3,DSF=N
SVA PSIZE=640K,SDL=300,GETVIS=768K
/+
/*
⋮
```

This example increases the VSIZE of the VSE operating system from 75MB to 95MB. It reserves virtual addresses 900 through 906 for fixed block architecture virtual disks, and it sets aside an additional 29 cylinders of 3390 DASD on DOSRES volume starting at address 410 for system paging DASD. (108 cylinders were already being used at addresses 209, and 398. Also, there are 180 4KB pages in every 3390 cylinders. So 29 cylinders is equal to 20MB.)

Step 3, Define and Initialize a Virtual Disk: Create a procedure (to be invoked as part of the background initialization JCL, for example $0JCL) that will do the following:

- Define data space size using SYSDEF command
- Initialize virtual disks using // VDISK command.

For example:

```
⋮
*  DEFINE THE SIZE OF THE DATASPACE
// SYSDEF DSPACE,DSIZE=20M
*  DEFINE AND INITIALIZE EACH VIRTUAL DISK
// VDISK UNIT=900,BLKS=40320,VOLID=QPVDS1,VTOC=008
/*
/+
⋮
```

This example reserves approximately 20MB of virtual storage for virtual disks. (Remember you do not need 20MB of real storage to support 20MB of virtual storage.) The virtual disk at address 900 uses approximately 20MB of that storage (40320 512-byte blocks, with 8 512-byte blocks of that reserved for the VTOC). (The virtual disk addresses were defined in Step 2.)

Note: Because of the structure of a virtual disk, blocks must be allocated in multiples of 960 blocks. So instead of 2048 512-byte blocks for 1MB of storage, you can only allocate 1920 blocks.

Step 4, Define a Backup File: Define a sequential file on a real disk for use later when backing up the VSAM user catalog (defined on the next step on a virtual disk). For example:

```
⋮
*  DEFINE A SEQUENTIAL FILE FOR VDISK UCAT BACKUP
// EXEC IDCAMS,SIZE=AUTO
   DEFINE NONVSAM (NAME(VDISK1.UCAT.BKUP) -
          DEVICETYPES(3390) VOLUMES(SYSWK1))
/*
⋮
```

This example creates a backup file called `VDISK1.UCAT.BKUP`.

Note: The above JCL does not work if it is used on FBA DASD. If only FBA DASD is used, do one of the following:

- Skip this step
- Use a VSAM ESDS file for the backup file.

Step 5, Define a VSAM User Catalog: Define a VSAM user catalog on the virtual disk using the DEDICATE option. For example:

```
⋮
*  DEFINE A VSAM USER CATALOG
// EXEC IDCAMS,SIZE=AUTO
 DEFINE USERCATALOG ( -
        NAME (VDS1.USER.CATALOG  ) -
        DEDICATE -
        VOLUME (QPVDS1))
/*
⋮
```

This example creates a VSAM user catalog called `VDS1.USER.CATALOG` on volume `QPVDS1` and dedicates the entire volume for this VSAM user catalog. (Volume `QPVDS1` was defined in Step 3.)

Step 6, Define a Virtual Disk Dbextent: Define a dbextent (VSAM cluster) on the virtual disk. For example:

```
⋮
*  ADD CLUSTERS FOR THE DATA BASE VIRTUAL DISK
// DLBL VDSUC1,'VDS1.USER.CATALOG',,VSAM
*  DEFINE CLUSTERS
// EXEC IDCAMS,SIZE=AUTO
   DEFINE CLUSTER (NAME(SQL34.DDSK8.VDSK.DB) NONINDEXED REUSE -
            CNVSZ (4096) BLOCKS(39360) VOL(QPVDS1) -
            RECSZ (4089 4089) SHR(2)) CAT(VDS1.USER.CATALOG)
/*
⋮
```

This example defines a dbextent named SQL34.DDSK8.VDSK.DB in the VSAM user catalog VDS1.USER.CATALOG (The VSAM user catalog was defined in Step 5.)

Step 7, Add a Label for the Virtual Disk Dbextent: Update the cataloged procedure to include a DLBL statement for the new dbextent. For example:

```
⋮
*  CATALOG DATABASE DBEXTENT LABELS
// EXEC LIBR,PARM='MSHP'
ACCESS S=IJSYSRS.SYSLIB
CATALOG DTLDVDSK.PROC R=Y
*  *****************  SQL/DS DBEXTENT LABELS  **************
// DLBL TSQLUC,'TSQL.USER.CATALOG',,VSAM
// DLBL VDSUC1,'VDS1.USER.CATALOG',,VSAM
// DLBL BDISK,'SQL34.BDISK.DTLD.DB',,VSAM,CAT=TSQLUC
// DLBL LOGDSK1,'SQL34.LOGDSK1.DTLD.DB',,VSAM,CAT=TSQLUC
// DLBL DDSK1,'SQL34.DDSK1.DTLD.DB',,VSAM,CAT=TSQLUC
⋮
// DLBL DDSK7,'SQL34.DDSK7.DTLD.DB',,VSAM,CAT=TSQLUC
// DLBL DDSK8,'SQL34.DDSK8.VDSK.DB',,VSAM,CAT=VDSUC1
/+
/*
⋮
```

This example adds a DLBL statement for DDSK8 that identifies dbextent SQL34.DDSK8.VDSK.DB. (The dbextent was created in Step 6.) DDSK7 is one of the dbextents that belonged to the original internal dbspace storage pool.

Step 8, Add Dbextents to a New Storage Pool: Add dbextents to the new storage pool by following instructions included in the *DB2 Server for VSE System Administration* manual. (Refer to “adding and deleting dbextents.”)

To avoid using too much real storage, it is recommended that you include at least two dbextents in the new pool. The first must be the virtual disk. The second, should be a physical dbextent that can accommodate the overflow from the virtual disk. You can use some of the dbextents from the original pool that contained the internal dbspaces (in this example DDSK7). Make the total size of both dbextents large enough to accommodate your current internal dbspaces and make the virtual disk as large as possible without over committing real storage.

Also, ensure that you add the virtual dbextent before you add any physical dbextents. The database manager searches the dbextents for a free page in the order that they were added.

Attention: Do not accidentally place the virtual disk in an existing storage pool that contains anything other than internal dbspaces. You will lose valuable data and a full database or pool level restore will be required.

The following is an example of the ARISADD member, which specifies how procedure ARIS250D will add and delete dbextents to and from pools:

```
POOL 9
ADD 8 9
DELETE 7
ADD 7 9
ARCHIVE
```

This example adds dbextent 8 (DDSK8) to storage pool 9. (DDSK8 was identified in Step 7.) Ensure that you add the virtual disk dbextent to a **new** storage pool (reserved only for internal dbspaces). It also removes dbextent 7 (DDSK7) from the original pool that contained the internal dbspaces. It then adds it to the new pool (pool 9) that contains the virtual disk dbextent.

Step 9, Back Up the Virtual Disk VSAM User Catalog: Back up the VSAM user catalog defined on the virtual disk into the sequential file on a real disk. For example:

```
⋮
// LIBDEF PHASE,SEARCH=IJSYSRS.SYSLIB
*  THIS JOB UNLOADS A VSE/VSAM CATALOG USING THE REPRO COMMAND
// DLBL IJSYSCT,'VSAM.MASTER.CATALOG',,VSAM
// ASSGN  SYS001,DISK,VOL=SYSWK1,SHR
// DLBL   CATOUT,'VDISK1.UCAT.BKUP',999
// EXTENT SYS001,SYSWK1,1,0,33315,75
// DLBL   IJSYSUC,'VDS1.USER.CATALOG',                              X
               ,VSAM
// EXEC   IDCAMS,SIZE=AUTO
       REPRO   INFILE (IJSYSUC) -
               OUTFILE (CATOUT -
                 ENVIRONMENT ( -
                   BLOCKSIZE (2068) -
                   RECORDFORMAT (VARBLK) -
                   RECORDSIZE (516) -
                 ) -
               )
/*
⋮
```

This example unloads the VSAM user catalog `VDS1.USER.CATALOG` to the backup file `VDISK1.UCAT.BKUP`. (The backup file was created in Step 4.) The backup file will be used to restore the VSAM user catalog on the virtual disk whenever the VSE system is restarted. Restoring the VSAM user catalog redefines the VSAM space and cluster previously defined on the virtual disk and sets the high used RBA to what is was before the system restart, thus allowing the database manager to successfully use it.

Step 10, Add Internal Dbspaces to the New Pool: Invoke the application server to add only internal dbspaces into this new pool. For example:

```
⋮
*  ADD INTERNAL DBSPACES TO THE DATABASE
// LIBDEF *,SEARCH=PRD2.SQL340
// EXEC PROC=DTLDVDSK
// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,STARTUP=S'
   INTERNAL 50 1024 9
/*
⋮
```

This example JCL creates 50 internal dbspaces, each of 1024 4KB pages, in storage pool 9. (Storage pool 9 was created in Step 8.)

Step 11, Add Conditional JCL to Application Server Startup: Create conditional JCL that does the following:

- Check that the dbextent defined earlier on the virtual disk still exists (1 and 2). If it does not, do the following:
  - Disconnect the VSAM user catalog from the master catalog (3)
  - Redefine the VSAM user catalog on the virtual disk (4)
  - Restore the VSAM user catalog from the sequential file (5)
  - If the restore fails for any reason cancel the job (6).
- Start the application server.

For example:

```
⋮
// DLBL IJSYSCT,'VSAM.MASTER.CATALOG',,VSAM
// ASSGN SYS001,DISK,VOL=SYSWK1,SHR
// DLBL VDSBKUP,'VDISK1.UCAT.BKUP'
// EXTENT SYS001,SYSWK1,1,0
// DLBL IJSYSUC,'VDS1.USER.CATALOG',,VSAM
// EXEC IDCAMS,SIZE=AUTO
   LISTCAT CAT(VDS1.USER.CATALOG) ENT(SQL34.DDSK8.VDSK.DB) ALL  (1)
   IF LASTCC NE 0 THEN DO                                       (2)
      EXPORT VDS1.USER.CATALOG DISCONNECT                       (3)
      DEFINE USERCATALOG ( NAME(VDS1.USER.CATALOG) -            (4)
                  DEDICATE VOLUME (QPVDS1))
      IF LASTCC NE 0 THEN CANCEL JOB
      REPRO  INFILE (VDSBKUP ENVIRONMENT -                      (5)
                 (BLOCKSIZE (2068) -
                  RECORDFORMAT (VARBLK) -
                  RECORDSIZE (516))) -
             OUTFILE (IJSYSUC)
      IF LASTCC GT 4 THEN CANCEL JOB                            (6)
   END
/*
⋮
```

Place this sample JCL in front of your current application server startup job. It checks for the existence of dbextent SQL34.DDSK8.VDSK.DB in the VSAM user catalog VDS1.USER.CATALOG. If it no longer exists, the JCL restores the VSAM user catalog from the backup created in Step 4. The application server will then start normally and will use the virtual disk for internal dbspaces.

Step 12, Archive Your Database: The ADD DBEXTENT and DELETE DBEXTENT operations are not recorded in the log. Since these operations update the directory (but not the database itself), problems can be encountered if you normally archive the database and then try to restore that archive with the ADD DBEXTENT or DELETE DBEXTENT occurring in between the archive and the restore. Archiving before and after you make changes to the virtual disk will assist you if problems occur. For more information, refer to the *DB2 Server for VSE System Administration* manual.

Virtual Disk Support for VM/ESA for Internal Dbspaces

Your internal dbspaces can use a virtual disk to improve their performance. *Virtual Disk Support* lets you use a data space as a virtual disk. A virtual disk is much faster than a conventional disk because it uses main storage instead of DASD. The virtual disk appears to any program or job as just another disk, only faster.

However, virtual disk storage is temporary. All data on a virtual disk is lost when it is detached from a user ID or when the user ID logs off. For this reason, **DO NOT** use virtual disk for anything other than internal dbspaces. These dbspaces are only used as temporary workspace, so it does not matter if their contents is lost. The storage pool containing the virtual disk must **NOT** be used for any permanent dbspaces.

While the use of a virtual disk is restricted to internal dbspaces, they can be used to improve the performance of index creations, joins, sorts, and other operations that require temporary workspace.

Remember, anytime you increase your use of virtual storage, you can realize significant performance improvements, but only if you have the real storage to support it.

Note: It is recommended that database generation be done with real minidisks only. If you decide to generate a database which uses a virtual disk, ensure that the virtual disk is used in a pool containing only internal dbspaces.

Using Virtual Disks with Internal Dbspaces: To use a virtual disk with internal dbspaces, you must:

1. Take an archive of your database, before making any changes to it.
2. Define a virtual disk in the database manager's VM Directory entry.
3. Run the SQLADBEX EXEC to add the virtual disk as the first dbextent of a **NEW** storage pool. The virtual disk **must** be the first dbextent in the new pool. If you have dbextents in the old storage pool where your internal dbspaces are currently defined, you should delete some of these dbextents from the old pool and add them to the new pool. Alternatively, you could add one or more real disk dbextents to the new pool.
4. Run the SQLADBSP EXEC to *move* your internal dbspaces to the new pool.
5. Modify the 'CP LINK' command for the virtual disk in the 'dbname SQLFDEF Q' file.
6. Modify the database manager's PROFILE EXEC or database start up EXEC to CMS FORMAT and RESERVE the virtual disk and make a duplicate LINK to this virtual disk. (this is explained further in the following detailed steps)
7. Archive your database after making the above changes, so that you have an archive that reflects these changes.

A detailed example of how to complete these steps appears below. Please read all the steps before executing any of them.

Step 1, Archive Your Database: This will be needed if problems arise during the setup for using a virtual disk and you need to restore your database to its previous state without any virtual disk.

Step 2, Define a Virtual Disk in the Database Manager's VM Directory Entry: Determine the size of the virtual disk to be added. It should be large enough for most sorts, but must not be so large as to cause excessive VM paging. Define this virtual disk in the VM Directory, similar to:

```
      MDISK 0329 FB-512 V-DISK nnnnnnnn M
```

For information about defining a virtual disk in a VM Directory, refer to the *VM/ESA: Planning and Administration* manual.

Step 3, Add Dbextents to a New Storage Pool: Add the virtual disk, and possibly other dbextents, to a new storage pool. For information about using the SQLADBEX EXEC, refer to the *DB2 Server for VM System Administration* manual.

To avoid using too much real storage, it is recommended that you include at least two dbextents in the new pool. The virtual disk **must** be the first dbextent in the new pool. Other dbextents should be real minidisks to accommodate the overflow from the virtual disk. You can use some of the dbextents from the pool that originally contained the internal dbspaces. Make the total size of the dbextents in the new pool large enough to accommodate your current internal dbspaces.

Also, ensure that you add the virtual dbextent before you add any real minidisks. The database manager searches the dbextents for a free page in the order they were added.

If you are also using DB2 Server DSS, you should use SEPINTDB=Y to use your internal dbspaces in a data space, instead of using a virtual disk. If you still want to use a virtual disk, you ***MUST*** update your Storage Pool Specification file to specify `'BLK'` and `'SEQ'` for the storage pool containing the virtual disk. VM does not allow a virtual disk to be mapped to a data space. In this case, message ARI2018E will be issued identifying the virtual disk address. For more information, see the *DB2 Data Spaces Support* manual.

Attention: Do not accidentally place the virtual disk in an existing storage pool containing permanent dbspaces. You will lose valuable data and a full database or pool level restore will be required.

When you run the SQLADBEX EXEC, you specify the actions to be taken by answering the prompts. When you see message ARI6145D, reply 1 (yes), to view the `'dbname SQLADBEX A'` file created from the prompts.

Assume you have the following set up and are changing to use a virtual disk for internal dbspaces:

- Your internal dbspaces are currently defined in pool 3, which also contains permanent dbspaces.
- Pool 3 currently has 3 dbextents, addresses 324, 325, and 326, which correspond to dbextent numbers 4, 5, and 6.

- You have a virtual disk defined at address 329 and you want to add it to the new pool number 8, as dbextent number 9.
- You want to 'move' dbextents 5 and 6 in pool 3 to the new pool 8.
- You DO want to take an archive after these changes.

  Note: Running the SQLADBEX EXEC will cause a break in the continuity of your log archives, so a database archive should **always** be taken.

After starting the SQLADBEX EXEC and entering the information at the prompts, reply 1 (yes) to message ARI6145D. This will display the 'dbname SQLADBEX A' file in XEDIT; its contents can be reviewed, modified, or both. This file specifies the sequence of actions that SQLADBEX will perform. Given our assumptions above, the file will appear as follows:

```
ADD 9 8                               <-- add virtual disk to pool 8
DELETE 5 3                            <-- delete dbextent 5 from pool 3
DELETE 6 3                            <-- delete dbextent 6 from pool 3
ADD 5 8                               <-- add dbextent 5 to pool 8
ADD 6 8                               <-- add dbextent 6 to pool 8
ARCHIVE                               <-- an archive will be taken
```

Refer to the *DB2 Server for VM System Administration* manual for more details, cautions and warnings concerning the adding and deleting of dbextents.

The last step of the SQLADBEX EXEC will update the 'dbname SQLFDEF Q' file, to match dbextents that have been added, deleted, or both. The 'CP LINK' command for the virtual disk in this file must be updated; this is documented in a following step.

Step 4, Move the Internal Dbspaces to the New Storage Pool: This step will 'move' the internal dbspaces from the old pool to the newly added pool which contains the virtual disk, by using the SQLADBSP EXEC. You may also add permanent dbspaces to the database at this time (except into the new pool) or you can simply redefine the pool where internal dbspaces will be placed. Remember, this new pool, with the virtual disk dbextent, can ONLY contain internal dbspaces. For information about using the SQLADBSP EXEC, refer to the *DB2 Server for VM System Administration* manual.

Step 5, Modify the 'dbname SQLFDEF Q' File: In this step, you will edit and modify the 'dbname SQLFDEF Q' file to change the 'CP LINK' mode for the virtual disk dbextent. This is required to allow the virtual disk to only be formatted and reserved once per IPL CMS of the database manager. If this step is not performed, the virtual disk must be formatted and reserved prior to each start up of the database manager (for example, SQLSTART).

This is done by having the virtual disk linked to the database manager virtual machine twice. When the 'dbname SQLFDEF Q' file detaches the virtual disk, the second link remains attached to the machine and the formatting of the virtual disk is not lost. This second link and the formatting is done from the database manager's PROFILE EXEC (this is set up in the next step).

Be sure you have the production minidisk (normally 'Q') accessed R/W. XEDIT the 'dbname SQLFDEF Q' file. Locate the line containing the 'CP LINK *userid cuu cuu* W' statement for the address ('*cuu*') of the virtual disk. Change the CP LINK 'MODE' character from 'W' to 'M'.

Attention: If you delete the virtual disk extent and add it again (through the SQLADBEX EXEC), you **must** again change the LINK Mode character from `'W'` to `'M'`.

Step 6, Modify the PROFILE EXEC: In this step you will modify the database manager's PROFILE EXEC so that the virtual disk will be CMS formatted and reserved each time the database manager virtual machine IPL's CMS. In addition, a second link to the virtual disk will be set up (see the previous step).

Attention: If an error occurs such that the virtual disk is not usable, the database cannot be brought up. In this situation, you must correct the error to make the virtual disk usable, or you must replace the virtual disk with a real minidisk at the same address (and at least the same size). The replacement minidisk must be formatted and reserved, as usual, before the database is brought up.

It is recommended that a separate EXEC be created to perform the LINK, FORMAT and RESERVE commands, and that this EXEC be called from the PROFILE EXEC. You can place the following statements in your PROFILE EXEC to initialize the virtual disk for usage:

```
'EXEC PREPVDSK'                /* Call EXEC to Prepare Virtual Disk */
If rc ¬= 0 Then Do; Say "PREPVDSK rc =" rc; Exit rc; End
```

Note: The `'If'` statement above will cause the PROFILE EXEC to end if an error is returned from the PREPVDSK EXEC. This assumes that the PROFILE EXEC will eventually invoke the SQLSTART EXEC after the virtual disk has been initialized. This is done because the database cannot be started if the virtual disk is not properly initialized. You may need to tailor this processing to suit you particular operational environment.

The following is a sample `'PREPVDSK EXEC'`:

```
/* REXX */ Trace 'O ';   Address 'COMMAND'
/* Use this EXEC to FORMAT and RESERVE a Virtual Disk,              */
/* that is used as the FIRST Dbextent of a Storage Pool containing   */
/* ONLY INTERNAL DBSPACES.                                           */
/*                                                                   */
/* ATTENTION: This process, to use a Virtual Disk, requires that the */
/*            CP Link Mode letter be changed from 'W' to 'M' in the  */
/*            SQLFDEF file for the CP LINK command for the DATABASE  */
/*            Address of the Virtual Disk.                           */
/*                                                                   */
/*  This EXEC should be called from the PROFILE EXEC of the Database */
/* Virtual Machine, to prepare the Virtual Disk for use.             */
/*           (once per LOGON/IPL of the Database Machine)            */
/*                                                                   */
/*  The Virtual Disk MDISK is Linked R/O with an unused address,     */
/* (which is refered to below as the PERANENT ADDRESS) so that       */
/* that subsequent Detaches of the normal address (refered to below  */
/* as the SQLFDEF ADDRESS) by this EXEC and the SQLFDEF file will    */
/* NOT lose the FORMAT/RESERVE information.                          */
/*                                                                   */
/*  The Virtual Disk MDISK is Linked again, R/W, with its SQLFDEF    */
/* address, for the FORMAT/RESERVE processing. This address is then  */
/* Detached.  It will be Linked again later by the SQLFDEF file when */
/* the database machine runs the SQLSTART EXEC.                      */
/*                                                                   */
/*  If the R/O Link of the Virtual Disk is detached by mistake,      */
/*  you MUST run this EXEC before running SQLSTART again.            */
/*                                                                   */
/********* UPDATE THE FOLLOWING 5 VARIABLES AS APPROPRIATE:  *********/
dbname = 'dbname  '             /* Database Name                     */
pdisk  = '0cuu'                 /* Virtual Disk PERMANENT Address    */
vdisk  = '0cuu'                 /* Virtual Disk SQLFDEF  Address     */
vlabel = 'DDKnn   '             /* Virt Disk Label (Dbextent Number)*/
ufm    = 'Z'                    /* Unused Filemode Letter            */

z=Diagrc(8?,'CP DETACH' vdisk)  /* Be sure SQLFDEF Addr is NOT Linked */
z=Diagrc(8?,'CP LINK *' vdisk pdisk 'RR')   /* Get PERMANENT R/O Link */
Parse Var z cprc . ; If cprc ¬= 0 Then Exit rc
z=Diagrc(8?,'CP LINK *' vdisk vdisk 'M')    /* Get SQLFDEF   R/W Link */
Parse Var z cprc . ; If cprc ¬= 0 Then Exit rc
'SET CMSTYPE HT'; 'RELEASE' ufm; 'SET CMSTYPE RT'
Push vlabel
Push '1'
'FORMAT' vdisk ufm '(BLKSIZE 4096 NOERASE'        /* FORMAT the Vdisk */
If rc ¬= 0 Then Exit rc
Push '1'
'RESERVE' dbname vlabel ufm                      /* RESERVE the Vdisk */
If rc ¬= 0 Then Exit rc
'SET CMSTYPE HT'; 'RELEASE' ufm; 'SET CMSTYPE RT'
z=Diagrc(8?,'CP DETACH' vdisk)   /* Detach the SQLFDEF address again, */
Exit 0        /* ... it will be re-Linked by SQLFDEF during SQLSTART. */
```

Step 7, Archive Your Database: Neither the ADD DBEXTENT nor the DELETE DBEXTENT operation is recorded in the log. Since these operations update the directory (but not the database itself), problems can be encountered if you normally archive the database and then try to restore that archive with the ADD DBEXTENT or DELETE DBEXTENT occurring in between the archive and the restore. For more information about this problem, refer to the *DB2 Server for VM System Administration* manual. Archiving before and after you make changes to the virtual disk will assist you if problems occur.

DASD Storage

How you manage DASD storage affects performance in four ways:

How Storage Is Divided
How you divide a limited amount of storage between indexes and data, and among dbspaces and among storage pools determines to a large degree how each will perform in different situations.

Wasted Storage
Wasted storage in itself may not affect the performance of the system that is using it, but it may represent a resource that could be used to improve performance elsewhere.

Distributing DASD I/O
How well you balance the demand for DASD I/O across several DASD devices, controllers and channels can affect how fast the database manager can retrieve information from DASD.

Running out of Storage
While running out of storage can disrupt your users because you are forced to bring down the application server to add storage, just getting close can degrade performance. (If you reach the application server's short on storage level you trigger unnecessary SOSLEVEL checkpoints, refer to “Short on Storage Cushion” on page 62.)

In VSE

The directory, logs, and dbextents are VSAM Entry Sequenced Data Sets (ESDS) with a control interval size of 512 bytes for the directory and 4096 bytes for the logs and dbextents. The database manager uses VSAM Control Interval processing to read and write records to the VSAM ESDS.

In VM

The directory, logs, and dbextents are CMS reserved minidisks with a blocksize of 512 bytes for the directory and 4096 bytes for the logs and dbextents (the directory may have a blocksize of 4096 bytes, if Data Spaces Support is used). These minidisks have CMS-like files that are in a format to be used with the IUCV *BLOCKIO I/O system that reads and writes records to these files. These minidisks are called reserved because they have been processed by the CMS RESERVE command. It 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.

Mapping of Dbspaces to DASD

Logical dbspaces must be mapped to physical dbextents on DASD. The database manager does this by maintaining page map table(s), for each dbspace, which map a given dbspace page to its location on DASD. The page map table is stored in the DB2 Server for VSE & VM directory. There can be multiple page map tables per dbspace. Each page map table block is equivalent to 128 pages in a dbspace.

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.

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, the database manager allocates pages in a storage pool in sequence, usually allocating all the pages in one dbextent before using the next dbextent. **With the DB2 Server DSS Feature**, the database manager can distribute pages evenly across all the extents, refer to “Striping” on page 93.

Managing Storage Pool Space

Short on Storage Cushion

The short on storage (SOS) cushion helps you avoid completely filling a storage pool. If the database manager is running:

- In SUM with LOGMODE not equal to N or
- In MUM

and the percentage of space available in one pool falls below the SOS level, the database manager performs a checkpoint to release shadow pages (refer to “Shadow Pages” on page 66). If this does not release enough pages to fall below the SOS level, a warning message is sent to the operator. If you are already short on storage and need more storage in a pool, refer to “Running out of Dbspace Pages” on page 68.

Tuning Parameter (SOSLEVEL): While it is acceptable to reach the SOSLEVEL initialization parameter occasionally, do not let any of your storage pools hover around it. SOSLEVEL initiated checkpoints are unnecessary overhead. If they occur frequently, it is a good sign that you should either free space in the overloaded pools, or increase their size by adding dbextents.

Do not just lower the SOSLEVEL to avoid checkpoints. If you have less than 10% free space in a storage pool the database manager will initiate a checkpoint during a rollback even if you set SOSLEVEL below 10%.

Instead, set SOSLEVEL to at least 15% and try to keep at least 25% free space in each storage pool. This ensures that even if you accumulate a large number of shadow pages in a pool, the database manager will not initiate unnecessary checkpoints.

Performance Indicator (SHOW POOL): Use the SHOW POOL operator command to determine what percentage of each storage pool is full. If the free space in a pool falls below the SOSLEVEL parameter, the `SHORT ON STORAGE` flag appears in the report for that pool. (Refer to page 26.)

Also watch the CHKPOINT counter. If you notice an excessive number of checkpoints occurring during insert or update transactions, the database manager may be doing the following:

- Reaching the SOSLEVEL and performing a checkpoint.
- The checkpoint releases just enough shadow pages for the pool to fall below the SOSLEVEL.
- Subsequent processing quickly refills the pool to the SOSLEVEL and another checkpoint is taken.

The database manager may spend so much time processing SOSLEVEL checkpoints that it can perform little useful work. Changing the SOSLEVEL will not help this problem. Instead, add storage to the pool, refer to “Running out of Dbspace Pages” on page 68.

Types of Pages

There are four types of pages that can reside in a dbspace:

Header Pages
These pages contain an inventory of all the dbspace attributes, tables and indexes created in the dbspace.

Data Pages
These pages contain table rows that may be from several different tables in the dbspace.

Index Pages
These pages contain index entries. Each page contains information for one specific index on one specific table.

Shadow Pages
These pages are used to ensure that the database manager can reconstruct changes to the database after a system failure, refer to “Shadow Pages” on page 66.

Number of Header Pages

Because there are never many header pages in a single dbspace, never more than eight, they do not represent a significant impact on performance. We suggest that they remain at the system default of eight.

Proportion of Index to Data and Header Pages

The amount of space you reserve for index pages, depends on how many indexes you expect to create and on the number and size of columns included in the indexes. You can use the following as a guideline:

Read-Only Data
: Since many indexes are recommended for read-only data, you should reserve at least the default of 33% and as much as 50% for the pages in a dbspace for index pages.

Update Intensive Data
: You can reserve less than the default of 33% of the pages in a dbspace for index pages, since you may not use as many indexes for this type of data. (It is expensive to update indexes every time data is updated, so it is suggested that you use fewer indexes with this data.)

If you are unsure whether your data is read-only or update intensive, use the default of 33% index pages.

Tuning Parameter (PCTINDEX): You can set the proportion of index to data and header pages in a dbspace when you acquire it using the PCTINDEX parameter. For example, the following statement acquires a dbspace and reserves 50% of its pages as index pages:

```
ACQUIRE PUBLIC dbspace NAMED test_dbspace (PCTINDEX=50)
```

Performance Indicator (PCTINDX): To determine the current percentage of reserved index pages in a dbspace, look in the PCTINDX column of the SYSTEM.SYSDBSPACES catalog table for your dbspace.

Free Space in Data Pages

You can reserve a percentage of each data page for updates that make the changed row longer than it was before. This free space is not used for inserts. You can reclaim the free space for inserts through an ALTER DBSPACE statement. The percentage of free space you choose will depend on the type of activity being carried out on the data in the dbspace:

High Insert/Low Update Activity
: This is the situation where there will be few updates, or all columns are fixed length and non-nullable in the tables. Here, you would set the percentage of free space to a high value before loading the data; then lower it to a low value. The difference between the original value and the final value can then be used by insert activity.

Low Insert/High Update Activity
: In this situation, PCTFREE should be set to a low to medium value, depending on the likelihood of updated rows increasing in length. (Increase PCTFREE in proportion to the likelihood of increasing row length.) The space saved by PCTFREE will be used by the update activity only if the update increases the size of the row and the free space will accommodate the new row.

Low Insert/Low Update Activity Or Read-Only Data
: Read-only data is data that is loaded into a dbspace and then never modified or updated, only retrieved using query statements. In this situation, set PCTFREE to a low value or zero, before you load any data into the dbspace.

High Insert/High Update Activity

In this situation, set PCTFREE to a high value while you load data into the database and then lower it. This would allow space for use by both update and insert activities.

One purpose of PCTFREE is to minimize overflow because of 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 the row has to be moved again, the pointer is set to mark the newest location. Therefore, the database manager never reads more than two pages for one row.

The other purpose of PCTFREE is to reserve space on a page when data is loaded. After loading, PCTFREE can be lowered to allow the free space to be used for inserts (to help keep the data clustered).

Tuning Parameter (PCTFREE): You can set the percentage of space on each page that is kept free when data is inserted in the dbspace, when you acquire it using the PCTFREE parameter. For example, the following statement acquires a dbspace and reserves 20% of the space on each page for inserts:

```
ACQUIRE PUBLIC dbspace NAMED test_dbspace (PCTFREE=20)
```

You need to know the PCTFREE setting for a dbspace before you can calculate the number of rows you can effectively store on a single data page. For a complete description of how to calculate this, refer to “Estimating the Number of Data Pages Required” in the *DB2 Server for VM Database Administration* or the *DB2 Server for VSE Database Administration* manuals.

Performance Indicator (FREEPCT): To determine the current percentage of space on each page that is kept free when data is inserted in the dbspace, look in the FREEPCT column of the SYSTEM.SYSDBSPACES catalog table for your dbspace.

Performance Indicator (AVGROWLEN): To determine average length of the rows in a table, look in the AVGROWLEN column of the SYSTEM.SYSCATALOG catalog table.

Performance Indicator (NOVERFLOW): To determine how many rows are overflowing onto new pages, look in the NOVERFLOW column of the SYSTEM.SYSCATALOG catalog table. As a rule of thumb, if the number of overflow rows in a table (NOVERFLOW) exceeds 5% of the total number of rows in the table (ROWCOUNT), it is probably time to reorganize the table. Refer to “Reorganizing Data” on page 73.

If you decide to reorganize the table because of this, you may also want to alter the dbspace to give it a larger PCTFREE value.

Free Space in Index Pages

You can reserve a percentage of space in each index page for future index entries, which allows index maintenance to take place without splitting of index pages. Its default is 10 percent, which is a good value for most purposes. If you expect much insert or update activity after the creation of the index, you might want to override the default by setting the percentage to a higher value. If you expect no insert or update activity after the creation of the index, you might want to override the default by setting the percentage to zero. Usually, a low value (5% to 10%) is a good choice when creating an index, as this allows enough room to accommodate a low level of maintenance.

Tuning Parameter (PCTFREE): You can set the percentage of space in each index page for future index entries, when you create it using the PCTFREE parameter. For example, the following statement creates an index and reserves 20% of the space on each page for future entries:

```
CREATE INDEX test_index ON test_table (test_column) PCTFREE=20
```

You can change a current PCTFREE value by either dropping the index and recreating it with a new PCTFREE, or you can reorganize it with the DBS Utility. For example, the following command will reorganize index `index_number_one` created by `smith` and give it a PCTFREE value of 50:

```
REORGANIZE INDEX (smith.index_number_one) PCTFREE=50
```

For more information on reorganizing indexes, refer to “Reorganizing Fragmented Indexes” on page 77.

Performance Indicator (IPCTFREE): To determine the current percentage of space in each index page for future index entries, look in the IPCTFREE column of the SYSTEM.SYSINDEXES catalog table for your index.

Shadow Pages

Shadow pages are used whenever you make changes to your database. They use space in a storage pool that is only released during a checkpoint. If you are not careful to leave enough free space in your pools, shadow pages can fill them before the space is reclaimed at the next checkpoint. This is true even if you are only modifying rows and not adding new ones.

Each permanent (not internal) dbspace page has two entries in the page map table. One points to the *current* page while the other points to the *shadow* page. The current page contains any updates made to the page since the last checkpoint. The shadow page contains the original page as it was at the time of that checkpoint. (See “Choosing the Checkpoint Interval” on page 111 for a discussion of checkpoints.) If there have been no changes to the table since the last checkpoint, both entries point to the current page.

The database manager uses this system to reconstruct changes to the database after a system failure. When the database manager is restarted after a system failure, it will use the page map table entries that point to the shadow pages. This effectively resets the database to its state at the last checkpoint. The LOG is then used to re-apply updates for LUWs committed after the last checkpoint.

When the database manager updates a page the following occurs:

1. A new physical page is allocated from a storage pool. This uses one physical 4KB page in a storage pool. It does not deplete the available data pages of the dbspace.
2. The current page map table entry is set to the new page location.
3. The new page is created in the local buffer pool.

At the next checkpoint the following occurs:

1. The new page in the buffer pool is written to the new physical location in a storage pool and the buffer page is released for reuse.
2. 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: Do not confuse shadow page recovery with rollback work processing. Shadow pages are **NOT** released during a ROLLBACK. During a rollback the contents of the log are read and any changes to the database are undone. For example the database manager will undo a CREATE TABLE with a DROP TABLE. If a ROLLBACK were accomplished by falling back on shadow pages, you could not recover if a system failure occurred during ROLLBACK processing.

Determining the Number of Shadow Pages in Use: If you want to know how many pages are used by a specific update transaction, you can compare the number of PAGES USED (SHOW POOL) before and after the transaction. For example, consider a storage pool with two dbextents. Force a checkpoint (drop a dbspace created for the purpose), and then enter a SHOW POOL command:

```
show pool

POOL NO.  1:     NUMBER OF EXTENTS = 2

EXTENT   TOTAL    NO. OF      NO. OF      NO. OF     %
 NO.     PAGES  PAGES USED  FREE PAGES  RESV PAGES  USED
   1       855        245         610               28
   2       855          0         855                0
TOTAL     1710        245        1465          20   14
ARI0065I Operator command processing is complete.
⋮
```

This pool has 245 PAGES USED. This total includes data, header, and pages index. However, since you have just forced a checkpoint it does not include any shadow pages.

If you now perform a transaction (for example an UPDATE statement) you can determine how many shadow pages it uses by reissuing the SHOW POOL operator command. For example, enter another SHOW POOL after the checkpoint:

```
show pool

POOL NO.  1:     NUMBER OF EXTENTS = 2

EXTENT   TOTAL    NO. OF      NO. OF      NO. OF     %
 NO.     PAGES  PAGES USED  FREE PAGES  RESV PAGES  USED
   1      855        249         606                 29
   2      855          0         855                  0
TOTAL     1710       249        1461          20     14
ARI0065I Operator command processing is complete.
⋮
```

This time the pool has 249 PAGES USED. This means that your transaction used 4 shadow pages (249-245).

If you force another checkpoint, the database manager will now release all the shadow pages in the pool and reclaim the space. (There will be 245 PAGES USED and 610 FREE PAGES.)

Note: The above procedure will not be accurate if the transaction performs enough database modifications to cause a checkpoint to occur.

Running out of Dbspace Pages

If you have run out of pages in a dbspace its because of one of two conditions, either the logical dbspace is full or the storage pool to which is assigned no longer has any unallocated physical pages.

Storage Pool Full: If all the pages in a storage pool have been allocated, reorganize the dbspaces allocated to it by dropping and recreating (reorganizing) them. This will reclaim space wasted because of fragmentation, refer to “Reorganizing Data” on page 73. If this does not reclaim enough space, you must add a dbextent to the pool. For instruction on how to do this, refer to the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manuals.

You cannot make more pages available by deleting tables or rows in another dbspace using the same pool. Deleted data pages are not returned to their pool. After a data page in a storage pool has been assigned to a specific dbspace, it cannot be used by another dbspace using the same pool until the entire dbspace is dropped.

You can use the SHOW DBEXTENT operator command (refer to page 25) to monitor the storage available in each storage pool, and you can use the SHOW POOL operator command (refer to page 26) to monitor the number of pages available in each dbextent in the pool.

Dbspace Full: A dbspace cannot be extended after it is defined (either during initial database generation or when it is added to a storage pool). Your only choices are to delete rows or tables in the dbspace itself, or unload the contents of the dbspace and reload them into a new dbspace that is larger than the original.

Alternatively, just over allocate the dbspace when you acquire it (refer to the ACQUIRE dbspace command in the *DB2 Server for VSE & VM SQL Reference* manual). A dbspace is only a logical allocation of space in the form of directory page tables. You will not actually consume the total number of pages in the storage pool that you defined for all the dbspaces in the pool. Define the size of dbspace based on how large they may become, but define the size of the storage pool based on how much storage you need right now, which includes shadow pages. As your tables grow and you need more pages, just add dbextents to the pool.

Note: The amount of storage you need right now includes space for shadow pages.

You can use the SHOW DBSPACE operator command (refer to page 26) to monitor the number of header, data, and index pages allocated to the dbspace and the percentage of each actually in use.

Shadow Pages: You may find that even though you have not added a significant number of new rows to a dbspace it may become full. This occurs because every time you modify a existing index or data page (or create a new one) a *shadow* page is created. These pages require additional storage that is not reclaimed until the next checkpoint, refer to “Shadow Pages” on page 66. To avoid this problem, ensure that there is enough free space in your storage pools to accommodate shadow pages. You will usually require between 15% and 25% free space measured immediately after a checkpoint. For example:

```
show dbextent

POOL     TOTAL    NO. OF      NO. OF      NO. OF     %    NO. OF
NO.      PAGES  PAGES USED  FREE PAGES  RESV PAGES  USED  EXTENTS SOS
   1      1710     1410         300         20       83      2
FREE    268626
ARI0065I Operator command processing is complete.
```

This dbextent has 17% free space (100-83). You may want to add space to ensure that shadow pages will not become a storage problem.

Ever Increasing Index: One reason for running out of index pages is that you are using an “ever increasing index.” For example, consider a table where you only keep data for three months. Every month you delete any rows that were created more than three months earlier. To keep track of the creation dates, you use a date column, or timestamp and create an index on that column.

Unfortunately in this example, even though you delete old rows, the pages that contain their index keys remain allocated to the table. They are not released for reuse. They also remain allocated to the same range of values (or dates), so in this example, they may never be reused. For example, if one index page contained keys for a range of dates from March 1, 1996 to April 14, 1996, it will only ever be reused for that range.

If you have this type of index, you must constantly monitor the percentage of free index pages in the dbspace. You can use the SHOW DBSPACE operator command, refer to “Proportion of Available Pages” on page 26.

To recover the wasted storage used by an ever increasing index, you must reorganize it. Refer to “Reorganizing Fragmented Indexes” on page 77.

Data Clustering

Clustered Indexes

You can say that an index is clustered if the **data** is logically stored in an order which closely matches the sequence of the index. That means that, ideally, when you retrieve the rows following the order of a clustered index, the database manager can do so by looking at a minimum number of pages.

Consider the following; all the rows in a table are retrieved in the sequence of an index. As the database manager retrieves each row, it counts the number of times it needs to access a different page than the one it is currently using.

In the best case, the number of pages 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.

Remember that saying an index is clustered really means that the table is clustered relative to the index. If the database manager can use the index to sequentially retrieve the rows in the table by looking at a minimal number of data pages, the index is clustered. Another index acting on the same data may or may not be considered clustered.

The Clustering Index

The first index created on a table is, by default, the clustering index. It inserts new rows into data pages so that as many pages as possible are clustered relative to the clustering index, refer to “Clustered Indexes.”

Default Clustering versus Clustering Index Strategy: 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. Essentially the default logic places any new rows at the end of the table, while the clustering index logic places a new row in index sequence, as much as possible. While the clustering strategy tries to keep a clustering index clustered, the only way to ensure that it is completely clustered is to reorganize the data, refer to “Reorganizing Data” on page 73.

The **default logic strategy** is used if a clustering index is not available (indicated by a “D” in the CLUSTERTYPE column in SYSTEM.SYSCATALOG for the table). This strategy uses the value in the CLUSTERROW column 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 preprocess this kind of program to update the CLUSTERROW value in the package.

The **clustering index strategy** is used if a clustering index is available (indicated by a “I” in the CLUSTERTYPE column in SYSTEM.SYSCATALOG for the table). 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, refer to “Reorganizing all the Tables in a Dbspace” on page 75, 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 (refer to “Reorganizing Fragmented Indexes” on page 77) 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 usually drop all indexes on the table, create the new clustering index as the first index, and then create any other indexes. Refer to “Reorganizing a Single Table” on page 74 and “Reorganizing all the Tables in a Dbspace” on page 75. You can also change the clustering index by updating the SYSTEM.SYSINDEXES catalog table, refer to “Changing the Clustering Index without Dropping Indexes” on page 76.

How Indexes Become Unclustered

Indexes become unclustered when:

- A significant number of rows were added to a table since the clustering index was first created or since the data was last reorganized
- And there was insufficient free space available to put the rows into their optimal locations.

You can increase the number of rows that you can add before the index becomes unclustered by increasing the PCTFREE setting when you reorganize your data. Refer to “Reorganizing a Single Table” on page 74.

Identifying Unclustered Indexes

Deciding whether an index is clustered requires some judgement. First, you need to execute an UPDATE STATISTICS statement against the table the index belongs to. Second, you need to look at the CLUSTERRATIO and the CLUSTER column in the SYSTEM.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:

```
                        ROWCOUNT - PAGE JUMPS
CLUSTERRATIO = 10000 * ---------------------
                        ROWCOUNT - 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 CLUSTERRATIO value ranges between 0 and 10000, and indicates the percentage of time that the table's row, when retrieved using that index, are in logical page sequence.

The **CLUSTER** value, in addition to giving a general idea about whether the index is clustered, is also used to identify the clustering index for the table.

Table 2. CLUSTER values

CLUSTER Value	Clustered	Clustering
F	Yes	Yes
C	Yes	No
W	No	Yes
N	No	No

The CLUSTER column will show an index to be clustered if the following is true:

```
        PAGE JUMPS
110% <  ---------- x 100
        PAGE COUNTS
```

(The number of jumps per page is greater than 1.1.)

Clustering VIEW: You can include all the important information about clustering and indexes in one VIEW. For example, the VIEW should contain the following information:

- The name of the index and its creator
- The CLUSTERRATIO of the index
- The CLUSTER value of the index
- The number of rows in a table that the index acts on
- The number of pages in the table that the index acts on

For example:

```
SELECT i.iname,
       i.clusterratio,
       i.cluster,
       t.rowcount,
       t.npages
       FROM system.sysindexes i, system.syscatalog t
       WHERE t.tname = i.tname and
             t.creator = i.creator
```

to help you determine if your index is clustered, your view should also include:

- The number of jumps
- The number of jumps per page.

While the CLUSTERRATIO and CLUSTER values are very useful in determining how clustered a index is, you may find it useful to see how many jumps the database manager makes for each page it reads. Remember each additional jump per page represents an unnecessary I/O. You may also want to compare the number of jumps per page to the number of rows per page. Unfortunately there is no concrete rule you can use to decide when an index is unclustered. However, the more information you have available the better you will be able to get a “feel” for the state of the index.

To calculate the number of jumps, rearrange the clusterratio calculation to solve for jumps instead of clusterratio. For example:

```
                                CLUSTERRATIO
PAGE JUMPS = ROWCOUNT - ------------- X ( ROWCOUNT - PAGE COUNT)
                                    10000
```

If the page jumps calculation was included in a SELECT with the name of the index, the SELECT would look like this:

```
SELECT i.iname,
       (t.rowcount-( i.clusterratio/10000.0*(t.rowcount-t.npages)))
       FROM system.sysindexes i, system.syscatalog t
       WHERE t.tname = i.tname and
             t.creator = i.creator
```

The following SELECT statement:

- Combines the previous two SELECT statements and adds the jumps per page ratio.
- Orders the results by the jumps per page ratio, so you can see the indexes with the worst ratio first
- Excludes indexes created against empty tables.

```
SELECT i.iname,
       i.clusterratio,
       i.cluster,
       t.rowcount,
       t.npages,
       (t.rowcount-(i.clusterratio/10000.0*(t.rowcount-t.npages))),
       ((t.rowcount-(i.clusterratio/10000.0*(t.rowcount-t.npages))))/t.npages
       FROM system.sysindexes i, system.syscatalog t
       WHERE t.tname = i.tname and
             t.creator = i.creator and
             t.npages > 0 and
             t.rowcount > 0
       ORDER BY 7 desc
```

Reorganizing Data

You should reorganize data for one of four reasons:

- A table's clustering index has become unclustered. Refer to “Identifying Unclustered Indexes” on page 71.
- The number of overflow rows in a table (NOVERFLOW) exceeds 5% of the total number of rows in the table (ROWCOUNT). Refer to “Free Space in Data Pages” on page 64.
- To change which index acts as the clustering index.
- To return once populated but now empty data pages to the storage pool, refer to “Storage Pool Full” on page 68.

The first two conditions indicate that the rows can no longer be efficiently retrieved.

Essentially, reorganizing involves unloading the data and reloading it. Unload the data, making sure that the clustering index exists. If a clustering index is available, the data is unloaded following its sequence. Drop the clustering index and reload the data (it will be reloaded in the order of the clustering index). Then recreate the clustering index. This reclusters the data according to the clustering index, and reclaims space that was lost because of row overflow.

The following instructions assume that you are using the DBS utility to unload and reload tables. For more information on its use, refer to the *DB2 Server for VSE & VM Database Services Utility* manual.

There are several questions you need to ask before you choose a way to reorganize your data.

- If you are reorganizing all the tables in a dbspace at once and there are not many tables in that dbspace with field procedures), and it is **not** difficult for you to recreate all the indexes, referential constraints, and unique keys in that dbspace, follow the instructions in “Reorganizing all the Tables in a Dbspace” on page 75. While this set of instructions requires you to recreate the entire dbspace, the actual process of reloading the data is faster than the following alternative.
- If you only need to reorganize one or two tables or if your table contains columns with field procedures , or if it is too much work to recreate all their indexes, referential constraints, and unique keys, follow the instructions in “Reorganizing a Single Table.” However, if you also want to change which index acts as the clustering index, you will have to drop and recreate all the table's indexes.
- If you only want to change the clustering index in a single table that uses several other indexes, follow the instructions in “Changing the Clustering Index without Dropping Indexes” on page 76. Unlike the previous procedure, this set of instructions does not require you to drop and recreate all the table's indexes.

Reorganizing a Single Table

The following method reorganizes a single table. It uses the DBS Utility RELOAD PURGE command. While it is not as fast as the RELOAD NEW command, you do not need to manually drop and recreate any indexes, referential constraints, and unique keys (unless you want to change the clustering index).

1. If you want to change which index acts as the clustering index, do the following:
   a. Drop all indexes for the table by issuing a DROP INDEX statement for each one.
   b. Create a new index (using the CREATE INDEX statement). This index will act as the clustering index.
2. Unload the table (usually to tape), by issuing a DBS Utility UNLOAD TABLE command. The rows are automatically unloaded in the key sequence of the clustering index.
3. Set the PCTFREE value of the dbspace to a high enough value to allow space on pages for future clustered insertion of rows.
4. Set UPDATE STATISTICS ON if you want to automatically collect statistics during the RELOAD, or set it OFF if you plan to UPDATE ALL STATISTICS after the RELOAD. Refer to “Automatic Statistics Collection” on page 121.
5. Reload the table by issuing a DBS utility RELOAD command with the PURGE option specified.

During RELOAD command processing with the PURGE option specified, all rows of the specified table are deleted. As part of the PURGE, the DBS utility drops the clustering index, deactivates any active primary keys, active foreign keys, and active unique keys, and deletes all indexes on the table before deleting and reloading the data. After the table has been reloaded, the DBS utility recreates the clustering index, primary key, and unique keys, and recreates the remaining indexes. It then reactivates all the foreign keys it dropped. Since packages are invalidated because of table index deletions, they are dynamically repreprocessed the next time someone attempts to execute the package.

6. Reduce PCTFREE to make the free space available for use on normal INSERT activity.
7. If you set UPDATE STATISTICS OFF, collect statistics for all columns by issuing the UPDATE ALL STATISTICS command.
8. If you changed which indexes acts as the clustering index, recreate the other table indexes required, using CREATE INDEX statements. The definition of all indexes on a table can normally be determined by querying the SYSTEM.SYSINDEXES system catalog table, as long as the length of the column names on which the index is defined is less than 100 characters.

Reorganizing all the Tables in a Dbspace

The following method reorganizes all the tables in a dbspace.

1. Record any index, referential constraint, unique key definitions, or field procedures authorizations in the dbspace.
2. If you want to change which index acts as the clustering index for any tables in the dbspace, do the following for those tables:
   a. Drop all indexes for the table by issuing a DROP INDEX statement for each one.
   b. Create a new index (using the CREATE INDEX statement). This index will act as the clustering index.
3. Unload the dbspace (usually to tape), by issuing a DBS Utility UNLOAD DBSPACE command. The tables will be unloaded in the order of the clustering index.
4. Drop and recreate the dbspace.
5. Set the PCTFREE value of the dbspace to a high enough value to allow space on pages for future clustered insertion of rows.
6. Set UPDATE STATISTICS ON if you want to automatically collect statistics during the RELOAD, or set it OFF if you plan to UPDATE ALL STATISTICS after the RELOAD. Refer to “Automatic Statistics Collection” on page 121.
7. Reload the dbspace by issuing a DBS Utility RELOAD DBSPACE command with the NEW option specified.

   The NEW option assumes that none of the tables you are reloading currently exist in the dbspace. A program that accesses a table, the index of which was dropped, is re-preprocessed when it is next executed, which ensures that it takes advantage of the new clustering properties.

   If a table in the dbspace has field procedures associated with it, the table should be dropped and recreated to include the field procedures and reloaded

| using the PURGE parameter. It is not necessary to unload the table again, as
| the table can be reloaded from the unloaded dbspace file.

8. Recreate the clustering index.
9. Reduce PCTFREE to make the free space available for use on normal INSERT activity.
10. If you set UPDATE STATISTICS OFF, collect statistics for all columns by issuing the UPDATE ALL STATISTICS command.
11. Recreate the other table indexes, any referential constraints and any unique keys.

Changing the Clustering Index without Dropping Indexes

The following method reorganizes a single table, and changes which index will act as the clustering index. It eliminates the need to individually drop and recreate all indexes on the table. (The steps can be performed in a single execution of the DBS Utility.)

1. On the SYSTEM.SYSINDEXES table entry for the original clustering index, update the CLUSTER column value of “F” or “W” to “N.”
2. Change the value in the CLUSTERRATIO column to 1000 (10.00%).
3. If the new clustering index does not exist, create it with a CREATE INDEX statement.
4. On the SYSTEM.SYSINDEXES catalog table entry, update the CLUSTER column for the new clustering index to the value “W.”
5. Change the value in the CLUSTERRATIO column to 7500 (75.00%).
6. Unload the table by issuing a DBS Utility UNLOAD TABLE command.
7. Set UPDATE STATISTICS ON if you want to automatically collect statistics during the RELOAD, or set it OFF if you plan to UPDATE ALL STATISTICS after the RELOAD. Refer to “Automatic Statistics Collection” on page 121.
8. Reload the table by issuing a DBS Utility RELOAD command with the PURGE option specified.

   During RELOAD command processing with the PURGE option specified, all rows of the specified table are deleted and the table index (if one exists) is dropped and recreated. A program that accesses a table, the index of which was dropped, is re-preprocessed when it is next executed, which ensures that it takes advantage of the new clustering properties.
9. If you set UPDATE STATISTICS OFF, collect statistics for all columns by issuing the UPDATE ALL STATISTICS command.

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.

Reorganizing Fragmented Indexes

To determine whether an index should be reorganized, enter the SHOW DBSPACE operator command to see how many index pages are occupied in the dbspace, and what the actual percentage of free space in the occupied pages is. Next, determine the *expected* percentage of free space by averaging the PCTFREE settings of all the indexes. If the actual free space is appreciably higher than the expected amount, index fragmentation or skewed index values are the likely cause.

There are two ways to reorganize an index. One is to obtain all index definitions from the catalog tables, drop the index with the DROP INDEX statement, then recreate it with the CREATE INDEX statement.

The other is to enter the following DBS Utility command:

```
REORGANIZE INDEX (index-name)
```

You must be the owner of the index or have DBA authority.

The advantages of the REORGANIZE INDEX utility are:

- A dbspace scan is not required to retrieve the rows of the table.
- A sort of the index key columns is not required.
- Dependent packages are not invalidated and therefore do not require re-preprocessing.
- The clustering property of a clustering index is not lost. (If there is more than one index on a table, and a clustering index is reorganized by being dropped and re-created, it is no longer the clustering index.)

For more information on the REORGANIZE INDEX utility, see the *DB2 Server for VSE & VM Database Services Utility* manual.

Notes:

1. You must use the ALTER TABLE statement to reorganize an index that was created by the database manager to enforce the uniqueness of a primary key or a unique constraint (see the *DB2 Server for VM Database Administration* or the *DB2 Server for VSE Database Administration* manuals).
2. A different utility is provided to reorganize the catalog table indexes (see the *DB2 Server for VSE Database Administration* or the *DB2 Server for VM Database Administration* manuals.).

3. Reorganizing an index is not a solution for an unclustered index. To correct an unclustered index, you must reorder the data to match the index sequence, refer to “Reorganizing Data” on page 73. In addition, issuing the REORGANIZE INDEX command does not return freed pages to the storage pool. The freed pages are only returned to the storage pool if you drop the dbspace.

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. 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 *DB2 Server for VSE & VM 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** (SQLSTATE 23505) 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 *DB2 Server for VSE & VM Operation* manual.

Reorganizing an Invalid Index

Use the SHOW INVALID operator command to display all invalid indexes in the database, as well as the reason why each index is invalid.

Use the REORGANIZE INDEX utility to revalidate an invalid index that is invalid because you encountered a `NO ROOM IN THE STORAGE POOL` message.

If the invalid index was created to support a primary key or a unique constraint, it can be reorganized with the ALTER TABLE *table_name* ACTIVATE *key_name* command.

When reorganizing an invalid index, the database manager must scan the dbspace and sort the index keys, because the invalid index may not contain all the keys.

You cannot use the REORGANIZE INDEX utility to revalidate a unique index that contains duplicates causing it to be marked invalid. You must drop this index, remove the duplicates, and re-create it. If the index was created to support a primary key or a unique constraint, you must deactivate the primary key or unique constraint with the ALTER TABLE *table_name* DEACTIVATE *key_name* command, remove the duplicates, and reactivate the primary key or unique constraint with the ALTER TABLE *table_name* ACTIVATE *key_name* command.

DASD Balancing

How well you balance the demand for DASD I/O across several DASD volumes can affect how fast the database manager can retrieve information from DASD.

Do not spend a lot of time and effort balancing the utilization of your DASD channels and controller. Instead, concentrate on balancing the utilization of your DASD volumes. You can then simply allocate an even number of volumes to each controller.

Evenly Distributing Workload across Physical Volumes

Moving Dbextents

To evenly distribute your workload across all volumes of DASD, use the following method as a guide:

1. Measure the current utilization of your DASD volumes.
2. Select the highest utilized volume. While DASD balancing based on utilization may not necessarily give optimal performance (it assumes all your volumes perform equally well), it is an excellent place to start. (You can also select a volume based on average service time. Choose the volume with the highest average service time. Balancing this way ensures that you will drive faster DASD harder.)
3. If there is **more than one dbextent** on the volume, move one dbextent to the lowest utilized volume. In VM use DDR, and in VSE use VSAM backup and restore. (While you can use the copy dbextent facility that is supplied with the DB2 Server for VSE & VM product to move a dbextent, DDR and VSAM are much faster.)
4. If there is only **one dbextent**, examine the assignment of dbspaces to pools to dbextents. Refer to “Reassigning Dbspaces” on page 80.

5. Measure the current utilization of your DASD volumes again.
   a. If you find a significant improvement, return to step 2.
   b. If you do not find a significant improvement, return to step 3 and select a different dbextent to move.
   c. If there is no significant difference between the utilization of the highest and the lowest utilized volumes they are balanced. Occasionally, measure the utilization of your DASD volumes to ensure that they are still balanced.

Reassigning Dbspaces

To reassign your dbspaces, first determine which storage pool the dbextent belongs to, then choose one of the following options:

- Move a dbspace from one storage pool to another.
- Move a table from one dbspace to another. This choice is not valid if this is already the only table in the dbspace. Also make sure that if you move the table you do not put more than one highly used table in the same dbspace.
- Change the dbextent(s) in the storage pool to which the dbspace is allocated.

Moving Dbspaces

1. Select a dbspace to move. While you can use the SHOW DBSPACE operator command to see how many pages from the storage pool have been allocated to a dbspace, you cannot easily determine how utilized the dbspace is. You must rely on your knowledge of how the table(s) in the dbspace are used.
2. Unload all the tables in that dbspace.
3. Acquire a dbspace in a new storage pool. This storage pool should have dbextents on the lowest utilized volumes. To accomplish this, you may have to add a dbextent or dbspace or both.
4. Reload the tables.
5. If a table in the dbspace has field procedures associated with it, the table should be dropped and recreated to include the field procedures and reloaded using the PURGE parameter.
6. Drop the old dbspace.
7. Recreate the indexes, views, and authorities.
8. Recreate any referential integrity constraints.

Moving Tables

1. Select a table to move. While you can use the NPAGES column in the SYSTEM.SYSCATALOG table to see how many pages from the dbspace have been allocated to a table, you cannot easily determine how utilized the table is. You must rely on your knowledge of how the table(s) in the dbspace are used.
2. Unload the table.
3. Select a dbspace in a new storage pool.
4. If the table has field procedures associated with it, recreate the table to include the field procedures.
5. Reload the tables.
6. Drop the old table.

7. Recreate the indexes, views, and authorities.
8. Recreate any referential integrity constraints.

Change Dbextents: You can either let the database manager do most of the work for you, or you can do it yourself:

Let the Database Manager Do it

1. Add dbextents to the storage pool until there is more free space in the pool than on the dbextent to be deleted (allowing sufficient space for shadow pages and an adequate SOSLEVEL).
2. Delete the dbextent on the most used volume. The database manager will automatically move data from the extent to be deleted onto the remaining dbextents in the pool.

Do it Yourself

1. Unload all the tables in all the dbspaces in a storage pool.
2. Drop all the dbspaces in the storage pool.
3. Re-assign dbextents to the storage pool.
   - One simple technique is to split one dbextent into two smaller dbextents on two separate volumes. One dbextent remains on the highly utilized volume and the other is allocated to a low utilized volume. You cannot use the DB2 Server for VSE & VM copy dbextent facility to do this.
   - Unless you are using the DB2 Server DSS Feature with striping turned on, do not just add a new dbextent to the pool. That will not result in any usage of the new dbextent until the previous dbextents are full.
4. Acquire dbspaces in the storage pool.
5. Reload the tables.

   Note: Unless you are using striping, data is added to the dbextents in the order that they were created. The database manager will fill the first dbextent before it proceeds to the next one.
6. If a table in the dbspace has field procedures associated with it, the table should be dropped and recreated to include the field procedures and reloaded using the PURGE parameter.
7. Recreate the indexes, views, and authorities.
8. Recreate any referential integrity constraints.

General Considerations

There are other things to consider when you organize your dbspaces, storage pools, dbextents and physical DASD.

Place the **database catalog tables** into their own pool. At database generation time, the catalog is placed in pool number one. All other, non-catalog tables, should be moved to different pools.

Place **internal dbspaces** in their own pool. Performance should benefit greatly for large complex queries if you use data spaces with this pool. In **VSE/ESA**, assign the pool to a set of dbextents that includes a VSE virtual disk. Refer to “Virtual Disk

Support for VSE/ESA for Internal Dbspaces" on page 50. In **VM/ESA**, if you have DB2 Data Spaces Support, use unmapped data spaces support for internal dbspaces. Refer to the *DB2 Data Spaces Support* manual.

Caching is best used where data is frequently reused. For example, the database directory is primarily read from and will benefit from caching, while the log is primarily written to and will not benefit from it. Any highly utilized dbextent disk that contains tables that are primarily used for read only transactions will benefit from caching.

Attention: The amount of frequently-reused-data should not exceed the size of the cache.

If you have **faster storage devices** available, use them for your highest utilized dbextents.

Place the database directory on a separate volume from your storage pool dbextents. Because you may use all of these at the same time, if you do not separate them you may create a bottleneck. You can place the directory and the log(s) in the same volume, but it is better to separate them. If you use dual logging, be sure to put each log on a different physical device (and controller and channel, if possible).

If you are using the DB2 Server DSS Feature with striping turned on, make sure that each dbextent in a storage pool is on a separate volume. Refer to "Striping" on page 93.

If you are not using DB2 Server DSS striping, **place dbextents consecutively** on the same physical volume. This avoids unnecessary head movement. In a **VM** system, you can control exactly where a minidisk is placed. However, if you want to place dbextents consecutively in a **VSE** system, you need to backup all VSAM datasets on a particular disk and then reallocate them consecutively.

VM

Fair Share Scheduling

VM was originally designed to support a large number of equally important virtual machines. To ensure that each user receives an equal allotment of its resources, VM's scheduler attempts to give each machine in the system a fair share of the processor's time.

However, if you are only using one or two database machines that use most of your system's resources, fair share scheduling may keep them from receiving the processor time they need. The database and user machines may receive approximately the same resources to perform their tasks. However because the database machines are performing work for many users they may need much more resource than the users. The database machines may become a bottleneck, because the user machines are spending more time waiting for them than for processor time.

Fortunately, there are several parameters that let you shift fair share scheduling in the database machine's favor. Use these carefully. It is easy to over adjust them and virtually lockout all other users in your system.

SET SHARE You can use the SET SHARE command or the SHARE directory statement to control the percentage of system resources a virtual machine receives. These resources include processors, real storage, and paging I/O capability. A virtual machine receives a proportion of any scarce resource according to its share setting.

You can use this command to ensure that a database machine receives an absolute minimum share of system resources and that the remaining resources are divided among the rest of the user machines. (However, remember that by allocating an absolute share of system resources to a single machine, you will also limit its share to that amount.

If the database machine is not the only multiple user server on the system, give it a relative (instead of absolute share).

SET QUICKDSP You can use either the SET QUICKDSP command or the QUICKDSP operand of the OPTION directory statement to designate virtual machines that will not wait in the eligible list when they have work to do. Instead, a virtual machine with a quick dispatch setting (QUICKDSP) is added to the dispatch list immediately without first waiting in the eligible list.

You should always use this command to ensure that a database machine never waits longer than absolutely necessary for another machine when it has work to do.

For more information on either QUICKDSP or SHARE, refer to the *VM/ESA: Planning and Administration* manual.

VSE

Dispatching Priority

The database partition should be configured according to the following guidelines:

- Set its priority lower than any CICS partitions that are accessing it.
- Set its priority immediately below the CICS partition with the lowest priority that is accessing it.
- Set its priority higher than any batch partitions accessing it.
- Do not include it in a *partition balancing* pool.

Fast CCW Translation

Do not use fast CCW translation (FASTTR job control option) in the database or CICS partitions. Include the following option card in the application server start up job:

```
// OPTION NOFASTTR
```

With DB2 Server for VSE, the VSAM I/O buffer address normally changes every time it does an I/O, hence will suffer from using FASTTR. It performs I/O directly from its local buffer.

Virtual Addressability Extension (VAE)

While there is a performance advantage to placing the database partition and the CICS partition in the same address space, it is nearly impossible. Instead, place the database partition and the CICS partition in separate address spaces. This will increase the contention between these two partitions and introduce additional overhead for address space switching. However, you will have a significant amount of space for:

- Additional agent structures
- Larger buffer pools
- More locks
- A larger package cache

All of these will improve your server's performance if you have enough main storage to avoid increased paging.

31 Bit Addressing

You can use 31 bit addressing to increase the database partition size above the normal 16MB limit (refer to “Storage Above 16MB (31 Bit Addressing)” on page 47). However, because only one partition in the address space can use storage above the 16MB line, it is still impractical to place CICS and the database partition in one address space. Fortunately, with 31 bit addressing the cost of address space switching becomes a relatively minor performance concern.

Compile Partition Size

Ensure that the partition you intend to use to compile application programs is large enough. Preprocessing tends to produce relatively large source programs, and compiles can take as much as ten times longer than necessary if your partition is too small. Start with a partition size of 1.2MB and expand it if necessary.

CICS

AMXT/MXT

If you are adding DB2 Server for VSE work to an existing CICS environment, consider increasing the CICS DFHSIT macro AMXT value. The optimal level of CICS subtasking may now be higher than it was. Each active ISQL user requires two active tasks within CICS. If the AMXT limit is reached, response time is adversely affected.

ISQL

Transaction Name

Use transaction name ISQ2 rather than CISQ for ISQL. The ISQL transaction will attempt to start a second transaction called ISQ2. If it cannot find ISQ2 it will look for the CISQ transaction. Using the name ISQ2 avoids the additional processing involved in searching for both ISQ2 and CISQ. (The ISQL transaction first looks for a second transaction whose name is constructed by replacing the last character of the first transaction ID with 2. In this case it would be ISQ2.)

Number of Concurrent Users

Consider limiting the number of concurrent ISQL users. If the database manager is only used from the CICS environment through ISQL, you can limit the number of concurrent users by limiting the number of links to the application server when you start the DB2 Server for VSE online support (CIRB transaction).

If you plan to use the database manager from the CICS environment through both ISQL and preplanned transactions, you can do this using the CICS CMXT parameter. This is done by assigning the CISQ transaction to its own CICS class and setting CMXT for that class to the desired limit.

Do not place a CMXT limit on the ISQL transaction ID. It may cause problems with long queries. The ISQL transaction will temporarily end in the middle of a long query, leaving the CISQ transaction active while it waits for a reply from the application server. If more ISQL users logon, the number of ISQL transactions can reach the CMXT limit. When the CISQ transaction eventually gets a reply from the server, it try's to restart its partner ISQL transaction. This will fail if the CMXT limit has been reached.

You can use CMXT to allocate CICS-DB2 Server for VSE links for CICS production work that requires access to DB2 Server for VSE data. For example, if 6 CICS-DB2 Server for VSE links are defined, and CMXT limits the number of ISQL users to 4, at least 2 links are always available for other DB2 Server for VSE requests.

Instead of limiting the total number of ISQL users, you can also limit the number of ISQL users by group. For more information refer to the *DB2 Server for VSE System Administration* manual.

Temporary storage

Auxiliary versus Main

Consider using AUXILIARY storage if you expect to run large routines. All ISQL routines are read into CICS temporary storage (either MAIN or AUXILIARY) before the first command in the routine is run. Using MAIN temporary storage improves performance but uses more virtual storage. Using AUXILIARY temporary storage slightly degrades performance, but reduces the amount of virtual storage required.

To use MAIN storage, code TSP=1$ on the CICS SIT or to use AUXILIARY storage, code TSP=2$.

Guest Sharing with VSE under VM

VSE users can access a VM application server if the VSE system is running as a second level guest under VM. While all the tuning suggestions for a native VSE application server also apply to a second level guest, there is an additional consideration.

Distributed Configuration Considerations

In a VM System

DRDA enabled users can access DB2 Server for VM and non-DB2 Server for VM DRDA application servers located either in a TSAF collection or in an SNA network.

A DB2 Server for VM DRDA enabled application server can accept requests from DB2 Server for VM or DRDA application requestors located either in a TSAF collection or in an SNA network. They can also accept requests from VSE users in a second level VSE system. (Non-DRDA DB2 Server for VM servers cannot accept requests from non-DB2 Server for VM requestors.)

In a VSE System

In native VSE, Online CICS application programs can access and manipulate data managed by any remote application server that implements the DRDA architecture, as well as local DB2 for VSE servers. Batch application programs can access only local DB2 for VSE servers within the same VSE processor. They cannot access remote DRDA servers. In a guest sharing VM system, VSE users in a second level VSE system can access a DB2 Server for VM server within the same VM processor, or a remote DB2 Server for VM server located in a TSAF collection or in an SNA network (but not to a non-DB2 Server for VM server).

A DB2 Server for VSE DRDA enabled server can accept requests from any DB2 Server for VSE requesters within the same processor, or from DB2 Server for VM or non-DB2 Server for VSE & VM DRDA requestors.

Performance Implications

How you configure a distributed system can have a significant impact on the performance of all the processors in the network. While this guide cannot describe all possible distributed installations, nor can it suggest the best possible installation for you, it does include some basic guidelines and several simple examples.

See the *VM/ESA: Connectivity Planning, Administration, and Operation* manual for your operating system for details on optimizing performance in a TSAF collection or SNA network. For information on both SNA networks and the connectivity issues that are relevant in IBM distributed database systems, see the *Distributed Relational Database Connectivity Guide* manual.

Applications Planning

If your application program needs to interact with a remote processor, there are several things that you can do to minimize the communication traffic between the requester and the server.

Fetch and Insert Blocking

Blocking groups multiple row insertions or retrievals into one request. Instead of sending a separate instruction for each insert or fetch done by a cursor, instructions are grouped together and sent in one communication block. This reduces message traffic and overhead. (However, it is not supported in single user mode, or with DRDA.) For more information, refer to “Fetch and Insert Blocking” on page 118.

Hold File

The creation of hold files is a technique allowing you to save the results of a query (database information) in CMS or CICS files. Subsequent requests for this information are satisfied by retrieving it from the CMS or CICS files.

Local Copy

If your application requires information from a database on another processor that is not periodically updated, consider copying the information into temporary tables in a local database. For example, if you need access to a monthly sales summary, simply unload the summary data from the remote server once a month and load it into your local server.

Chapter 4. Configuring the Application Server and Requester

Database Manager Storage

Database I/O

Before a page of data can be used by the database manager, it must be located in its data page *buffers*. The buffers are two areas of storage in your database machine or partition, which are allocated when you start the database manager. One area called the *directory buffer pool* is reserved for pages from the DB2 Server for VSE & VM directory disk. The size of the pool is determined by the NDIRBUF initialization parameter. The other area called the *local buffer pool* is reserved for pages from the storage pools. Its size is determined by the NPAGBUF initialization parameter.

When the database manager needs a page, it looks for it in its buffer pool. If it does not find it there, it uses a service (IUCV *BLOCKIO or paging in VM, and VSAM in VSE) to read the page from DASD into a free space in its pool.

Since the buffer pools are part of a primary address space, the operating system treats them like part of the database manager code. If a buffer page is not referenced frequently, it may be moved out to system paging DASD by the VM or VSE paging system. In VM the page may also be moved out to expanded storage if it is available. (Refer to “Auxiliary Storage” on page 43.)

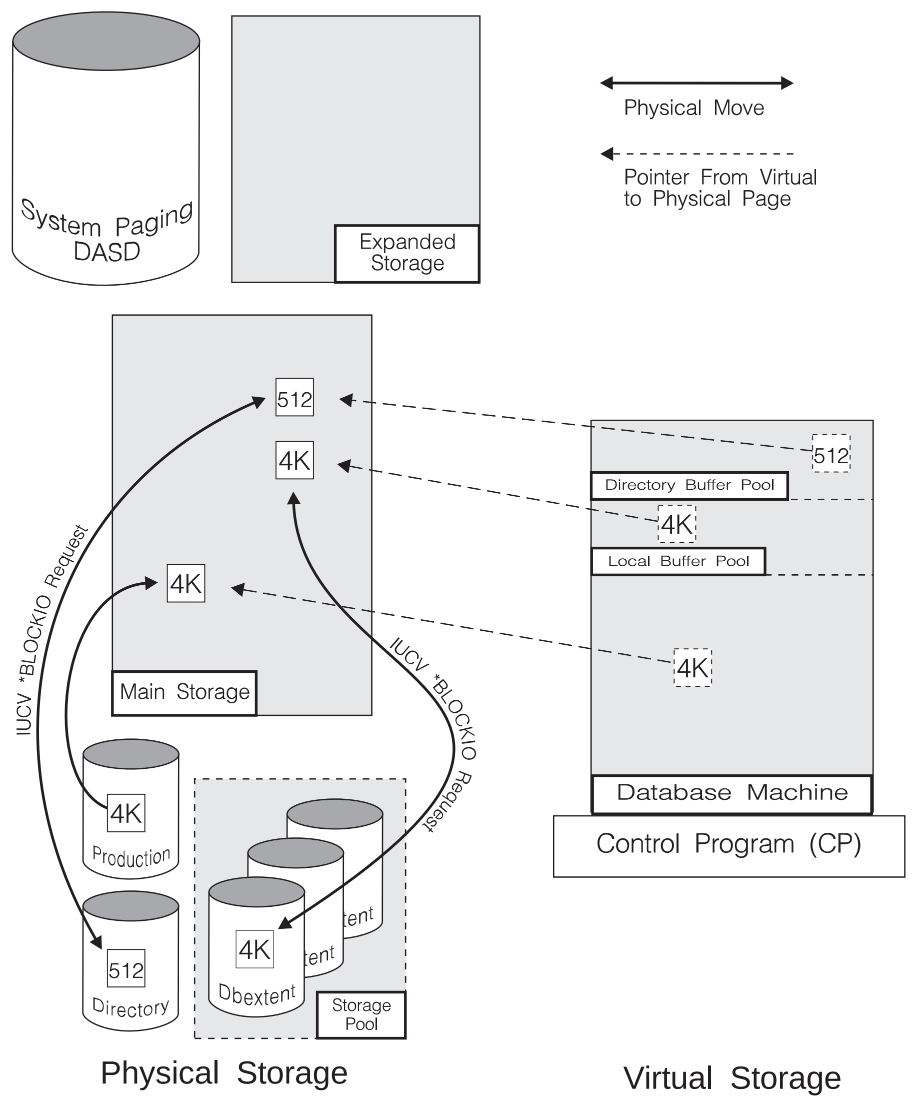


Figure 13. The Standard DB2 Server for VM DASD I/O System. The database manager explicitly directs the operating system to move pages to and from DASD. Once database machine pages are in main storage, they may be moved out to system paging DASD by the paging system. In VSE pages are moved by VSAM and the database machine is a database partition.

When the database manager needs a buffer for another page, it overwrites the "oldest" unmodified page in the pool with a new page. This is referred to as *releasing* a page or *stealing* a buffer.

While a page is in the buffer pool, the database manager may modify it. To ensure the integrity of your data, a modified page will not be released until it has been written back to DASD. If the database manager needs a buffer occupied by a modified page, it first writes the page to DASD, then loads the buffer with a new page.

Tuning Parameters

The sizes of these buffer pools are among the more important factors determining performance. You can significantly improve performance by optimizing these values. Unless your system's main storage is extremely constrained, the default values are probably too low.

Buffer pool sizes are set by initialization parameters:

- NDIRBUF, which is the number of 512-byte blocks in the directory buffer pool
- NPAGBUF, which is the number of 4KB pages in the local buffer pool.

The optimal buffer pool size is governed by the trade-off between database I/O and system paging I/O (refer to “Auxiliary Storage” on page 43). In general, an increase in the buffer pool sizes improves performance only if the resulting increase in system paging is small. Stated another way, the buffer pools should be backed up by a corresponding amount of available main storage.

Using a Large Buffer Pool: The database manager is designed to efficiently manage its buffer pools no matter how large they are. Very large buffer pools can be an excellent tuning choice **if** sufficient virtual and real storage is available.

Using a Small Buffer Pool: At the other extreme, if your environment is characterized by limited real storage and a relatively high paging rate, consider using smaller buffer pool sizes. Avoid extremely small buffer pools: they increase the likelihood that work has to be backed out because of buffer pool contention. Twenty buffer pages per real agent (20*NCUSERS) is an absolute minimum, and is usually too low for most applications.

Performance Indicator

The performance information available through the COUNTER operator command is helpful in guiding the selection of buffer pool sizes. Two especially useful measurements are the local buffers effective use and the directory buffer effective use values. (Refer to “COUNTER Operator Command” on page 23.)

There are no fixed guidelines as to what constitutes a good or bad value, because this depends upon the availability of main storage to back up the buffer pools, as described above. Of more interest are their relative values under different conditions. For example, before and after observations can be used to find out how effective an increase in the buffer pool size was in reducing database I/O. A large decrease in I/O indicates that the change was effective, whereas a small increase would suggest that the change was not worthwhile. Alternatively, calculate your buffer hit ratios (see “Measurements” on page 7) before and after your change.

Because directory buffers are eight times smaller than the local buffer pages, you can afford to be much more generous with them. Consider increasing NDIRBUF enough to cause the directory read rate (DIRREAD/sampling interval) to be very low. On a well tuned system, the directory pool effective use tends to be much higher than the local buffer effective use.

DB2 Data Spaces Support (VM/ESA only)

Consider using DB2 Data Spaces Support (DB2 Server DSS) instead of significantly increasing the size of your buffer pools.

With DB2 Data Spaces Support, if the database manager cannot find a page in its buffer pools, it “retrieves” data from a data space and stores it in a free buffer in its pool. When this happens, the operating system actually does most of the work. If the page is already in main storage, the operating system can move it directly to the buffer pool. If the page is in expanded storage or in DASD, the operating system moves it into main storage, and then copies it into a buffer. If your processor supports *Enhanced Move Page for VM*, pages are moved from expanded storage directly into a buffer.

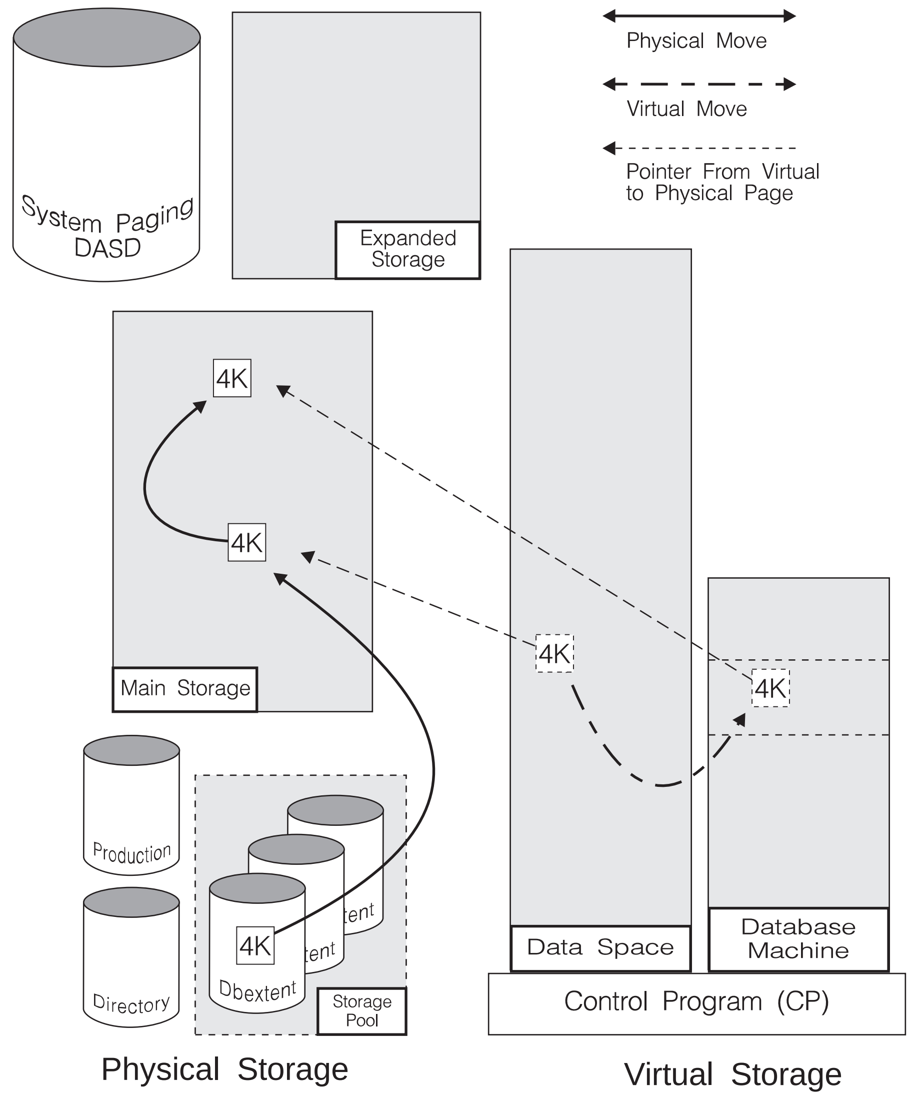


Figure 14. Page Movement with Data Spaces Support. The figure shows a page being read into main storage from DASD and then into a buffer pool.

The VM Paging subsystem can be much faster and more efficient than the standard DASD I/O system. The data spaces act like a large DASD cache, keeping the most recently used data in the fastest storage. While this is similar to using a large pool of buffers or minidisk caching, there are significant advantages to using Data Spaces Support over these two methods.

Some of the advantages are:

- Shorter path length
- Asynchronous page fault processing
- Striping
- Blocking and prefetching
- Dynamic working storage size management
- More asynchronous writes.

These are described in turn below.

Shorter Path Length: There is a series of internal processes between when the database manager requests a page from DASD, and when the operating system transfers it to main storage. This series is shorter when you use Data Spaces Support than when you use the standard DASD I/O system.

Asynchronous Page Fault Processing: Since the operating system treats the buffers like part of the database manager code, it may page them out to system paging DASD if it needs main storage. Without DB2 Server DSS, whenever the database manager needs a piece of code (or a buffer) that has been moved to paging DASD, it and all its users must wait for that page to return from DASD.

With Data Spaces Support, you can use a smaller pool of local buffers, decreasing the chance of a buffer being paged out. If a page fault occurs in a data space (the operating system cannot find the page in main or expanded storage) the database manager can proceed with other users and return to the original user when the fault has been resolved.

Striping: Without striping, the database manager completely fills one dbextent in a pool before using the next dbextent in the pool. When you use striping, the database manager tries to evenly distributed pages across all the dbextents in the pool. Striping spreads these groups of 16 pages across all the dbextents in a storage pool. If the dbextents are on separate physical devices, the operating system can read several groups of pages at the same time (asynchronously). This improves blocking and prefetching (see below), and helps you balance the load between DASD packs.

Striping also tries to keep related data pages physically close together on DASD. (It allocates pages in groups of 16.) Thus, when the operating system needs to retrieve related pages from DASD, there is a good chance that the pages will be located together. The operating system can then read in a whole series of pages with one I/O operation, which improves the performance of your system.

Blocking and Prefetching: When you use the Data Spaces Support, the operating system tracks the way you access pages. It records which pages you have used together (in a *block*) and the order in which you use them. Then, when the database manager requests a page from a data space, if the page is on DASD, CP will start retrieving (*prefetching*) other pages in the same block in the order you previously followed. Because DASD I/O can proceed in parallel (because of striping), this effectively places pages in main storage before the database manager needs them.

In some cases, the database manager will pass information to the operating system about how it expects to use pages. The operating system uses this information to modify its own reference pattern and thereby further improve prefetching.

Dynamic Working Storage Size Management: You can dynamically manage how the database manager uses main and expanded storage:

- You can set a target working storage size to control how much main and expanded storage your database machine uses.
- You can favor some storage pools over others by setting their working storage residence priority. This lets you improve the performance of critical storage pools, even if you have a limited amount of main and expanded storage.
- You can set a save interval. When the number of blocks of modified pages in a data space exceeds this parameter, the database manager directs the operating system to write all the modified pages in that data space to DASD. This reduces the number of modified pages in storage. As a result, there are fewer pages to be saved during checkpoint processing, which reduces checkpoint processing time.

More Asynchronous Writes: With Data Spaces Support, the database manager can write modified pages back to DASD “more” asynchronously than without it.

With Data Spaces Support off, if the database manager needs a buffer occupied by a modified page, it first writes the page to DASD, then loads the buffer with a new page.

When it does this, it puts the current agent into an I/O Wait State until the write is complete. Because the database manager continues to service agents that are not in wait states, this process is asynchronous *between agents*.

With Data Spaces Support on, when the database manager writes a modified page to a data space, the current agent is not put into a wait state. The operating system ensures that the page is eventually written to DASD (before the next checkpoint) without stopping the current agent. This process is asynchronous *within an agent* and therefore more asynchronous than without Data Spaces Support.

Using Virtual Disks

Your internal dbspaces can use a virtual disk to improve their performance. Virtual Disk Support lets you use a data space as a virtual disk. A virtual disk is much faster than a conventional disk because it uses main storage instead of DASD. A virtual disk appears to any program or job as just another disk, only faster. Refer to “Virtual Disk Support for VSE/ESA for Internal Dbspaces” on page 50 and 56.

Package Cache

The package cache works much the same as the buffer pools, except that instead of storing data pages, the package cache stores packages. When a package is loaded into the database machine's virtual storage, users can use it consecutively without reloading it each time. Unfortunately, separate users cannot use the same package at the same time. If a package is already in use when a user requests it, an additional copy will be loaded.

You need to trade-off the advantage of reducing your DASD I/O by having a large cache capable of storing a large number of packages, against the storage the packages consume.

Tuning Parameters

The package cache has a series of slots that contains information about the packages loaded into the database machine or partition. One slot is used for each package. The total number of slots available is determined at application server startup by two initialization parameters:

- NPACKAGE, which defines the maximum number of packages available for each real agent.
- NCUSERS, which is the number of real agent structures.

The number of slots in the package cache is calculated as follows:

```
NPACKAGE X NCUSERS
```

For example, if NPACKAGE is 10 and NCUSERS is 5, the number of slots in the cache is 50 (10X5). While NCUSERS is part of the calculation, do not use it to tune the size of the cache. Instead, increase or decrease NPACKAGE and set NCUSERS based on your requirements for real agents. (Refer to “Agents.”)

You can also set a **package cache threshold** that limits the number of packages that will remain in the cache. At the end of a logical unit of work (LUW), the database manager checks the number of packages in the cache. If that number exceeds the threshold, the database manager releases the package that has been in the cache the longest to make room for a new one.

The package cache threshold is determined at startup by an initialization parameter (NPACKPCT) and is calculated as follows:

```
                        NPACKPCT
NPACKAGE X NCUSERS X ----------
                          100
```

For example, if there are 50 slots in the cache and NPACKPCT is 80%, the package cache threshold is 40 packages (50X80/100). While NPACKAGE and NCUSERS appear in the calculation, do not tune the threshold with them, rather, use NPACKPCT.

Concurrency

Agents

The database manager uses a set of control blocks called an agent structure (or real agent) to service requests from multiple users accessing a common database.

There are always at least two 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.

Allocating Users to Agent Structures

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

In single user mode (SUM) this process is quite simple. There are three agents created: the Operator, Checkpoint, and User. At initialization time, the Operator agent performs the initialization functions. When initialization is complete, it becomes dormant and control is passed 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.

The User and Checkpoint agent alternate until the User agent finishes its work. Then the Checkpoint agent performs a final checkpoint and the Operator agent shuts down the application server.

In multiple user mode (MUM), when initialization has been completed, all the user agents start dormant. When a user first issues an SQL statement, a connection is established between the user and the database manager. The connection remains in effect until an explicit or implicit (COMMIT WORK RELEASE) release occurs.

VSE: In VSE, a batch user is connected to the application server by connecting a batch partition directly to a real agent in the database partition. An interactive user is connected to the server by establishing at least one link between the CICS partition and a real agent in the database partition. A remote DRDA user is connected to a pseudo agent in the server. The pseudo agent then connects to a real agent when one becomes available (refer to "Pseudo-Agents" on page 97).

VM: *In a IBM VM* system, a connection is established between the user machine and a pseudo-agent in the database machine. The pseudo agent then connects to a real agent when one becomes available (refer to "Pseudo-Agents" on page 97).

Tuning Parameters (NCUSERS)

In both VSE and VM the number of real agents is determined by the DB2 Server for VSE & VM initialization parameter NCUSERS. If you have the resources to support more users, increase NCUSERS. For example, ten users want to share the server, but there are only four real agents available. Six users must wait for one of the four to finish a logical unit of work before they receive access to a real agent. If you have the processing power to concurrently service all ten, there is no reason to make some wait.

However, remember that by increasing the level of concurrency in your system, you are also increasing overhead. Each additional real agent requires a minimum of 110KB of storage and, if you use the default size for your buffer pools, an additional 18KB of storage (four 4KB local buffers and four 512-byte directory buffers). Additional real agents can also:

- Increase the system paging in your database machine or partition, "Auxiliary Storage" on page 43
- Increase overall DASD I/O and buffer looks, "Database I/O" on page 89
- Increase locking contention, deadlocks, and lock escalations, refer to "Locking" on page 100
- Increase the size of the package cache, "Package Cache" on page 94.

Performance Indicator

SHOW CONNECT: Use the SHOW CONNECT or SHOW USERS operator command to see how many real agents are in use, and when appropriate how many pseudo agents are waiting for real agents. If all your real agents are never in use at the same time, you should reduce NCUSERS to save resources. If you consistently have more than five pseudo agents waiting for a real agent consider increasing NCUSERS by one if you have the resources to support an additional real agent. Refer to page 30.

For CICS users, refer to the performance indicator discussion under the heading "CICS."

CICS

Before a CICS user can access your application server, you must establish at least one link between the CICS partition and the database partition.

Tuning Parameter (CIRB)

The CIRB transaction defines one or more links, each of which is exclusively attached to a real agent until it is terminated by the CIRT transaction.

Because each link requires its own real agent it is important to establish an appropriate number of links. If you establish too many links for the number of concurrent users you expect to access your server through CICS, you will needlessly tie up real agents, and the resources they require. However, if you do not establish enough links, CICS users will be forced to wait for a free link. As well as causing a delay, not having enough links increases your overhead. Storage and processor time are consumed to concurrently manage these links. CICS must manage all the links in a queue and select one user each time a link becomes available.

To help you decide whether you should err on the side of too many links or not enough, consider which resources are more constrained, — those of CICS or those of your application server. If CICS resources are constrained, increase the number of links. If your server's resources are constrained, decrease the number of links.

Performance Indicator

There are two tools for CICS links, the CICSPARS/VSE report, and the CIRD transaction. The CICSPARS/VSE report presents historical information on how many waits for links occurred during a monitoring interval. In contrast, the CIRD transaction provides a snapshot of the same information.

Pseudo-Agents

Pseudo-agents allow many users to share, but not concurrently, a few real agent structures. This saves a significant amount of storage because each real agent requires a minimum of 110KB of storage, a pseudo-agent uses less than 600 bytes. (The 110KB value increases depending on how the real agent is currently being used, refer to the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manuals.)

Differences between VM and VSE

While pseudo agents are always used in a VM system, they are only used in a VSE system for remote DRDA users.

When a user CONNECTs to an application server, that user is allocated a pseudo-agent. The pseudo-agent is assigned to a real agent (assuming one is available) when the user sends an SQL statement to the server. 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.

Table 3. Real and Pseudo Agents

	Real Agent	**Pseudo Agent**
Storage per Agent	Minimum 110KB	600 bytes
Number (VM)	NCUSERS (DB2 Server for VM initialization parameter)	MAXCONN – minidisks – # of active Stored Procedure Servers - 1 (CP directory)
Number (VSE)	NCUSERS (DB2 Server for VSE initialization parameter)	RMTUSERS (remote DRDA users)
When assigned	At SQL command request	At CONNECT
When freed	End of LUW	End of connection
When no agents are available	Pseudo agent waits in FIFO queue	No connection, error message

Guest Sharing: With Guest Sharing, the DB2 Server for VSE 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.

Note: For CMS users to simultaneously access the database, NCUSERS must be greater than the number of CICS links.

Tuning Parameter

The only reason you would want to restrict the number of pseudo agents is to send an error message to a user indicating that no real agents are available instead of putting this user in a queue.

If this is not a problem, set the number of pseudo-agents to the maximum number of users that could possibly wish to connect to the application server at one time. The virtual storage requirement of 600 bytes per connection should be negligible.

VM (MAXCONN): The number of pseudo-agents allocated is equal to the value of MAXCONN specified in the CP directory minus the number of CMS minidisks used for the database machine and minus one for the connection to *IDENT.

VSE (RMTUSERS): Pseudo-agents are not used for CICS users nor are they used for batch partition users. However, they are used for remote DRDA users. The RMTUSERS initialization parameter sets the number of pseudo agents available to remote DRDA users. This limits the number of remote DRDA users that can access the server at any one time because each DRDA user requires one pseudo agent. To calculate the number of real agents that can be shared between remote DRDA users, subtract the number of active batch partition users and the number of CIRB initiated CICS connections from the number of real agents (NCUSERS). For example, consider the following:

- NCUSERS is 10, refer to “Tuning Parameters (NCUSERS)” on page 96
- There are 5 CIRB initiated CICS connections, refer to “Tuning Parameter (CIRB)” on page 97
- Two batch partitions are active
- RMTUSERS is 20.

This means that there are three (10-5-2) real agents that must be shared between as many as 20 pseudo agents (each connected to a remote DRDA user).

Privileged Remote DRDA User (VSE Server)

You can specify that a remote DRDA user is *privileged*. Normally, a remote DRDA user is assigned one pseudo agent during the time it is connected to an application server, and shares the available real agents with the other pseudo agents. It always releases a real agent at the end of a logical unit of work (LUW) (refer to “Agents” on page 95). However, a privileged remote DRDA user holds a real agent until it disconnects from the server. Only specify a user as privileged if you expect it to constantly submit work to the server. For example, large batch applications. Interactive applications are not good candidates.

To specify a privileged remote DRDA user, you need to update the TPN entry in the DBNAME directory for that user. For instructions, refer to the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manuals.

Performance Indicator (SHOW USERS)

You can use the SHOW USERS operator command to look for consistently free pseudo agents. This indicates that you could probably reduce MAXCONN for VM, or RMTUSERS for VSE. However, constantly busy pseudo agents may indicate that you should increase them.

The most effective indicator that either parameter is set too low is to look for complaints from your users that they cannot connect to the application server. They will receive SQLCODE=-933 with SQLSTATE=57030.

Dispatching Agents

Real agents are placed in a queue and wait there until they receive a slice of the processor time from the database manager (referred to as being dispatched). Before an agent is dispatched, it must:

- Not be in a wait state. For example it cannot be waiting for an I/O operation to complete or waiting for a lock held by another user.
- Be at the top of the queue. This is determined by two processes: Prioritization, and Fair Share Auditing.

Prioritization

After each dispatch, the agents in the queue are reprioritized so that shorter LUWs are moved to the top of the queue. “Special purpose” or “system” agents, such as the operator or checkpoint agents, 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.

Fair Share Auditing

Fair Share Auditing is invoked at regular intervals, during which the dispatch queue is scanned for a “deprived” agent and, if one is found, it is moved to the top of the queue. A deprived agent is one that has referenced the buffer pools less than a calculated *fair* value. This value is based on the average number of references per LUW and per real agent.

Tuning Parameter (DISPBIAS)

You can affect the frequency of Fair Share Auditing with the DISPBIAS initialization parameter. You can set it from 1 to 10 and it defaults to 7. The higher the number the less frequent the audits. 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 to short LUWs.

Performance Indicator

While there are no quantifiable indicators for this parameter, you can determine if it needs tuning by listening to your users. Can you differentiate between users that use short LUWs and those that use long LUWs? If people with short LUWs complain, try increasing DISPBIAS. If people with long LUWs complain, try decreasing DISPBIAS.

Startup Mode

If there are periods where you only need to support large sequential jobs (for example, an overnight batch window or long DBS utility job), consider running your application server in single user mode (SUM). You will reduce the overhead of both concurrent processing and in communications.

In single user mode, pseudo agents are not created and there is no overhead associated with prioritizing real agents or with fair share auditing. Also, since the application server does not need to communicate with a user machine or partition, you reduce overhead by eliminating the APPC/VM (in VM) or XPCC (in VSE) conversations.

Locking

To optimize your application server's performance, you need to minimize the overhead of locking, while you maintain the integrity of your data. Concentrate on three areas:

- Reduce lock contention by reducing the number of locks you require and the duration of each lock, refer to “Locking Contention” on page 101.
- Reduce the number of lock escalations, refer to “Lock Escalation” on page 107.
- Reduce the number of potential deadlocks, refer to “Deadlock” on page 108.

There are a number of techniques to help you with each area.

Locking Contention

Lock contention occurs when an agent tries to lock an object that is already locked with a conflicting mode by another agent. The more locks the database manager uses and the longer each lock lasts the greater the probability that contention will occur.

To understand when locks come into contention, you must first understand:

- Locking Hierarchy
- Lock Modes
- Lock Compatibility

Locking Hierarchy

The database manager locks objects in the database according to a hierarchy.

```
                  DBSPACE
                     |
                     V
                   TABLE
                     |
              ┌──────┴───────┐
         DATA PAGE       INDEX PAGE
              |              |
              V              V
             ROW        INDEX KEY VALUE
```

Figure 15. Locking Hierarchy

The database manager grants locks in the order of the hierarchy. Thus, an agent accessing a row, has to first obtain a lock on the dbspace, the table and the page that contain the row before it can obtain a lock on the row itself. (After the first row is locked, it is not necessary to get the DBSPACE and table locks again since they will be held until the end of the LUW.)

Lock Modes

There are eight types of locks, or *lock modes*:

Share (S)
This type locks an object (dbspace, table, page or row) for reading when using the repeatable read (RR) or the cursor stability (CS) isolation levels. Other agents can obtain share locks on the same object and look at it simultaneously.

Exclusive (X)
This type locks an object for updating. Objects locked in X mode can be read by applications using isolation level UR.

Super Exclusive (Z)
This type locks an object (dbspace, table, page or row) for updating. Other agents cannot obtain any other locks on the same object and cannot read or manipulate it in any way, even if using isolation UR.

Intent Share (IS)
This type indicates that a share lock is being used on an object lower in the hierarchy. For example, if an agent needs a share lock on a table it

first needs to obtain an intent share lock on the dbspace that contains the table.

Intent Exclusive (IX)
This type indicates that an exclusive lock is being used on an object lower in the hierarchy.

Intent None (IN)
This type indicates that no locks are held on objects lower in the hierarchy for reading using isolation level UR. For example, an application using isolation level UR to read a row will get an IN lock on the dbspace and table, but will then not hold any locks on the page or row. See the *DB2 Server for VSE Diagnosis Guide and Reference* or *DB2 Server for VM Diagnosis Guide and Reference* manuals for more information.

Share with Intent Exclusive (SIX)
This type indicates that a share lock was held on this object but that an exclusive lock is now being used on an object lower in the hierarchy.

Update (U)
This type of lock is used during a FETCH when the cursor is declared FOR UPDATE. It locks an object for reading, but indicates that an update may be required. If the agent finishes with an object without updating it, the lock is downgraded to share. If an update is required, the lock is upgraded to exclusive. While the update lock is held, other agents can obtain share locks on the same object to look at it simultaneously, but they cannot obtain update or exclusive locks on it.

Lock Duration

Locks can be held and released almost instantly or held until the end of the current logical unit of work. The lock duration depends on the lock mode, the type of internal data manipulation call, and the isolation level (refer to “Isolation Level” on page 104). A detailed table including the relationships between all of these is included in the *DB2 Server for VM Diagnosis Guide and Reference* or the *DB2 Server for VSE Diagnosis Guide and Reference* manuals.

Lock Compatibility

The main purpose of having different lock modes is to be able to define which requests to access a certain object are compatible with other requests and which are incompatible. The matrix in Table 4 on page 103 indicates which lock modes 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. Either way a lock contention occurs.

Table 4. Compatibility of Lock Modes

	MODE OF LOCK REQUEST							
MODE OF LOCK HOLD	IN	IS	IX	S	U	SIX	X	Z
IN	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
IS	Yes	Yes	Yes	Yes	Yes	Yes	No	No
IX	Yes	Yes	Yes	No	No	No	No	No
S	Yes	Yes	No	Yes	Yes	No	No	No
U	Yes	Yes	No	Yes	No	No	No	No
SIX	Yes	Yes	No	No	No	No	No	No
X	Yes	No	No	No	No	No	No	No
Z	No	No	No	No	No	No	No	No

Public and 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.

Number of Concurrent Users

The amount of locking contention is directly related to the number of concurrent users allowed to access the application server (NCUSERS). While you could reduce locking contention by reducing NCUSERS, this would affect your overall response time, refer to “Tuning Parameters (NCUSERS)” on page 96. You need to find a balance between having users wait for a lock and having them wait for access to the server.

Minimum Lock Level

The smallest possible lock level is a single row in a table. However, you can increase the minimum lock level in a dbspace to be a single page or the dbspace itself. Using a larger minimum lock level reduces the number of locks required, which reduces locking overhead but may increase locking contention. (The default lock level is a single page.)

Tuning Parameter: You can define a larger lock level with the ACQUIRE DBSPACE statement or change an existing size with ALTER DBSPACE.

The default lock level (PAGE) should be appropriate for most applications. However, you may consider using the DBSPACE lock level if your application is read-only and accesses the data primarily through dbspace scans.

Only consider ROW level locking for applications that access small answer sets through index scans. Also, this level should not be used with an application performing a dbspace scan using cursor stability.

Performance Indicator: You can find the minimum lock size for a given dbspace in the SYSTEM.SYSDBSPACES catalog table in the LOCKMODE column. Refer to “SYSTEM.SYSDBSPACES” on page 40.

Indexes

Remember that index pages and key values are locked for any SQL statement that uses the index or whenever the table that they are indexing is updated. If your indexes are not critical to fast access to your data, you can reduce lock contention by reducing the number of indexes defined on your data. In other words, do not create unnecessary indexes.

Unique Indexes and Row Level Locking: If you are using row level locking, make sure that your tables contain a unique index. They perform better, with respect to locking contention, because the database manager is able to determine the exact row it needs to lock. Without a unique index it may unnecessarily lock several rows at once. If you cannot create a unique index on a single column, create a unique multicolumn index. For example if you want to sort the EMPLOYEE table by the JOB column, create an index on JOB and EMPNO (JOB,EMPNO). The index will still sort by JOB, but it can also be a UNIQUE index.

Access Path

You can usually reduce the number of locks required for a particular SQL statement by ensuring that it is accessing the data as efficiently as possible. For information on access path selection, refer to Chapter 5, “Improving Data Access Performance” on page 125.

Logical Unit of Work

Since a lock will never last longer than a logical unit of work (refer to “Logical Units of Work” on page 110), it is critical that you make your LUWs as short as possible. Commit work frequently, even if you are only reading tables.

Do not use ROLLBACK WORK to release locks. While rolling back a LUW also releases locks, a roll back involves more work and processor time than a commit work and should only be used when you want to undo updates, inserts, or deletions.

Isolation Level

You can set the isolation level for a particular application program during preprocessing. It represents the degree of independence that the application program will have from other programs. A lower isolation level maximizes concurrency and performance but increases the risk of inconsistent data appearing in applications. There are three isolation levels: *repeatable read*, *cursor stability*, and *uncommitted read*.

Repeatable Read (default): A repeatable read application program locks every object it accesses until the end of the current logical unit of work. It guarantees that within a logical unit of work it can repeatedly read the same row of data without having it changed by some other user. With repeatable read, a user is completely isolated from interference by other applications. Other users must wait until your logical unit of work is complete before they can modify the data you were using.

Cursor Stability: A cursor stability application program only locks an object for as long as it is directly accessing it. This allows more than one user to work on the same data at the same time. It is possible to issue the same query twice within a logical unit of work and get different results. That is, rows in a table, or pages in a DBSPACE, that you have already read are subject to change by other users. It also means that the data may appear “inconsistent.” If this is not a problem, and will not affect the integrity of your application program, seriously consider using cursor stability. It can significantly reduce the locking contention in your system.

Cursor stability only applies to tables in PUBLIC DBSPACEs with PAGE or ROW level locking. An application program that accesses tables in PRIVATE DBSPACEs or PUBLIC DBSPACEs with DBSPACE level locking always act like a repeatable read program.

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.

Uncommitted Read: Many uncommitted read (UR) application programs can query the same data simultaneously while the data is being updated by another application. This isolation level prevents read-only applications from waiting on applications that have changed or may change the data about to be read. Uncommitted read provides the lowest degree of isolation and hence greater concurrency and throughput.

Since isolation level UR gives applications the ability to read data that is not necessarily committed, data can appear to be inconsistent. For example, it is possible for you to issue the same query twice within a logical unit of work and get different results. You must be very careful when deciding to use uncommitted read for your applications. Only choose it for an application if it is not important that the data read is necessarily committed.

Note: Uncommitted read 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.

Isolation level uncommitted read (UR) is defined as follows:

- An application can see uncommitted changes made by other application processes
- An application cannot update uncommitted changes made by other application processes
- The re-execution of a statement can be affected by other application processes
- Uncommitted updated rows cannot be updated by other application processes
- Uncommitted updated rows can only be read by application processes using UR
- Accessed rows can be updated by other application processes
- Accessed rows can be read by other application processes
- The current row of a read-only cursor can be changed by other application processes

- The current row of an updatable cursor cannot be changed by other application processes.

User Defined: While you normally set the isolation level for a program when you preprocess it, you may allow the application program to dynamically set its isolation levels during execution. For more information on this, refer to the USER isolation level in the *DB2 Server for VM Application Programming* or the *DB2 Server for VSE Application Programming* manuals.

Isolation Level and Updates:

Note: The isolation level does not affect the duration of the locks held on data that have 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.

Guidelines for Selecting an Isolation Level: We recommend that you use cursor stability whenever possible because it reduces the duration of locks for the application program that uses it. For even further reductions in locking and lock durations, you may consider using uncommitted read. Only use this isolation level for applications in which data integrity is not important. The effects of cursor stability and uncommitted read can be very subtle. Specific guidelines for selecting isolation levels are in the appropriate DB2 Server for VSE & VM manuals. For guidelines on selecting an isolation level in application programs, see the *DB2 Server for VM Application Programming* or the *DB2 Server for VSE Application Programming* manuals. For guidelines that apply to the DBS utility, see the *DB2 Server for VSE & VM Database Services Utility* or the *DB2 Server for VSE & VM Database Services Utility* manuals. For ISQL guidelines, see the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* or the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manuals.

Catalog Tables

Catalog tables can be exclusively locked by:

- Data definition statements
- Data control statements (granting authorizations)
- Preprocessing
- Dynamic repreprocessing
- Inserts (loading tables and dbspaces)
- Extended dynamic CREATE PROGRAM, PREPARE, or DROP STATEMENT
- UPDATE STATISTICS

Try to avoid any or all of these during peak load periods and try not to include them in your application programs. If you cannot avoid them, at least COMMIT WORK after each statement.

Performance Indicator

To display locking contention as it occurs, use the SHOW LOCK operator commands. They can help you identify agents that are locking other agents out of critical data and solve immediate locking problems.

To test the frequency of lock contentions after they occur, use the COUNTER operator command. Specify the WAITLOCK counter to get the number of lock requests that resulted in a wait.

Lock Escalation

The database manager uses internal control blocks called lock request blocks (LRBs) to manage locking. 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 initialization parameter NLRBU. The sum of the number of LRBs held by all agents cannot exceed the limit defined by the NLRBS initialization parameter. 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: 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.

The lock manager requests a lock on the chosen DBSPACE. The lock mode requested is the same as the most restrictive lock that the agent holds in the DBSPACE. For example, if the agent holds any Z locks, a Z lock is requested. The next choice would be an X lock, followed by an S lock. 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.

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.

Tuning Parameters (NLRBU, NLRBS): If an application program is causing too many lock escalations, consider the following alternatives:

- Change the locking level of some of the dbspace(s) used by the application (for example, from ROW to PAGE) by using either the SQL ALTER DBSPACE or the SQL LOCK statement. This will reduce the number of locks required by the application.

  Note: Using a larger minimum lock level can increase locking contention. Refer to “Minimum Lock Level” on page 103.

- Reduce the duration of the locks by changing the application: add SQL COMMIT WORK statements to the application.
- If appropriate, consider running the application by itself: either in single user mode, where no locking is required, or in multiple user mode with a reduced NCUSERS.
- If you are currently using the repeatable read isolation level, consider using cursor stability or uncommitted read.

If you cannot reduce the number of lock escalations, you may need to increase the number of available lock request blocks by increasing the the NLRBU, and NLRBS initialization parameters.

To establish the lock request block requirements for running an DB2 Server for VSE & VM preprocessor, or for an application that is causing escalation problems:

1. Start the application server in multiple user mode with NCUSERS=1, NLRBU about five times its current setting, and NLRBS set to the same value as NLRBU.
2. Start the application and allow it to complete processing.
3. Verify that no escalation occurred by displaying the ESCALATE and LOCKLMT counters. If no escalation occurred, enter the SHOW LOCK MATRIX operator command. MAX USED BY LUW will show the number of lock request blocks required.
4. If an escalation did occur, set NLRBU to a value greater than or equal to MAX USED BY LUW, then start the application server again, and rerun the application.

Performance Indicators (COUNTER, SHOW LOCK MATRIX): To test the frequency of lock escalations, use the **COUNTER operator command**, refer to “COUNTER Operator Command” on page 23. Specify both the ESCALATE and the LOCKLMT counters to get the number of successful escalations and the number of unsuccessful escalation attempts respectively. (An escalation can fail if the LUW that reached the lock limit is rolled back because of a deadlock, or if a sufficient number of lock request blocks cannot be freed.)

Note: ESCALATE and LOCKLMT may increase during preprocessing, because locks are required then as well.

You can also use the **SHOW LOCK MATRIX operator command** that displays information about lock request block usage, refer to “Lock Escalation” on page 38. You can determine whether unexpected delays are caused by locking; monitor how the database manager is using lock request blocks; and determine the lock request blocks required to preprocess a single application.

One of the values displayed by SHOW LOCK MATRIX is called MAX USED BY LUW. It is the maximum number of lock request blocks used by any one application during a logical unit of work. (When any LUW exceeds NLRBU and the escalation process occurs, MAX USED BY LUW is set to zero.)

In addition you can look for SQLCODE **-912** (SQLSTATE 57028), or SQLCODE **-915** (SQLSTATE 57029). These indicate rollbacks that occur because of, insufficient lock request blocks for the database manager, or insufficient lock request blocks for a user application, respectively.

Deadlock

The database manager performs deadlock detection prior to placing any agent into a lock wait. A deadlock occurs when 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. For example, consider two users, LAWRENCE and VERONICA:

1. LAWRENCE selects rows from the EMPLOYEE table, placing a SHARE (S) lock on the table.
2. VERONICA also selects from the same table, also placing a SHARE (S) lock on it.
3. LAWRENCE tries to UPDATE the employee table, but cannot because he is placed in a lock wait. The EXCLUSIVE (X) lock he needs before he can update the table is incompatible with VERONICA's SHARE (S) lock.
4. VERONICA also tries to UPDATE the same table, but cannot. The EXCLUSIVE (X) lock she needs before she can update the table is incompatible with LAWRENCE's SHARE (S) lock. However, before she is placed in a lock wait, the database manager detects a potential deadlock.
5. The database manager rolls back VERONICA's logical unit of work because it is younger than LAWRENCE's LUW.
6. LAWRENCE receives the lock he needs because VERONICA loses her SHARE (S) lock when her LUW ends, and VERONICA receives the following message:

```
UPDATE SQLDBA.EMPLOYEE SET SALARY=60000 WHERE LASTNAME='HAAS'
ARI7955I THE SYSTEM ENDED YOUR QUERY RESULT TO PROCESS YOUR COMMAND.
ARI0503E AN SQL ERROR HAS OCCURRED.
         THE CURRENT LOGICAL UNIT OF WORK HAS BEEN
         ROLLED BACK DUE TO A DEADLOCK.  IT WAS WAITING
         FOR A PAGE LOCK IN DBSPACE = 17
         HELD BY USER LAWRENCE.
ARI0505I SQLCODE = -911 ROWCOUNT = 0
ARI0504I SQLERRP: ARIXRSS SQLERRD1: -110 SQLERRD2: -99
ARI0502I FOLLOWING SQL WARNING CONDITIONS ENCOUNTERED:
         NULLWHERE NOLUW
ARI7021E THE APPLICATION SERVER HAS ISSUED A ROLLBACK
         STATEMENT. ALL WORK ENTERED FOR PROCESSING SINCE
         THE LAST COMMIT STATEMENT WAS ROLLED BACK.
         YOU MAY HAVE TO REENTER SOME STATEMENTS.
```

While the database manager does not allow deadlocks to occur, the more potential deadlock situations that you create the more resources are required to avoid them.

Note: The time required to detect potential deadlocks increases exponentially (power of two) with the number of real agent structures in your database manager. For example, it takes 100 times longer to process deadlocks when NCUSERS=20 than it does when NCUSERS=2.

Tuning Parameters

Application Design: Look for two applications that access the same data in the opposite order. If you can, switch the order of access for one application so they both use the same order.

Reschedule Applications: If you find that two applications often create deadlocks, try to reschedule them to run at different times of the day.

Reduce Lock Contention: The other way to reduce potential deadlocks is to simply reduce the number and duration of locks that your database manager needs to use, refer to “Locking Contention” on page 101.

Reduce Lock Escalation: Escalation can also cause deadlocks. For example, suppose two users are updating tables in a dbspace. When the lock size is escalated to a dbspace level, both users can be locked out, with each waiting for the other to complete a logical unit of work. Refer to “Lock Escalation” on page 107.

Performance Indicator

To determine if deadlocks are a problem, look for users receiving SQLCODE **-911** (SQLSTATE 40001, rollback due to deadlock).

Note: This message may also be received during preprocessing, as the locks are required then as well.

To test the frequency of deadlocks, use the COUNTER operator command and specify the DEADLOCK counter. It displays the number of deadlocks detected, each of which causes a rollback.

Recovery

Logical Units of Work

When a user or an application program has made a change or a group of related changes to the database, and if the application in question completed successfully, the user or program issues an SQL COMMIT WORK statement to the application server, to commit these changes to the database. If the application did not complete successfully, the user instead issues an SQL ROLLBACK WORK statement, which undoes all the changes made up to the point of the error since the last COMMIT WORK statement, or since the start of the program or session.

A group of SQL statements is called a *logical unit of work (LUW).* An LUW can be as small as one statement, or as large as an entire application execution (or ISQL session). All SQL statements are executed within an LUW. If no LUW exists when a statement is issued, then the database manager creates one implicitly.

CMS Work Units (VM)

Users working on a VM operating system can take advantage of CMS work units, which allow them to maintain more than one logical unit of work (LUW) at a time. With separate CMS work units, application programs can be independent of one another. For example, a user can run a program, and in the middle of an LUW, have that program call a second program which runs in a separate CMS work unit. When work is committed in the second program, it does not affect the active LUW in the first program.

Note: CMS work units require extra processing overhead, so should only be used when necessary. If an application does not need this support, set the WORKUNIT option of the SQLINIT command to NO.

Checkpoints

A checkpoint is an internal operation where the database manager writes modified data and status information to DASD, and writes a summary status record to the log.

What occurs during the Checkpoint Process?

When the database manager takes a checkpoint:

- It writes the contents of the local and directory buffer pools to DASD.
- It frees all *shadow pages*. (Whenever it “modifies” a page in a storage pool, it creates a new page in the same pool, and keeps the original as a shadow page. Refer to “Shadow Pages” on page 66).
- If LOGMODE=Y (no archive), the database manager clears space in the log up to the beginning of the oldest LUW still active when the checkpoint is taken.
- It updates the directory pages to account for released shadow pages and updated page allocation maps.

When do Checkpoints Occur?

A checkpoint is scheduled when:

- The number of log pages specified by the CHKINTVL initialization parameter have been written to the log, refer to “Choosing the Checkpoint Interval.”
- 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.)
- A COMMIT WORK is processed in single user mode with no logging (LOGMODE=N)
- The percentage of free pages in a storage pool reaches the minimum specified by the SOSLEVEL initialization parameter, refer to “Short on Storage Cushion” on page 62. (This does not apply when LOGMODE=N.)
- A DROP DBSPACE is issued.
- Soft recovery processing is complete during startup.
- An archive (both before and after) is performed.
- A shutdown is issued, either in multiple user mode (MUM) or single user mode (SUM).
- A log-full condition occurs, refer to “Log Cushion and Automatically Initiated Archives” on page 115.
- An LUW that updates data in a nonrecoverable storage pool ends.

Performance Implications

A checkpoint has two performance implications:

- It performs a high amount of I/O to DASD. It writes all the modified buffer pages and data space pages back to DASD, and updates the directory disk.
- It holds up processing. User agents must wait until the checkpoint is finished before they can proceed.

Choosing the Checkpoint Interval

To control the duration between checkpoints, use the CHKINTVL initialization parameter. This parameter specifies how many log pages the database manager will fill before it takes its next checkpoint.

Setting the Time Between Checkpoints: The time between checkpoints depends on the number of modifications you make to the database. If logging is turned on, the database manager writes to the log every time you perform an insert, update, or delete. The more modifications you make, the faster you will reach a checkpoint. If you only perform queries, the database manager may never perform a checkpoint.

We recommend that you adjust the CHKINTVL parameter so that the database manager takes a checkpoint every 10 to 15 minutes. Should you experience a system failure, it should take you no longer than 10 to 15 minutes to restart the database manager once you have recovered your system. If you adjust CHKINTVL so that checkpoints occur less frequently, for example every four hours, it may take up to or more than four hours to restart your database.

Many installations find that the optimum CHKINTVL setting is between 50 and 300. Installations with large, randomly modified databases are in the lower end of that range, and installations with small databases tend to be in the upper end of that range. Large databases having a relatively low frequency of random modifications also tend to be in the upper end of that range.

If you set the CHKINTVL parameter **too low**, you minimize the risk of filling the log or storage pools. However, while each checkpoint is faster, you increase the overall number of checkpoints.

If you set it **too high**, you lower the overhead associated with checkpoint processing. However, consider the following adverse affects:

- It may take longer to recover from a system failure.
- You risk filling the log and storage pools if you are running with LOGMODE=Y.

  This consideration does not apply if you are doing archiving (LOGMODE=A or L), because in that situation log space is reclaimed only when the database or log is archived.
- You may see an increase in the time required to complete a single checkpoint.

  However, unless your standard workload includes a significant amount of random data modifications over a relatively large area (more than 100MB), you probably will not notice significant delays. The effect is unimportant if the database is small or if there is very little random data modification activity. Bulk sequential data modifications also do not generally cause problems.
- You will probably require more storage to support additional shadow pages.

  Whenever the database manager modifies a page in a storage pool, it creates a new page in the same pool, and keeps the original as a shadow page. Therefore the longer the period between checkpoints, the more modified pages will accumulate in your storage pools. 25% free storage in each storage pool is generally sufficient, refer to “Short on Storage Cushion” on page 62.

Forcing Checkpoints: You can avoid checkpoint processing during peak periods by manually forcing them to occur when required. For example, consider a bank that processes a large number of transactions when its customers are on break for lunch, between 11:00AM and 1:00PM. A checkpoint could lock tellers out of the database for several minutes frustrating both the tellers and the bank's customers.

To avoid a checkpoint, the checkpoint interval is set very high. Just before the lunch rush, an empty dbspace, created for the purpose, is dropped to force a checkpoint. After the rush, the dbspace is recreated and dropped again to force another checkpoint and ensure that the lunch time work is saved to DASD.

If you plan to use this method to control when checkpoints occur, create a plan that specifically indicates when each is to occur and make sure that it is followed. Not performing any checkpoints can cause more performance problems in the long run than you will avoid in the short run.

DB2 Data Spaces Support: The DB2 Server DSS feature can make checkpoint processing faster by limiting the number of modified pages in main and expanded storage. When the number of modified pages in a data space exceeds an initialization parameter called the save interval (SAVEINTV), the database manager directs the operating system to save all the modified pages in that data space to DASD. Unlike the save that occurs during checkpoint, the database manager can continue to service users while this is being done. For more information on the save interval, refer to the *DB2 Data Spaces Support* manual.

Logging and Archiving

Log

| A log is a file maintained on DASD that records all the changes to the database.
| Each time a DML statement (for example, INSERT, DELETE, UPDATE) is
| processed by the database manager, the old and new values are written in the log.
| If any changes to the database must be undone or redone, you can use the log to
| restore the data to its proper state.

Archive

An archive is a copy of data in your database at the time the archive was made. You can archive an entire database, a portion of the database, or even the log. Typically, you use archives to recover from a DASD failure.

You can create three different archives:

Selective Archive
: Is a copy of individual tables or even dbspaces. You can create a selective archive using either the DBS utility or IBM DB2 for VM Control Center (refer to page 9).

Database Archive
: Is a copy of the entire database at a specific time. A database archive includes the database directory and all dbextents, but excludes the log.

Log Archive
: Is a copy of the current log on either tape or disk.

| Note: By using the DB2 for VSE & VM Data Restore Feature, you can
| also create incremental archives and restore individual storage pools.

Dual Logs

A database must have at least one log. Optionally, you can define a second log to keep a duplicate copy of the database changes. Then, if a DASD failure occurs on one log, the other is still available.

Note: To ensure that you really have true dual log protection, each log file or minidisk must reside on a separate DASD volume. For information on this "dual log" option, see the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manuals.

Choosing a Logmode

You can choose from four different log mode values:

LOGMODE=Y
: A log records all changes to the database. These are stored in the log until a **checkpoint** saves the changes to DASD. It is totally your responsibility to schedule archives of the database, because the database manager never initiates any for you. If the application server or your operating system abends, you can recover up to and including the last complete LUW. If you suffer a database DASD failure, you can recover from your last archive. This value is the default.

LOGMODE=A
: A log records all changes to the database. These are stored in the log until a database archive occurs. This ensures that you can recover from either an abend or a DASD failure up to and including the last complete LUW. The database manager automatically initiates a **database archive** when the log is nearly full. While you can wait for this to occur, it is more efficient to perform regular archives yourself, refer to “Log Cushion and Automatically Initiated Archives” on page 115.

LOGMODE=L
: A log records all changes to the database. These are stored in the log until a log archive occurs. This lets you recover using the last database archive plus subsequent log archives. You can recover up to and including the last complete LUW.

Note: Before you can use LOGMODE=L, you must create a database archive.

The database manager automatically initiates a **log archive** when it is nearly full. While you can wait for this to occur, it is more efficient to perform regular archives yourself, refer to “Log Cushion and Automatically Initiated Archives” on page 115.

LOGMODE=N
: Indicates that nothing is recorded in the log. This option is not recommended for normal operation and it is only available in single user mode.

When you choose a log mode, decide how much protection you want, and the amount of time you can spend in recovering data.

If you are running in single user mode (SUM) and you do not need to protect your data from either system or DASD failures, specify LOGMODE=N. The application server will run faster because it will not require the extra time to create archives or maintain a log.

If you do not need to protect your data from DASD failures, specify LOGMODE=Y. The application server will run faster because it will not require the extra time to create archives and you can maintain a smaller log.

LOGMODE=Y cannot protect you from DASD failures because the contents of the log are only saved until the next checkpoint. After the checkpoint, the current contents of the log can be overwritten by new changes. If several checkpoints have occurred since your last database archive, you cannot use the contents of the log to recover.

If you must have the ability to recover from DASD failures, choose either mode A or L. With LOGMODE=A an archive of the entire database is created periodically, so you can restore your entire database or individual storage pools by using the latest database archive along with the contents of the current log. With LOGMODE=L, archives are also taken but you can create archives of the database less frequently than with mode A, because you have log archives as well. If a DASD failure occurs, you can restore the entire database or individual storage pools by using the latest database archive, the sequence of log archives that follow it, and the contents of the current log. If you are doing a log archive for the first time you will be prompted to do a database archive first. You will not be prompted again. You must schedule any subsequent archives yourself.

To decide between LOGMODE=A or L, consider:

- How important it is to recover quickly after a DASD failure. You recover more quickly with LOGMODE=A.
- How much time you can devote to taking archives. Because the log is usually smaller than the database, log archives require less time to create than database archives. You can create both archives when you stop the application server, or while users are still accessing data in the database. If you create a database archive when users are accessing data in the database, they must wait longer for the application server to process their requests.

When you choose a log mode, use it whenever you start the system. Do not change the log mode without thought and planning. If you must do so, you may have to carry out additional procedures. For more information, see the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manuals.

Tuning Parameters

Log Cushion and Automatically Initiated Archives: The **SLOGCUSH** initialization parameter defines when automatically initiated log-full processing begins. It is expressed in terms of a percentage of the log. When the log fills to the SLOGCUSH value, the database manager aborts the oldest active logical units of work until enough log space is freed to bring the percentage of the log below the SLOGCUSH level.

The **ARCHPCT** initialization parameter defines when automatically initiated archives will occur. It is also expressed in terms of a percentage of the log. When the log fills to the ARCHPCT value, the database manager forces either a log or database archive depending on whether it is running in LOGMODE A or L.

Ideally, you should never reach SLOGCUSH or ARCHPCT. Log-full processing and automatically initiated archives reduce performance, and often occur during peak workloads, so avoid them by:

- Ensuring that your log is large enough.
- Trying to maintain enough free log space through regular log or database archives (LOGMODE=A or L) and through regular checkpoints (LOGMODE=Y).

  Note: Checkpoints only free space in the log when you run your application server with LOGMODE=Y.
- COMMITing WORK frequently to avoid long running LUWs.
- Running very long LUWs in single user mode without logging (LOGMODE=N).

Since performing an archive impacts performance less than log-full processing avoid the latter by:

- Setting SLOGCUSH > ARCHPCT.
- Ensure that there is a comfortable difference between SLOGCUSH and ARCHPCT. (If the SLOGCUSH percentage is reached during an online archive operation, all SQL processing is suspended until the archive operation is complete.)
- Setting SLOGCUSH high enough to avoid log—full processing but not too high to risk completely filling the log.
- Enable archiving (LOGMODE=A or L). SLOGCUSH has no effect if archiving is disabled.

Performance Indicator (SHOW LOG)

You can use the SHOW LOG operator command to determine if archiving has been enabled and what percentage of the log is full. Also, if archiving is enabled, it displays the percentage of the log remaining before ARCHPCT is reached. If archiving is disabled, it displays the percentage of the log remaining before SLOGCUSH is reached.

Communications

DRDA Performance Considerations (VM)

This section discusses how to use the PROTOCOL parameter and different block sizes in a DRDA protocol environment to obtain maximum performance from the database manager. For information on setting up this type of environment, refer to the *DB2 Server for VM System Administration*or the *DB2 Server for VSE System Administration* manuals.

PROTOCOL Performance Considerations

The PROTOCOL parameter specifies the types of protocols that the application server can process and the types of protocol under which the application requester runs.

On the application server, the PROTOCOL parameter is specified in the SQLSTART EXEC. The PROTOCOL parameter has two options on the application server, SQLDS and AUTO. When PROTOCOL=SQLDS is specified, the DB2 Server for VM application server allows access from DB2 Server for VM application requesters only. (The application requesters and application servers can be in either a local or remote environment.) This is the default option. When PROTOCOL=AUTO is specified, the DB2 Server for VM application server allows access from DB2 Server for VM application requesters and non-DB2 Server for VM application requesters.

On the application requester, the PROTOCOL parameter is specified in the SQLINIT EXEC. The PROTOCOL parameter has three options on the application requester, SQLDS, AUTO and DRDA. When PROTOCOL(SQLDS) is specified, the DB2 Server for VM application requester cannot connect to a non-DB2 Server for VM application server. This is the default option. When PROTOCOL(AUTO) or PROTOCOL(DRDA) is specified, the DB2 Server for VM application requester can connect to DB2 Server for VM application servers and non-DB2 Server for VM application servers.

When a connection is made between the application requester and the application server, the combination specified by these parameters determines the protocol to be used (either SQLDS protocol or DRDA protocol).

Table 5 shows the protocol used between the application requester and the application server.

Table 5. Protocol Used Between the application requester and the application server

Application Requester		Application Server		
		DB2 Server for VM (SQLSTART)		Non-DB2 Server for VM (including DB2 Server for VSE)
		SQLDS	AUTO	
DB2 Server for VM (SQLINIT)	SQLDS	SQLDS	SQLDS	Not Allowed
	AUTO	SQLDS	SQLDS	DRDA
	DRDA	Not Allowed	DRDA	DRDA
Non-DB2 Server for VM		Not Allowed	DRDA	Not Applicable

When the DB2 Server for VM application server is started with PROTOCOL=AUTO, DRDA “handshaking” occurs (**unless** the application requester is a DB2 Server for VM application requester that has been initialized with PROTOCOL(SQLDS)). Handshaking is an identification exchange between the application server and the application requester. During this handshaking sequence, information is exchanged between the application requester and the application server. This exchange includes CCSID information and generation of an LU 6.2 LUWID.

For more information on handshaking, see the discussion on accessing a remote relational database manager in the *Distributed Relational Database Architecture Reference* manual.

The PROTOCOL parameters used also affect CCSID conversion. If either the application requester or application server specifies SQLDS for the PROTOCOL parameter, the application requester default CCSIDs are ignored, and the application server CCSIDs are assumed.

Application requester CCSIDs are used when:

- Both the application server and the application requester specify the AUTO option, or
- The application server is started with PROTOCOL=AUTO option, and the application requester specifies PROTOCOL(DRDA) on the SQLINIT EXEC.

When communication is between a DB2 Server for VM application server and a DB2 Server for VM application requester, the AUTO option yields the same performance advantages as the SQLDS option except that it has a slight overhead when establishing a connection with the application server. Specifying the AUTO option on the DB2 Server for VM application server has many advantages. This option allows the application server to receive both SQLDS protocol or DRDA protocol, from both DB2 Server for VM application requesters and non-DB2 Server for VM application requesters. If you specify the AUTO option on the DB2 Server for VM application requester, it makes the necessary adjustments for both DB2 Server for VM application servers and non-DB2 Server for VM application servers.

When PROTOCOL(DRDA) is specified on the application requester, DRDA protocol is forced for connections, even if the target is a DB2 Server for VM application server. The DRDA option is useful when you are doing prototype testing between a DB2 Server for VM application requester and a DB2 Server for VM application server to model problems that may occur in communications with a non-DB2 Server for VM application server. You can also use the DRDA option to test SQL extensions only available in a DRDA protocol environment, for example, a larger block size.

Fetch and Insert Blocking

The database manager lets you use blocking for row insertion and row retrieval. Blocking improves performance in multiple user mode because data is sent between your program and the database manager in *blocks of rows* (rather than one row at a time). This reduces overhead from communications between the application server and the requester. Most applications that do multiple-row insertions or retrievals would benefit from blocking.

Implementing Blocking

To use blocking, specify the BLOCK parameter when preprocessing the program. (For extended dynamic statements, specify the BLOCK parameter on the CREATE PROGRAM statement.) When you run the program, blocking is automatically used for:

- Insert cursors (those that use OPEN, PUT, and CLOSE statements); and,
- Fetch cursors (those that use OPEN, FETCH, and CLOSE statements).

It is unnecessary for programs to explicitly handle the blocks because they are managed by the database manager. With the SQLDS protocol, 8KB blocks are used for both fetch and insert blocking. With the DRDA protocol, insert blocking is disabled and the rows are inserted one at a time; however, the block size for fetch blocking can be set by the application requester from 1K-byte to 32K-byte by using the QRYBLKSIZE option of the SQLINIT EXEC (see the *DB2 Server for VM Database Administration* or the *DB2 Server for VSE Database Administration* manuals).

For VSE Users

In VSE with the DRDA protocol, insert blocking is disabled and the rows are inserted one at a time; however, the block size for fetch blocking can be set by the application requester from a minimum value of 512 bytes to a maximum value of 32KB - 1 byte (32767).

Note: Blocking is only useful if the block size is sufficiently large that many rows can be blocked (that is, it must be greater than the maximum row length).

For retrievals, as many rows as the block will hold are sent to the application requester on the first fetch (with the DRDA protocol, the first block is sent with the OPEN statement). When the program fetches all the rows in the block, the next fetch that it issues causes another block to be sent. The program never needs to explicitly request a block.

For insertions, the blocks are also handled automatically. Whenever the program issues a PUT, a row is added to the block. When another row cannot fit into the block, the resource adapter sends the block to the database manager.

Suppressed Blocking

Single User Mode: The database manager does not do blocking for single user mode applications. Because both it and the application run in the same partition (for VSE) or machine (for VM) there is no cross-partition/machine communication overhead to be saved. Programs that have been preprocessed using the BLOCK parameter do *not* need to be re-preprocessed to run in single user mode. There is an automatic suppression of the blocking; no warning is sent to the program at run time. Some programs, however, process SQL statements dynamically at run time by using the PREPARE statement. These programs, when preprocessed with the BLOCK option, will receive a runtime warning if a dynamically processed statement is disqualified for blocking.

Multiple User Mode: In some instances, there is also suppressed blocking in multiple user mode. Suppressed blocking for a cursor occurs when:

- There is not enough virtual storage to get one block.
- Two rows cannot fit into one block.
- The cursor retrieves long fields (LONG VARCHAR or LONG VARGRAPHIC or VARCHAR(*n*) or VARGRAPHIC(*n*) where *n* is greater than 254, or 127, respectively).
- The cursor contains a FOR UPDATE clause.
- The cursor is operated on by a DELETE ... WHERE CURRENT OF CURSOR statement.
- The cursor is operated on by an UPDATE ... WHERE CURRENT OF CURSOR statement.

In all cases, a warning is sent to the program, in the SQLCA, to let it know that blocking was suppressed, and execution continues. Notice that the database manager suppresses blocking on a cursor level. It may be doing blocking for some cursors in a program even though the blocking for other cursors is suppressed.

The advantages of fetch and insert blocking are not limited to user programs. DB2 Server for VSE & VM facilities take advantage of blocking as well. The DBS utility and ISQL take advantage of blocking. Refer to the *DB2 Server for VM Database Administration*, the *DB2 Server for VSE Database Administration*, and the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manuals. (ISQL use of blocking is limited to fetch blocking.)

One minor performance disadvantage to using blocking is that the database manger uses extra virtual storage (equal to the block size) for every open cursor. The storage is freed when the user closes the cursor or when the user ends the logical unit of work (whichever comes first). This not only applies to user applications, but to the DBS utility. (ISQL only has one cursor open at a time because a user can only issue one SELECT statement at a time.)

Another minor performance disadvantage is that in not using a block worth of data, you pay the overhead of that block. For example, only 10 rows are returned in a block capable of holding 200 rows.

For more information on using fetch or put operations in programs with blocking, refer to the *DB2 Server for VSE & VM SQL Reference* manual, the *DB2 Server for*

VM Application Programming, and the *DB2 Server for VSE Application Programming* manuals.

Synchronous Communications (VM)

The SYNCHRONOUS parameter of SQLINIT EXEC determines whether synchronous or asynchronous communication is used between the user and database machines. Synchronous communication performs better than asynchronous communication but has the following restrictions:

- You cannot use SQLHX or CANCEL to cancel SQL statements. The only ways to terminate an unwanted LUW is to use the operator command FORCE, or to re-IPL CMS.
- You cannot use the SQLQRY command to query the status of the application that you are currently running on the DB2 Server for VSE & VM user machine.

We recommend that you use synchronous communication primarily when running a well-tested production batch application against local application servers. Always use the default, asynchronous communication, with interactive programs such as ISQL.

Considerations for ISQL and Adhoc Queries

Adhoc queries or the Interactive SQL facility (ISQL) can significantly affect how your database system performs.

AUTOCOMMIT

You should do all ISQL work in AUTOCOMMIT ON mode, the default. In AUTOCOMMIT ON mode, ISQL internally issues a COMMIT WORK, thus freeing DB2 Server for VSE & VM resources between query requests for possible use by others.

The COMMIT WORK is done immediately after a statement completes successfully. The only exception is for INSERT, DELETE, or UPDATE statements that change more than one row. For these statements, ISQL will give you a chance to rollback.

Use AUTOCOMMIT OFF mode, if you must have explicit control over committing work. For example, if two SQL statements update data in two tables simultaneously (as with debit and credit operations) and these updates must be synchronized to prevent inconsistent data, use AUTOCOMMIT OFF mode. If you are using this mode, package SQL statements into an ISQL routine so that terminal read delays are minimized or eliminated.

You can cancel any SQL statement if it is still in progress by issuing the ISQL CANCEL statement. (This statement causes a ROLLBACK WORK RELEASE to be executed.) That is, you can enter CANCEL if you are prompted to clear the screen, or prompted to enter CANCEL. You can also enter CANCEL to any ISQL message requesting a reply.

Isolation Levels

To minimize contention on shared resources, do all adhoc query work with the isolation level set to cursor stability (CS) unless the work being performed requires the repeatable read (RR) isolation level to ensure consistent data.

Alternatively, if it is not important that the data you are reading has necessarily been committed, consider setting the isolation level to uncommitted read (UR).

Adhoc users should be aware that when they are viewing a query result they may be delaying other users especially if querying the catalog tables.

Temporary Tables

If a long series of adhoc queries is expected against certain data in a large database, it may be best to copy that data into one or more temporary tables and query the copy. The queries will run faster and indexes can be created without being concerned with the effect of additional index maintenance on production work that is updating the data.

If a series of adhoc queries is expected against data in several related tables, consider creating a temporary table that contains the joined results of those tables. Queries run against this temporary table will run faster and be easier to formulate. An added benefit is that the temporary table can be created in a PRIVATE DBSPACE where locking overhead during query execution is negligible. The INSERT using subselect form of the SQL INSERT statement can be used to create the copy.

Views

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. In addition to the security benefits, this is an effective strategy for reducing the amount of processor and input/output usage that can be generated by indiscriminate querying of the data.

DBS Utility Considerations

Automatic Statistics Collection

Unless you specify otherwise, statistics are automatically collected and updated *during* execution of the RELOAD, and RELOAD DBSPACE commands. If you are performing a DATALOAD, statistics will also be automatically collected if you load data into a single empty table with no indexes. The database manager automatically issues an implicit UPDATE STATISTICS statement following the DATALOAD. This can be time-consuming, if the number of active data pages in that DBSPACE is large.

Note: This type of automatic collection only updates statistics for columns with indexes. For multicolumn indexes it only updates the leading column. If you want to ensure that all the columns have their statistics up to date, suppress automatic collections and enter UPDATE ALL STATISTICS immediately following a dataload. Refer to page 148.

Suppressing Automatic Statistics Collection

The automatic collection of statistics can be suppressed by specifying SET UPDATE STATISTICS OFF in the DBS input file before the DATALOAD. In cases where the database manager will not implicitly issue the UPDATE STATISTICS statement (but rather collects statistics during the load), there is no advantage in explicitly suppressing statistics collection. Otherwise, consider suppressing the UPDATE STATISTICS statement if either of the following conditions apply:

- There are many DATALOADs into the same table. UPDATE STATISTICS could be executed after the last one, or on a periodic basis.
- You know the statistics are not going to change significantly (for example, a small amount of data is being added to a large table). In such cases, you could postpone updating the statistics until more substantial changes have occurred.

In an IBM VM system

In the case of UNLOAD, a block size greater than 8244 bytes for tape output files is recommended for improved performance. Specify the block size in the CMS FILEDEF command associated with the OUTFILE statement.

Lock Escalation

When running the DBS utility in multiple user mode to load (INSERT) or unload (SELECT) rows from a database, you may encounter lock escalation. SQL LOCK DBSPACE or LOCK TABLE statements override the automatic locking mechanism; they can be used to avoid deadlock conditions.

A user-issued SQL LOCK statement is useful only during multiple user mode processing for table data in a public dbspace that is not defined with locking at the dbspace level. A user-acquired database lock remains in effect until the end of the logical unit of work in which it was issued.

DATALOAD and RELOAD Locking Considerations

If you insert many rows into the database with a RELOAD command or a DATALOAD command without the COMMITCOUNT option specified, consider using the SQL LOCK DBSPACE statement to eliminate or reduce lock escalation. An exclusive lock on the dbspace where the tables being loaded are defined does not appreciably increase lock contention and reduces the likelihood of deadlock with another user.

Note: An exclusive lock on a **table** being loaded does not prevent lock escalation and is not recommended.

You can also avoid lock escalation during multiple user mode DATALOAD processing by issuing a SET AUTOCOMMIT ON command before the DATALOAD command and specifying a sufficiently low COMMITCOUNT value in the DATALOAD INFILE subcommand. Use of DATALOAD COMMITCOUNT processing reduces the likelihood of the locking required by DATALOAD processing delaying other users accessing the table being loaded or other tables in the same dbspace where the table being defined resides. If the target table is in a dbspace defined with ROW level locking, a COMMITCOUNT value of approximately 200 should be sufficiently low. If the dbspace is defined with PAGE locking, the COMMITCOUNT value can be higher (1000, for example) and lock escalation is still avoided. Do not arbitrarily set the COMMITCOUNT value too low because frequent commit points increase DATALOAD run time.

SELECT, DATAUNLOAD, and UNLOAD Locking Considerations

If you are running with an isolation level setting of repeatable read (the default processing mode) and you know that a particular SELECT, DATAUNLOAD, or UNLOAD operation is going to access many rows from one or more tables in the database, lock escalation then normally occurs. You should consider acquiring a SHARE lock on the table(s) being accessed. If all the tables being accessed reside in the same dbspace, you should consider acquiring a SHARE lock on the dbspace being accessed. This action can reduce lock contention and the likelihood that a SELECT, DATAUNLOAD, or UNLOAD causes a deadlock with another user. Other users can modify other tables in the same dbspace where the table being accessed resides.

UNLOAD and RELOAD PACKAGE Considerations

To obtain the best performance when using the UNLOAD PACKAGE command and the RELOAD PACKAGE command, consider doing the following:

- Unload or reload large numbers of packages in your system's off-peak usage time or in single user mode.
- If you are unloading or reloading packages in multiple user mode, use blocking (by ensuring that the DBS utility was initialized with the BLOCK option).

These actions improve performance by preventing interruptions by other users.

PROGRAM is a synonym for PACKAGE. Therefore, UNLOAD or RELOAD PROGRAM, and UNLOAD or RELOAD PACKAGE are equivalent commands.

When unloading or reloading a modifiable package, an exclusive lock is held on the catalog table SYSACCESS. This may cause a performance deterioration for other users wanting to run the exclusively locked package.

Chapter 5. Improving Data Access Performance

User applications can access data without being dependent on how the data is stored or on the types of access paths available to locate the data. The optimizer determines an efficient access path to the data. This capability makes data more readily available for use by many diverse applications; however, you can experience a wide range of performance characteristics for the variety of possible application requests.

Ideally, the user of a relational database need not be concerned with how data is accessed. This is probably true for end users who write SQL queries quickly for one-time or occasional use. However, for those who plan transaction programs that may be executed thousands of times a day, some knowledge about the database manager and how it chooses among various access paths and evaluation sequences can enable them to significantly improve performance.

You can directly influence the access path to data in several ways (the first five are discussed in this chapter):

- Creating or dropping indexes
- Maintaining up-to-date statistical information on your database
- Changing the number of tables in a dbspace
- Updating the catalog statistics used to estimate access costs
- Rewriting a query in a more efficient form
- Reorganizing data, refer to "Reorganizing Data" on page 73
- Reorganizing indexes, refer to "Index Fragmentation" on page 76.

Access Paths and Indexes

To evaluate a query, the database manager determines an *access plan* that consists of a set of *access paths* (one for each table listed in the query) and other actions (for example, a sort). Five types of access paths are described here:

- Dbspace scans
- Nonselective index scans
- Selective index scans
- Index-only access scans
- Unique index with key matching predicate(s).

For each method, a model query is given that refers to a generic table T, with columns C1, C2, C3, and so on.

Notes:

1. The examples below use `SELECT *` because they are modeling arbitrary queries. For actual queries, the use of `SELECT *` is not recommended: all queries should only select the columns that are required in the answer set, in order to reduce the cost of processing the query, and to provide additional access path opportunities.
2. Considerations regarding the number of data pages read may not apply to tables with long fields.

Dbspace Scans

Assuming that T has no indexes, the model query is:

```
SELECT * FROM T WHERE C1 = 42
```

Because a page can contain rows from any table in a dbspace, the database manager must read every active data page in the dbspace to locate every row of T and to determine whether its value of C1 matches the given value. If there are other tables besides T in the dbspace, they will have to be read as well. If the fraction of the dbspace occupied by T is small, then most of the pages read will contain few or no rows from T.

It is a good idea to make a dbspace scan as inexpensive as possible. This can be accomplished by having one table in a dbspace and reorganizing its rows so that there are none that overflow from their original page onto another page. For information, see "Reorganizing a Single Table" on page 74. Another factor is the amount of free space left on each page. For information, see the *DB2 Server for VM Database Administration* or the *DB2 Server for VSE Database Administration* manuals.

Index Scans

For an index scan, the model query is:

```
SELECT * FROM T WHERE C2 = 42
```

An index scan improves performance by enabling the database manager to avoid the following:

- Reading all of the active data pages in the dbspace.
- Reading data pages that do not contain desired rows.
- Sorting the result.

An index scan performs better than a dbspace scan in many situations. However, it has the following drawbacks:

- If VARCHAR or VARGRAPHIC columns are selected, or if not all columns referenced are in the index, then the index scan must read the index pages as well as the data pages. (A dbspace scan reads only data pages.)
- If the index is not clustered (even if the index is a clustering index), some data pages may be read more than once. Refer to "Clustered Indexes" on page 70.

There are two types of index scans: nonselective and selective. An index scan on T is *selective* if C2 is the first column of the index key. All other index scans are *nonselective*.

Nonselective Index Scans

If T has an index on C1, the database manager can use the index to pick out only those pages that contain rows of table T.

Be aware that if T is the only table in its dbspace, this method may be no better than a dbspace scan. It is only more efficient in those cases where only a portion of the pages in the dbspace contain rows from table T, the result needs to be sorted on the index key, or index sargs can be applied to the index keys.

Selective Index Scans

If T has an index on C2, the database manager will be able to use the index to pick out only those rows from table T where C2 = 42. That is, the only pages that will be accessed are index leaf pages that contain keys where C2 = 42, and non-leaf pages that must be traversed to navigate to these leaf pages.

A selective index scan is generally the most efficient access path. This is true even in the case where T is the only table in the dbspace, if only a portion of the data pages contain rows where C2 = 42. (If all or nearly all pages contained rows where C2 = 42, then a dbspace scan would likely be more efficient).

Index-Only Access Scans

Although in general the database manager has to read data pages for a table to evaluate a query, there are cases where all the columns referenced are present in the index and the predicates do not require the data page. If these conditions are met, then only index pages will be read. This is called index-only access, and is possible for both selective and nonselective index scans. The model query is:

```
SELECT C2, C3 FROM T WHERE C2 = 50
```

(It is assumed here that an index exists on columns C2 and C3.)

Clustering Index

There is no advantage to using a clustered index with index-only access, because clustered indexes are only valuable when the database manager uses an index to access data pages.

Examples of Index only Access

The following are examples of queries that use index-only access. It is assumed that a multicolumn index exists on columns C1, C2, C3, and C4.

```
SELECT COUNT(*) FROM T WHERE C2 = 5
```

The database manager scans the entire index looking for C2=5, but no data pages are read.

```
SELECT C2 FROM T
```

The database manager scans the entire index, but no data pages are read.

```
SELECT MIN(C1) FROM T
```

The database manager does not read the entire index; just a single value.

```
SELECT MAX(C1) FROM T
```

For MAX column functions, C1 must be defined as NOT NULL so that a single value is read rather than the entire index.

```
SELECT C1 FROM T WHERE C1 = 42 AND C4 = 100
```

The database manager reads only the index entries where C1 = 42 and then scans for C4=100, but no data pages are read.

Index-only access is not possible when a VARCHAR or VARGRAPHIC column appears in the SELECT list or in a residual predicate.

Creating Indexes

In some cases, it may be reasonable to create an index that includes data just to improve the performance of certain common queries. For example, the sample ACTIVITY table identifies each activity by an activity number (ACTNO). It also contains a 6-character activity keyword (ACTKWD). If the table were often used to decode activity numbers by retrieving the corresponding keywords, it might be useful to have an index on both columns. The model query would be:

```
SELECT ACTKWD FROM ACTIVITY WHERE ACTNO = 42
```

Another case where index-only access is beneficial is a table with very long rows, where the portion of the row retrieved is small compared to the size of the row. If a query needs only three or four relatively short columns of that data, an index on those columns might be worthwhile merely to avoid the cost of scanning all data pages and extracting the useful data.

Unique Index with Key Matching Predicate(s)

The model query is:

```
SELECT * FROM T WHERE C1 = 42
```

Here, access is most direct if there is a unique index on column C1. In this case, the database manager reads only as much of the index as needed to locate one entry, and then at most one data page. Furthermore, instead of using a scan, it uses a more efficient operation to return a single row. Refer to “Key-matching Predicates” on page 132.

Indexes for Sorting

The primary use of indexes is to provide selective access to data, but they are also used to sort data in a specified way. Consider this query:

```
SELECT * FROM EMPLOYEE
  WHERE WORKDEPT LIKE 'A%'
    ORDER BY EMPNO
```

The database manager can access the rows needed through an index on WORKDEPT, but then it would have to sort all of those rows by EMPNO. It might estimate that it would be more efficient to access all rows in order by an index on EMPNO, then check the value of WORKDEPT in each one, but eliminate the sort.

The database manager can use indexes for ORDER BY and GROUP BY, but not always for SELECT DISTINCT. It can avoid a sort for SELECT DISTINCT if a unique index is used, or if there is a GROUP BY list that is a subset of the SELECT list.

Note: If an application program contains a SELECT DISTINCT statement that is preprocessed using a unique index, the preprocessor records that the package has a dependency on the unique index. If the unique index ever becomes invalid, the entire package will be invalidated and it will be dynamically repreprocessed the next time it is executed.

Recommendations for Indexes

The nature and purpose of your data will determine what indexes you should create, but the following very general guidelines may be of some help:

- If you delete rows from or update the primary key on parent tables, define indexes on foreign keys.
- Define primary keys or unique constraints wherever they apply. The database manager automatically defines unique indexes for these.
- Use indexes to speed up the most frequent queries to tables with more than 15 data pages; and for tables with more than 10 pages that are primarily accessed for reading only.
- Create indexes on fixed-length rather than varying length columns.
- Create indexes to include columns frequently queried to allow for index-only access.
- Whenever possible, create unique indexes. If you cannot create a unique index on a single column, create a unique multicolumn index. For example if you want to sort the EMPLOYEE table by the JOB column, create an index on JOB and EMPNO (JOB,EMPNO). The index will still sort by JOB, but it can also be a UNIQUE index.
- In a multicolumn index, place the “most” unique column first.

Disadvantages of Indexes

The above descriptions of the various types of access paths should suggest to you that indexes can reduce access time significantly. But before you begin creating them, carefully consider their costs:

- They require storage space.
- It takes time to create and maintain them.
- There is overhead associated with keeping them synchronized with the tables they index. It takes more time for the database manager to update a table that uses an index or insert new data into it.
- They may increase locking contention.
- They increase the time required for recovery.

Placing Tables into Dbspaces

Each large table should be placed in its own dbspace, so that rows from other tables do not have to be examined during a dbspace scan. Another advantage is that if you later wish to eliminate that table, you can do so with a DROP DBSPACE statement, which will run very fast because the data, index, and header pages do not have to be examined.

Very small tables may be grouped together in the same dbspace, because relatively few additional pages have to be read during a dbspace scan. However, avoid page level locking in this situation.

Dbspace scans are done during CREATE INDEX, DROP TABLE, and UPDATE STATISTICS processing, and may be used to satisfy other SQL requests, depending on index availability.

By default, locking takes place at the page-level. This is usually the best trade-off between concurrency and locking overhead. You should consider locking at the row-level when many applications access one small part of the database. The tables there could be put in their own dbspaces, for which you would request row-level locking (using an ACQUIRE DBSPACE or ALTER DBSPACE statement).

When you request row-level locking for a dbspace, key-level locking is also done for indexes in that dbspace. Key-level locking on indexes reduces contention, but increases overhead.

Organizing Referential Structures

Because referential operations (update of a primary or foreign key, deletion of a parent row, or insertion of a foreign key) involve access to more than one table, when organizing a referential structure you should carefully consider the implications of concurrency. (The issues discussed here are equally applicable to any set of related tables.)

Referential operations require access to multiple tables, and possibly to multiple rows of dependent tables. This characteristic increases the possibility of deadlock situations. When the primary key of a parent table is modified (DELETE, UPDATE), all dependent rows are accessed (and possibly also modified). Conversely, when a foreign key in a dependent table is modified (INSERT, UPDATE), the parent table is accessed. If two such operations run concurrently in different logical units of work (LUWs), a deadlock situation could result, which would trigger the automatic rollback of the later LUW.

Note: A similar potential deadlock situation would be encountered whenever logically related data is concurrently accessed in opposing ways.

Similarly, when multiple users access referential structures, lock contention increases (because the users are accessing the same tables, and the number of rows accessed can be quite large). This contention may reduce concurrency.

A user who understands the nature of referential operations can minimize their effect on concurrency, by reducing the chances of multiple users performing logically unrelated operations contending for locks. (Contention cannot be avoided if users are performing logically *conflicting* operations.) Consider the following ways to improve concurrency:

- Do not put tables from different referential structures in the same dbspace. In general always try to place only one table in each dbspace.
- Create an index on a foreign key. (Whenever possible this should be a unique index that exactly matches the columns of the foreign key. If necessary make the index unique by creating a multicolumn index.) This provides the most selective access path possible to dependent rows whose foreign key has a particular value. Operations on a parent row can use this index to scan the dependent table, thus avoiding the need for a dbspace scan or a nonselective index scan.

  You should use discretion in creating such indexes, especially on tables with several foreign keys. For example, you may consider creating them only on the most often referenced foreign keys. These indexes will be of particular use in reducing deadlock situations in environments where parent tables and dependent tables are being modified concurrently, and will also provide faster execution of all referential operations against parent rows.

Predicate Processing

Search conditions contain predicates joined with AND, OR, and NOT. A predicate is a search condition in a WHERE or HAVING clause of an SQL statement. Examples include `C1 = 10`, `C2 BETWEEN 10 AND 20`, `EXISTS(subquery)`, and `C4 NOT LIKE 'A%'`. Only those rows that satisfy a predicate are returned.

Predicates are resolved in one of two categories: residual or sargable. (Sargable is a term derived from the words "search argument.") Sargable predicates are applied at the Database Storage Subsystem (DBSS) level; residual predicates are applied at the Relational Data System (RDS) level.

Figure 16 shows the hierarchy of predicates.

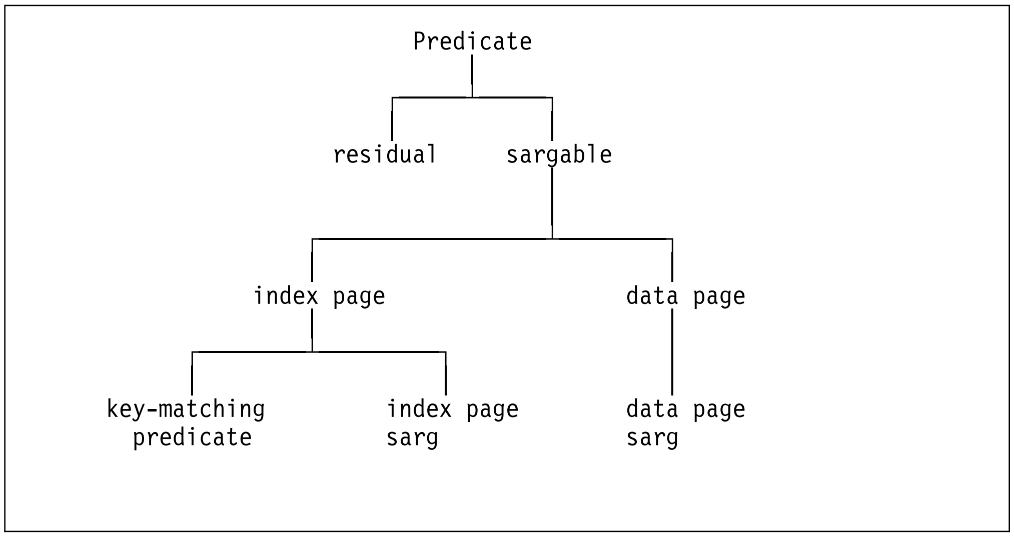


Figure 16. Predicate Hierarchy

Sargable predicates are further divided into two categories: those that use the index, and those that do not. The former are called either key-matching predicates or index page sargs. The latter are called data page sargs.

A key-matching predicate, which is applied directly to the index key, is created when the columns referenced in the predicate form an initial substring of an index on the table.

An index page sarg is resolved using the index page, but is not used to search the index key. It is created when the columns referenced in the predicate are *not* an initial substring of an index, but are contained in the index.

A data page sarg does not use the index, and always requires the data pages be read. It is created when the columns referenced in the predicate are not contained in the index.

Column Attributes

The next sections deal with predicates in the WHERE clause. For these predicates, it is important that the data types and CCSIDs of any columns and literals match whenever possible. That is, numeric values should use the same representation, including the same precision and scale for DECIMAL values. Character and graphic values should have the same length. Columns and literals should use the same CCSID, refer to “Impact of CCSIDs on Sargability” on page 141. Adhering to this rule will always give the database manager the greatest flexibility in choosing an efficient access path. All the examples assume that this rule has been followed. For more information, refer to note number one on page 1 on page 137.

Key-matching Predicates

Before a predicate can be considered key-matching it must be in the correct form and a suitable index must be available.

Form of Key-matching Predicates

Some types of predicates can match index entries; other types cannot. For example, if the EMPLOYEE table has an index on the column SEX, it matches the predicate in this query:

```
SELECT * FROM EMPLOYEE WHERE SEX = 'M'
```

On the other hand, the same index does *not* match the predicate in this query:

```
SELECT * FROM EMPLOYEE WHERE SEX < > 'F'
```

We call a predicate *key-matching* if it can match the entries in a suitable index. Table 6 on page 136 shows which predicate types are key-matching.

If a predicate fails to match the index, it may still be applied to the index, but not used to search it.

Only one predicate per column can be chosen as the key, however, other predicates on that column are eligible to be a sargable predicate. For predicates that are joined with AND, one per column is chosen as the key. The one with the best filter factor establishes the path and the other is turned into a sargable predicate. For example, consider a table with three columns C1, C2, and C3. A multi-column index is created (C3, C2, C1) and the following WHERE clause is used in a SELECT statement:

```
...WHERE C1>1 AND
       C1<2 AND
       C2=2 AND
       C3=3
```

Only the first, third, and fourth predicates are chosen as key-matching predicates (C1>1, C2=2, C3=3). The second predicate (C1<2) is not chosen as key-matching but it is sargable.

For maximum efficiency, use key-matching predicates and create suitable indexes. The database manager may not always use an index to apply a key-matching predicate—other factors may intervene. But the first step in reducing the processing cost of a query is to use key-matching predicates where possible and then create suitable indexes.

In general, when you create an multi-column index, put the column with the most distinct values first, and continue in order to the least distinct values.

One exception to this rule is the case where the index provides a necessary ordering of the data. With this query:

```
SELECT * FROM EMPLOYEE
  WHERE EDLEVEL > 14 AND JOB = 'CLERK'
    ORDER BY EDLEVEL
```

an index on EDLEVEL,JOB enables the database manager to access data in the order required by the ORDER BY clause, thus saving a sort at the end. This may be enough to justify scanning index entries for rows that are rejected.

Suitable Index for Key-matching Predicate

For a simple predicate, an index is fully matched if the column in the predicate is the first column of the index. For example, the predicate C1=10 matches an index on columns C1, C2, C3, as well as an index on column C1 alone. If there are additional predicates on columns C2 and C3, they may also be evaluated through the multicolumn index.

For a search condition where all the predicates are joined by an AND, it is enough if the index includes the set of columns as an initial substring. For example, an index on columns C1, C3, C4, and C6 is fully matched by the search condition C1=10 AND C4='A' AND C3=7 AND C6=9, as long as all but the last column are matched with equality predicates. The last predicate can be either an equality or a range predicate.

The same index is *not* fully matched by the search condition C1=10 AND C4='A' AND C6=9, because the set of columns in that search condition (C1,C4,C6) is not an initial substring of C1, C3, C4, C6. However, the database manager can use the index for the parts of the search condition that do form an initial substring; in the example, it can apply the predicate C1=10 through the index. In addition, it can still use the index to evaluate the predicates on C4 and C6, so that data pages do not need to be accessed.

Similarly, such an index is *not* fully matched by the search condition C1=10 AND C3=7 AND C4>'A' AND C6=9, because the predicate C4>'A' is not an equality predicate. An index can only be matched up to and including the first non-equality predicate. Thus, the database manager can apply the predicates C1=10 AND C3=7 AND C4>'A' as key-matching predicates to the index. Again, the predicate on C6 can be evaluated as an index sarg so that data pages do not need to be accessed.

Hence, the order of the index columns is important; it should take into account the kinds of queries used. For example, suppose the Spiffy Computer department intends to query its employee table regularly with predicates such as EDLEVEL > 14 AND JOB = 'CLERK'. With an index on EDLEVEL,JOB, the database manager finds the first index entry with EDLEVEL greater than 14 and scans the remainder of the index from there upward. But with an index on JOB,EDLEVEL, it scans only the entries for clerks having EDLEVEL > 14, giving a shorter access path.

Note: If you created an index in order to improve the performance of an SQL statement, you should probably check that the database manager actually uses the index for that statement. To find out what access and processing methods it has chosen, use the EXPLAIN statement (see “Using Explanation Tables to Evaluate Performance” on page 151).

Sargable and Residual Predicates

Rows that are retrieved go through two stages of processing. Predicates can be applied at the first stage are called *sargable predicates*; those that cannot be applied until the second stage are called *residual predicates*. Predicates in the HAVING clause are always residual. Resolution of predicates and the predicate hierarchy are detailed in “Predicate Processing” on page 131. Table 6 on page 136 shows which predicates are sargable and which are not.

There is a definite performance advantage in using sargable predicates: they require fewer CPU instructions than do residual predicates, because they eliminate rows that would otherwise be passed from first to second stage processing. Thus, whenever possible, avoid a residual predicate by rewriting your SQL statement.

Example

Table T contains 1000 rows, and column C6 contains the integers from 1 to 1000. Consider this query:

```
SELECT * FROM T WHERE INTEGER(C6/7) = 2
```

Because the column in the predicate is involved in an arithmetic expression, the predicate is residual. The first stage must access 1000 rows and pass them all back to the second stage. If you instead write the predicate as WHERE C6 BETWEEN 14 AND 20, then only seven rows are passed back to the second stage. Furthermore, the predicate C6 BETWEEN 14 AND 20 is key-matching. If there is an index on C6, the first stage need only access seven rows.

Sargable predicates are better than residual predicates, but a suitable index is better still. Avoiding the processing cost of a residual predicate won’t help you much if you have to access ten million rows without an index. (You can use EXPLAIN to tell whether a sargable predicate exists for a particular column, refer to the REFERENCE EXPLAIN table in the *DB2 Server for VSE & VM SQL Reference* manual.)

Join Predicates

In general, any predicate involving more than one table is a join predicate. In the database manager, a condition of the form T1.C1=T2.C2 (the equijoin) is handled specially by the optimizer. For information on joins, see “Methods of Joining Two or More Tables” on page 143.

Search Conditions and their Processing Characteristics

Table 6 shows the different types of search conditions, and their processing characteristics. The following conventions are used:

- A search condition consists of one or more predicates
- Predicates are combined using the logical operators AND/OR
- NOT can be applied to either predicates or search conditions
- *Expression* is any expression involving arithmetic operators, concatenation, scalar functions, or column functions
- *Value* is a literal or host variable
- *Litexpr* is any *value* or *expression*
- *Anyexpr* is any column, *value* or *expression*
- *Char* is any character string that does not begin with the ‘%’ or ‘_’ special characters
- *pattern* is any character string that begins with the `'%'` or ‘_’ special characters

- *Op* is one of the operators: <, <=, >, >=, =, <>, ¬=
- *Rop* is one of the range operators: <, <=, >, >=
- *Q* is one of the quantifiers: ANY, ALL, SOME
- <> represents <> or ¬=
- [ ] indicates parts of the predicate that are optional.

Note: If the predicate falls in two different categories, choose the more specific category.

Table 6. Search Conditions and their Processing Characteristics

Search Conditions	**Key-Matching?**	**Sargable?**	**Default Filter Factor (FF)**	**Notes**
COL = *value*	Yes	Yes	1/25	1,16
COL IS NULL	Yes	Yes	1/25	2
COL *rop value*	Yes	Yes	1/3	1,16,19
COL BETWEEN *value1* AND *value2*	Yes	Yes	1/10	1,4,16,19
COL LIKE '*char*'	Yes	Yes	1/10	5,19
COL IN (*value1,...*)	Yes	Yes	1/25*size	1,3,6,16,17
COL <> *value*	No	Yes	24/25	1,7,16
COL IS NOT NULL	No	Yes	24/25	2
COL NOT BETWEEN *value1 AND value2*	No	No	9/10	7,8,19
COL NOT IN (*value1,...*)	No	Yes	1 - 1/25*size	1,7,9,16,17
COL NOT LIKE '*value*'	No	No	9/10	19
COL LIKE '*pattern*'	No	No	1/10	19
COL LIKE *host variable*	No	No	1/10	16,19
T1.COL = T2.COL (different tables)	Yes	Yes	1/25	1,15
T1.COL *rop* T2.COL (different tables)	Yes	Yes	1/3	1,15,19
T1.COL <> T2.COL (different tables)	No	Yes	24/25	1,15
T1.COL1 = T1.COL2 (same table)	No	No	1/25	
T1.COL1 *rop* T1.COL2 (same table)	No	No	1/3	19
T1.COL1 <> T1.COL2 (same table)	No	No	24/25	
COL = [Q] (uncorrelated scalar subquery)	Yes	Yes	1/25	1,10
COL *rop* [Q] (uncorrelated scalar subquery)	Yes	Yes	1/3	1,10,19
COL <> [Q] (uncorrelated scalar subquery)	No	Yes	24/25	1,10
litexpr = [Q] (uncorrelated scalar subquery)	No	No	1	10
litexpr *rop* [Q] (uncorrelated scalar subquery)	No	No	1/3	10,19
litexpr <> [Q] (uncorrelated scalar subquery)	No	No	1	10
COL = (subquery)	No	No	1/25	11
COL <> (subquery)	No	No	24/25	11
COL *rop* (subquery)	No	No	1/3	11,19
litexpr = (subquery)	No	No	1	11
litexpr <> (subquery)	No	No	1	11
anyexpr *op* Q (subquery)	No	No	1	11
anyexpr [NOT] IN (subquery)	No	No	1	11
[NOT] EXISTS (subquery)	No	No	1	11
COL = *expression*	No	No	1/25	12
COL <> *expression*	No	No	24/25	12
COL *rop expression*	No	No	1/3	12,19
anyexpr NOT BETWEEN *anyexpr* AND *anyexpr*	No	No	9/10	12,18,19
anyexpr BETWEEN *anyexpr* AND *anyexpr*	No	No	1/10	12,18,19
anyexpr <> *expression*	No	No	24/25	12
litexpr rop anyexpr	No	No	1/3	12
litexpr = anyexpr	No	No	1	12
litexpr <> *anyexpr*	No	No	~0	12
litexpr NOT IN (*value,...*)	No	No	~0	12,17
litexpr IN (*value,...*)	No	No	1	12,17
search condition AND *search condition*	Yes	Yes	FF1*FF2	13
search condition OR *search condition*	No	Yes	FF1+FF2-FF1*FF2	14
NOT *search condition*	No	No	1-FF	

Notes to Table 6: Search conditions which are listed as key-matching or sargable are only potentially so; they may not be treated as such because of the following factors:

1. The *value* must be of the same or compatible type as the column. Adhere to this rule whenever possible. Numeric data types have the following hierarchy:

   ```
   SMALLINT < INTEGER < DECIMAL < FLOAT
   ```

   A value's data type can be converted to any higher data type. For example, INTEGER can be converted to a DECIMAL (given sufficient precision) or FLOAT, but not to SMALLINT. Similar compatibility considerations exist for character and graphic data lengths, as well as for the precision and scale of decimal data.

   If the data type of the column is CHAR(n) or GRAPHIC(n), the predicate that references it is sargable if the length of that predicate value is less than or equal to "n." The column and the predicate should also have the same CCSID, refer to "Impact of CCSIDs on Sargability" on page 141.

   If the data type of the column is DECIMAL(m,n), it must be possible to accommodate the number of decimal digits before and after the decimal point in the target decimal field (P1-S1 <= P2-S2 and S1<=S2 where P is precision and S is scale).

   Even-precision DECIMAL variables are supported by the DB2 Server for VSE & VM product for the assembler preprocessor. You can use even-precision DECIMAL columns in tables that are referenced by Assembler programs. Host variables in even-precision will be left as is by the preprocessor. Therefore, when these programs access even-precision DECIMAL data, predicates become sargable instead of RESIDUAL. Your performance may improve when using SQL statements that use even-precision packed DECIMAL columns.

2. The NULL predicate must be applied to a column *without* the NOT NULL attribute in order to be key-matching or sargable. Otherwise, the predicate is residual.

3. If a multicolumn index exists, at most one IN predicate can be used to match columns of the index. For example, if a table T1(C1, C2, C3) and an index C1, C2, C3 exist, the following query will have only one key-matching predicate, not three:

   ```
   SELECT * from T1 where C1 IN (:HV1, :HV2) AND C2 IN (:HV3, :HV4) AND C3=5
   ```

4. If *value1* and *value2* are equal, then for filter factor calculation purposes the predicate is treated as though it were the equality predicate COL = *value1*.

5. Although the LIKE predicate is a residual predicate, the database manager takes advantage of the character argument to generate a BETWEEN predicate which is both key-matching and sargable. This BETWEEN predicate is then applied by the first stage either as a key-matching or a sargable predicate. This transformation does not apply if the pattern is a host variable or the ESCAPE clause exists. It also does not apply if the character data is mixed data.

6. If there are no host variables in the list, then a BETWEEN predicate will be generated using the lowest and highest values. This predicate is sargable, and can be used to reduce the number of rows returned to the second stage.

7. Whenever possible, avoid negating a predicate using NOT. Instead, use an equivalent form that distributes the negation. In some cases, the database

manager will perform this transformation for you. For example, the predicate NOT COL = *value* is treated like COL ¬= *value*.

8. This predicate, although residual as stated, can be rewritten to eliminate the NOT BETWEEN into COL < *value1* OR COL > *value2*, which is sargable. (See note 14.)
9. Because this predicate is residual when more than one value is used, it might be beneficial to rewrite it as COL ¬= *value1* AND COL ¬= *value2* AND ... which is sargable.
10. An uncorrelated scalar subquery can return at most one value, and can be evaluated before the query that contains it. This returned value is then used to replace the subquery. The predicate is scalar only if the subquery statement specifies a COLUMN function and the subquery does not contain a GROUP BY clause, or if the predicate containing the subquery is not quantified.
11. Predicates that reference correlated subqueries or subqueries that can return more than one row are always residual.
12. An expression makes any predicate residual. Sometimes a query can be rewritten to avoid the presence of an expression. For example, instead of “SALARY+200 = 20000,” write “SALARY = 198000.” The second form is executed more efficiently.
13. For this kind of search condition to be key-matching, all predicates must refer to columns that form an initial substring of the index columns. All but the last column must be matched with equality predicates; the last predicate can be either an equality or a range predicate.

    In search conditions containing multiple predicates on the same column, only one predicate can be chosen as the key. The predicate providing the best filtering establishes the path, and the others are turned into sargable predicates.
14. All predicates in a search condition that contains an OR remain sargable only if all the individual predicates are sargable; otherwise, they are all treated as residual. In other words, a single residual OR will cause all the predicates in a search condition to be residual.

    If all the predicates refer to the same column and the column is indexed, the search condition can be rewritten using the IN predicate.

    For example, instead of:

    ```
    SELECT * FROM EMP_ACT
      WHERE ACTNO=90 OR ACTNO=100
    ```

    write:

    ```
    SELECT * FROM EMP_ACT
      WHERE ACTNO IN (90,100)
    ```

    If different columns are referenced and the columns are indexed, then a UNION may be a more efficient form of the query.

    In the following example, the database manager will have to examine all rows in the EMPLOYEE table to find those that satisfy the two predicates:

    ```
    SELECT * FROM EMPLOYEE
      WHERE JOB = 'CLERK'
      OR LASTNAME = 'JONES'
    ```

The same request can be processed more efficiently if it is reformulated as the UNION of two SELECT statements:

```
SELECT * FROM EMPLOYEE WHERE JOB = 'CLERK'
UNION
SELECT * FROM EMPLOYEE WHERE LASTNAME = 'JONES'
```

15. A join is accomplished by first accessing the outer table and looking for rows that satisfy all predicates on that table only. For each such row, the inner table is then accessed to find all rows there that match that row's join column value. Because a specific value is being used, a join predicate of "colname = colname" becomes "colname = value". This is why join predicates can give selective access to a table, if the table is the inner table.

    A join predicate is only sargable if the data types of the two columns are identical (disregarding whether the columns support NULLS). If the data types are CHAR(n), VARCHAR(n), GRAPHIC(n), and VARGRAPHIC(n), the lengths must match. If they are DECIMAL(m,n), precision and scale must both match.

    The database manager path selection takes this into account when it decides which table should be accessed first.

16. Predicates using indicator variables are sargable only if they meet the following criteria:

    - The predicate is of the form COL = :HV1:IND1 or of the form COL = ?
    - COL is a nullable column

    Predicates using indicator variables which do not meet the above criteria are always residual.

17. When the IN predicate contains only one value in the list, it is converted to an EQUAL predicate.

18. All BETWEEN and NOT BETWEEN predicates that do not have a column as the first argument and values as the second and third arguments, are residual.

19. For the following cases the default filter factor is determined from the COLCOUNT value in the SYSTEM.SYSCOLUMNS catalog table (refer to "SYSTEM.SYSCOLUMNS" on page 39):

    - COL rop *host variable*
    - COL rop COL
    - COL like *predicate*
    - COL BETWEEN *anyexpr* AND *anyexpr*

Table 7. Filter Factors

COLCOUNT Value	Filter Factor for *rop*	Filter Factor for LIKE, BETWEEN
>=100,000,000	1/10000	3/100000
>=10,000,000	1/3000	1/10000
>=1,000,000	1/1000	3/10,000
>=100,000	1/300	1/1000
>=10,000	1/100	3/1000
>=1000	1/30	1/100
>=100	1/10	3/100
<100	1/3	1/10
=-1	1/3	1/10

Filter Factors

The objective of a predicate is to return to the user only those rows satisfying a particular search condition. Every predicate is treated like a filter that reduces the number of rows returned. The degree to which the predicate reduces the size of the answer set is the *filter factor* (FF). The filter factor is an estimate of the proportion of rows that remain after a predicate has “filtered out” the rows that do not satisfy it.

The filter factor is a value between 0 and 1. If it is 1, the whole table is selected, and the predicate has no filtering effect; if it is 0, no rows are returned.

The database manager estimates a filter factor for every predicate. If the predicate is either too complex (contains an expression), uses disjunction (OR), uses host variables, or if there are no statistics available for the columns it references, then a default filter factor is used. The defaults for various predicates are shown in Table 6 on page 136.

Sometimes the optimizer will not use the index when a predicate uses host variables. This is because the optimizer is forced to make assumptions about the values that are not available when the statement is being preprocessed. In such cases, you can usually improve performance by executing the SQL statement dynamically with fixed values.

Examples of Predicate Processing

The following examples of predicates illustrate the general rules shown in Table 6 on page 136. In each case, assume that there is an index on columns C1, C2, C3, C4 of the table.

WHERE C1 = 5 AND C2 = 7
: Both predicates are sargable, and both can be applied as key-matching predicates to the index.

WHERE C1 = 5 AND C2 > 7
: Both predicates are sargable, and both can be applied as key-matching predicates to the index.

WHERE C1 > 5 AND C2 = 7
: Both predicates are sargable, but only the first can be applied as a key-matching predicate. Because all the predicates reference columns in the index, the second predicate will be applied as an index page predicate.

WHERE C1 > 5 OR C2 = 7
: Both predicates are sargable, and the combination is sargable. The OR prevents the use of key-matching predicates. The index can *not* be used for a selective index scan. However, both predicates will be applied as index page predicates.

WHERE C1 IN (*subquery*) AND C2 = C1
: Both predicates are residual. The index is not considered for a selective index scan, and both predicates are evaluated residually.

WHERE C1 = 5 AND C2 = 7 AND C3+5 = 7
: Only the first two predicates are sargable and can be applied as key-matching predicates. The third predicate is residual. The index is considered for selective access. All rows satisfying those two predicates are passed to residual processing to evaluate the third predicate.

WHERE C1 = 5 OR C2 = 7 OR C3+5 = 7
: The third predicate is residual; hence, the combination is residual. All three predicates are evaluated residually.

WHERE C1 = 5 OR (C2 = 7 AND C3 = C4)
: The third predicate is residual, so, the combination of the second and third predicates (in parentheses) is also residual. Hence, the total combination is residual. All predicates are evaluated residually.

WHERE (C1 > 5 OR C2 = 7) AND C3 = C4
: The combination of the first two predicates is sargable, but the OR prevents the use of key-matching predicates. The third predicate is residual. The index is not considered for a selective index scan, but the combined predicate (in parentheses) is sargable and will be applied as index page predicates. All rows satisfying those two predicates are passed to residual processing to evaluate the third predicate.

WHERE C1 > 5 AND C2 = 7 AND C5 = 8
: All predicates are sargable, but only the first can be applied as a key-matching predicate. Because the remaining predicates reference columns in both the index and data pages, the remaining two predicates are applied as data page sargs.

Impact of CCSIDs on Sargability

The *DB2 Server for VSE & VM SQL Reference* manual lists the rules used to decide which operand will undergo Coded Character Set Identifier (CCSID) conversion in a comparison operation. These rules will help you to maintain sargability whenever possible. The rules were defined to ensure that a column operand will only undergo CCSID conversion if it is absolutely necessary.

Whenever a column operand is chosen to undergo CCSID conversion, the predicate becomes residual because CCSID conversion is performed from RDS, and not from DBSS. In most cases, this is a necessary consequence of using the CCSID support. In other cases, it can be avoided by understanding how the rules apply and by changing the application or the data.

Consider the search condition: COL = *value* (where *value* is a host variable). This search condition is normally sargable (see Table 6 on page 136). The database manager attempts to keep this search condition sargable by always performing CCSID conversion on the host variable operand. There is a case, however, when the rules state that the column operand is the one that should be converted: when the subtype of the column is SBCS and the subtype of the host variable CCSID is mixed.

If it is possible that the host variable will contain mixed data, then the column operand must undergo CCSID conversion and the predicate must become residual. One way to make this predicate sargable is to set the column subtype to mixed. This may not always be possible or desirable, but you should consider this situation when setting the subtype of new columns.

If the host variable will never contain mixed data, then it is possible to make this predicate sargable by changing the subtype of the host variable from mixed to SBCS. To do this, change the default CCSID values on the application requester. Refer to the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manuals for information on how to change the CHARNAME setting for an application requester. This situation may not be possible or desirable, especially if graphic or mixed data is used elsewhere in the query.

Tuning Queries with Several Tables

The process of combining rows of one table with rows of another is called a *join*. It is often possible to write a query against two or more tables either as a join or as one or more nested SELECT clauses. The first method is usually more efficient, as this gives the optimizer more choices during access path selection.

The following query retrieves data about all designers in departments that are responsible for projects that are part of a major project MA2100. It will be used here to illustrate different access methods in detail.

```
FROM   EMPLOYEE E, PROJECT P
WHERE  E.JOB = 'DESIGNER'
  AND  E.WORKDEPT = P.DEPTNO
  AND  P.MAJPROJ = 'MA2100'
```

In this example, the following assumptions are made:

- Each table is in its own dbspace
- Table EMPLOYEE
  - Has 10 000 rows on 500 pages
  - Has an index on EMPNO with 25 pages
  - Has an index on WORKDEPT with 10 pages
- Table PROJECT
  - Has 3000 rows on 60 pages
  - Has an index on PROJNO with 8 pages
  - Has an index on RESPEMP with 8 pages
- COLCOUNT for MAJPROJ is 100.

- COLCOUNT for WORKDEPT is 1000; for JOB, 50

Methods of Joining Two or More Tables

To join two tables in a single query, the database manager chooses the less costly of a *nested loop join* and a *merge scan join*. The two methods are described below.

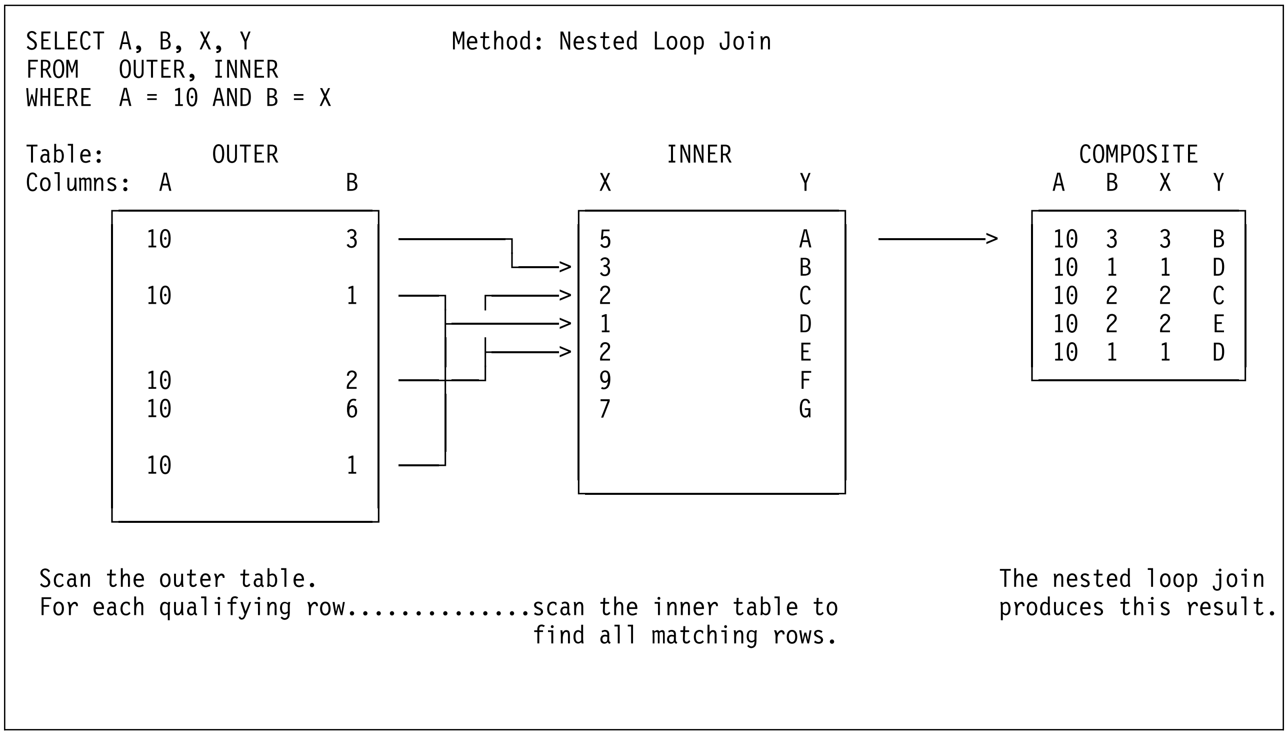
Nested Loop Join (Type 1)

In nested loop joins, the rows of one table (the “outer” table) are retrieved one by one. Sargable and residual predicates are applied to eliminate unqualified rows. For each qualified row of the outer table, the database manager opens a cursor on the second table (the “inner” table), and retrieves all rows that satisfy both the join predicate connecting the two tables and any local predicates on the inner table.

Either table can be scanned by a dbspace or index scan. The outer table is scanned once, while the inner table is scanned as many times as the number of qualifying rows in the outer table. Hence, the nested loop join is most efficient when the inner table has an efficient access path, and when only a few rows of the outer table remain after applying predicates to it. For a nested loop join:

```
Join cost = cost of outer table scan
          + ((Estimated number of qualifying records in outer table)  x
             (cost of inner table scan))
```

If the inner table is small enough to fit into its share of the buffer pool, the database manager anticipates that the entire inner table will remain in buffers throughout the operation. On this assumption, the I/O cost in the second term of the join cost is estimated as no more than the cost of scanning the inner table once. The nested loop join is illustrated in Figure 17.



```
SELECT A, B, X, Y              Method: Nested Loop Join
FROM   OUTER, INNER
WHERE  A = 10 AND B = X

Table:        OUTER                          INNER                    COMPOSITE
Columns:  A             B                X             Y            A   B   X   Y

          10            3                5             A            10  3   3   B
                                         3             B            10  1   1   D
          10            1                2             C            10  2   2   C
                                         1             D            10  2   2   E
                                         2             E            10  1   1   D
          10            2                9             F
          10            6                7             G

          10            1

 Scan the outer table.                                              The nested loop join
 For each qualifying row..............scan the inner table to       produces this result.
                                      find all matching rows.
```

Figure 17. Nested Loop Join

Merge Scan Join (Type 2)

For this method, there must be one or more predicates of the form TABLE1.COL1 = TABLE2.COL2, where the two columns have the same data type and length attribute. One of the predicates is chosen as the *merge join predicate*. The approach is to scan both tables in the order of the merge join columns, and to merge the result together whenever matching rows are found.

If the outer table has no efficient index on the join columns, an intermediate table is built by sorting the outer table on the join columns, applying any local predicates, and eliminating unused columns. The inner table is handled similarly. The database manager then reads the first row of both ordered tables (applying any predicates that remain). If the merge join predicate matches, the database manager returns the combined result. It then reads the next row of the inner table, which might match the same row of the outer table, and continues to read rows from the inner table and return the results until the merge join predicate fails to match. When there is no longer a match, the database manager reads the next outer table row. If that row has the same join predicate value, the database manager goes back and reads the matching group of records from the inner table again. If the outer row has a new join predicate value, the database manager searches ahead in the inner table until it finds either:

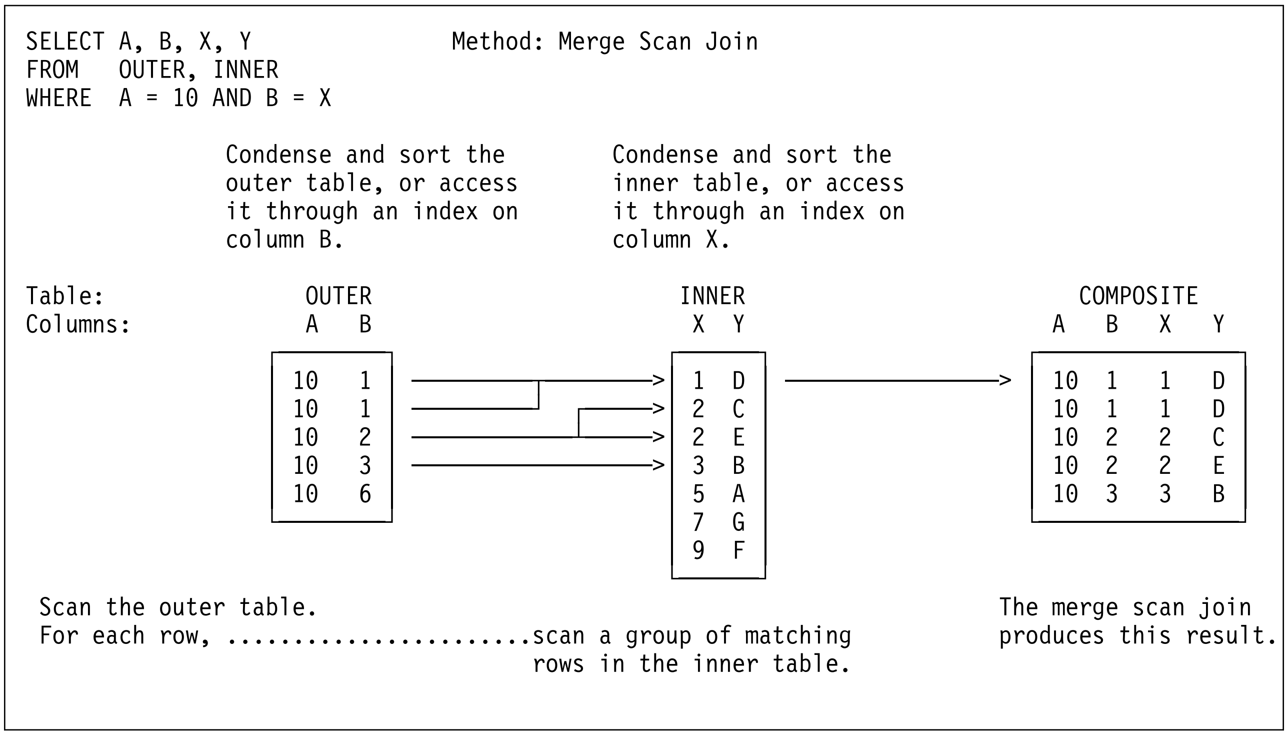
- A matching inner row, in which case the matching process is repeated.
- An inner row with a higher value than the join predicate. Then the database manager discards the unmatched row of the outer table and searches through the outer table until either a matching row or a row with a higher join predicate value is found. If a matching row is found, the matching process is again repeated. If a higher join predicate is found, the search moves back into the inner table.

Hence, for a merge scan join,

```
Join cost = cost of outer table scan
          + cost of sorting and reading outer table (if needed)
          + cost of inner table scan
          + cost of sorting and reading inner table (if needed)
```

If an efficient index does exist on the join column, and this index has been used to retrieve the rows of either table, the rows of that table are already in sequence. In this case, no sort of the table is required.

Merge scan join is illustrated in Figure 18 on page 145.



```
SELECT A, B, X, Y             Method: Merge Scan Join
FROM   OUTER, INNER
WHERE  A = 10 AND B = X

               Condense and sort the        Condense and sort the
               outer table, or access       inner table, or access
               it through an index on       it through an index on
               column B.                    column X.

Table:               OUTER                       INNER                        COMPOSITE
Columns:             A   B                        X  Y                      A   B   X   Y

                   10   1  ─────────────────>   1  D  ────────────────>  10  1   1   D
                   10   1  ─────────────────>   2  C                     10  1   1   D
                   10   2  ─────────────────>   2  E                     10  2   2   C
                   10   3  ─────────────────>   3  B                     10  2   2   E
                   10   6                       5  A                     10  3   3   B
                                                7  G
                                                9  F

 Scan the outer table.                                                The merge scan join
 For each row, .......................scan a group of matching         produces this result.
                                      rows in the inner table.
```

Figure 18. Merge Scan Join

Choosing an Access Method

When choosing an access method for the query shown on page 143, which entails a simple join of only two tables, four access methods must be considered: the nested loop join and the merge scan join, each with both possible choices of the outer and inner table. The choice will be based on a comparison of their costs.

Nested Loop, with EMPLOYEE as Outer Table

With no index on JOB, a scan occurs on all 10000 (500 pages) rows of the employee table. For each row, it first evaluates the predicate JOB = 'DESIGNER'; an estimated 200 (1/50 x 10000) rows remain. For each of those 200 rows, a scan takes place on all 3000 rows (60 pages) of the project table to find rows with WORKDEPT = DEPTNO and MAJPROJ = 'MA2100'. The major costs are:

```
Join cost = 10 000 row scan (500 page I/Os)
          + (200 x 60 page I/Os)              = 12 500 page I/Os
```

Nested Loop, with PROJECT as Outer Table

With no index on MAJPROJ, the database manager must scan all 3000 rows of the project table. For each row, it first evaluates the predicate MAJPROJ='MA2100'; an estimated 30 (1/100 x 3000) rows remain. For each of those 30 rows, the database manager must find all the rows in the EMPLOYEE table with WORKDEPT = DEPTNO and JOB = 'DESIGNER'. But instead of scanning the entire employee table, it can use the index on WORKDEPT each time. Each department has an average of 10 employees (10 000 rows / 1000 distinct departments), so the I/O cost is 1 index leaf page and 10 data pages for each value of WORKDEPT. The major costs are:

```
Join cost = 3000 row scan (60 page I/Os)
          + (30 x 11 page I/Os)                = 390 page I/Os
```

Merge Scan, with EMPLOYEE as Outer Table

Here the database manager reads the project table and applies the local predicate MAJPROJ = 'MA2100'. The remaining 30 rows are sorted and placed into a temporary table (a write and read of 1 page). Then, the database manager reads the employee table in department number order using the index on WORKDEPT (10 index pages + 500 data pages). For each row of the employee table, the database manager first evaluates the predicate JOB = 'DESIGNER'; if the row qualifies, it reads rows in the inner (temporary) table that match on department number. The costs are:

```
Join cost = cost of outer table index scan        510 page I/Os
          + cost of accessing inner table          60 page I/Os
          + cost of sorting and reading inner       2 page I/Os

                                                = 572 page I/Os
```

Merge Scan, with PROJECT as Outer Table

If the outer table is the project table, it cannot be accessed in department number order, and thus must be sorted. Before sorting, the database manager eliminates the rows that do not satisfy MAJPROJ = 'MA2100', leaving 30 rows. The estimated costs are:

```
Join cost = cost of outer table scan               60 page I/Os
          + cost of sorting and reading outer       2 page I/Os
          + cost of accessing inner table         510 page I/Os

                                                = 572 page I/Os
```

The nested loop scan with the project table as the outer table would be the best choice here. Only this access path makes use of the employee table index on WORKDEPT to avoid a scan of the entire table while not requiring a sort of the project table.

The two merge scan costs are shown as identical here because CPU costs and the presence of duplicates have been ignored. In reality, both would affect the actual cost calculated by the database manager.

Multiple Joins

Multiple joins are performed by logically joining two tables at a time, using either the nested loop or merge scan join method. This does not mean that each join necessarily produces an actual intermediate table; in a number of cases, no intermediate tables are required.

For example, a query joining tables T1, T2, and T3 may use a nested loop join to join T1 to T2, and a merge scan join to join the logical result (T1-T2) to T3. In this case, an intermediate or *composite* table might be created, and would appear in PLAN_TABLE if you examined the query with EXPLAIN. In addition, the composite table may require sorting to participate in the merge scanjoin. T3 may also require sorting (unless an index exists over the join columns). If the query has no ORDER BY clause, requiring sorting of the final result, then the T1-T2 composite table can be joined (using merge scan) with T3, returning each qualifying result row as it is found.

In short, whether composite tables are formed or not depends on a number of factors, including the SQL statement in question, the availability and type of indexes, and the catalog statistics. Any number of access plans can produce the correct answer, but the optimizer will choose the lowest estimated total cost solution.

Keeping Database Statistics Current

The catalog tables hold statistical information on data stored in the database, and the database manager uses these statistics to determine how it will access data for each individual SQL request. If the statistics are unavailable, then default values are used. Table 8 shows the key statistics used for access path selection, and identifies which catalog table they are in.

Table 8. Key Catalog Statistics Used for Path Selection

Table Name Column Name	**Description**	**Default Value Used by Optimizer When Catalog Value is -1**
SYSCATALOG		
ROWCOUNT	Total number of rows for this table.	100
NPAGES	Number of pages in the dbspace that contain rows of this table.	3
SYSDBSPACES		
NTABS	Number of tables in the dbspace.	
NACTIVE	Number of active pages in the dbspace.	3*NTABS
NPAGES	Number of usable pages in the dbspace.	
SYSCOLUMNS		
COLCOUNT	Number of distinct values in this column.	
HIGH2KEY	Second highest value in this column.	
LOW2KEY	Second lowest value in this column.	
AVGCOLLEN	Average length of the column.	
SYSINDEXES		
FULLKEYCOUNT	Number of distinct values of the full key.	
FIRSTKEYCOUNT	Number of distinct values of the first column of the key. Equals COLCOUNT for the index column.	
NLEAF	Number of leaf pages in the index.	
NLEVELS	Number of levels in the index.	
CLUSTERRATIO	Measure of how clustered an index is.	
SYSCOLSTATS		
VAL10	The value at the tenth percentile.	
VAL50	The value at the fiftieth percentile.	
VAL90	The value at the ninetieth percentile.	
FREQ1VAL	The most frequent value in the column.	
FREQ1PCT	Number of rows that contain that column value, given as a percentage of the total number of rows.	
	The second most frequent value in the column.	
FREQ2VAL	Number of rows that contain that column value,	
FREQ2PCT	given as a percentage of the total number of rows.	

Notes:

- Default values are only assigned to the base table. The values of NPAGES and NACTIVE determine PCTPAGES.
- Column statistics are used primarily for calculating filter factors. When these statistics are not available (a value of -1), the optimizer uses a default filter factor. These default filter factors are listed in Table 6 on page 136.

It is impractical for these statistics to be maintained on every INSERT, UPDATE, and DELETE operation; therefore, you must periodically update them in the catalog tables with the UPDATE STATISTICS or the UPDATE ALL STATISTICS statement.

Update your statistics whenever a table's contents change significantly.

Suppose you enter the following UPDATE STATISTICS statement:

```
UPDATE STATISTICS FOR TABLE MYTABLE
```

The database manager updates the statistics for MYTABLE in the catalog tables. However, the statistics are updated only for indexed columns. (For indexes having multicolumn keys, only the first column is updated.) Similarly, whenever you create a new index, the database manager automatically updates the statistics for index columns.

To update the statistics for *all* columns in MYTABLE (even those that are not indexed), enter the following statement:

```
UPDATE ALL STATISTICS FOR TABLE MYTABLE
```

The complete set of statistics produced by the ALL option may result in a better access strategy being selected by the optimizer component. However, the ALL option can greatly increase the processing time required to run the UPDATE STATISTICS statement. UPDATE ALL STATISTICS is recommended where queries have non-indexed columns with local predicates or queries have multi-column indexes with local or join predicates that are not on the first column of an index. The *DB2 Server for VSE & VM SQL Reference* manual describes exactly what operations affect each statistic. The SYS0001 DBSPACE, which contains the catalog tables, is a candidate for UPDATE ALL STATISTICS processing.

It is recommended that you schedule UPDATE STATISTICS activities during off-peak hours.

When working with preplanned application programs, ensure that the programs are re-preprocessed whenever the tables accessed by the application have significantly changed (for example, a 10%-20% or more change). Before re-preprocessing, ensure that statistics have been updated so that the optimizer is provided with the new characteristics of the data.

The DBS utility DATALOAD and RELOAD commands automatically collect the statistics for a table as part of the load operation; thus, it is not necessary to issue a separate UPDATE STATISTICS statement (although see the *DB2 Server for VSE & VM Database Services Utility* or the *DB2 Server for VSE & VM Database Services Utility* manuals for restrictions). If you want, you can suppress this automatic updating of statistics through the DBS Utility SET UPDATE STATISTICS command.

Note: Statistics are *not* updated when you use DATALOAD to load data into a view.

Using Catalog Statistics

The following suggestions introduce the techniques of using the catalog statistics to influence the choice of access methods made by the database manager. For more information see “Tuning Queries with Several Tables” on page 142.

Modelling your Production System

If you have DBA authority, you are allowed to update the statistical values stored in the catalog tables. See Table 8 on page 147 for a description of the columns in question. This ability lets you create a model of your production system on a smaller test system. You can then use the EXPLAIN statement to determine how your production tables would be accessed for some set of SQL statements. In the same way, you can model a future production system by making assumptions about the size and nature of the database structure.

To create a model of your production system, the same database structure must be in place on your test system. This means the same dbspaces, tables, indexes, referential constraints, and so on must be defined on the test system. You can then modify the statistics in your test system catalog tables to be identical to those in the production system. The optimizer will choose the same paths to access your tables on the test system as it would on the production system.

After you have a model of the production system established, you can discover how the optimizer will react to changes in the database structure, such as adding new indexes by updating the catalog table statistics. Using EXPLAIN will tell you whether (and how) a new index would be used for a particular SQL statement, and how it impacts the expected costs of executing that statement. With this information, you can decide whether you should add the index to the production system.

Similarly, you can now use your test system to discover how rewriting an SQL statement into an alternate form affects the path chosen and the estimated cost for executing the statement.

A Warning about Updating Statistics

If you supply the COLCOUNT value for an index column without running UPDATE STATISTICS, you should also supply HIGH2KEY and LOW2KEY for the index. If the data is not uniformly distributed you should also supply the additional values in the SYSCOLSTATS table. These columns are defined as CHAR, so an UPDATE statement must provide a character or hexadecimal value. Although the columns have a length of 12, only 8 bytes of information should be stored. Entering a character value is quite straightforward—SET LOW2KEY = `'ALAS'`, for instance. But to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 Server for VSE & VM internal format. To determine the proper hexadecimal data to use for these data types, create a table with columns of the required types and insert the values you want to use for HIGH2KEY and LOW2KEY into it. Then display the internal format of these values by using the HEX column function in the select list. For example:

```
SELECT HEX(column_name) FROM ...
```

Be sure to allow for a NULL indicator in keys that allow NULLS by making the first character `'00'`X. If values being set are less than 8 bytes long (including the `'00'`X NULL indicator byte) pad them on the right with `'00'`X bytes.

If the NPAGES column of SYSTEM.SYSDBSPACES is updated (to allow testing of the access plan generation) and then an ACQUIRE DBSPACE command attempts to acquire this dbspace, an error message may result. Updating NPAGES does not actually change the size of a dbspace, it changes the information supplied to the optimizer used in access plan generation.

Determining the Cost of Access Methods

The access method cost has two parts: a processing cost and an I/O cost. Depending on the hardware environment, a query can be either CPU or I/O bound. You may want to compare the cost characteristics of your queries to equivalent alternatives.

Processing Cost

The database manager estimates processing cost as a result of:

- The number of rows considered
- The number of residual predicates checked for each row
- The number of rows that satisfy the residual predicates.

This estimate of the number of rows that remain after applying all the predicates is dependent on assumptions on the distribution of data within the column. For the first column of an index, the database manager records additional information to help it recognize non-uniform data distributions. Otherwise, it assumes that the data values are evenly distributed, and uses the following rules:

For a predicate of the form WHERE *column* = *value***:** The number of rows is estimated as either ROWCOUNT/COLCOUNT or ROWCOUNT*(1/COLCOUNT); that is, the total number of rows divided by the number of distinct values in the column. The term 1/COLCOUNT is the filter factor.

For a predicate that uses a range operator: The number of rows is estimated using the ratio of the range encompassed in the predicate to the range of values in the column. Thus, if LOW2KEY and HIGH2KEY are respectively 10 and 90, then the predicate *column* > 70 is given a filter factor of, approximately, (90 - 70)/(90 - 10), or 0.25. Only the first 8 bytes of the column are stored for HIGH2KEY and LOW2KEY so it is important that columns be distinct within the first 8 bytes. (7 bytes if the column is nullable.)

I/O Cost

The I/O cost is estimated by the number of index pages, the number of data pages, and the number of directory pages to be read.

For a dbspace scan, the number of index pages read is zero. Otherwise, it is determined from the number of leaf pages and levels in the index (NLEAF and NLEVELS) and the filter factors of the matching predicates.

A dbspace scan reads all data pages. The total number of pages is given by NACTIVE.

If access is through an index, the proportion of pages read depends on the filter factor for the predicates applied through the indexes and on the extent to which the data rows are clustered by the index.

Using Explanation Tables to Evaluate Performance

The explanation tables produced by the EXPLAIN statement allow you to get information about the structure and execution performance of SQL statements. This information can help you analyze how existing database designs perform, or how future designs will perform. Specifically, you can use explanation tables to:

- Find out the indexes that are used for a given statement, the number of index columns used selectively, whether index-only access was sufficient to fulfill the request, and whether a fetch operation was required
- Find out the sorts that are required, and the reason for the sorts
- Analyze request loads
- Estimate the size of responses
- Separate queries into their subquery structures
- Obtain costs for statements and access paths
- Assist in database design
- Determine when a program must be preprocessed again.

After you complete your design, and construct a prototype, you can use explanation tables to see how well real queries will work against the design. (You can select explain processing explicitly using the EXPLAIN statement, or implicitly using the EXPLAIN(YES) preprocessing parameter or the USING EXPLAIN(YES) option of the CREATE PACKAGE statement.)

Explain Processing

Explain processing accepts an SQL statement as an argument, analyzes it, and inserts information about the structure and execution of that statement into the explanation tables, which you must create. (This includes the cost of internally generated statements.) You can then query the explanation tables.

You can select explain processing explicitly by using the EXPLAIN statement, or implicitly using the EXPLAIN(YES) preprocessor parameter or the EXPLAIN(YES) USING OPTION of the CREATE PACKAGE statement.

Note: Explain processing does **not** execute the SQL statement. It only explains how the statement will work when you actually execute it.

Using the EXPLAIN Statement

You can use the SQL EXPLAIN statement in an application program, the DBS utility, or ISQL to estimate execution performance. For the syntax of the EXPLAIN statement and the structure of the explanation tables produced, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

To explicitly process the EXPLAIN statement, you must either:

- Preprocess the program every time you change the statement that EXPLAIN is to analyze; or,

- Use multiple EXPLAIN statements; or,
- Have the program build the EXPLAIN statement and then execute it using the dynamic prepare statements.
- Replace all the host variables with parameter markers and then issue an EXPLAIN for it. For specific rules regarding the use of parameter markers, see the *DB2 Server for VSE & VM SQL Reference* manual.

Each time the EXPLAIN statement is executed, rows are appended to the specified explanation tables. Any existing rows are not affected.

When the EXPLAIN statement is issued for INSERT, UPDATE, and DELETE statements that change tables in a referential structure, information is returned not only on the INSERT, UPDATE, and DELETE statements, but also on internally generated statements. Refer to the *DB2 Server for VM Application Programming* or the *DB2 Server for VSE Application Programming* manuals for more information on internally generated statements.

Each of the explanation tables has a column called QUERYNO (query number). The QUERYNO column has a data type of INTEGER. With the SET QUERYNO clause, you can place an integer value in the QUERYNO columns of the rows inserted by the EXPLAIN statement. Thus, you can use QUERYNO to identify new rows, and to mark them as corresponding to a particular statement.

For *integer* in SET QUERYNO, you must specify an integer constant that is not preceded by a sign. You cannot use a host variable in the SET QUERYNO clause, even in application programs. However, you can use the “&n” place-holder variables in ISQL.

The SET QUERYNO clause is optional. If you omit it, a NULL value is placed in the fields of the rows inserted by the EXPLAIN statement. If you set the QUERYNO to some initial value, this value identifies the query for which the EXPLAIN is issued. Because QUERYNO is an INTEGER field, an error is returned if its value exceeds 2 147 483 647.

You can enter the EXPLAIN statement from **ISQL**. When you enter EXPLAIN from ISQL, you must use a character constant if you execute the statement immediately. Alternatively, you can store the EXPLAIN statement by placing it in a routine or by using an ISQL STORE statement. This lets you use a place-holder (for example, &1) for *explainable_sql_statement* (refer to the *DB2 Server for VSE & VM SQL Reference* manual for the syntax of the EXPLAIN statement, including *explainable_sql_statement*). Thus, you can execute EXPLAIN for different SQL statements without having to key in the entire EXPLAIN statement each time. When using this technique, however, you should keep ISQL limits in mind. For example, when the place-holder is in a routine table, the length of input to a parameter is limited by the length of the COMMAND column of the routine table. At most, input to a place-holder can be 254 characters. This number is further reduced if you do not put the “&n” place-holder on a line by itself.

When you enter EXPLAIN from the **DBS utility**, you must use a character constant for *explainable_sql_statement*. The utility does not allow the use of host variables or place-holders in any SQL statement.

Using the EXPLAIN Option

You can select explain processing implicitly using the EXPLAIN(YES) preprocessing parameter or the EXPLAIN(YES) using option of the CREATE PACKAGE statement. If you select explain processing implicitly, explanatory information is provided for all SQL statements in a package that can be explained and for all internally generated statements. The name of the package and the name of the owner of the package are stored in the explanation tables.

Because you cannot assign a QUERYNO when you select explain processing implicitly, the section number assigned to the statement being preprocessed is used as the query number. This number corresponds to the position of the query in the application. Using the preprocessor listing file, you can determine the section number assigned to a statement and use it to determine the corresponding rows in the explanation tables. The following variable names are used as section numbers for the languages specified.

Table 9. Variable Names for Section Number (Query Number)

Language	Structure Name	Variable Name
ASM	RDIIN	RDISECT#
COBOL	RDIIN	SQL-SECTION-NUM
C	RDIIN	SECTION_NUM
PLI	RDIIN	SECTION_NUM
FORTRAN	SQLCTL	SQLSTMT

For further information, see the *DB2 Server for VSE & VM SQL Reference*, the *DB2 Server for VM Application Programming*, and the *DB2 Server for VSE Application Programming* manuals.

Comparing Implicit and Explicit Explain Processing

Implicit and explicit explain processing insert the same kind of information into the explanation tables during explain processing. The package name and package owner columns of the explanation tables, however, contain information only if implicit explain processing is used.

There is also a difference between implicit and explicit explain processing for the query number of an application. For explicit explain processing, if you do not supply the query number, it is set to NULL. For implicit explain processing, you cannot provide query numbers for SQL statements in the middle of an application, so the section number assigned to the statement when it is processed is used as the query number. You can then use the preprocessor listing file to determine the section number assigned to each statement and the corresponding rows in the explanation tables.

During explicit explain processing, rows are added to the explanation tables when a program is preprocessed, or dynamically repreprocessed. During implicit explain processing, rows are added when a program is preprocessed, but not when it is dynamically repreprocessed. In all situations, explicit explain processing overrides implicit explain processing.

When you processes an application program using the **implicit** EXPLAIN(YES) option, the preprocessor checks for the existence of the EXPLAIN tables once. If it does not find them, processing is terminated and SQLCODE -649 (SQLSTATE = 42704) is issued.

Each time the preprocessor encounters an **explicit** EXPLAIN statement, it checks for the existence of the explain tables. If its does not find them the explicit EXPLAIN is not processed and, SQLCODE -204 (SQLSTATE = 42704) or SQLWARNING +204 (SQLSTATE = 01532) is issued. This check is repeated for every explicit EXPLAIN statement that the preprocessor encounters.

Using Explanation Tables

There are four explanation tables: REFERENCE_TABLE, STRUCTURE_TABLE, COST_TABLE, and PLAN_TABLE. The definitions of these tables and the EXPLAIN statement syntax are in the *DB2 Server for VSE & VM SQL Reference* manual. A DBS utility job file, ARISEXP, to generate explanation tables, indexes, and views is shipped with the DB2 Server for VSE & VM product. Instructions for generating the tables using the ARISEXP file are provided at the top of the file. For further information on the contents of the explanation tables, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

When you execute an EXPLAIN statement, information is placed in your tables, and is independent of any other user's explanation tables. You can review and summarize the information placed in your explanation tables just as you can other tables. However, because explanation tables only insert rows, you also have the responsibility to delete unnecessary information yourself.

In the following descriptions of each table, the term *query block* is used. A query block is a part of a query. Query blocks are used to distinguish the parts of a subquery. For example, when a query does not involve a subquery, there is only one query block: query block 1. When there is a subquery, there are two query blocks, the outer-level query and the subquery. They are referred to as query block 1 and query block 2, respectively. Because subqueries may be nested within each other, there may be many query blocks in a statement; each query block corresponds to separate (but interacting) parts of the statement.

The SELECT statement in Figure 19 is used in the following descriptions of the explanation tables. This SELECT statement has only one query block.

```
SELECT X.DEPTNAME, Y.FIRSTNME, Y.MIDINIT, Y.LASTNAME, Y.PHONENO
FROM DEPARTMENT X, EMPLOYEE Y
WHERE X.MGRNO = Y.EMPNO
  AND X.DEPTNO = Y.WORKDEPT
```

Figure 19. SELECT Statement for Explanation Table Descriptions

Assume that user Smith owns tables DEPARTMENT and EMPLOYEE where:

- DEPARTMENT has columns DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT.
- EMPLOYEE has columns EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, and COMM.

Using the Cost Table: This table is updated by EXPLAIN COST or EXPLAIN ALL. The information in this table provides the cost estimate of the statement for which the EXPLAIN is issued and for any internally generated statement used to enforce referential integrity. In addition, you can compute the contribution of each subquery (if any) to the total cost estimate of the statement. (To compute the subquery cost estimates, you will need to use information provided by the EXPLAIN STRUCTURE statement.)

For each query block in the statement, EXPLAIN inserts one row into COST_TABLE. The information depends on the existing indexes and catalog statistics. If indexes are added or dropped after you issue EXPLAIN for the statement, then the COST_TABLE entry for the command is not valid.

If you need a description of the columns in the COST_TABLE, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

The value in COST is referred to as the cost estimate (occasionally referred to as the optimizer cost estimate or the resource cost estimate). All of these terms refer to the same thing: the internal value that the optimizer uses to represent the resource cost of executing an SQL statement for which the EXPLAIN is issued and for any internally generated statement used to enforce referential integrity. The value is a relative value that incorporates I/O requirements with a weighted factor of processor requirements for a query.

Aside from the COST column, there are two other ways to get this value for a given SQL statement. One way is to examine the SQLCA after preparing a dynamically defined SQL statement. The cost estimate is kept in the SQLERRD(4) field. A second way to see the cost estimate is using ISQL. ISQL displays the query cost estimate integer. The integer results from dividing the real internal value by 1000 and adding 1 to it. This produces a number that is easier to grasp. This is a valid technique because the numbers are relative to each other; they do not represent real physical consumption directly.

It follows, then, that it is futile to try to develop an algorithm that directly maps the cost estimate to a real physical unit such as time. Too many other factors are involved (for example, overall system workload). It is best to use the cost estimate as a general indicator.

For the SELECT statement in Figure 19, there would be only one row entered into COST_TABLE, because there is only one query block:

```
QUERYNO  RINO  QBLOCKNO  PKGNAME  PKGOWNER  COST
-------  ----  --------  -------  --------  ------------------
      1     0         1                     1.4388885498046E+01

TIMESTAMP
--------------------------
1996-08-26-09.49.25.601721
```

Figure 20. Results of COST_TABLE Query

Using the Plan Table: This table is updated by EXPLAIN PLAN or EXPLAIN ALL. The information in this table describes the order in which tables are accessed by the statement for which the EXPLAIN is issued and by any internally generated statement used to enforce referential integrity. In addition, the PLAN_TABLE table describes the indexes used to access the tables, and specifies whether indexes alone were used, the methods that the database manager used to do joins, the sorts done as part of runtime processing, and the reasons for the sorts.

As with the COST_TABLE, the PLAN_TABLE results depend on the existing indexes and catalog statistics at the time the EXPLAIN statement is executed. If indexes are added or deleted, then the PLAN_TABLE entry for the statement is not valid.

For each step in the plan determined by the database manager for processing the query, EXPLAIN inserts one row into the PLAN_TABLE. There is one step for each table reference in a query block. There are additional steps if the database manager must perform additional sorts at the end of processing for the query block, or if any internally generated statements are to be processed.

The steps in the plan are ordered by the value of the PLANNO column of PLAN_TABLE, and for each step, the TNAME column identifies the table accessed. The phrase “previous steps of the plan” refers to PLAN_TABLE rows with smaller values of PLANNO. The action described in a step is either a join of a table to those previously joined, or it is a sort. (Joins themselves may involve performing sorts.) The term “composite” refers to the result of all previous steps; the term “new” refers to the new table that is being accessed and joined as part of a particular plan step.

If you need a description of the columns in the PLAN_TABLE, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

The PLAN_TABLE for the query in Figure 19 is shown below. Because there are many columns in PLAN_TABLE, the display of the table is split to fit on the page:

```
QUERYNO  RINO  QBLOCKNO  PKGNAME  PKGOWNER  PLANNO  METHOD  CREATOR
-------  ----  --------  -------  --------  ------  ------  -------
      1     0         1                          1       0  SMITH
      1     0         1                          2       1  SMITH

TNAME       TABNO  ACCESSTYPE  MATCHCOLS  ACCESSCREATOR
----------  -----  ----------  ---------  -------------
DEPARTMENT      1  W                   0  SMITH
EMPLOYEE        2  I                   0  SMITH

ACCESSNAME         INDEXONLY  SORTNEW  SORTCOMP  SORTN_UNIQ  SORTN_JOIN
----------         ---------  -------  --------  ----------  ----------
MGRNOI             N          N        N         N           N
PKEYB1H9ZR8CD51W   N          N        N         N           N

SORTN_ORDERBY  SORTN_GROUPBY  SORTC_UNIQ  SORTC_JOIN  SORTC_ORDERBY
-------------  -------------  ----------  ----------  -------------
N              N              N           N           N
N              N              N           N           N

SORTC_GROUPBY  TIMESTAMP                   REMARKS
-------------  --------------------------  -------
N              1996-08-26-09.49.25.601721
N              1996-08-26-09.49.25.601721
```

Figure 21. Results of PLAN_TABLE Query

A Type 1 (or nested loop) join is performed on tables EMPLOYEE and DEPARTMENT. The database manager accesses DEPARTMENT as the outer table of the join (the first table accessed), and EMPLOYEE as the inner table of the join.

The row with PLANNO=1 indicates that the database manager accesses DEPARTMENT using the index MGRNOI (which, as it happens, was created on the MGRNO column).

The entry with PLANNO=2 indicates that the database manager has performed an action on the EMPLOYEE table, based upon the conditions included in the query. An index has been generated internally on the primary key EMPNO of the EMPLOYEE table. This index performs the matching of EMPNO (in the EMPLOYEE table) to MGRNO (in the DEPARTMENT table). This index can be used because the value in MGRNO of DEPARTMENT, which must be matched by the EMPNO value of EMPLOYEE. Retrieval of rows from an inner table of a join will often, though not always, use an index on a join column of the inner table.

No sorts are used in the plan for this query. However, if the query had demanded SELECT DISTINCT, instead of SELECT, the plan would have an additional row, with PLANNO=3, which would have TABNO=0, METHOD=3, SORTC_UNIQ=‘Y’ and SORTCOMP=‘U’.

Using the Reference Table: This table is updated by EXPLAIN REFERENCE or EXPLAIN ALL. The database manager inserts one row in REFERENCE_TABLE for each column referenced in the statement (in certain ways, as explained below) and for any internally generated statement used to enforce referential integrity. Even if the column is referenced more than once for a table, there is still only one row inserted for the column and that row is for the most selective predicate. However, multiple appearances of a table in a query (as when a table is joined to itself) can lead to multiple descriptions of their columns.

One row is entered for each table reference, one for the statement as a whole, and one that indicates the way in which the column is used in the query. For a description of the columns in the REFERENCE_TABLE, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

For the example statement presented in Figure 19 on page 154, the new rows entered into REFERENCE_TABLE by EXPLAIN REFERENCE might be:

```
QUERYNO  RINO  QBLOCKNO  PKGNAME  PKGOWNER  REFTYPE  CREATOR  TNAME
-------  ----  --------  -------  --------  -------  -------  ----------
      1     0         1                     SELECT
      1     0         1                     TABLE    SMITH    DEPARTMENT
      1     0         1                     TABLE    SMITH    EMPLOYEE
      1     0         1                     COLUMN   SMITH    DEPARTMENT
      1     0         1                     COLUMN   SMITH    DEPARTMENT
      1     0         1                     COLUMN   SMITH    DEPARTMENT
      1     0         1                     COLUMN   SMITH    EMPLOYEE
      1     0         1                     COLUMN   SMITH    EMPLOYEE
      1     0         1                     COLUMN   SMITH    EMPLOYEE
      1     0         1                     COLUMN   SMITH    EMPLOYEE
      1     0         1                     COLUMN   SMITH    EMPLOYEE
      1     0         1                     COLUMN   SMITH    EMPLOYEE

TABNO  CNAME  COLNO  FILTER               DBSSPRED  JOINPRED  ORDERCOL
-----  -----  -----  -------------------  --------  --------  --------
    0             0                0.0E0                              0
    1             0                0.0E0                              0
    2             0                0.0E0                              0
    1  DEPTNAME   2              1.0E+00  N         N                0
    1  DEPTNO     1   1.111111448837E-01  Y         Y                0
    1  MGRNO      3            3.125E-02  Y         Y                0
    2  EMPNO      1            3.125E-02  Y         Y                0
    2  FIRSTNME   2              1.0E+00  N         N                0
    2  LASTNAME   4              1.0E+00  N         N                0
    2  MIDINIT    3              1.0E+00  N         N                0
    2  PHONENO    6              1.0E+00  N         N                0
    2  WORKDEPT   5  1.1111110448837E-01  Y         Y                0

GROUPCOL  UPDATECOL  TIMESTAMP
--------  ---------  -------------------------
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
       0             1996-08-26-09.49.25.601721
```

Figure 22. Results of the REFERENCE_TABLE Query

These rows indicate that the statement is a SELECT statement with no subqueries, joining two tables, SMITH.DEPARTMENT and SMITH.EMPLOYEE. The columns MGRNO, DEPTNO from DEPARTMENT, and the columns EMPNO, WORKDEPT from EMPLOYEE appear together in the 'WHERE' clause (identified by a Y in the DBSSPRED column), permitting indexes to be used. These columns are the JOIN columns (identified by a Y in the JOINPRED column). FILTER may be misleading, because the filtering depends on the order in which tables are processed.

Referential Integrity (RINO Value)

RINO is set to zero for the original statement and is automatically incremented by one for each internally-generated statement that is processed for referential integrity or cascade delete. For example if you perform an EXPLAIN against a statement that deletes a department from the DEPARTMENT table, the following REFERENCE table is generated.

```
        QNO   RINO QBLOCK REFTYPE TNAME              CNAME              ...
----------- ------ ------ ------- ------------------ ------------------...
          1      0      0 DELETE                                        ...
          1      0      1 TABLE   DEPARTMENT                            ...
          1      1      0 UPDATE                                        ...
          1      1      1 TABLE   EMPLOYEE                              ...
          1      1      1 COLUMN  EMPLOYEE           WORKDEPT           ...
          1      2      0 SELECT                                        ...
          1      2      1 TABLE   PROJECT                               ...
          1      2      1 COLUMN  PROJECT            DEPTNO             ...
```

Notice that the DELETE statement that was written (RINO=0) produces two other statements: First an UPDATE that changes the WORKDEPT column for any employee in the deleted department to NULL (RINO=1), and second a SELECT that checks that any departments to be deleted do not have any projects assigned to them (RINO=2).

The REFERENCE_TABLE and the PLAN-TABLE can be used together to indicate whether materialization was used to generate a view. View materialization lifts a number of restrictions on the use of views, including the use of column functions operating on the column of a view when the definition of the view already contains a column function. For example:

```
CREATE VIEW V1(DPT,MAXSAL) AS
    SELECT   WORKDEPT, MAX(SALARY)
    FROM     EMPLOYEE
    GROUP BY WORKDEPT
EXPLAIN ALL FOR SELECT DPT FROM V1
```

Because view materialization is used for view V1, the TNAME column in the REFERENCE_TABLE and PLAN_TABLE will contain the name of the view. Keeping in mind that view materialization is generally more expensive than merging the SELECT statement of the view with that of the query, the information on the EXPLAIN tables can be helpful in performance tuning.

Using the Structure Table: This table is updated by EXPLAIN STRUCTURE or EXPLAIN ALL. The database manager inserts one row in STRUCTURE_TABLE for each query block in the statement.

If you need a description of the columns in the STRUCTURE_TABLE, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

If the following SELECT statement is issued, only one row is entered in STRUCTURE_TABLE, as shown in Figure 23 on page 160, because there is only one query block.

```
   EXPLAIN ALL FOR SELECT * FROM EMPLOYEE
```

```
QUERYNO  RINO  QBLOCKNO  PKGNAME  PKGOWNER  ROWCOUNT      TIMES
-------  ----  --------  -------  --------  --------  ---------
            0         1                           32      0.0E0

PARENT  ATOPEN  TIMESTAMP
------  ------  --------------------------
     0  N       1996-08-26-09.49.26.001720
```

Figure 23. Results of the STRUCTURE_TABLE Query

A more complicated example is provided in the following sections, where we show how to separate the costs for individual query blocks using STRUCTURE_TABLE and COST_TABLE together.

Using Subquery Blocks: A query may have subqueries, which in turn may have subqueries. The database manager separates this tree of subqueries into pieces, called query blocks. Each query block has its own tables, columns, and rowcount. EXPLAIN STRUCTURE, COST lets you look at combined information, or to separately examine the information for each query block.

The PARENT field gives the logical parent for each query block, which is not always obvious from the query itself. Sometimes, a query block has no correlation to the query where it immediately appears, so it is executed only once, when some ancestor query block is first entered, rather than many times.

The following example has a large number of query blocks, but references only one table (many times). It illustrates the meanings of PARENT and ATOPEN, as well as the method for decomposing COST values into separate costs for query blocks.

```
                                             QBLOCKNO
                                            ---------
SELECT * FROM DEPT X                        *** 1 ***
WHERE DNAME > ALL
  (SELECT DNAME FROM DEPT                   *** 2 ***
   WHERE X.DNO = DNO
   AND LOC = 32)
AND DNO =
  (SELECT DNO FROM DEPT Y                   *** 3 ***
   WHERE MGR =
     (SELECT MGR FROM DEPT Z                *** 4 ***
      WHERE DNAME IN
        (SELECT DNAME FROM DEPT             *** 5 ***
         WHERE NEMP = X.NEMP)
      AND DNO =
        (SELECT DNO FROM DEPT W             *** 6 ***
         WHERE NEMP > Z.NEMP
         AND LOC IN
           (SELECT LOC FROM DEPT            *** 7 ***
            WHERE DNAME = Y.DNAME))
       AND LOC = 32 )
   AND Y.NEMP <
     (SELECT AVG(NEMP) FROM DEPT            *** 8 ***
      WHERE Y.MGR = MGR ));
```

Here are results from STRUCTURE_TABLE for this query. (ROWCOUNT is not shown.)

```
QUERYNO  RINO  QBLOCKNO  PKGNAME  PKGOWNER   TIMES  PARENT
-------  ----  --------  -------  --------  ------  ------
      ?     0         1                      9.000       0
      ?     0         2                      0.500       1
      ?     0         3                      9.000       1
      ?     0         4                      0.500       3
      ?     0         5                      2.000       3
      ?     0         6                      1.500       4
      ?     0         7                      1.000       4
      ?     0         8                      1.000       3

ATOPEN  TIMESTAMP
------  --------------------------
     N  1996-08-26-09.49.27.011719
     N  1996-08-26-09.49.27.011719
     N  1996-08-26-09.49.27.011719
     N  1996-08-26-09.49.27.011719
     Y  1996-08-26-09.49.27.011719
     N  1996-08-26-09.49.27.011719
     Y  1996-08-26-09.49.27.011719
     N  1996-08-26-09.49.27.011719
```

Figure 24. Example STRUCTURE_TABLE

Note: A ? indicates a NULL value.

Although query block 5 is physically nested within query block 4, it references neither Y nor Z. Hence, its value can be computed once each time query block 3 is first entered, with a particular X row.

Query block 5 is executed once, at open time of query block 3.

Query block 7 is physically within query block 6, but can be evaluated once when the database manager first enters query block 4. Query block 7 does not need to be re-executed again until new values for rows outside query block 4 are required. This is different from query block 2, for example, which must be executed once for each row of query block 1, rather than just once when execution of query block 1 begins.

TIMES may be a fraction (and may be less than 1) because it represents the estimated number of times, per execution of a query block, that its dependent query blocks will be executed.

The most important use of this query block breakdown involves computation of costs for individual query blocks, instead of costs for the query as a whole. This is described next.

Computing Block Costs: Here are the COST_TABLE entries for the query in the preceding section:

```
QUERYNO  RINO  QBLOCKNO  PKGNAME  PKGOWNER       COST
-------  ----  --------  -------  --------  ---------
      ?     0         1                      3021.000
      ?     0         2                        10.000
      ?     0         3                       324.000
      ?     0         4                        24.000
      ?     0         5                         4.000
      ?     0         6                         6.000
      ?     0         7                        10.000
      ?     0         8                        10.000

TIMESTAMP
--------------------------
1996-08-26-09.49.27.011719
1996-08-26-09.49.27.011719
1996-08-26-09.49.27.011719
1996-08-26-09.49.27.011719
1996-08-26-09.49.27.011719
1996-08-26-09.49.27.011719
1996-08-26-09.49.27.011719
1996-08-26-09.49.27.011719
```

Figure 25. Results of COST_TABLE Query

The cost displayed is the total cost for each query block, including costs associated with all query blocks that are below it in the logical tree of query blocks. Thus, the cost of executing the statement is approximately 3021. (For simplicity in the calculations that follow, the values are shown as integers, but they need not be.)

The cost for executing the entire statement helps you understand the effect of the statement on system load, but it hides the blocks of the query that are contributing the most to the cost of the query.

A more useful set of figures might be those listed in the following table:

Note: The following table is only an example. It is not stored in the database, and INDIVCOST and MULTCOST are not columns in COST_TABLE.

```
QBLOCKNO  INDIVCOST     MULTCOST
--------  ---------   ----------
       1     15.000       15.000
       2     10.000       90.000
       3     14.000      126.000
       4     11.000      891.000
       5      4.000       36.000
       6      6.000      243.000
       7     10.000      810.000
       8     10.000      810.000
```

Figure 26. Example COST_TABLE

INDIVCOST represents the cost of one execution of the individual query block itself, not including the costs of any of its subqueries. MULTCOST not only counts the cost of the individual query block, but multiplies INDIVCOST by the number of times the query block is expected to be executed in the query. This is a better measure of the cost importance of the query block than either COST or INDIVCOST. Notice that the MULTCOST column adds up to COST(1), the total cost of the entire query.

Thus, the following formula can be used to derive INDIVCOST:

```
COST(I) = INDIVCOST(I) + the sum, over all blocks J that have
                    I as logical parent, of either:

                        TIMES(I) * COST(J)

                    if J is not done AT OPEN of I, or

                        COST(J)

                    if J is done AT OPEN of I.
```

For example:

```
COST(3) = 324 = INDIVCOST(3) + 9*24 + 4 + 9*10,
```

so INDIVCOST(3) is 14.

Here is another example:

```
COST(4) = 24 = INDIVCOST(4) + 0.5*6 + 10,
```

so INDIVCOST(4) is 11.

This formula can be used to derive MULTCOST from INDIVCOST:

```
MULTCOST(I) =
   INDIVCOST(I) * the product of TIMES(I) for
         all logical ancestors of I,
               if I is not done AT OPEN of its parent,
           or
   INDIVCOST(I) * the product of TIMES(I) for
         all logical ancestors of I EXCEPT its parent,
               if I is done AT OPEN of its parent.
```

For example:

```
MULTCOST(7) = 10 * TIMES(3) * TIMES(1) = 810.
```

We do not multiply TIMES(4) into that product, because 4 is 7's immediate parent, and 7 is done AT OPEN of 4.

This demonstrates that the most important component of the estimated cost comes from block 4, and you might choose indexes that make processing this query block cheaper. By inspecting the query, or from the rows in REFERENCE_TABLE, you might decide that indexes on one of DNAME, DNO or LOC might reduce the cost of processing.

Estimating Sizes of Responses

Because ROWCOUNT (estimated number of rows in result, for queries, or of affected rows, for updates and deletes) is stored in STRUCTURE_TABLE, it is easy to gauge the estimated size of responses. If your structure includes a DELETE CASCADE rule, EXPLAIN will include the cost of the cascading effects of a DELETE. This can help you understand whether requests are reasonable, and whether the statistics in the database catalog tables that estimate ROWCOUNT seem up-to-date.

ROWCOUNT can also help determine space requirements when responses are being stored in program data structures. However, ROWCOUNT, like all estimates made by the system, is neither precisely accurate, nor even an upper bound on the actual number of rows in the response.

Using EXPLAIN for Database Design

A systematic analysis of many statements in the workload of the system can help the administrator plan the access paths for the database. The analysis should consider costs when different combinations of indexes exist. It should also consider the costs of performing updates, which are not reflected in the COST column of COST_TABLE, and limits on space for indexes.

You may load the database with tables ordered on certain columns. Indexes on such columns (called CLUSTERING indexes) enable the database manager to maintain this ordering. Because the ordering of rows within tables strongly affects the costs of execution, it may be worthwhile to reload the database to improve performance. For each statement, join, ORDER BY, and GROUP BY columns may be good candidates for ordering. Also, tables that are often joined might be interleaved on join columns when the database is loaded.

If the cost of executing a statement (as determined by EXPLAIN COST, or by running the statement) is higher than expected, a user or administrator may want to look at the procedure that the database manager chose to execute that statement. Building additional access paths or altering the layout of tables may be necessary to achieve good performance for the statement. For example, if a relation scan, that is ACCESS TYPE=‘R’, is performed on a large table, it may be better to build an index on some column of that table; EXPLAIN REFERENCE provides hints about which indexes might help. Adding new indexes makes updates more expensive, so this decision must be considered carefully.

The PLAN_TABLE can also help the administrator determine which indexes are *not* being used, so that he or she may decide which indexes might be dropped. This assumes that the administrator knows not only the significant statements in programs, but also the significant statements issued by users directly at their terminals.

Modifying Table Designs to Enhance Performance

The primary consideration for the performance of access to data in the database is the number of DASD input/output requests required to access the table rows. The indexing and clustering techniques discussed in the previous sections enhance data access performance by reducing DASD input/output requests without impacting your logical data (table) design. However, other techniques can be used, if you are willing to reconsider your logical table design.

Keeping Together Frequently Updated Columns: Keep frequently updated columns close together in the same row. This helps to reduce the amount of data that has to be logged because only the portion of the row from the first column updated to the last column updated is recorded in the log.

Storing Joins of Tables (Redundant Data): The evaluation of a join of two tables involves combining information from corresponding rows of the tables. Ideally, the corresponding rows are on the same page. However, such clustering of rows from separate tables is difficult to establish and maintain. Then, such clustering may not be in the best interest of query accesses to the individual tables. Assuming tables are clustered on different pages, multiple input/output requests are required to evaluate the join of corresponding rows of the two tables.

For example, suppose the PROJECT table is clustered on DEPTNO. Retrieving information about a department (a DEPARTMENT row) and its corresponding project name (PROJECT rows), involves an access to one page to get the DEPARTMENT row and another access to a different page to get the corresponding PROJECT rows.

If most of such joins are done just to pick up the DEPTNAME information out of the DEPARTMENT row, it may be worthwhile to store DEPTNAME in both tables. This eliminates the need to join the two tables to obtain department names (DEPTNAMEs) in retrieval of project names. This, in turn, eliminates accesses to the DEPARTMENT table pages. This, in effect, reduces database input/output requests by “storing the join” of the two tables.

You can use the ISQL INSERT with Subselect (Format 2) to combine tables.

The extreme case of “storing joins” is to replace both tables with the complete join (SELECT *) of the two tables. This is rarely cost effective.

The cost of storing joins is the DASD space consumed and the extra activity required to maintain the redundant data. Table 10 shows the cost of storing the DEPTNAME column in the PROJECT table.

Table 10. Cost of Storing DEPTNAME in the PROJECT Table

Cost Factor	Approximate Cost
DASD Storage	The average column length of the DEPTNAME values times the number of rows in PROJECT (approximately 20-bytes per PROJECT row) or approximately a 35% increase in the average row length for PROJECT
INSERT into PROJECT	This requires the application to first access the PROJECT table to obtain the DEPTNAME for inclusion in the PROJECT row.
UPDATE of DEPTNAME	This requires an update to the PROJECT table (but this is not a frequent operation).
DELETE of a PROJECT	No extra cost.
DELETE of a DEPTNAME	No extra cost.

The cost of storing the DEPTNAME information redundantly is not excessive when compared to the input/output cost for frequent selecting of department names with queries on project information. The cost of DASD storage looks high (about a 35% increase in the size of the PROJECT table). However, a dbspace page will still hold about 140 PROJECT rows. Because most departments have less than 140 project names, it is still reasonable to expect all project names for a department to be on the same page for most departments.

Storing a Logical Table as Two Tables: Another way to reduce the number of pages occupied by a table is to reduce the size of the table. There is not much you can do to make a table smaller than it really is; however, you can achieve a similar effect by separating frequently used columns from the infrequently used columns.

You can do this by splitting the table into two (or more) tables. One table would contain the frequently used columns and the other(s) would contain the infrequently used columns. You could then cluster the rows of frequently used columns on a fewer number of pages.

The cost of splitting a table is the overhead added to queries that need all columns (which, by definition, is infrequent). Splitting a table also produces redundant data. That is, both (all) tables would have to contain the necessary column(s) to support the join.

Bibliography

This bibliography lists publications that are referenced in this manual or that may be helpful.

DB2 Server for VM Publications

- *DB2 Server for VM Application Programming*, SC09-2661
- *DB2 Server for VM Database Administration*, SC09-2654
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2663
- *DB2 Server for VM Diagnosis Guide and Reference*, LC09-2672
- *DB2 Server for VSE & VM Overview*, GC09-2806
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2674
- *DB2 Server for VM Master Index and Glossary*, SC09-2666
- *DB2 Server for VM Messages and Codes*, GC09-2664
- *DB2 Server for VSE & VM Operation*, SC09-2668
- *DB2 Server for VSE & VM Quick Reference*, SC09-2670
- *DB2 Server for VM System Administration*, SC09-2657

DB2 Data Spaces Support Publications

- *DB2 Server Data Spaces Support for VM/ESA*, SC09-2675

DB2 Server for VSE Publications

- *DB2 Server for VSE Application Programming*, SC09-2662
- *DB2 Server for VSE Database Administration*, SC09-2655
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2663
- *DB2 Server for VSE Diagnosis Guide and Reference*, LC09-2673
- *DB2 Server for VSE & VM Overview*, GC09-2806
- *DB2 Server for VSE Installation*, GC09-2656
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2674
- *DB2 Server for VSE Master Index and Glossary*, SC09-2667
- *DB2 Server for VSE Messages and Codes*, GC09-2665
- *DB2 Server for VSE & VM Operation*, SC09-2668
- *DB2 Server for VSE System Administration*, SC09-2658
- *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2669
- *DB2 Server for VSE & VM SQL Reference*, SC09-2671

Related Publications

- *DB2 Server for VSE & VM Data Restore*, SC09-2677
- *DRDA: Every Manager' s Guide*, GC26-3195
- *IBM SQL Reference, Version 2, Volume 1*, SC26-8416
- *IBM SQL Reference*, SC26-8415

VM/ESA Publications

- *VM/ESA: General Information*, GC24-5745
- *VM/ESA: VMSES/E Introduction and Reference*, GC24-5837
- *VM/ESA: Installation Guide*, GC24-5836
- *VM/ESA: Service Guide*, GC24-5838
- *VM/ESA: Planning and Administration*, SC24-5750
- *VM/ESA: CMS File Pool Planning, Administration, and Operation*, SC24-5751
- *VM/ESA: REXX/EXEC Migration Tool for VM/ESA*, GC24-5752
- *VM/ESA: Conversion Guide and Notebook*, GC24-5839
- *VM/ESA: Running Guest Operating Systems*, SC24-5755
- *VM/ESA: Connectivity Planning, Administration, and Operation*, SC24-5756
- *VM/ESA: Group Control System*, SC24-5757
- *VM/ESA: System Operation*, SC24-5758
- *VM/ESA: Virtual Machine Operation*, SC24-5759
- *VM/ESA: CP Programming Services*, SC24-5760
- *VM/ESA: CMS Application Development Guide*, SC24-5761
- *VM/ESA: CMS Application Development Reference*, SC24-5762
- *VM/ESA: CMS Application Development Guide for Assembler*, SC24-5763
- *VM/ESA: CMS Application Development Reference for Assembler*, SC24-5764

- *VM/ESA: CMS Application Multitasking*, SC24-5766
- *VM/ESA: CP Command and Utility Reference*, SC24-5773
- *VM/ESA: CMS Primer*, SC24-5458
- *VM/ESA: CMS User’s Guide*, SC24-5775
- *VM/ESA: CMS Command Reference*, SC24-5776
- *VM/ESA: CMS Pipelines User’s Guide*, SC24-5777
- *VM/ESA: CMS Pipelines Reference*, SC24-5778
- *VM/ESA: XEDIT User’s Guide*, SC24-5779
- *VM/ESA: XEDIT Command and Macro Reference*, SC24-5780
- *VM/ESA: Master Index and Glossary*, SC09-2398
- *VM/ESA: Quick Reference*, SX24-5290
- *VM/ESA: Performance*, SC24-5782
- *VM/ESA: Dump Viewing Facility*, GC24-5853
- *VM/ESA: System Messages and Codes*, GC24-5841
- *VM/ESA: Diagnosis Guide*, GC24-5854
- *VM/ESA: CP Diagnosis Reference*, SC24-5855
- *VM/ESA: CP Diagnosis Reference Summary*, SX24-5292
- *VM/ESA: CMS Diagnosis Reference*, SC24-5857
- *VM/ESA: CMS Data Areas and Control Blocks*, SC24-5858
- *VM/ESA: CP Data Areas and Control Blocks*, SC24-5856
- *IBM VM/ESA: CP Exit Customization*, SC24-5672
- *VM/ESA REXX/VM User's Guide*, SC24-5465
- *VM/ESA REXX/VM Reference*, SC24-5770

C for VM/ESA Publications

- *IBM C for VM/ESA Diagnosis Guide*, SC09-2149
- *IBM C for VM/ESA Language Reference*, SC09-2153
- *IBM C for VM/ESA Compiler and Run-Time Migration Guide*, SC09-2147
- *IBM C for VM/ESA Programming Guide*, SC09-2151
- *IBM C for VM/ESA User's Guide*, SC09-2152

Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA) Publications

- *IBM VSE/ESA Administration*, SC33-6505
- *IBM VSE/ESA Diagnosis Tools*, SC33-6514
- *IBM VSE/ESA General Information*, GC33-6501
- *IBM VSE/ESA Guide for Solving Problems*, SC33-6510
- *IBM VSE/ESA Guide to System Functions*, SC33-6511
- *IBM VSE/ESA Installation*, SC33-6504
- *IBM VSE/ESA Messages & Codes*, SC33-6507
- *IBM VSE/ESA Networking Support*, SC33-6508
- *IBM VSE/ESA Operation*, SC33-6506
- *IBM VSE/ESA Planning*, SC33-6503
- *IBM VSE/ESA System Control Statements*, SC33-6513
- *IBM VSE/ESA System Macros User's Guide*, SC33-6515
- *IBM VSE/ESA System Macros Reference*, SC33-6516
- *IBM VSE/ESA System Utilities*, SC33-6517
- *IBM VSE/ESA Unattended Node Support*, SC33-6512
- *IBM VSE/ESA Using IBM Workstations*, SC33-6509

CICS/VSE Publications

- *CICS/VSE Application Programming Reference*, SC33-0713
- *CICS/VSE Application Programming Guide*, SC33-0712
- *CICS Application Programming Primer (VS COBOL II)*, SC33-0674
- *CICS/VSE CICS-Supplied Transactions*, SC33-0710
- *CICS/VSE Customization Guide*, SC33-0707
- *CICS/VSE Facilities and Planning Guide*, SC33-0718
- *CICS/VSE Intercommunication Guide*, SC33-0701
- *CICS/VSE Performance Guide*, SC33-0703
- *CICS/VSE Problem Determination Guide*, SC33-0716
- *CICS/VSE Recovery and Restart Guide*, SC33-0702
- *CICS/VSE Release Guide*, GC33-0700
- *CICS/VSE Report Controller User's Guide*, SC33-0705
- *CICS/VSE Resource Definition (Macro)*, SC33-0709
- *CICS/VSE Resource Definition (Online)*, SC33-0708
- *CICS/VSE System Definition and Operations Guide*, SC33-0706
- *CICS/VSE System Programming Reference*, SC33-0711
- *CICS/VSE User's Handbook*, SX33-6079
- *CICS/VSE XRF Guide*, SC33-0704

CICS/ESA Publications

- *CICS/ESA General Information*, GC33-0155

VSE/Virtual Storage Access Method (VSE/VSAM) Publications

- *VSE/VSAM Commands and Macros*, SC33-6532
- *VSE/VSAM Introduction*, GC33-6531
- *VSE/VSAM Messages and Codes*, SC24-5146
- *VSE/VSAM Programmer's Reference*, SC33-6535

VSE/Interactive Computing and Control Facility (VSE/ICCF) Publications

- *VSE/ICCF Administration and Operation*, SC33-6562
- *VSE/ICCF Primer*, SC33-6561
- *VSE/ICCF User's Guide*, SC33-6563

VSE/POWER Publications

- *VSE/POWER Administration and Operation*, SC33-6571
- *VSE/POWER Application Programming*, SC33-6574
- *VSE/POWER Installation and Operations Guide*, SH12-5329
- *VSE/POWER Networking*, SC33-6573
- *VSE/POWER Remote Job Entry*, SC33-6572

Distributed Relational Database Architecture (DRDA) Library

- *Application Programming Guide*, SC26-4773
- *Architecture Reference*, SC26-4651
- *Connectivity Guide*, SC26-4783
- *DRDA: Every Manager' s Guide*, GC26-3195
- *Planning for Distributed Relational Database*, SC26-4650
- *Problem Determination Guide*, SC26-4782

C/370 for VSE Publications

- *IBM C/370 General Information*, GC09-1386
- *IBM C/370 Programming Guide for VSE*, SC09-1399
- *IBM C/370 Installation and Customization Guide for VSE* GC09-1417
- *IBM C/370 Reference Summary for VSE*, SX09-1246
- *IBM C/370 Diagnosis Guide and Reference for VSE* LY09-1805

VSE/REXX Publication

- *VSE/REXX Reference*, SC33-6642

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
- *VM/Directory Maintenance Licensed Program Operation and User Guide Release 4*, SC23-0437
- *IBM Distributed Relational Database Architecture Reference*, SC26-4651
- *IBM Systems Network Architecture, Format and Protocol*
- *SNA LU 6.2 Reference: Peer Protocols*
- *Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM Systems Network Architecture, Logical Unit 6.2 Reference: Peer Protocols*
- *Distributed Data Management (DDM) List of Terms*
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526

CCSID Publications

- *Character Data Representation Architecture, Executive Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

C/370 Publications

- *IBM C/370 Installation and Customization Guide*, GC09-1387
- *IBM C/370 Programming Guide*, SC09-1384

Communication Server for OS/2 Publications

- *Up and Running!*,GC31-8189
- *Network Administration and Subsystem Management Guide* SC31-8181
- *Command Reference*, SC31-8183
- *Message Reference*, SC31-8185
- *Problem Determination Guide*, SC31-8186

Distributed Database Connection Services (DDCS) Publications

- *DDCS User's Guide for Common Servers*, S20H-4793
- *DDCS for OS/2 Installation and Configuration Guide* S20H-4795

VTAM Publications

- *VTAM Messages and Codes*, SC31-6493
- *VTAM Network Implementation Guide*, SC31-6494
- *VTAM Operation*, SC31-6495
- *VTAM Programming*, SC31-6496
- *VTAM Programming for LU 6.2*, SC31-6497
- *VTAM Resource Definition Reference*, SC31-6498
- *VTAM Resource Definition Samples*, SC31-6499

CSP/AD and CSP/AE Publications

- *Developing Applications*, SH20-6435
- *CSP/AD and CSP/AE Installation Planning Guide*, GH20-6764
- *Administering CSP/AD and CSP/AE on VM*, SH20-6766
- *Administering CSP/AD and CSP/AE on VSE*, SH20-6767
- *CSP/AD and CSP/AE Planning*, SH20-6770
- *Cross System Product General Information*, GH23-0500

Query Management Facility (QMF) Publications

- *QMF General Information*, GC26-4713
- *QMF VSE/ESA Setup and Usage Guide*, GG24-4196
- *Managing QMF for VSE/ESA*, SC26-3252
- *Installing QMF on VSE/ESA*, SC26-3254
- *QMF Learner's Guide*, SC26-4714
- *QMF Advanced User's Guide*, SC26-4715
- *QMF Reference*, SC26-4716
- *Installing QMF on VM*, SC26-4718
- *QMF Application Development Guide*, SC26-4722
- *QMF Messages and Codes*, SC26-4834
- *Using QMF*, SC26-8078
- *Managing QMF for VM/ESA*, SC26-8219

DL/I DOS/VS Publications

- *DL/I DOS/VS Application Programming*, SH24-5009

COBOL Publications

- *VS COBOL II Migration Guide for VSE*, GC26-3150
- *VS COBOL II Migration Guide for MVS and CMS*, GC26-3151
- *VS COBOL II General Information*, GC26-4042
- *VS COBOL II Language Reference*, GC26-4047
- *VS COBOL II Application Programming Guide*, SC26-4045
- *VS COBOL II Application Programming Debugging*, SC26-4049
- *VS COBOL II Installation and Customization for CMS* SC26-4213
- *VS COBOL II Installation and Customization for VSE* SC26-4696
- *VS COBOL II Application Programming Guide for VSE* SC26-4697

Data Facility Storage Management Subsystem/VM (DFSMS/VM) Publications

- *DFSMS/VM User's Guide*, SC26-4705

Systems Network Architecture (SNA) Publications

- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *SNA Format and Protocol Reference: Architecture Logic for LU Type 6.2*, SC30-3269
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *SNA Synch Point Services Architecture Reference* SC31-8134

Miscellaneous Publications

- *IBM 3990 Storage Control Planning, Installation, and Storage Administration Guide*, GA32-0100
- *Dictionary of Computing*, ZC20-1699
- *APL2 Programming: Using Structured Query Language*, SH21-1056
- *ESA/390 Principles of Operation*, SA22-7201

Related Feature Publications

- *Control Center Installation and Operations Guide for VM*, GC09-2678
- *Control Center Installation and Operations Guide for VSE*, GC09-2679
- *IBM Replication Guide and Reference*, S95H-0999

Index

Special Characters

***IDENT 29**

Numerics

16MB line
- storage above 47
- storage queue 46
- virtual addressability extension 84

31 bit addressing
- advantages 47
- storage queue 46
- virtual addressability extension 84

31-bit addressing 27

A

absolute share 83
access path
- choosing 145
- dbspace scan 126
- disadvantages of indexes 129
- index scan 126
- index-only access 127
- influencing 125
- locking 104
- types 125, 127
- unique index with key matching predicate 128
- with unclustered index 126

accounting
- measurement tool 9

ACQUIRE DBSPACE
- allocating dbspace storage 68
- minimum lock level 103

ADD 51
address space
- data space 48
- primary space 48
- size 47
- virtual addressability extension 84
- virtual disk 50, 56

addressability extension
- virtual 84

addressing 43
adhoc query
- isolation level 121
- temporary table 121
- view 121

agent
- checkpoint 95
- deprived 100
- dispatching 99

agent *(continued)*
- operator 95
- pseudo 97
- real 95
- structure
  - virtual addressability extension 84
- user 95

allocating users to agents 96
ALTER DBSPACE
- free space in a data page 64
- lock escalation 107

AMXT/MXT
- CICS 84

analyzing
- SQL statements 151

application program
- adhoc query 120
- DBS utility considerations 121
- deadlock 109
- distributed database 87
- ISQL considerations 120
- response time 5

application requester
- configuring 89
- DRDA 86
- PROTOCOL parameter 116

application server
- configuring 89
- DRDA 86
- PROTOCOL parameter 116
- response time 4

archive
- checkpoint 111
- database 51, 55, 57, 60
- log 113
- selective 113

archiving
- as overhead 12

ARCHPCT
- tuning parameter 115

ARIS41DB 20
ARISQLDS 20
arithmetic operator
- in syntax diagrams xiii

asynchronous
- communication 120
- page fault processing 93
- writes 94

auditing
- fair share 100

AUTO PROTOCOL option 117

AUTOCOMMIT
- command 120
- ISQL 120

automatic statistics collection 121
- suppressing 122

auxiliary storage
- CICS temporary 85
- expanded storage 43
- system paging DASD 43

availability 1, 5
average row length 39
AVGROWLEN 39

B

back-up
- file 52
- virtual disk 54
- VSAM 79

balancing
- DASD 79

batch user, VSE 96
BEGINLUW counter 23
BLOCK
- initialization parameter 118

blocking 93
- disadvantages 119
- DRDA protocol 118
- fetch and insert 118
- ISQL 119
- maximum row length 118
- single user mode (SUM) 119
- suppressed 119

buffer pool
- checkpoint 111
- data page 89
- DB2 Server DSS 92
- NCUSERS 91, 96
- size 91

C

caching
- minidisk 82
- package 94
  - NCUSERS 96

CANCEL
- ISQL 120
- synchronous communication 120

catalog
- procedure
  - virtual disk 53
- table 38

catalog table
- automatic statistics collection 121
- column information 39

catalog table *(continued)*
- dbspace information 40
- description 9
- index information 41
- locking 106, 121
- placement 81
- statement 147
- statistics 147
  - influencing data access 149
  - updating 149
- SYSTEM.SYSCATALOG 38, 65
- SYSTEM.SYSCOLUMNS 39
- SYSTEM.SYSDBSPACES 40, 64, 65
- SYSTEM.SYSINDEXES 41
- table information 38
- UPDATE STATISTICS 147

CCSID
- CHARNAME 142
- column 40
- conversion
  - PROTOCOL 117
- DRDA handshaking 117
- impact on sargability 141

CCW
- fast translation 83

channel 79
CHARNAME
- CCSID 142

checkpoint
- agent 95
- as overhead 12
- forcing 67, 112
- interval 111
- invalid index 78
- load measurement 23
- occurrence 111
- overview 110
- performance implication 111
- process 67
- shadow page 66, 111
- short on storage 62
- storage queue 46

CHKINTVL
- checkpoint 111
- choosing a value 111

CHKPOINT counter 23
choose
- in syntax diagrams xiv

CICSPARS 8, 18
CIRB
- initiated CICS connections
  - concurrency 99
- transaction 97

CIRD transaction 8, 22
CIRT transaction 97

CISQ
  transaction name 84
cluster ratio
  catalog table 42
  evaluating 71
clustered
  catalog table 42
  index 70
    access path 126
  unclustered index 71
clustering index 70
  catalog table 42
  changing
    while reorganizing a dbspace 75
    while reorganizing a table 74
    without dropping indexes 76
  data 70
  PCTFREE 65
  reorganize index 77
  unclustered 71
  view 72
CLUSTERTYPE
  clustering 70
column
  > 254 119
  catalog information 39
  length 40
  null type 40
  type 40
COMMIT WORK
  checkpoint 111
  ISQL 120
  lock escalation 107
  log cushion 115
  statement 110
COMMITCOUNT
  locking 122
communication
  blocking 118
  DRDA protocol 116
  synchronous 120
compatibility
  lock 102
compile partition size 84
concurrency
  agent 95
  allocating agent structure 96
  as overhead 11
  CICS 97
  CICSPARS 97
  CIRB transaction 97
  concurrent ISQL users 85
  cost of additional real agents 96
  description 95
  differences between real and pseudo agents 98
  dispatching agents 99
concurrency *(continued)*
  DISPBIAS 100
  fair share auditing 100
  guest sharing 98
  long versus short LUW 100
  MAXCONN 98
  NCUSERS 96
  operator SHOW commands 28
  prioritization 100
  privileged remote DRDA user 99
  pseudo-agents 97
  RMTUSERS 99
  SHOW CONNECT 97
  SHOW USERS 97, 99
  startup mode 100
  VM user 96
  VSE batch user 96
  VSE interactive user 96
  VSE remote DRDA user 96
conditional JCL
  startup for virtual disk 55
configuration
  consideration, distributed 86
console log 21
constraint
  removing 12
controller 79
conventions
  example xvii
  syntax diagram notation xii
Conversational Monitor System (CMS)
  work unit 110
cost
  estimating
    obtaining 162
  explain table 155
  of monitoring 7
cost-benefit ratio 1
COUNTER
  CP indicate user
    I/O 16
    READS 16
    RES 16
    WRITES 16
  CP query time
    CTIME 16
    TOTCPU 16
    VTIME 16
  example 23
  operator command 9, 23
COUNTER POOL
  DB2 Server DSS 10
CP
  monitor 8
  time 16

CP QUERY TIME 8
CPU
as a resource 10
load measurement 15, 17
time used 30
CREATE INDEX
reorganize a single table 75
CREATE PROGRAM
blocking 118
CSTT transaction 19
CTIME counter 16
cursor
fetch 118
insert 118
stability (CS)
isolation level 104
ISQL 121
minimum lock level 103
suppressed blocking 119
cushion
log 115
Customer Information Control System (CICS)
AMXT/MXT 84
CICS partition
31 bit addressing 84
CCW translation 83
connection 96
dispatching priority 83
system paging DASD 45
virtual addressability extension 84
CICS statistics 8
CICSPARS 8, 18, 97
CIRB transaction 97
CIRD transaction 8, 22
CMXT parameter 85
concurrent ISQL users 85
CSTT transaction 19
DFHMCT ID keywords and clocks 19
DFHMCT macro 19
DFHSIT macro 19
link to database partition 97
measurement tools 8
online resource adapter 98
pseudo agent 99
temporary storage 85
transaction time usage 19

D
DASD I/O
*BLOCKIO 46
as a resource 10
balancing 79
channel 79
checkpoint 111
controller 79
DASD I/O *(continued)*
DASD I/O system 89
DB2 Server DSS 92
even distribution 93
general consideration 81
load measurement 15, 18
NCUSERS 96
package cache 94
page format 61
VSAM 46
DASD storage
as a resource 10
CMS reserved minidisk 61
cost of monitoring 7
dbextent placement in a volume 82
distributing 61
dividing 61
failure 113
running out 61, 68
VSAM entry sequenced data sets (ESDS) 61
wasting 61
DASDIO counter 23
DASDREAD counter 23
DASDWRIT counter 23
data clustering 70
data location
physical 20
data page
description 63
free space 64
proportion to other pages 64
data set placement (VSE) 20
data space
DB2 Server DSS 48
virtual disk 50, 56
database
archive 113
design
using EXPLAIN 164
generation
MAXPOOLS keyword 62
machine
31 bit addressing 47
address space size 47
auxiliary storage 43
buffer pool 89
CPU load 17
DASD I/O load 15
fair share scheduling 82
load on 23
overall performance 16
SHOW STORAGE 27
virtual storage load 15
manager
storage 89

database partition
31 bit addressing 47, 84
address space size 47
auxiliary storage 44
balancing pool 83
batch 83
buffer pool 89
CICS
dispatching priority 83
fast CCW translation 83
virtual addressability extension 84
compile size 84
DASD I/O load 15
deactivation 46
load on 23
overall performance 16
SHOW STORAGE 27
DATALOAD
automatic statistics collection 121
clustering index 70
free page space 41
locking 122
PCTFREE 65
DATAUNLOAD
locking 123
date
ever increasing index 69
DB2 for VM Control Center Installation and Operations Guide, IBM 9
DB2 Server DSS
asynchronous processing 93
asynchronous writes 94
blocking 93
checkpoint 113
counters 25
dynamic storage management 94
increasing virtual storage 48
paging system 92
path length 93
prefetching 93
save interval 94
striping 93
target working storage size 94
working storage residence priority 94
dbextent
adding 68
allocation to dbspace 80
caching 82
consecutive placement 82
mapping to dbspace 62
moving 79
restructuring 81
splitting 81
storage 26
virtual disk 53, 58
DBNAME directory 99
DBS utility (Database Services utility)
automatic statistics collection 121
lock escalation 122
performance considerations 121
reorganize 77
REORGANIZE INDEX 77
unload and reload package 123
UNLOAD block size 122
dbspace
active pages 41
adding 68
available pages 40
catalog information 40
DROP
checkpoint 111
dropping 62
dropping for force a checkpoint 67
extending 68
forcing checkpoint 112
full 68
over allocating 68
shadow page 69
LOCK DBSPACE statement 122
lockmode 41
mapping to dbextent 62
move 80
pages for indexes 41
PCTFREE 65
percentage of page free 41
private
lock 103
reassign 80
releasing empty pages 62
reorganize 68, 75
scan 126
minimum lock level 103
reorganize index 79
row level locking 105
storage pool 41
table placement 129
DBSSCALL counter 23
DDR 79
deactivation of partition 46
DEADLCK counter 23
deadlock
application design 109
COUNTER 110
example 108
hit ratio 24
lock contention 109
lock escalation 110
NCUSERS 96, 109
overview 108
performance indicator 24, 110
reschedule applications 109

deadlock *(continued)*
rollback 110
time required to detect 109
DEDICATE parameter 52
default
in syntax diagrams xv
deprived agent 100
design
data 151
database 164
evaluation 151
modifying tables 165
device utilization
load measurement 15
DFHMCT CICS macro 19
DFHSIT
macro instruction for CICS
AMXT/MXT 84
CICSPARS 19
diminishing returns, law of 1
directory
buffer
DASD I/O system 89
NDIRBUF 91
performance indicator 24
caching 82
checkpoint 111
page map table 62
placement 82
solutions 9
DIRREAD counter 23
DIRWRITE counter 23
disk locations, VM 20
dispatching
agents 99
fair share auditing 100
prioritization 100
priority (VSE) 83
SET QUICKDSP 83
DISPBIAS
agent dispatching 100
distributed configuration
application planning 87
performance implication 86
VM 86
VSE 86
distributing
DASD 61
dividing
DASD 61
processor time 82, 83
DRDA protocol
blocking 118
distributed consideration 86
DRDA option 118
fetch and insert blocking 87
DRDA protocol *(continued)*
handshaking 117
performance considerations 116
PROTOCOL parameter 117
DRDA user
privileged remote 99
RMTUSERS 99
VSE pseudo agent 96
DROP DBSPACE
checkpoint 111
command 129
dual log 113
duration
lock 102
dynamic
page allocation 62
storage size management 94

E
elapsed time 4, 16
elements of performance 1
empty log
invalid index 78
engineering change (EC) 2
ESCALATE 23
counter 38, 108
estimating
size of response 164
evaluating
logical data design 151
example
conventions xvii
exclusive lock
description 101
expanded storage
auxiliary storage 43
buffer pool 89
DB2 Server DSS 49
EXPLAIN
cost table 155
database design 164
explicit and implicit 153
plan table 156
reference table 157
referential constraint 159
statement 151
structure table 159
explanation table
general description 151
using 151, 154
extension
virtual addressability 84

F

fair share
  auditing 100
  scheduling 82
fallback procedures 2
fast CCW translation 83
FASTTR
  job control option 83
fault
  page 45
fetch blocking 118
  distributed database 87
filter factor 140
fold file
  distributed database 87
forcing
  a checkpoint 67
fragment of syntax
  in syntax diagrams xv
fragmentation
  index 76

G

guest sharing
  concurrency 98
  distributed database 86
guidelines
  tuning 1

H

handshaking
  DRDA 117
hardware
  problems 2
  upgrading 2
header page
  description 63
  number 63
  proportion to other pages 64
hierarchy of locks 101
HIGHSTOR 28
hit ratio
  deadlock 24
  description 7
  directory buffer 24
  local buffer 24
  waitlock 24
hold file
  distributed database 87
host variable
  in syntax diagrams xiii

I

I/O
  cost 150
  counter 16
  DASD balancing 79
  DASD I/O as a resource 10
  DASD I/O system 89
  DB2 Server DSS 92
  fast CCW translation 83
  general consideration 81
  Wait State 94
IBM DB2 for VM Control Center 9
IDCAMS
  backup 52
  dataset location 21
improvement process
  performance 2
index
  access path 125
  catalog information 41
  causes of invalid 79
  cluster ratio 42
  clustered 42, 70
  clustering 42, 70
    default versus clustering 70
    unclustered 73
  clustering strategy 70
  creating 128
  default strategy 70
  disadvantages 129
  ever increasing 69
  foreign key 130
  fragmentation 76
  free space
    catalog table 42
    fragmentation 76
  fully matched 133
  index-only access 127
  invalid 78
  lock contention 104
  lockmode 42
  page
    description 63
    free space 66
    proportion to other pages 64
    reserved in a dbspace 41
    running out 69
  PCTFREE 76
  recommendation 129
  release level 42
  reorganize 77
    invalid 79
  scan 126
    minimum lock level 103
  sorting 128

index *(continued)*
transient 78
unclustered
identifying 71
unique
access path type 128
row level locking 104
with key matching predicate 128
INDICATE USER
command 16
description 8
example 16
indicators
performance 2
indicators, performance 4
individual device utilization
load measurement 15
influencing data access
with catalog statistics 149
initialization parameter
BLOCK 118
CHKINTVL 111
LOGMODE 111
NCUSERS 95, 96
NDIRBUF 89
NLRBS 107
NLRBU 107
NPACKAGE 95
NPACKPCT 95
NPAGBUF 89
PCTFREE 66, 75
PROTOCOL 116
retrieval 9, 21
inner table 143
insert
blocking 87, 118
distributed database 87
multiple row 118
INSERT logic 70
intent
exclusive lock
description 102
none lock
description 102
share lock
description 101
interactive
interface
VSE 8
user
VSE 96
Interactive Structured Query Language (ISQL)
asynchronous communication 120
AUTOCOMMIT 120
blocking 119
CANCEL 120
Interactive Structured Query Language (ISQL) *(continued)*
CICS 84
concurrent users 85
isolation level 121
performance considerations 120
temporary table 121
transaction name 84
view 121
internal dbspace
adding to a virtual disk pool 54
data spaces 81
placement 81
virtual disk 50, 56, 81
interval, monitoring 6
invalid index 78
package 78
SHOW INVALID 78
IPL procedure 51
isolation level
considerations for adhoc users 121
description 104
ISQL 121
lock duration 106
lock escalation 107
selecting 106
user defined 106
ISQ2
transaction name 84
IUCV *BLOCKIO
DASD I/O system 89
page format 61
IUCV conversations 100

J

join
merge scan 144
multiple 146
performance considerations 142
predicate 134
stored 121, 165

K

key-matching 132
keyword
in syntax diagrams xiii

L

law of diminishing returns 1
LDIRBUFF counter 23
length, column 40
limits
system tuning 3

LISTCAT 8, 21
load
  leveling 46
  workload 3
load measurement
  checkpoint 23
  CPU 15, 17
  DASD I/O 18
  database machine or partition DASD I/O load 15
  description 7
  I/O 150
  individual device utilization 15
  logical unit of work 23
  main storage 17
  operating system 15
  processor 15, 17
  real and virtual storage 15
  system paging DASD 15, 17
  tools 15
local
  buffer
    DASD I/O system 89
    new page 67
    NPAGBUF 91
    performance indicator 24
    virtual addressability extension 84
  copy
    distributed database 87
locations
  physical data 20
LOCK
  lock escalation 107
lock contention
  access path 104
  catalog table 106
  deadlock 109
  index 104
  isolation level 104
  lock compatibility 102
  logical unit of work 104
  measurement 36
  minimum lock level 103
  NCUSERS 103
  operator SHOW command 36
  row level locking 104
lock escalation
  ALTER DBSPACE 107
  COUNTER operator command 108
  DBS utility 122
  deadlock 110
  isolation level 107
  LOCK 107
  measurement 38
  NCUSERS 96
  operator SHOW command 38
  overview 107
lock escalation *(continued)*
  SHOW LOCK MATRIX 108
  trading lock level 107
lock request block
  determining number required 108
  insufficient 108
  NLRBS 107
  NLRBU 107
  number in use 38
  performance indicator 24, 108
locking
  as overhead 11
  compatibility 102
  considerations
    reducing lock escalation 122
  contention
    overview 101
    performance indicator 106
  cost of 12
  DBS utility
    DATALOAD and RELOAD 122
    deadlock 123
    UNLOAD PACKAGE and RELOAD PACKAGE 123
  duration 102
    isolation level 106
  hierarchy 101
  lock level 103
  lockmode
    dbspace 41
    index 42
  minimum lock level 130
  mode 101
  NCUSERS 96
  operator SHOW commands 36
  overview 100
  private dbspace 103
  SHARE lock
    DBS utility 123
  virtual addressability extension 84
LOCKLMT
  counter 38
  example 23
log
  archive 113
  caching 82
  checkpoint 111
  cushion 115
  dual 113
  file 113
  full
    checkpoint 111
  invalid index 78
  placement 82
  SHOW LOG 116

logging
as overhead 12
logical data design
evaluation of 151
logical unit of work
load measurement 23
lock duration 102
locking 104
long versus short 100
recovery 110
storage queue 46
logmode
changing 115
initialization parameter 111
selecting 114
short on storage 62
LOGREAD counter 23
LOGWRITE counter 23
LPAGBUFF counter 23
LU 6.2 LUWID
DRDA handshaking 117

M
main storage
buffer pool 91
DB2 Server DSS 49
dedicating 43
load measurement 17
page count 16
temporary CICS 85
virtual addressability extension 84
virtual disk 50, 56
maintenance
statistics 147
manual
organization xi
prerequisite xii
who should use xi
matched index 133
MAXCONN 29
increasing virtual storage 47
pseudo agent 98
MAXPOOLS
keyword 62
measurement
production system 7
relative nature 7
test system 7
to indicators 16
tools 8, 9
catalog table 9
CICS 8
CICS monitoring facility 8
CICS statistics 8
CICSPARS 8
CIRD transaction 8
measurement *(continued)*
tools *(continued)*
COUNTER operator command 9
CP INDICATE USER 8
CP monitor 8
CP QUERY TIME 8
database manager 20
DB2 Server DSS COUNTER POOL 10
DB2 Server DSS SHOW TARGETWS 9
DB2 Server for VSE & VM 9
DB2 Server for VSE & VM accounting 9
DB2 Server for VSE & VM trace 9
IBM DB2 for VM Control Center 9
initialization parameter 9
operating system 15
RTM VM/ESA 8
SHOW operator command 9
VM 8
VM/PRF 8
VMMAP 8
VSAM LISTCAT 8
VSE 8
VSE interactive interface 8
measuring performance 15
merge scan join 144
message traffic
fetch and insert blocking 87
microcode assists 2
minidisk
caching 82
minidisk caching
DB2 Server DSS 93
minimum lock level 103
modifying table designs 165
Monitor Analysis Program (VMMAP) 8
monitor, CP 8
monitoring
cost 7
interval 6
master schedule 6
plan 6
performance 2, 6
real time 6
statistical 6
most recently used pages 45
multiple join 146
multiple user mode (MUM)
allocating user to agents 96
blocking 118
running 100

N
NCUSERS
buffer pool 91
deadlock 109

NCUSERS *(continued)*
determining number of real agents 96
example 29
increasing virtual storage 47
lock contention 103
number of real agents 95
package cache 95
RMTUSERS 99
NDIRBUF
directory buffer 91
directory buffer pool 89
increasing virtual storage 47
nested loop join 143
network
distributed 86
response time 4
SNA 86
NLRBS
increasing virtual storage 47
initialization parameter 107
NLRBU
increasing virtual storage 47
initialization parameter 107
NOFASTTR
job control option 83
nonrecoverable storage pool
checkpoint 111
nonselective index scan 126
NOVERFLOW 39
NPACKAGE
increasing virtual storage 47
package cache 95
NPACKPCT
package cache 95
NPAGBUF
increasing virtual storage 47
local buffer 91
local buffer pool 89
null 40

O

objectives
performance 2, 4
obtaining costs for statements 162
operating
mode
XA 47
XC 47
system measurements 15
operator
command 25, 33
operator agent 95
storage queue 46
optimizer 125
optional
default parameter
in syntax diagrams xv
item
in syntax diagrams xiii
keyword
in syntax diagrams xv
options file, start up 21
outer table 143
overflow
catalog table 65
reorganization 73
row expansion 65
overhead 11

P

package
cache 94
NCUSERS 96
threshold 95
virtual addressability extension 84
invalid 78
unload and reload 123
page
active in a dbspace 41
allocating from storage pool 62
available in a dbspace 40
DASD I/O system 89
data
description 63
free space 64
DB2 Server DSS 49
dbspace
running out 68
dynamic allocation 62
fault
asynchronous 93
system paging DASD 45
free percentage 41
header 63
index 63
reserved in a dbspace 41
running out 69
jumps 71
logical and physical 62
map table 62
shadow page 66
modified 90
most recently used 45
number used by a table 39
releasing 90
shadow
checkpoint 66
description 63
number in use 67

page *(continued)*
size 61
stealing 90
storage 43
type 63
update 66
PAGEREAD counter 23
PAGWRITE counter 23
parentheses
in syntax diagrams xiii
path length 93
PCTFREE
initialization parameter 65
reorganize a dbspace 75, 76
reorganize a single table 74
PCTINDEX
proportion of page types 64
PCTINDX
proportion of page types 64
performance
elements of 1
improvement process 2
indicator 4
measurement
description 7
translating to indicators 16
monitoring plan 2, 6
objectives 2, 4
trade-off 2
performance indicator
AVGROWLEN 65
CICSPARS 97
CIRD transaction 97
COUNTER 91
deadlock 24, 110
directory buffer 24
FREEPCT 65
IPCTFREE 66
local buffer 24
lock contention 106
lock request block 24, 108
lockmode 103
NOVERFLOW 65
operating system 15
PCTINDX 64
performance improvement process 2
SHOW POOL 63
SHOW USERS 99
tools 15
waitlock 24
Performance Reporting Facility, VM 8
permanent dbspace
shadow page 66
physical data locations 20
placement in dbspace 129
plan explain table 156
plan, performance monitoring 2, 6
predicate
CCSID 40
column attribute 132
filter factors 140
join 134
key-matching 132
processing 131
examples 140
residual 134
sargable 134
suitable indexes for 133
prefetching 93
preprocessing
compile partition size 84
prerequisite manual xii
priority
agent prioritization 100
dispatching (VSE) 83
private dbspace
cursor stability 105
privileged remote DRDA user, VSE 99
process
performance improvement 2
processing cost 150
processor
as a resource 10
load measurement 15, 17
time used 30
production workload 3, 7
PROFILE EXEC, modifying 59
program temporary fix (PTF) 2
proportion
of available pages 26
of index to data and header pages 64
proportion of index to data and header pages
PCTINDEX 64
PCTINDX 64
PROTOCOL
AUTO option 117
CCSID conversion 117
DRDA
communications 116
option 117, 118
SQLDS option 117
pseudo agent
description 97
number waiting 97
punctuation mark
in syntax diagrams xiii

Q

QRYBLKSIZE
blocking 118

query
  adhoc
    isolation level 121
    performance considerations 120
    temporary table 121
    view 121
  separating into subquery structures 160
  several tables 142
QUERY TIME
  command 16
  description 8
  example 16
queue, storage 27
quick dispatch
  SET QUICKDSP 83

R

ratio
  cost-benefit 1
  hit 7
RDSCALL counter 23
READS counter 16
real agent
  description 95
  number available 97
  storage queue 46
real storage
  as a resource 10
  description 43
  increasing virtual storage 47
  load measurement 15
Real Time Monitor VM/ESA program, (RTM VM/ESA) 8
real time monitoring 6
recovery
  archive 113
  as overhead 12
  checkpoint 110
  log 113
  logical unit of work 110
  shadow page 66
  storage queue 46
reducing
  lock escalation 122
redundant data 165
reentrant code 48
reference explain table 157
referential structure
  EXPLAIN 159
  organization 130
relative share 83
release level 42
releasing, page 90
RELOAD
  automatic statistics collection 121
RELOAD *(continued)*
  clustering index 70
  free page space 41
  locking 122
  NEW 75
  PCTFREE 65
  PURGE 74
  single table 74
RELOAD PACKAGE
  DBS utility 123
  usage considerations 123
remote, privileged DRDA user 99
reorganize
  all the tables in a dbspace 75
  clustering index 77
  dbspace 68
  index 77
  invalid index 79
  PCTFREE 74, 75
  reasons to reorganize data 73
  selecting a method 74
  single table 74
REORGANIZE INDEX
  advantages 77
  DBS utility command 77
  invalid index 79
repeat symbol
  in syntax diagrams xiv
repeatable read (RR)
  isolation level 104
  ISQL 121
required item
  in syntax diagrams xiii
RES counter 16
reserved words
  SQL xvi
RESET
  example 23
  operator command 23
residual predicate 131, 134
resources 10
response size
  estimating 164
response time
  application program 5
  application server 4
  components 4
  network 4
  performance 1, 4
restore virtual disk backup 55
RINO
  EXPLAIN 159
RMTUSERS
  concurrency 99
ROLLBACK 23

ROLLBACK WORK
checkpoint 111
invalid index 78
ISQL 120
locking 104
statement 110
row
expansion 65
length
blocking 118
level locking
unique index 104
overflow 39
total number 39
ROWCOUNT
catalog table 39
estimating response size 164
running out of
DASD 61

S

sargable
affect of CCSID 141
predicate 131, 134
save
interval 94
saved segments 48
SAVEINTV
DB2 Server DSS 113
scan
dbspace 126
index 126
scheduling
fair share 82
second level guest 86
segments, saved 48
selective
archive 113
index
scan 127
separating queries into subquery structures 160
SET QUICKDSP
fair share scheduling 83
SET SHARE
fair share scheduling 83
shadow page
accumulating 63
checkpoint 112
checkpoint process 111
dbspace full 69
description 63, 66, 111
number in use 67
page map table 66
short on storage 62
share
description 102
lock
description 101
with intent exclusive lock 102
SHARE lock 123
short on storage
checkpoint 111
example 27
initialization parameter 62
managing pool space 62
SHOW POOL 63
SOSLEVEL 62
shorter path length 93
SHOW
operator command 9, 25
SHOW ACTIVE
operator command 28
SHOW CONNECT
concurrency 97
operator command 30, 33, 35
SHOW DBEXTENT
available storage pool space 68
operator command 25
SHOW DBSPACE
operator command 26
SHOW INVALID
invalid index 78
transient index 78
SHOW LOCK
operator command 36
SHOW LOCK ACTIVE
operator command 36
SHOW LOCK GRAPH
operator command 37
SHOW LOCK MATRIX
lock escalation 108
operator command 36, 38
SHOW LOCK USER
operator command 37
SHOW LOG
log cushion 116
SHOW POOL
available dbextent space 68
operator command 26
shadow page 67
short on storage 63
SHOW STORAGE
operator command 27
SHOW TARGETWS
DB2 Server DSS 9
SHOW USERS
concurrency 97
operator command 29
pseudo-agents 99

shutdown
- checkpoint 111

single user mode (SUM)
- allocating users to agents 96
- blocking 119
- fetch and insert blocking 87
- logmode 114
- running 100

size of responses, estimating 164
SLOGCUSH
- tuning parameter 115

SNA network
- distributed database 86

software problems 2
Solutions Directory 9
sorting
- index 128

SQL statements, analyzing 151
SQLCODE
- -649 154
- -911 110
- -912 108
- -915 108

SQLDS protocol 117
SQLHX
- synchronous communication 120

SQLINIT
- PROTOCOL parameter 116
- QRYBLKSIZE 118
- SYNCHRONOUS 120

SQLQRY
- synchronous communication 120

SQLSTART 20
SQLSTATE
- 40001 110
- 42704 154
- 57028 108
- 57029 108

start up
- mode 100
- options file 21

statements
- obtaining costs 162

statistics
- automatic collection 121
- catalog table 38
- CICS 8
- influencing data access 149
- maintenance 147
- monitoring 6
- update all 148

stealing, page 90
storage
- 16MB line 27
- 31-bit addressing 27
- address space size 47

storage *(continued)*
- allocation above 16MB 27
- as a resource 10
- auxiliary 43
- available dbextent storage 26
- available storage pool space 25
- DASD 61
- DASD I/O system 89
- database manager 89
- DB2 Server DSS 48
- operator SHOW commands 25
- package cache 94
- page 43
- paging system 45
- proportion of available pages 26
- queue 27, 46
- real and virtual 43
- saved segments 48
- short on storage 62
- temporary CICS 85
- virtual 27
- virtual disk 50, 56
- virtual machine 43
- virtual partition 43

storage pool
- allocating page from 62
- assign DASD 79
- assigning dbextent 62
- available space 25
- catalog placement 81
- COUNTER POOL, DB2 Server DSS 10
- dbspace 41
- description 62
- directory and log placement 82
- existing pool and virtual disk 53
- free allocated pages 68
- full 68
- internal dbspace placement 81
- managing space 62
- nonrecoverable
  - checkpoint 111
- reassign dbspace 80
- virtual disk 53, 58

stored procedure
- executing 32

stored procedures program 5
storing
- joins of tables 165
- logical table as two tables 166

striping 93
structure explain table 159
subquery structures, determining 160
super exclusive lock
- description 101

supervisor mode
- ESA 47

supervisor mode *(continued)*
VMESA 47
suppressed blocking 119
suppressing
automatic statistics collection 122
SYNCHRONOUS
communication 120
syntax diagram
notation conventions xii
SYSDEF 51
system paging DASD
auxiliary storage 43
buffer pool 89
DB2 Server DSS 48
hidden cost 45
increasing virtual storage 47
load measurement 15, 17
monitoring load 47
NCUSERS 96
saved segments 48
virtual disk 50, 56
SYSTEM.SYSCATALOG
average row length 39, 65
catalog table 38
clustering 70
number of page used by a table 39
overflow rows 39, 65
total number of rows 39
SYSTEM.SYSCOLUMNS
catalog table 39
CCSID 40
column length 40
column type 40
null 40
SYSTEM.SYSDBSPACES
active pages 41
available pages 40
catalog table 40
ever increasing index 69
free space in a data page 41, 65
lockmode 41, 103
percentage of reserved index pages 41, 64
storage pool number 41
SYSTEM.SYSINDEXES
catalog table 41
changing clustering index 71
cluster ratio 42, 71
cluster value 71
clustered 42
clustering 42
free index space 42, 66
lockmode 42
page jumps 71
release level 42
row count 71

T

table
allocation to dbspace 80
average row length 39
catalog
automatic statistics collection 121
information 38
locking 106
design modification 165
explanation 154
LOCK TABLE statement 122
methods of joining 143
moving 80
nested loop join 143
number of pages 39
overflow rows 39
placement in dbspace 129
reorganize 74
total number of rows 39
target working storage size 94
temporary storage
local copy 87
virtual disk 50, 56
test workload 3, 7
thrashing 46
throughput 1, 5
time
components 4
elapsed 4
response 4
tools
measurement 8, 9
catalog table 9
CICS 8
CICS monitoring facility 8
CICS statistics 8
CICSPARS 8
CIRD transaction 8
COUNTER operator command 9
CP INDICATE USER 8
CP monitor 8
CP QUERY TIME 8
database manager 20
DB2 Server DSS COUNTER POOL 10
DB2 Server DSS SHOW TARGETWS 9
DB2 Server for VSE & VM 9
DB2 Server for VSE & VM accounting 9
DB2 Server for VSE & VM trace 9
IBM DB2 for VM Control Center 9
initialization parameter 9
operating system 15
RTM VM/ESA 8
SHOW operator command 9
VM 8
VM/PRF 8
VMMAP 8

tools *(continued)*
  measurement *(continued)*
    VSAM LISTCAT 8
    VSE 8
    VSE interactive interface 8
TOTCPU counter 16
TPN entry
  privileged remote DRDA user 99
trace
  measurement tool 9
trade-off, performance 2
trading
  lock level 107
transaction
  CIRB 97
  CIRD 8, 22
  CIRT 97
  CSTT 19
  name
    ISQL 84
  per second 5
  time, CICS 19
transient index 78
  SHOW INVALID 78
translation
  fast CCW 83
  of performance measurements to indicators 16
TSAF collection
  distributed database 86
tuning
  by levels 2
  guidelines 1
  queries 142
  trade-off 12
tuning parameter
  access path, lock contention 104
  ARCHPCT 115
  CIRB transaction 97
  deadlock 109
  DISPBIAS 100
  index lock contention 104
  isolation level 104
  lock level 103
  logical unit of work, lock contention 104
  MAXCONN 98
  minimum lock level 103
  NCUSERS 91, 95, 103
  NDIRBUF 91
  NLRBS 107
  NLRBU 107
  NPACKAGE 95
  NPACKPCT 95
  NPAGBUF 91
  PCTFREE 65, 66
  PCTINDEX 64
  PROTOCOL 116
tuning parameter *(continued)*
  RMTUSERS 99
  SLOGCUSH 115
  SOSLEVEL 62

U
unclustered index 71
uncommitted read (UR)
  isolation level 105
  ISQL 121
UNDO
  invalid index 78
unique constraint
  invalid index 79
unique index
  invalid index 78
  row level locking 104
  with key matching predicate 128
UNLOAD
  block size 122
  locking 123
UNLOAD PACKAGE
  DBS utility 123
update lock
  description 102
UPDATE STATISTICS
  ALL 148
  automatic statistics collection 121
  clustered index 71
  DBS utility 121
  reorganize a dbspace 76
  reorganize a single table 74
upgrading hardware 2
user agent 95
USER isolation level 106

V
VDISK 51
view 121
virtual
  addressability extension 84
  dbextent
    virtual disk 57
virtual disk
  ADD 51
  adding internal dbspace 54
  backing up 54
  backup file 52
  cataloged procedure 53
  conditional startup JCL 55
  define 51, 57
  existing pools 53
  increasing virtual storage 50, 56
  internal dbspace 50, 56

virtual disk *(continued)*
MDISK 57
number of dbextents 54
procedure to create 50, 56
PROFILE EXEC 59
restore backup 55
storage pool 53, 57, 58
SYSDEF 51
temporary storage 50, 56
VDISK 51
virtual and real dbextents 54
virtual dbextent 53, 57, 58
virtual VSAM cluster 53
VSAM user catalog 52
VSIZE 51
virtual machine
fair share scheduling 82
time 16
virtual storage 27, 43
as a resource 10
blocking 119
load measurement 15
virtual addressing 43
VM Monitor Analysis Program (VMMAP) 8
VM/ESA
31 bit addressing 47
DB2 Server DSS 48, 92
distributed database 86
SET QUICKDSP 83
SET SHARE 83
virtual disk 56
VM/Performance Reporting Facility (VM/PRF) 8
VMMAP 8
VSAM
backup 82
backup and restore 79
cluster
virtual disk 53
DASD I/O system 89
I/O buffer address 83
LISTCAT 8, 21
user catalog
virtual disk 52
VSE interactive interface 8
VSE/ESA
31 bit addressing 47
virtual addressability extension 84
virtual disk 50
VSIZE 51
VTIME counter 16

W

WAITLOCK
example 23
performance indicator 24
waitlock hit ratio 24
wasting
DASD 61
working storage
residence priority 94
size 94
workload
considerations 3
distributing 79
production 3
test 3
throughput 5
WRITES counter 16

X

XPCC conversations 100

Communicating Your Comments to IBM

DB2® for VSE & VM
Performance Tuning Handbook
Version 6 Release 1

Publication No. GC09-2669-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States and Canada: 416-448-6161
  - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
  - Internet: torrcf@ca.ibm.com
  - IBMLink: toribm(torrcf)
  - IBM/PROFS: torolab4(torrcf)
  - IBMMAIL: ibmmail(caibmwt9)

Readers' Comments — We'd Like to Hear from You

DB2® for VSE & VM
Performance Tuning Handbook
Version 6 Release 1

Publication No. GC09-2669-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
GC09-2669-00

Cut or Fold
Along Line

Fold and Tape　　**Please do not staple**　　Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK　ONTARIO　CANADA　M3C 1H7

Fold and Tape　　**Please do not staple**　　Fold and Tape

GC09-2669-00

Cut or Fold
Along Line

File Number: S370/4300-50
Program Number: 5648-A70

GC09-2669-00

Spine information:

DB2® for VSE & VM Performance Tuning Handbook *Version 6 Release 1*