DB2 Server for VSE & VM

Database Administration

Version 7 Release 1

SC09-2888-00

DB2 Server for VSE & VM

Database Administration

Version 7 Release 1

SC09-2888-00

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 275.

First Edition (September 2000)

This edition applies to Version 7 Release 1 Modification 0 of the IBM® DATABASE 2™ Server for VSE & VM Program, (product number 5697-F42) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC09–2654–00 and SC09–2655–00.

Contents

About This Manual

This book describes the tasks for planning and administering an application server in the following environments:

1. Virtual Storage Extended (VSE/ESA), 2.3.1 or above.
2. Virtual Machine/Enterprise Systems Architecture (VM/ESA), 2.3.0 or above
3. VM/ESA with Virtual Storage Extended (VSE) running as a guest under VM and accessing a VM application server.

The planning and administration of a DB2 Server for VSE & VM application server consists of designing, implementing, securing, and maintaining a database. To accomplish these tasks, you must know about:

- Database design
- Table design
- Index creation
- Structured Query Language (SQL)
- Relational concepts.

The first three areas are described in this book. For a description of the other topics, refer to the *DB2 Server for VSE & VM SQL Reference* manual, SC09-2989, and the *DB2 Server for VSE & VM Database Services Utility* manual, SC09-2983.

Note: The *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2987, contains information on database design techniques that you must know before you start to design your database. This information was previously in this manual under the chapter describing advanced database design and performance techniques.

Some Terminology

Throughout this book, the Customer Information Control System **(CICS)** refers to CICS/VSE Version 2 Release 3 or CICS Transaction Server Version 1 Release 1 or later for online support and for ISQL. **DB2 Server for VSE & VM** refers to DATABASE 2 Server for IBM VSE & VM Systems Version 7 Release 1, unless otherwise noted.

Components of the Relational Database Management System

Figure 1 on page viii depicts a typical configuration with one database and two users.

Figure 2 on page ix depicts a typical configuration with one database, one batch partition user, and a CICS® partition with several interactive users.

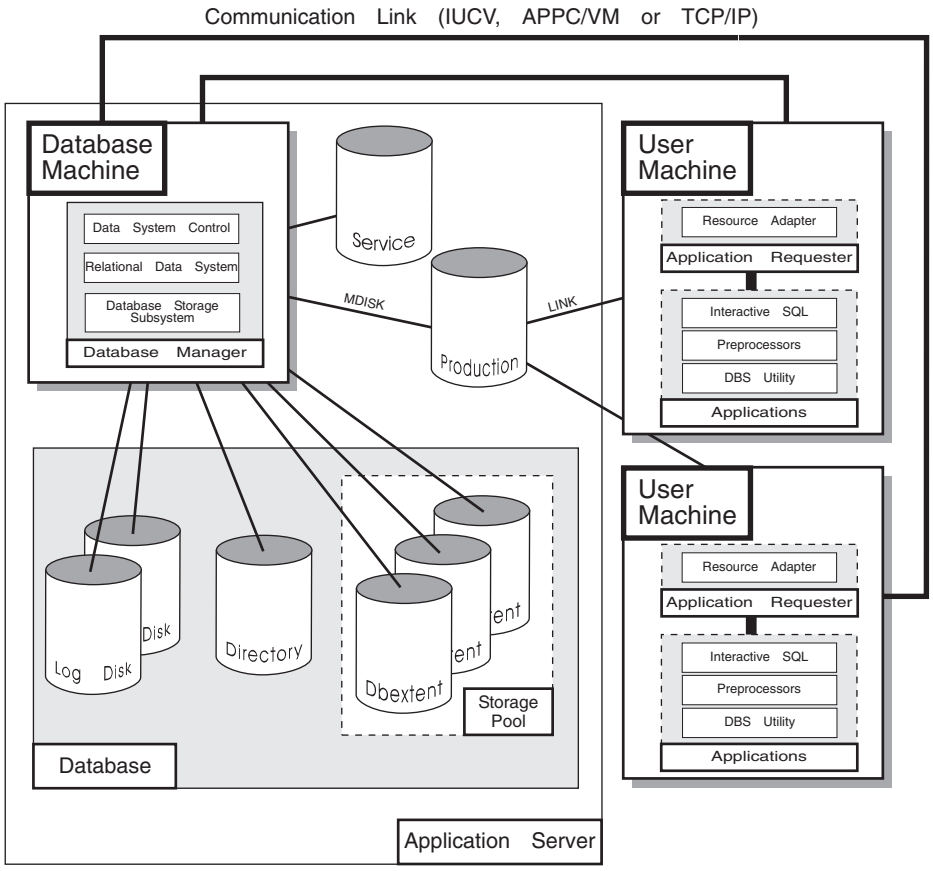


Figure 1. Basic Components of the RDBMS in VM/ESA

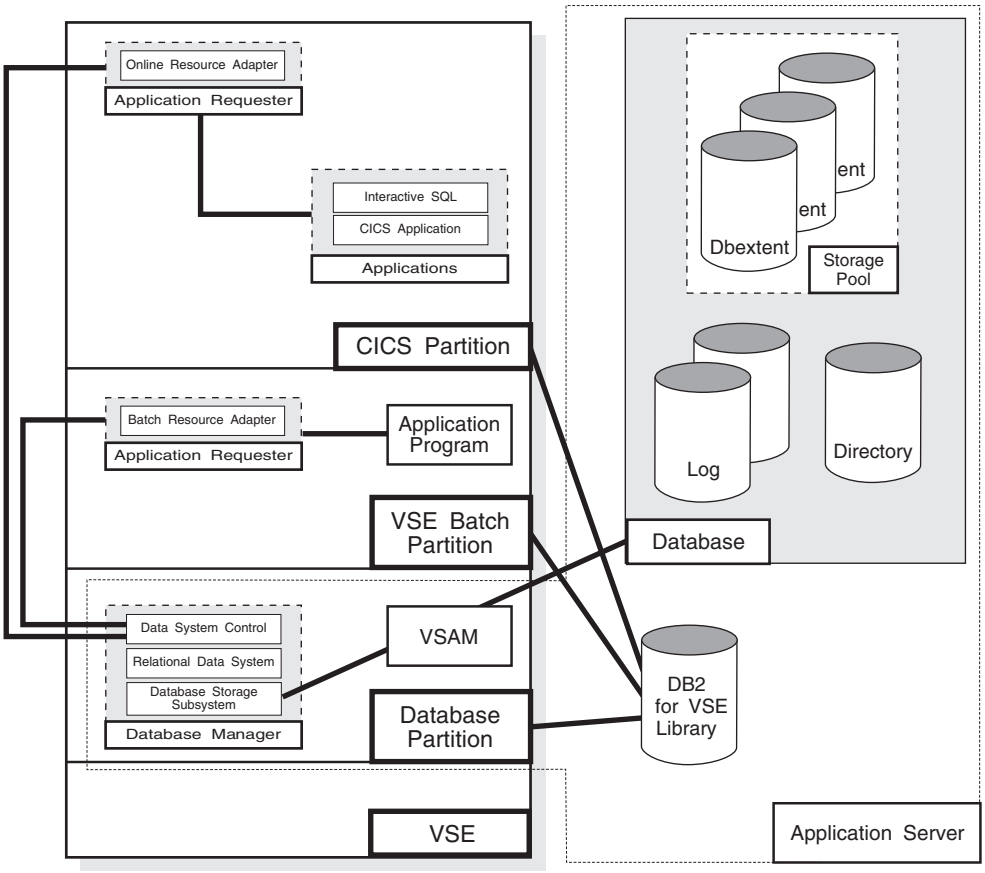


Figure 2. Basic Components of the RDBMS in VSE/ESA

The **database** is composed of :

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

The **database manager** is the program that provides access to the data in the database. In VM it is loaded into the database virtual machine from the production disk. In VSE it is loaded into the database partition from the DB2 Server for VSE library.

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

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

Note: General references to the database management system are assumed to apply to the database under discussion, any unique or specific references to other database systems will be explicitly made.

Organization

Summary of Changes.

This section summarizes the technical and library changes made to the DB2 Server for VSE & VM product for Version 7 Release 1.

Chapter 1. Designing a Database.

To store information in a database, you must first convert it into tables while maintaining any relationships that exist within it. This chapter outlines the steps for effective design of a database.

Chapter 2. Implementing Your Design.

This chapter describes how to estimate your storage requirements, use SQL commands to create objects (dbspaces, tables, views, and indexes) that support your design, and query the catalog tables.

Chapter 3. Maintaining Your Database.

After a database is implemented, it must be maintained. This chapter describes how to load data into tables, alter tables, and alter the design of dbspaces.

Chapter 4. Supporting Your Users.

This chapter describes activities that database administrators must consider to support users. The tasks described include adding, deleting, authorizing, and training users.

Chapter 5. Providing Security.

This chapter describes several security mechanisms that can help you protect your data from unauthorized access.

Chapter 6. Recovering from Failures.

This chapter describes facilities you can use to recover from failures and maintain the integrity of your data.

Chapter 7. Customizing the HELP Text and Messages Text.

This chapter discusses national languages used with the database manager.

Chapter 8. Application Design Considerations.

This chapter provides an overview of the ways that your data can be accessed, and discusses topics that you should consider when developing your applications.

Chapter 9. DB2 Server for VM Database Configurations.

Information can be stored in one or more DB2 Server for VM application server, and these application servers may be on one CPU or distributed among many. Furthermore, users can access an application server on the VM/ESA system from a VSE guest (this is called VSE Guest Sharing). This chapter describes these various types of configurations.

Chapter 10. Usage Environments in VSE.

This chapter provides an overview of five possible usage environments for which you can set up your DB2 Server for VSE system.

Chapter 11. Stored Procedures.

This chapter provides an overview of what stored procedures are, and how to use them.

Appendix A. Estimating Your Dbspace Requirements.

Dbspaces, which hold tables, must have sufficient storage capacities to meet the storage requirements of their tables. This appendix describes how to estimate the amount of storage the tables require, so that you acquire dbspaces with sufficient capacity.

Appendix B. CMS EXECs.

This appendix describes the EXECs provided for use in user VM/ESA virtual machines.

Appendix C. Querying the Status of an Application (VM Only).

This appendix describes the CMS SQLQRY command available in your VM/ESA system.

Appendix D. Maximums.

This appendix describes the logical data and ISQL maximums.

Appendix E. SQLGLOB Parameters (VSE Only).

This appendix describes the SQLGLOB VSAM file available in your VSE/ESA system.

Prerequisite IBM Publications

All readers of this book should be familiar with the content of the following manuals:

- *DB2 Server for VSE & VM Overivew*, GC09-2995
- *DB2 Server for VSE & VM SQL Reference*, SC09-2989
- *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2987.

Highlighting Conventions

This manual uses the following text highlighting conventions:

Italics Italic type is used for command variables, parameter values and their symbolic equivalents, titles of stand-alone manuals, strings of characters to be used exactly as they appear, and important terms that are being defined.

Boldface
Bold type is used for emphasis.

Monospace

Monospace type indicates material that is entered at a display station, or displayed, coded, or printed on a computer printing device.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other DB2 Server for VSE & VM documentation:

- Visit our home page at:

  `http://www-4.ibm.com/software/data/db2/vse-vm/`

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

  IBM CANADA LTD.
  DB2 Server for VSE & VM
  2S/240/1150/TOR
  1150 Eglinton Ave. East
  North York, Ontario
  Canada M3C 1H7

- Send your comments by electronic mail to one of the following addresses:

Format	Address
Internet	torrcf@ca.ibm.com
Facsimile	(416) 448-6161 (Attention RCF Coordinator)

  Be sure to include the name of the book, the form number (including the suffix), and the page, section title, or topic you are commenting on.

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

- Fill out the form at the back of this book and return it by mail, by fax, or by giving it to an IBM representative.

Syntax Notation Conventions

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

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

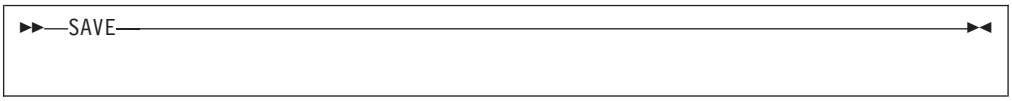
  The ►►— symbol indicates the beginning of a statement or command.

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

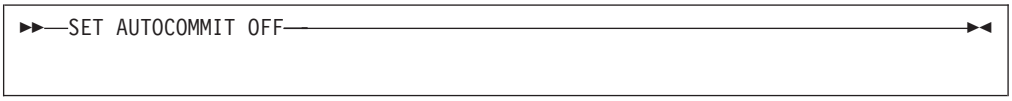
  The ►— symbol indicates that a statement is continued from the previous line.

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

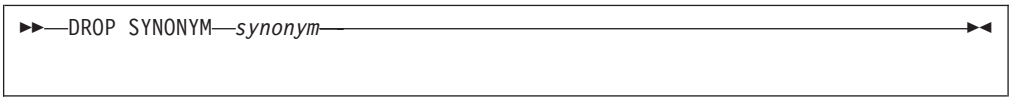
  Diagrams of syntactical units that are not complete statements start with the ►— symbol and end with the —► symbol.
- Some SQL statements, Interactive SQL (ISQL) commands, or database services utility (DBS Utility) commands can stand alone. For example:



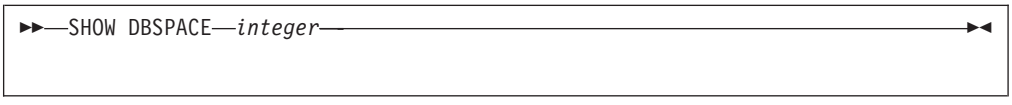
  Others must be followed by one or more keywords or variables. For example:



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



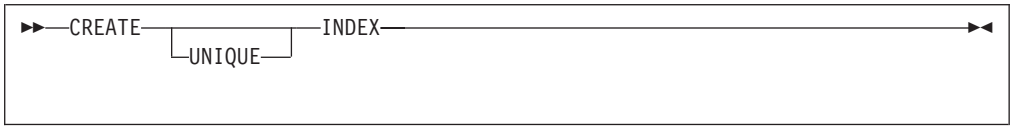
- Keywords appear in either uppercase (for example, SAVE) or mixed case (for example, CHARacter). All uppercase characters in keywords must be present; you can omit those in lowercase.
- Parameters appear in lowercase and in italics (for example, *synonym*).
- If such symbols as punctuation marks, parentheses,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



  This command might appear as:

```
SHOW DBSPACE 1
```

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



This statement could appear as either:

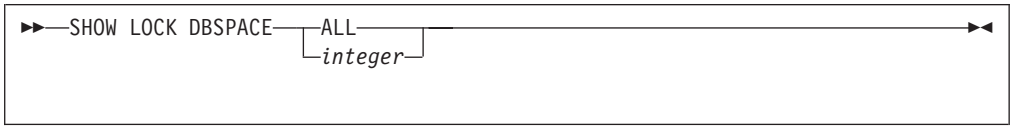
```
CREATE INDEX
```

or

```
CREATE UNIQUE INDEX
```

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

  If you must choose one of the items, one item appears on the main path. For example:



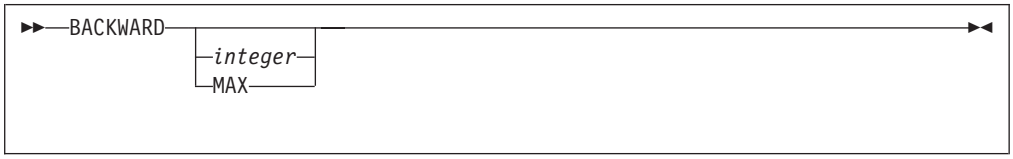
Here, the command could be either:

```
SHOW LOCK DBSPACE ALL
```

or

```
SHOW LOCK DBSPACE 1
```

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



Here, the command could be:

```
BACKWARD
```

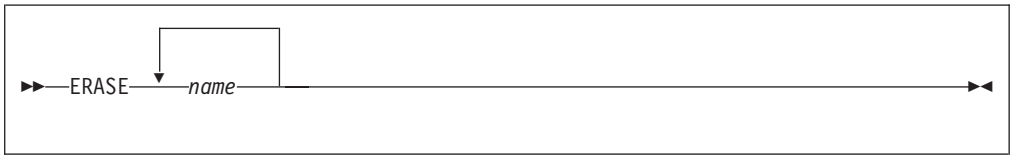
or

```
BACKWARD 2
```

or

```
BACKWARD MAX
```

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



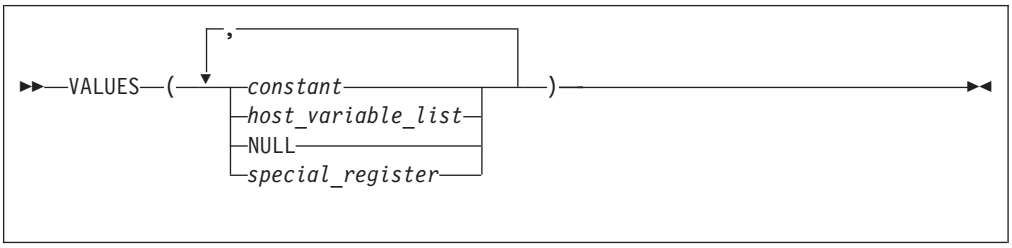
This statement could appear as:

```
ERASE NAME1
```

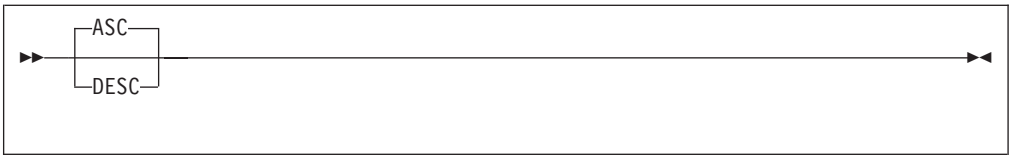
or

```
ERASE NAME1 NAME2
```

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

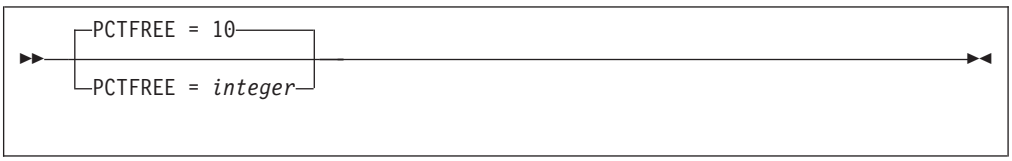


- If an item is above the main line, it represents a default, which means that it will be used if no other item is specified. In the following example, the ASC keyword appears above the line in a stack with DESC. If neither of these values is specified, the command would be processed with option ASC.

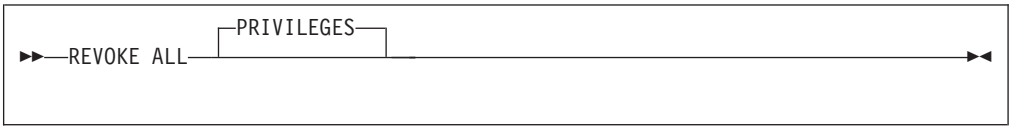


- When an optional keyword is followed on the same path by an optional default parameter, the default parameter is assumed if the keyword is not entered. However, if this keyword is entered, one of its associated optional parameters must also be specified.

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

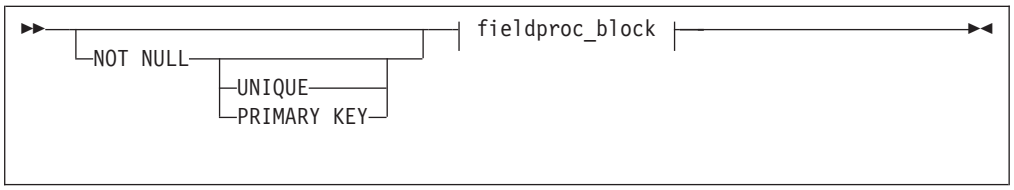


- Words that are only used for readability and have no effect on the execution of the statement are shown as a single uppercase default. For example:

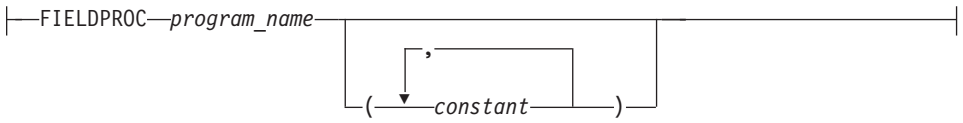


  Here, specifying either REVOKE ALL or REVOKE ALL PRIVILEGES means the same thing.

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



fieldproc_block:



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

CALL
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

Summary of Changes

This is a summary of the technical changes to the DB2 Server for VSE & VM database management system for this edition of the book. All manuals are affected by some or all of the changes discussed here. For your convenience, the changes made in this edition are identified in the text by a vertical bar (|) in the left margin. This edition may also include minor corrections and editorial changes that are not identified.

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.

Summary of Changes for DB2 Version 7 Release 1

Version 7 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 3 or later environment and on the Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA™) Version 2 Release 3 Modification 1 or later environment.

Enhancements, New Functions, and New Capabilities

TCP/IP Support for DB2 Server for VSE

TCP/IP support allows:

- VSE online and batch application programs to access remote application servers which support IBM's implementation of the DRDA architecture over TCP/IP.
- Remote application requesters which support IBM's implementation of the DRDA architecture to access the DB2 for VSE application server over TCP/IP.

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VSE & VM Database Administration*
- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE Program Directory*.

DRDA RUOW Application Requester for VSE (Batch)

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

This support provides VSE batch application programs with the ability to execute SQL statements to access and manipulate data managed by any remote application server that supports IBM's implementation of the DRDA architecture.

VSE batch application programs can access only one remote application server per unit of work, and must use TCP/IP communications.

For more information, see the following DB2 Server for VSE & VM documentation:

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

- *DB2 Server for VSE Program Directory*.

Stored Procedures Application Requester

A stored procedure is a user-written application program compiled and stored at the server. Stored procedures allow logic to be encapsulated in a procedure that is local to the database manager. The ability to use stored procedures provides distributed solutions that let more people access data faster. SQL statements and replies flowing across the network are reduced and performance is improved.

This support provides VM and VSE (online and batch) application programs with the ability to invoke stored procedures from any remote application servers that support IBM's implementation of the DRDA architecture. It also allows processing of result sets if supported by the remote DRDA application server.

For more information, see the following DB2 Server for VSE & VM documentation:

- *DB2 Server for VSE & VM Application Programming*
- *DB2 Server for VSE & VM SQL Reference*.

Simplified DB2 Server for VSE Installation/Migration

A REXX procedure Job Manager is supplied to assist in the DB2 Server for VSE Installation/Migration process. It controls the overall job flow based on the contents of the job list control tables and the parameter table (supplied as Z-type members). The job manager selects the job control member from the job list file (a Z-type member), extracts the member from the Installation Library, modifies the JCL, submits the job, evaluates the execution, posts the results, and then repeats the process as required. The users are required to modify the parameter table, according to their environment.

This support simplifies the process of installation and migration by reducing user intervention - the Job Manager submits the prepared jobs.

See the *DB2 Server for VSE Program Directory* for further details.

New Code Page and Euro Symbol Code Page Support

The following CCSIDs are now supported:

- 1137: Hindi
- 1142: E-Danish/Norweigan
- 1143: E-Finnish/Swedish
- 1145: E-Spanish.

Additional support has been added for conversions from Unicode (UTF-8) 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.

Control Center for VM Enhancements

The following is a list of enhancements that have been made to the Control Center for VM:

- QMF™ Tools: allow the user to list QMF objects, view and unload QMF queries and PROCS, schedule QMF PROCS to execute, and run explain on QMF queries.
- Table Create Tool: allows the user to create new tables.
- Search List improvements.
- Referential Integrity Report tool: A referential integrity map report can now be generated directly from the CMS command interface.

- PL/I prerequisite removal.
- New and improved tape hopper support.
- High density tape drive support: support for high density (non-CMS density) tape drives.

Control Center for VSE Enhancements

The following enhancements have been provided for Control Center for VSE:

- Additional Operator Command Support
- Installation of IBM-provided Stored Procedures.

QMF for VSE & VM Optional Feature

The following enhancements have been provided for QMF for VSE & VM:

- Application Requester support for VSE QMF users
- Command enhancements to default to object type
- Fast path to the QMF home screen
- Cross-platform install capability
- DB2 for AS/400 database access.

QMF for Windows® Optional Feature

The following enhancements have been provided for QMF for Windows :

- Java-based Query
- Aggregating, grouping and formatting directly within query results and automatic Form creation
- Personal portal user interface that launches centrally shared queries and reports, and sends results to spreadsheets, desktop databases, and browsers
- Procedures with REXX.

Reliability, Availability, and Serviceability Improvements

DBNAME Directory Restructuring

ARISDIRD has been restructured to improve readability and flexibility. Each DBNAME entry is now defined explicitly by its type (Local, Remote or Host VM (Guest Sharing)). CICS AXE Transaction TPNs (Transaction Program Names) are still included in the directory as a type of 'LOCALAXE'. The DBNAME Directory Builder program, ARICBDID has been rewritten as a REXX/VSE procedure with extensive error and dependency checking. Support for TCP/IP information is added and 'alias' DBNAMEs are supported. **ALL** DBNAMEs **must** be specified in the new DBNAME Directory, including the Product Default DBNAME "SQLDS". A migration REXX/VSE procedure, ARICCDID, is provided to assist in migrating to the new format. See the *DB2 Server for VSE System Administration* and *DB2 Server for VSE Program Directory* for additional information.

Migration Considerations

Migration is supported from SQL/DS™ Version 3 and DB2 Server for VSE & VM Versions 5 and 6. 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 Application Programming*
  - *DB2 Server for VSE Application Programming*

- *DB2 Server for VM Database Administration*
- *DB2 Server for VSE Database Administration*
- *DB2 Server for VSE Installation*
- *DB2 REXX SQL Interface Installation*
- *DB2 REXX SQL Reference*
- *DB2 Server for VM Diagnosis Guide and Reference*
- *DB2 Server for VSE Diagnosis Guide and Reference*
- *DB2 VM Data Spaces Support*

  Note: Information from this book can now be found in the *DB2 Server for VSE & VM Performance Tuning Handbook*
- *DB2 Server for VM Master Index and Glossary*
- *DB2 Server for VSE Master Index and Glossary*.

• The following books have been added to the library:
- *DB2 Server for VSE & VM Database Administration*
- *DB2 Server for VSE & VM Application Programming*
- *DB2 REXX SQL for VM/ESA Installation and Reference*
- *DB2 Server for VSE & VM Diagnosis Guide and Reference*
- *DB2 Server for VSE & VM Master Index and Glossary*.

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

Chapter 1. Designing a Database

This chapter describes the conceptual process of database design. The implementation of the design, that is, the actual creation of a set of objects, is discussed in "Chapter 2. Implementing Your Design" on page 17.

Sample Tables

The DB2 Server for VSE & VM database contains sample tables that are referenced throughout this book and are used to demonstrate various concepts and procedures.

Entities, Properties, and Occurrences

Some basic terms for database design are defined below. There is no universally accepted terminology for database design; these terms may be used differently elsewhere.

- An *entity* is anything about which information can be stored. In the sample database, some of the entities are employees, departments, and projects.
- *Properties* are types of information categories associated with an entity. In the sample table EMPLOYEE, the entity **employee** has properties, such as, employee number, job held, birth date, and salary amount, which appear as columns EMPNO, JOB, BIRTHDATE, and SALARY.
- The *occurrence* of an entity consists of the values in all the columns for that entity. In the sample table EMPLOYEE, each employee has a unique employee number; therefore, each value in the EMPNO column is unique and can be used to identify a particular occurrence.

Entities and properties are represented as columns, and occurrences are represented as values in the columns, as shown in Table 1.

Table 1. Occurrences and Properties of an Entity

ENTITY	PROPERTIES			
Employee	**EMPNO**	**JOB**	**BIRTHDATE**	**SALARY**
Sally Kwan	000030	Manager	1941-05-11	38250
William Jones	000210	Designer	1953-02-23	18270

Step 1: Select the Data to Record in the Database

To be effective, your database must be designed specifically to meet the data storage and retrieval needs of your organization.

The first step in designing an effective database is to identify the collection of information that it will contain. You must then organize this information into tables, with each column of a row related in some way to all other columns of that row. This approach will enable you to identify the relationships that exist between the different entities.

For example, the following data relationships are expressed in the sample tables:

- Employees are assigned to departments, for example:

Dolores Quintana is assigned to Department C01.
Heather Nicholls is assigned to Department C01.

- Employees earn money, for example:
  Dolores earns $23,800 per year.
  Heather earns $28,420 per year.
- Departments report to other departments, for example:
  Department C01 reports to Department A00.
  Department D01 reports to Department A00.
- Employees work on projects, for example:
  Dolores works on project IF1000.
  Heather works on projects IF1000 and IF2000.
- Employees manage departments, for example:
  Sally Kwan manages Department C01.

Before you design your tables, you must understand entities and their relationships. Table 2 shows an example.

Table 2. Relationships in the Sample Database

ENTITY	RELATIONSHIP	ENTITY
Employees	are assigned to	departments
Employees	earn	money
Departments	report to	departments
Employees	work on	projects
Employees	manage	departments

The relationship between the columns in a table is the same in each row of the table. For example, in Table 1 on page 1, the relationship between each entry in the Employee column and its corresponding entry in the Salary column is the same, because the Salary column describes the amount the employee earns.

Step 2: Define Tables for Each Type of Relationship

In a relational database, you can express several types of entity relationships. Consider the relationship between employees and departments. A given employee can work in only one department, so this relationship is *single-valued* for employees. On the other hand, one department can have many employees, so this relationship is *multivalued* for departments. Accordingly, this constitutes a *one-to-many* relationship. Relationships can be:

- One-to-one
- One-to-many
- Many-to-one
- Many-to-many.

If each employee can belong to several departments, the employees/departments relationship would be *many-to-many*.

You must define separate tables for different types of relationships.

One-to-One Relationships

One-to-one relationships are single-valued in both directions. A manager manages one department; a department has only one manager. The questions “Who is the manager of Department C01?” and “What department does Sally Kwan manage?” both have single answers. The relationship could be assigned to either the

department table or the employee table. Because all departments have managers, but not all employees are managers, it would be logical to add the manager to the department table, as shown in Figure 3.

Figure 3. Assigning One-to-One Facts to a Table

One-to-Many and Many-to-One Relationships

To define tables for each one-to-many and many-to-one relationship, you must:

- Group all the relationships for which the "many" side of the relationship is the same entity.
- Define a separate table for each group.

In Table 3, the "many" side of the first and second relationships is "employees", so we defined an employee table (EMPLOYEE). In Figure 4, "departments" is the "many" side, so we defined a department table (DEPARTMENT).

Table 3. Many-to-One Relationships

ENTITY	RELATIONSHIP	ENTITY
1. Employees	are assigned to	departments
2. Employees	earn	money
3. Departments	report to	(administrative) departments

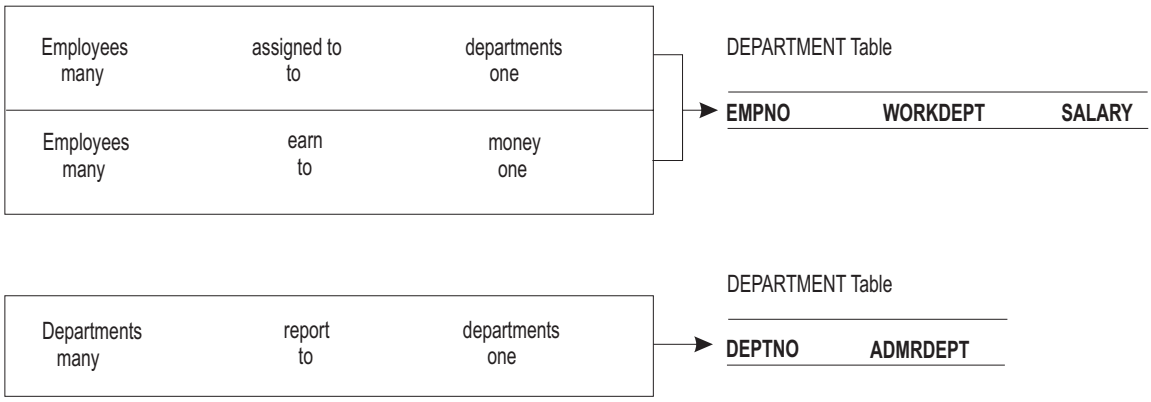


Figure 4. Assigning Many-to-One Facts to Tables

Many-to-Many Relationships

A relationship that is multivalued in both directions is many-to-many. An employee might work on more than one project, and a project might have more than one employee assigned to it. The questions "What does Dolores Quintana work on?" and "Who works on project IF1000?" both yield multiple answers. A many-to-many relationship can be expressed in a table with a column for each entity ("employees" and "projects"), as shown in Figure 5 on page 4.

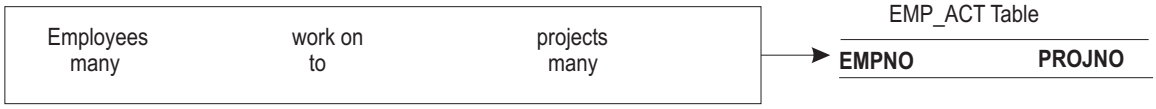


Figure 5. Assigning Many-to-Many Facts to a Table

Step 3: Provide Column Definitions for Tables

Defining a column in a table consists of:

- Choosing a name for the column
  Each column in a table must have a name that is unique within the table. For detailed information, see "Column Names" on page 31.
- Specifying the data type that is valid for the column
  The data type of a column indicates the length of the values in the column and the kind of data that is valid for it. For detailed information, see "Specifying Data Types" on page 31.
- Specifying the columns that can contain null values
  Some columns cannot contain meaningful values in all rows because some values may not be known at a particular time. For example, you may know a new employee's name but not his or her birth date. For detailed information, see "Specifying Data Types" on page 31.

Step 4: Identify One or More Columns as a Primary Key

If every row in a table represents relationships for a unique entity, the table should have a *primary key*: one column (or a set of columns) that provides a unique identifier for the rows of the table. A unique index of the columns of the primary key is created when the primary key is created. You can create the primary key when you create the table using the CREATE TABLE statement (see "Creating Tables" on page 27) or, if the table already exists, by using the ALTER TABLE statement (see "Altering the Design of a Table" on page 64). A primary key must not contain a nullable column or a long field.

Note: Long fields include the following data types: VARCHAR(n) with n>254, VARGRAPHIC(n) with n>127, LONG VARCHAR, or LONG VARGRAPHIC.

The primary keys of some of the sample tables are:

Table	**Key Column**
EMPLOYEE table	EMPNO
DEPARTMENT table	DEPTNO
PROJECT table	PROJNO

Figure 6 shows part of the PROJECT table with the primary key column indicated.

PROJNO	PROJNAME	DEPTNO
MA2100	WELD LINE AUTOMATION	D01
MA2110	W L PROGRAMMING	D11

Figure 6. A Primary Key on a Table

Figure 7 shows a primary key consisting of more than one column; it is a *multicolumn key*.

PROJNO	ACTNO	ACSTAFF	ACSTDATE
MA2100	10	0.5	82-01-01
MA2100	20	1.0	82-01-01
MA2110	10	1.0	82-01-01

Figure 7. A Multicolumn Primary Key. The three columns PROJNO, ACTNO, and ACSTDATE are all parts of the primary key.

If you have more than one candidate for a primary key, you can define a UNIQUE constraint on the column (or set of columns) that you do not select as the primary key. A column with a UNIQUE constraint is similar to a primary key in that a unique index on the column is created. It differs in that you can create more than one UNIQUE constraint on a table, and no foreign keys can reference a UNIQUE constraint (see "Foreign Key" on page 7).

Step 5: Ensure that Equal Values Represent the Same Entity

You can have more than one table describing properties of the same set of entities. For example, one table could give employees' job and salary information, as in the EMPLOYEE table, and another each employee's home address. To retrieve both sets of properties at once, you can join the tables on any set of matching columns, as shown in Figure 8 on page 6. If there are two employees named Sally Kwan, a join on *employee name* may not match the correct rows. Similarly, if one person has more than one authorization ID, a join on *ID* may not produce the correct match. Thus, for the purpose of retrieving information about an entity from more than one table, an equal value in each of those tables should represent that entity. This type of join is an *equijoin*.

Figure 8 shows a join between the DEPARTMENT and EMPLOYEE tables on columns of department numbers.

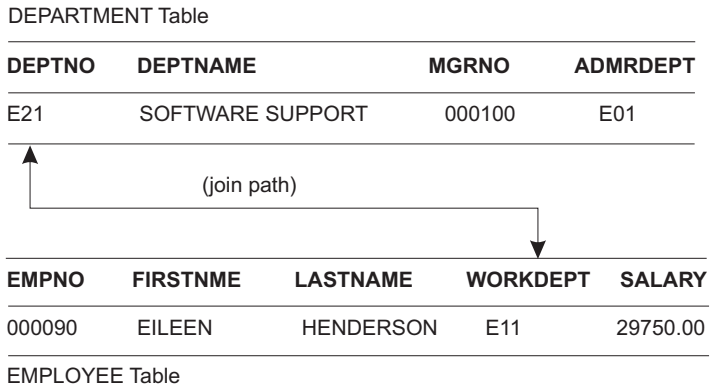


Figure 8. A Join Path between Two Tables

The connecting columns must be of the same data type. They can have different names (such as WORKDEPT and DEPTNO in Figure 8), or the same name (such as the two columns called DEPTNO in the DEPARTMENT and PROJECT tables). The latter case is illustrated in Figure 9.

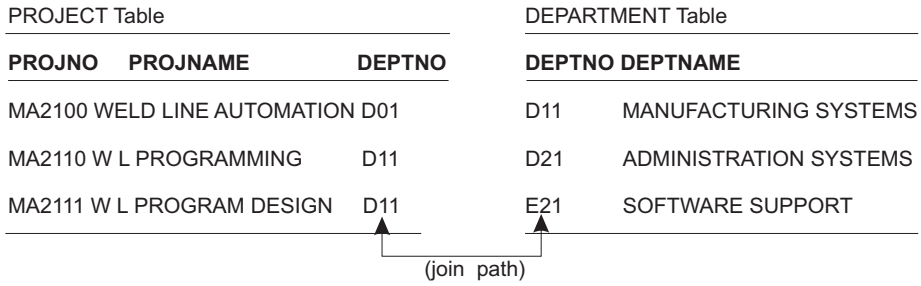


Figure 9. A Join Path on Columns with the Same Name

Step 6: Plan for Referential Integrity

A table can serve as a complete list of all occurrences of a single entity. In the sample database, the EMPLOYEE table serves that purpose for employees: only the numbers that appear in this table are valid employee numbers. Similarly, the DEPARTMENT table provides a master list of all valid department numbers, and the PROJECT table provides a master list of valid projects. When a table refers to an entity for which there is a master list, it should identify an occurrence of the entity that appears in the master list; otherwise, either the reference is incorrect or the master list is incomplete.

When all references from one table to another are valid, this condition is called *referential integrity*. Having referential integrity does **not** necessarily mean the data is correct. That the EMPLOYEE table shows every employee assigned to a **valid** department number is one thing; whether it shows every employee in the **correct** department is quite another.

Elements of Referential Integrity

You must consider many different elements to ensure referential integrity. The concepts of a *primary key* and a *unique constraint* were described in "Step 4: Identify One or More Columns as a Primary Key" on page 4. Other elements to consider when dealing with referential integrity are described in the following sections.

Foreign Key

A column or set of columns that refers to the primary key of another table is a *foreign key*. For example, the column Work Department (WORKDEPT) of the EMPLOYEE table is a foreign key; it refers to DEPTNO, the primary key of the DEPARTMENT table. The combination of the project number (PROJNO), activity number (ACTNO), and activity starting date (EMSTDATE) columns in the EMP_ACT table is a foreign key; it refers to the primary key of the PROJ_ACT table.

Referential Constraint

A referential constraint is a relationship between a primary key and a foreign key with certain deletion and update rules that define how the relationship is maintained. Refer to "DELETE, INSERT, and UPDATE Considerations" for information on deletion and update rules.

Parent and Dependent Tables

Establishing a referential constraint defines a relationship between two tables. The table containing the primary key is the *parent table*, and the one containing the foreign key is the *dependent table*. In a multilevel, hierarchical chain of dependent tables, a *descendent* table is any table below the top level. Such a table is a descendent of all the tables above it in the hierarchy.

A *referential cycle* is a set of referential constraints in which each table in the set is a descendent of itself. A table can be a parent of many tables, and it can also be a dependent or descendent of many parents.

Self-Referencing Table

A self-referencing table is one that contains both the primary key and the foreign key of a referential constraint. Conceptually, a self-referencing table is both the parent and the dependent table in a relationship. DB2 Server for VSE & VM does not support self-referencing.

DELETE, INSERT, and UPDATE Considerations

DELETE Rules

For Parent Tables: When an employee retires, you remove that person's EMPLOYEE record. The deletion affects the information in the PROJECT, DEPARTMENT, and EMP_ACT tables. For any particular relationship, one of the following deletion rules is enforced:

- RESTRICT

  You cannot delete any rows of the parent table that have dependent rows. In the DEPARTMENT-PROJECT relationship, using RESTRICT means that you cannot remove a department if any of its employees are assigned to a project.
- SET NULL

  When you delete a row of the parent table, the corresponding values of the foreign key in any dependent rows are set to NULL. This rule is used in the DEPARTMENT-EMPLOYEE relationship: when you delete a department record, the WORKDEPT column of dependent rows in the employee table is set to NULL, indicating that the employee is not assigned to a department.
- CASCADE

  When you delete a row of the parent table, any dependent rows in the dependent table are also deleted. This rule is useful when a row in the

dependent table is useless without a row in the parent table. For example, if you delete an employee there is no reason to maintain the associated EMP_ACT record.

Multiple levels of CASCADE are supported; that is, a delete operation on a parent table deletes **all** dependent rows in its dependent tables if the dependent tables are enforced by the CASCADE delete rule of referential constraint. If any of these dependent tables are also parent tables, the delete rule of referential constraint in turn applies between them and their dependent tables. All applicable delete rules are used to determine the result of a delete operation. A delete operation is subject to *rollback*, if the parent row has a dependent row in a referential constraint with a delete rule of RESTRICT, or if the deletion cascades to any descendent that has a dependent row in a referential constraint with a delete rule of RESTRICT.

For Dependent Tables: You may, at any time, delete rows from a dependent table without taking any action on the parent table. For example, you may no longer need EMP_ACT records after the project is completed. You can delete the record without affecting the EMPLOYEE or PROJ_ACT tables.

Restrictions When Using the DELETE Statement: To ensure referential integrity, the table specified in the subquery must not be affected by the delete on the object table of the DELETE statement.

For example, if B is the object table of a DELETE statement, and A is a table that is referenced in the FROM clause of a subquery of that statement, then the following rules apply:

- Table A cannot also be an object table of the deletion.
- Table A cannot be a dependent of table B in a relationship with a delete rule of CASCADE or SET NULL.
- Table A cannot be a dependent of any other table (for example, table C) in a relationship with a delete rule of CASCADE or SET NULL, if deletions from table B cascade to table C.

For more information on delete-connected tables, refer to "Restrictions on Keys and Referential Constraints:" on page 39.

INSERT Rules

For Parent Tables: You can insert a row at any time into a parent table without taking any action in the dependent table. For example, you can create a new department in the DEPARTMENT table without making any change to the EMPLOYEE table. For the insertion to be successful, the new primary key or unique key values must be unique.

For Dependent Tables: You cannot insert a row into a dependent table unless a row in the parent table contains a primary key value equal to the foreign key value you want to insert. If a foreign key has a null value, it can be inserted into a dependent table, but no logical connection exists.

UPDATE Rules

For Parent Tables: You cannot change a value in a primary key column if the associated row has a dependent row. For example, if a department number changes, the DEPTNO value in the DEPARTMENT table cannot be changed if there are employees in the EMPLOYEE table who are members of that department.

For Dependent Tables: You cannot change a value in a foreign key column of a dependent table unless the new foreign key value already exists in the primary key of the parent table. For example, when an employee transfers from one department to another, the department number must change. The new value must be the number of an existing department, or null.

Step 7: Normalize Your Tables

Normalization is the method of reducing data stored in tables so that the tables contain unique keys, each identifying a single entity. Each of these keys has an associated row of values that describes each entity. Complete normalization is **not** required for using the database manager.

The topic of normalizing tables draws much attention in database design. This section briefly reviews the rules for first, second, third, and fourth normal forms of tables, and describes some reasons why they should or should not be followed.

First Normal Form

Any relational table satisfies the requirement of first normal form: at each row-and-column position in the table, there exists only one value, never a set of values.

Second Normal Form

A table is in second normal form if each column not in the key provides a fact that depends on the entire key.

Second normal form is violated when a non-key column is a fact about a subset of a composite key, as in Figure 10. An inventory table records quantities of specific parts stored at particular warehouses; its columns are shown below.

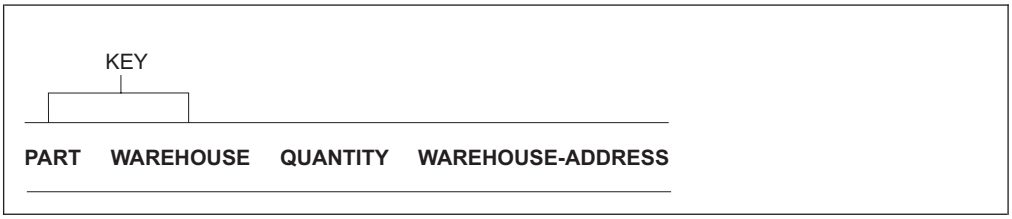


Figure 10. Key Violates Second Normal Form

The key here consists of the PART and the WAREHOUSE columns together. Because the column WAREHOUSE-ADDRESS depends only on the value of WAREHOUSE, the table violates the rule for second normal form. The problems with this design are:

- The warehouse address is repeated in every record for a part stored in that warehouse.
- If the address of the warehouse changes, every row referring to a part stored in that warehouse must be updated.
- Because of the redundancy, the data could become inconsistent, with different records showing different addresses for the same warehouse.
- If at some time there are no parts stored in the warehouse, there may be no row in which to record the warehouse address.

To satisfy second normal form, the information shown in Figure 10 on page 9 must be in two tables, as in Figure 11.

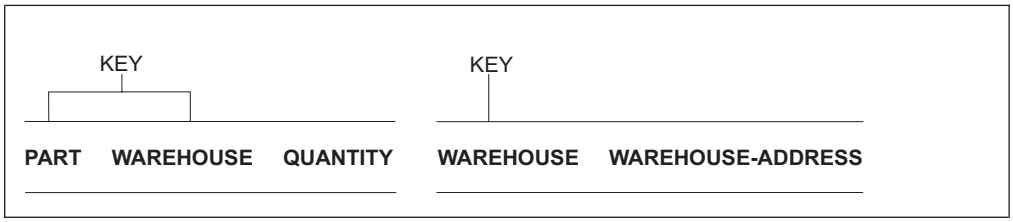


Figure 11. Two Tables Satisfy Second Normal Form

There is a performance disadvantage in having the two tables in second normal form, because programs that produce reports on the location of parts have to join both tables to retrieve the relevant information.

For further information on performance considerations, refer to "Considerations for Normalization" on page 29.

Third Normal Form

A table is in third normal form if each non-key column provides a fact that depends only on the key.

Third normal form is violated when a non-key column is a fact about another non-key column. For example, the first table in Figure 12 contains the columns EMPNO and WORKDEPT. Suppose a column DEPTNAME is added. The new column depends on WORKDEPT, whereas the primary key is the column EMPNO; thus, the table now violates third normal form.

Changing DEPTNAME for a single employee, John Parker, does not change the department name for other employees in that department. The inconsistency that results is shown in the updated version of the table in Figure 12.

EMPLOYEE-DEPARTMENT Table (EMPDEPT) Before Update

EMPNO	FIRSTNME	LASTNAME	WORKDEPT	DEPTNAME
000290	JOHN	PARKER	E11	OPERATIONS
000320	RAMLAL	MEHTA	E21	SOFTWARE SERVICES
000310	MAUDE	SETRIGHT	E11	OPERATIONS

EMPLOYEE-DEPARTMENT Table (EMPDEPT) After Update

EMPNO	FIRSTNME	LASTNAME	WORKDEPT	DEPTNAME
000290	JOHN	PARKER	E11	INSTALLATION MGMT
000320	RAMLAL	MEHTA	E21	SOFTWARE SERVICES
000310	MAUDE	SETRIGHT	E11	OPERATIONS

Figure 12. Update of an Unnormalized Table. Information in the table has become inconsistent.

The table can be normalized by providing a new table, with columns for WORKDEPT and DEPTNAME. In that situation, updating a department name is much easier as it only has to be made to the new table. But an SQL query that shows the department name with the employee name is more complex to write: it requires joining the two tables. It also takes longer to run than the query of a single table. As well, the entire arrangement takes more storage space, because the WORKDEPT column must appear in both tables.

Fourth Normal Form

A table is in fourth normal form if no row contains two or more independent multivalued facts about an entity.

Consider facts about employees, skills, and languages, where an employee may have several skills and know several languages. There are two relationships, one between employees and skills, and one between employees and languages. A table is not in fourth normal form if it represents both relationships, as in Figure 13.

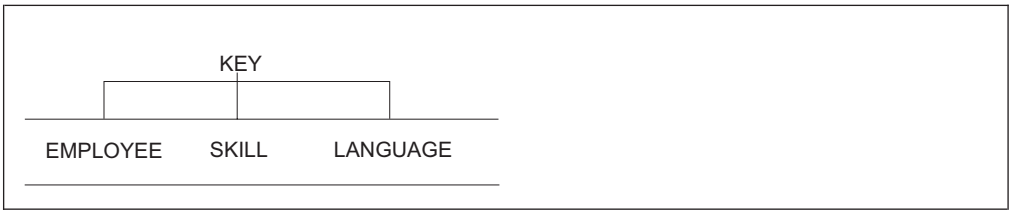


Figure 13. A Table That Violates Fourth Normal Form

Instead, the relationships should be represented in two tables, as in Figure 14.

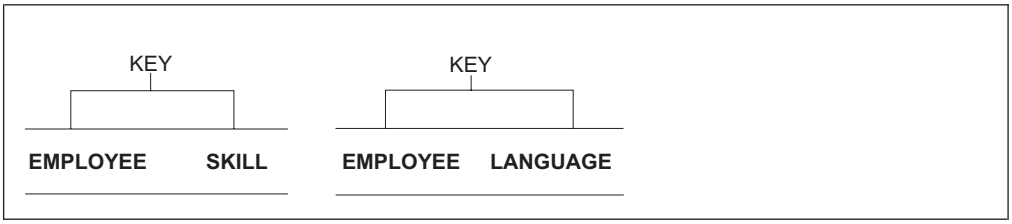


Figure 14. Tables in Fourth Normal Form

If, however, the facts are interdependent (that is, the employee applies certain languages only to certain skills), then the table should **not** be split.

Any data can be put into fourth normal form. A good rule when designing a database is to arrange all data in tables in fourth normal form, and then decide whether the result will give you an acceptable level of performance. If it will not, you are at liberty to undo the normalization of your design.

Step 8: Considerations for Distributed Data

Two types of access to DB2 Server for VSE & VM data are available. They are remote unit of work and distributed unit of work.

Remote unit of work, implemented in SQL/DS V3.3, for VM, and SQL/DS V3.4, for VSE, lets a user or application program on a Distributed Relational Database Architecture (DRDA) application requester to read or update data stored in a DB2 Server for VSE & VM DRDA application server. With remote unit of work, a user

or application program can have many SQL statements within a unit of work; accessing one database management system with each SQL statement; and accessing one database management system within a unit of work.

Distributed unit of work, implemented in DB2 Server for VSE & VM Version 5 Release 1 lets a user or application program on a Distributed Relational Database Architecture (DRDA) application requester to read or update data stored in multiple locations, where the DB2 Server for VSE & VM DRDA application server is one of the multiple sites where data is read or updated within a single unit of work. With distributed unit of work, a user or application program can have many SQL statements within a unit of work; accessing one database management system with each SQL statement; and accessing many database management systems within a unit of work. Commit and rollback are coordinated at all locations so that if a failure occurs anywhere in the system, data integrity is preserved. This type of coordinated approach is called two phase commit processing and is done by a Sync Point Manager. In phase one, the coordinating RDBMS (generally the requesting RDBMS) polls each participating RDBMS to vote to commit or rollback the transaction. In phase two, the coordinator directs the RDBMSs to commit or rollback based on the preceeding vote.

Access to DB2 Server for VSE & VM DRDA application servers by DRDA application requesters is possible only if the DRDA facility is installed on the DB2 Server for VSE & VM application server.

DB2 Server for VM implements the application server and application requester support for DRDA remote unit of work, and the application server support for DRDA distributed unit of work. VM application requesters can participate in remote unit of work activity but cannot participate in distributed unit of work activity.

Access to non-DB2 Server for VM application servers by DB2 Server for VM application requester is possible only if the DRDA facility has been installed on the DB2 Server for VM application requester and if the non-DB2 Server for VM application servers support IBM's implementation of the DRDA protocol.

DB2 Server for VSE implements the application requester support for DRDA remote and distributed unit of work for CICS/VSE online applications. VSE online application requesters can participate in remote and distributed unit of work activity. With distributed unit of work, a CICS/VSE online application is limited to accessing a single DRDA application server within one LUW. However, it can update another CICS resource, in addition to the remote DRDA application server it is accessing, within one LUW, provided both the DRDA application server and the CICS resource participates in two-phase commits.

DB2 Server for VSE implements the application requester support for DRDA remote unit of work for Batch applications. VSE batch application requesters can participate in remote unit of work activity, but cannot participate in distributed unit of work activity.

Access to remote application servers by a DB2 Server for VSE application requester is possible only if the DRDA facility has been installed on the DB2 Server for VSE application requester and if the remote application server supports IBM's implementation of the DRDA protocol.

Designing a distributed database management system involves making decisions about where to put the data, how to manage security and accounting, and how to handle problems, backup, recovery, and change control.

For general guidance on making these decisions, refer to the following manuals:

- *Planning for Distributed Relational Database*,
- *DB2 Connectivity Supplement*,
- *Connect Enterprise Edition Quick Beginnings*,
- *DB2 UDB Quick Beginnings*, and
- *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration*.

The decision to access distributed data has implications for many activities: application programming, data recovery, and authorization. This section introduces some of these considerations. Refer to the appropriate manual for information on particular tasks.

Definitions

The *application requester* is the component that accepts a request from an application and passes it to an application server. The *application server* is the component that receives and processes requests issued by the application requester.

In VSE an application server is *local* if it resides in the same VSE machine as the DB2 Server for VSE application requester. This can also be a DB2 Server for VM application server accessed through VSE guest sharing. This DB2 Server for VM server can be either on the same VM machine as the VSE guest, or on another VM machine accessed remotely through AVS or TSAF. A remote application server can be a DB2 Server for VSE application server *not* residing in the same VSE machine as the application program connecting to it, or a non-DB2 Server for VSE application server.

In VM, a system is *local* if the application requester and the application server reside on the same processor, and is *remote* if they reside on different processors. Remote does not necessarily mean at a distance; the application server and application requester may be at the same user site.

Two relational database systems are *like* if both the application requester and the application server are the same product (for example, both are DB2 Server for VSE or both are DB2 Server for VM). They are *unlike* if different products are involved (for example, a DB2 Server for VM application requester and a DB2 Server for VSE application server).

A DB2 Server for VM application requester can communicate with a like system, either local or remote, through the SQLDS protocol or the DRDA protocol. It can communicate with an unlike system through the DRDA protocol, if the Relational Database Management System (RDBMS) of the unlike system supports the protocol.

A DB2 Server for VSE application requester can communicate with a local DB2 Server for VSE application server through the SQLDS protocol or a DB2 Server for VM application server which is accessed using Guest Sharing through the SQLDS protocol. A DB2 Server for VSE application requester can communicate with a

remote application server through the DRDA protocol, if the Relational Database Management System (RDMS) of the remote application server supports the protocol.

Application Programming

Several categories of application programming considerations are:

- Character conversion

  Data and statements are converted if the connected systems are using different coded character set identifiers (CCSIDs). For example, an SQL statement originating in an ASCII environment that is sent to an EBCDIC environment must be converted for the DB2 Server for VSE & VM application server to process it. This conversion ensures that the application server correctly interprets the statement and the data, and displays the results using the appropriate character sets. For more information on character conversion, refer to either the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manual.

  It is important that the application server and application requester have the same CCSID value, unless there is a specific reason for them to be different. When the application server and application requester have different CCSID values, character conversion cannot be avoided. This conversion has an associated performance overhead, and causes performance degradation. For more information on performance, see the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

- Access limitations

  The limitations that exist for local multiple database applications apply to remote database applications with remote unit of work support. You cannot:

  – Access more than one application server in a single logical unit of work (LUW).
  – Join tables from multiple application servers.
  – Define referential constraints across application servers.

  These limitations also apply to remote database applications with distributed unit of work support. One exception though, is that with DUOW you can access more than one application server in a single logical unit of work (LUW).

  For the DRDA protocol restrictions, see the *DB2 Server for VSE & VM SQL Reference* manual.

- Performance considerations

  An obvious consideration for an SQL query that is transmitted to a remote application server is that the query and its reply must both be transmitted over an SNA or TCP/IP network, in VSE, or in VM, over a TSAF collection, VTAM network or TCP/IP network, conceivably as far as halfway around the world. This can increase the amount of processing and degrade the performance of the application in comparison with the same query run on your local application server. If the DRDA protocol is used, the DB2 Server for VSE & VM application requester has the option of increasing the block size used to return data. This can improve the performance of some applications. For more information in VM, see "SQLINIT EXEC" on page 235, in VSE, see "Appendix E. SQLGLOB Parameters (VSE Only)" on page 265.

  If the connected systems use different CCSIDs, performance can also be adversely affected, because additional processing is required to convert the data and statements.

- Cross-system differences.

Different relational database management systems use the SQL language, and strive to provide a consistent interface for applications. There are, however, some inconsistencies between systems. For example, the database manager does not support self-referencing constraints (a referential constraint in which both the primary key and the foreign key of the constraint are in the same table). On the other hand, it provides an EXPLAIN function, useful in tuning SQL statement performance, which is not provided by some RDBMS. These differences affect the portability of database designs and applications from system to system.

System Operations

Several commands for monitoring the operations of the DB2 Server for VSE & VM application server provide detailed information to the database administrator about users and their systems. For more information on these commands, see the *DB2 Server for VSE & VM Operation* manual.

You cannot effectively administer a remote application server from your local system, and sometimes must coordinate operations by means external to your local system. Both the application requester and application server must be defined in an SNA or TCP/IP network.

In VSE using SNA networks, Transaction Program Names (TPNs) can be used by remote application requesters to identify local DB2 Server for VSE application servers to which they want to connect on the local VSE system. These TPNs must be identified in the local DBNAME Directory and mapped to the appropriate server APPLID. Likewise, Remote Transaction Program Names (REMTPNs) can be used by the local system to identify the remote DRDA application server to which the local DB2 Server for VSE online (CICS) application requester wants to connect (Batch applications must use TCP/IP). These REMTPNs must be identified in the local VSE DBNAME Directory and mapped to the appropriate remote server SNA System ID (SYSID).

In VSE using TCP/IP networks, remote DRDA application requesters must know the local VSE TCP/IP Server's IP Address (or Host Name) and the local DB2 application server's Listener Port Number to access the local DB2 Server for VSE DRDA application server. Likewise, local VSE DRDA application requesters must know the remote DRDA application server's IP address (or Host Name) and Listener Port Number, which are identified in the local VSE DBNAME Directory.

For additional information on the VSE DBNAME Directory, refer to the *DB2 Server for VSE System Administration* manual.

In VM, all access to remote application servers through VTAM or TCP/IP require a CMS Communication Directory for the application requester. You must plan for creating and maintaining this directory on each VM system where the application requester resides. See the *VM/ESA: Connectivity Planning, Administration, and Operation* manual.

Similar considerations apply to users accessing other (non-DB2 Server for VSE & VM) application servers. Because each application server controls access to its own data, you must arrange to have valid user IDs on the other systems. As well, you must arrange for users to have proper authority and privileges on those application servers. Traces (used for problem determination) must also be coordinated with administrators at other sites, because traces must come from the system on which the data resides.

Distributing Existing Data

Although you can use the approaches previously described to distribute existing data, it is not a task to be undertaken lightly. Existing applications should only be distributed as part of an application redesign.

The best way to distribute data is the way used when the database was designed. However, the extent to which the preferred distribution method will affect existing applications must be considered in determining whether the preferred distribution should be implemented fully, partially, or at all.

Chapter 2. Implementing Your Design

After determining the design of your database, you can create objects to implement your design. These objects include dbspaces, tables, views, and indexes.

This chapter discusses the following topics:

1. Database Storage Concepts

   This section provides an overview of the physical database and explains the relationships between objects, dbspaces, and storage pools.
2. Database Generation

   When you create a database, its potential storage capacity is defined. You must do some planning to ensure that the database satisfies your data storage requirements.
3. Defining Dbspaces

   The task of defining dbspaces, which contain tables, views, and indexes, involves reserving logical space in the database, assigning the dbspace to a storage pool, and setting usage parameters. You must understand what these parameters are and how to select them so that the dbspace will best accommodate the data to be stored in it.
4. Creating Tables

   Information is stored in a database by placing it in *tables*. You must know how to create tables and how to define referential constraints.
5. Creating Views

   After you create tables, you can create *views*. A view is a logical, or virtual, table that is derived from one or more tables or other views. Using views can be advantageous in applications that have specific requirements for data tables.
6. Creating Indexes

   Indexes are optional: they improve the speed with which table rows are accessed.
7. Using the Catalog in Database Design

   The catalog tables contain information about the existing structure of the database, which can be helpful in database design.

Storage Concepts

A DB2 Server for VSE & VM database is a collection of user data objects (tables and indexes) and supporting information maintained by the database manager for that data. The supporting information includes control information (such as how each data table is formatted and where each is located), and data recovery information (restoring data to an earlier state). The database is composed of:

- A *Directory*: In VM this is a minidisk that contains database control information. In VSE it is a VSAM data set. It includes mappings of the dbspaces to their addresses on the DASD (that is, it relates the logical database image to the physical storage used).
- One or two *Logs*: In VM, these are minidisks and in VSE, these are VSAM data sets. These contain information about the changes made to the data. If any changes must be “undone” or “redone”, logs can be used to restore the data to its proper state.

- One or more *Storage Pools*: In VSE these are collections of VSAM data sets, and in VM these are collections of minidisks. Each is called a database extent or dbextent.

A dbextent is an allocation of actual DASD space. Storage pools are composed of one or more dbextents. The size of the storage pool can be increased or reduced by:

- adding more dbextents
- deleting existing dbextents
- In VM/ESA, moving dbextents to other devices.

Note: In VSE, each dbextent is the primary allocation of a VSAM data set (CLUSTER).

Storage pools can be defined to be either *recoverable* or *nonrecoverable*. The default is for them to be recoverable, whereby every change made to the pool is logged. For nonrecoverable storage pools, there is limited recovery; the database manager does not log updates, but takes a checkpoint for each logical unit of work (LUW) to ensure that the LUW's changes are written to DASD.

To maintain referential integrity, both tables in any referential constraint must be in either recoverable or in nonrecoverable storage pools: they cannot be spread across both types. This restriction is necessary because the portion of the relationship in the nonrecoverable pool might be lost, possibly invalidating the information remaining in the recoverable one. For more information about storage pools, refer to either the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual.

When a table is created, it must be assigned to a logical allocation of storage called a dbspace. The table creator can either do this assignment explicitly, or let the database manager use a default assignment. Any indexes created on that table will be stored in the same dbspace.

Figure 15 on page 19 shows how tables are stored in the database. It includes two tables and their indexes in dbspace A, two tables and their indexes in dbspace B, and one table with three indexes in dbspace C.

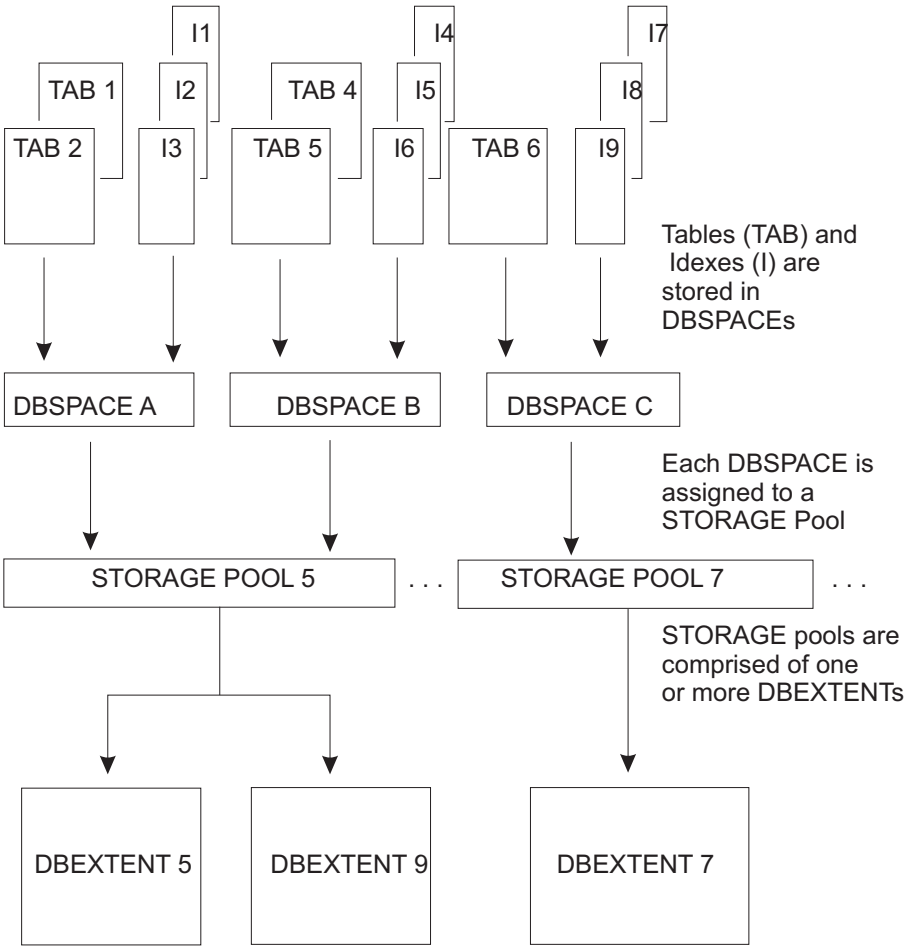


Figure 15. Table Storage in a Database

How Information is Stored in Dbspaces

A dbspace is not a real allocation of DASD space: it is a logical allocation of page map tables in the directory that relates logical dbspace pages to DASD locations. It holds data in 4096-byte blocks called *pages*, and can hold up to 255 tables, and their indexes. As dbspaces are assigned to the storage pool and their pages are filled, the physical DASD pages used are taken from the dbextents of the storage pool.

The database manager dynamically allocates real DASD storage space to support dbspace pages on a demand basis. Unused pages of a dbspace do not occupy DASD space. The potential capacity of a dbspace is fixed when it is defined.

The dbspace used to hold a table is determined when the table is created. A table cannot span (reside in) multiple dbspaces. However, two or more tables in a referential relationship may reside in separate dbspaces.

Figure 16 shows how information is stored in a dbspace.

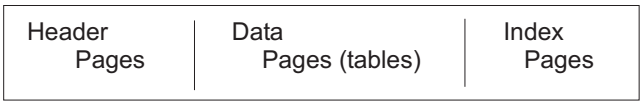


Figure 16. Table and Index Storage in a Dbspace

At the front of every dbspace are one to eight *header pages*, which contain control information about the tables and indexes stored in it. After the header pages are the *data pages*, which is where the rows of a table are stored. Index entries are stored in *index pages* at the back.

When you store multiple tables in the same dbspace, the database manager might store rows from different tables on the same data pages; however, it never puts index entries from different indexes on the same page.

Database Generation

This book does not describe how to create a database. That is the task of the system administrator, and is discussed in the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals. Because initial DASD allocations are assigned and the potential capacity for the database is established during that process, it is important that you analyze your storage requirements and inform the person responsible for generating the database. The information you provide should include the:

- Number of tables and views (objects) you intend to create
- Structure of those objects (such as number of columns, data type)
- Storage required for your objects.

Defining Dbspaces

Before defining a new dbspace, check to see if there are any already available having the properties that you require; if there are, you do not need to define a new one.

If you need to define one or more dbspaces, do the following:

1. Identify your requirements.

   Identify the data that the dbspace will contain and the way that it will be used.
2. Add the dbspace to the database.

   Add the dbspace to the database directory (if this has not already been done), using either the SQLADBSP EXEC in VM, or the ADD DBSPACE statement in VSE.
3. Acquire the dbspace.

   After a dbspace is established, enter the ACQUIRE DBSPACE statement to acquire it for your use.

Identifying Dbspace Requirements

To identify dbspace requirements, consider the tables that are to be stored and the way they will be used. If performance is a requirement, you can define a dbspace to support only one table and its indexes; often, however, dbspaces are defined to support several tables. Tables that have common requirements can be stored in the same dbspace.

Mapping Tables to Dbspaces

Table 4 shows the approach you should use for determining the way to map tables to dbspaces.

Table 4. General Approach to Mapping Tables to Dbspaces

Table Access	Type of Dbspace	Type of Data
Private tables	PRIVATE dbspaces (one per user, or user-application area)	End user data Application development data Data prototyping tables
Shared tables	PUBLIC dbspaces (one per user group, or table group)	Common end user data Application testing data Production application data

Dbspaces come in two types: PRIVATE and PUBLIC.

For private data, reserve one PRIVATE dbspace for each user. Private data is always locked at the dbspace level to eliminate unnecessary locking overhead when users are accessing their own private data.

Data kept in a PRIVATE dbspace can be shared, and concurrent read-only access to the data is possible.

For most users, one PRIVATE dbspace is sufficient; however, people doing application or data design for different application areas might want one for each area. Others might request additional storage as their data requirements grow. For these users, you can reserve additional PRIVATE dbspaces as needed.

For data that is to be shared, use PUBLIC dbspaces. These can be locked either at the row, page, or dbspace level. Thus, several users can access data at the same time. (See "Determining the Lock Size (LOCK)" on page 26.)

PUBLIC dbspaces support tables shared by a group of users. For example, a group of query users may have to share data. Rather than having each user keep a copy of the data, the extracted data could be directed to tables in a PUBLIC dbspace, where it could be accessed by all users.

For production application data, you should define one or more PUBLIC dbspaces, depending on logical groupings of tables. For further information on placing tables into dbspaces, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

Adding Dbspaces to the Database

To add a dbspace to a database you must reserve page tables in the directory, assign the dbspace to a storage pool, and specify the dbspace's type. These functions are described in the *DB2 Server for VSE System Administration* and *DB2 Server for VM System Administration* manuals.

Do not use SYS as the first three characters of a dbspace name; SYS denotes a dbspace reserved for database manager use.

Note: When you add dbspaces, you must be in single user mode.

Acquiring Dbspaces

After you have identified the mapping of tables to dbspaces, and the dbspaces have been added to the database, you can acquire them for use. Begin this process by identifying the parameters to be established for each dbspace. Table 5 summarizes these parameters; they are discussed in detail below.

Table 5. Derivation of Dbspace Parameters

Parameter	Derivation
Type	PUBLIC or PRIVATE, based on expected usage of tables.
SIZE (PAGES)	Sum of the potential sizes of each of the tables, plus the sum of the index size requirements, plus free space considerations.
STORPOOL	Consider device utilization of other dbspaces in the same pool and the availability of space in the pool. Also consider using nonrecoverable storage pools for read-only data.
NHEADER	Set based on the number of tables and indexes to be put in the dbspace.
PCTFREE	Set based on growth potential of the tables to be put in the dbspace.
PCTINDEX	Set based on the potential indexes to be created and their estimated sizes.
LOCK	Set based on the size of tables and the extent of their use.

Use the ACQUIRE DBSPACE statement to specify the parameters in Table 5. When acquiring a dbspace, you must specify whether it is to be PUBLIC or PRIVATE, and you can optionally set the number of pages in it, the level of recovery, the percentage of space to be reserved for updates and indexes, and the amount to be locked when accessed by users. See the *DB2 Server for VSE & VM SQL Reference* manual for more information on the ACQUIRE DBSPACE statement.

Determining Dbspace Type (PUBLIC or PRIVATE)

If any table is to be accessed by multiple users at the same time, and any one of the users will be doing UPDATEs, INSERTs, or DELETEs, then it should be placed in a PUBLIC dbspace. You need Database Administrator (DBA) authority to acquire a PUBLIC dbspace.

Only users with DBA or RESOURCE authority can create objects in PUBLIC dbspaces.

To acquire a PUBLIC dbspace, enter the ACQUIRE DBSPACE statement specifying your requirements. For example, to acquire a PUBLIC dbspace named *payroll* and using the defaults, enter:

```
ACQUIRE PUBLIC DBSPACE NAMED PAYROLL
```

You need DBA or RESOURCE authority to acquire a PRIVATE dbspace.

Only the *owner* of the PRIVATE dbspace, or a user with DBA authority, can create objects in the dbspace.

Every PRIVATE dbspace has an owner. To acquire the PRIVATE dbspace PERSONAL for user JOHN, enter the following:

```
ACQUIRE PRIVATE DBSPACE NAMED JOHN.PERSONAL
```

You cannot use the ALTER DBSPACE statement to change the type of a dbspace after you acquire it.

Determining the Size of the Dbspace (PAGES)

You need to ensure that the dbspace contains enough pages to hold the tables and associated indexes to be stored there.

The size of the dbspace should be based on the estimated current size of the tables and their indexes, plus an allowance for their expected growth. A dbspace cannot contain less than 128 pages. You must allocate pages in multiples of 128, otherwise the number is rounded up to the next highest multiple of 128. Algorithms for determining the number of pages needed are described in "Appendix A. Estimating Your Dbspace Requirements" on page 215.

Because you cannot extend a dbspace after it is defined, you should overestimate the required number of pages. Unused pages are not stored, so the cost of overestimating is nominal. In contrast, the cost of underestimating pages can be quite expensive because of the reorganization activities required to re-establish the data in a larger dbspace later.

Note: Two directory blocks of 512 bytes each are used for every 128 data pages defined.

Determining the Storage Pool (STORPOOL)

Storage pools come in two types: recoverable and nonrecoverable.

Consider assigning a dbspace to a nonrecoverable storage pool if the data in it will be read-only. Changes made to data in a nonrecoverable storage pool are not logged, which offers the advantages of requiring less log space, elapsed time, and CPU time. (There should be an alternative method of recovery available, such as reloading the storage pool.) The disadvantage is that data cannot be recovered when media failures occur (which may be acceptable for read-only data).

If you are using referential integrity, you must use recoverable storage pools. For nonrecoverable storage pools, ROLLBACK is not performed and no logging is in effect, so that some operations can be neither completed successfully nor rolled back. Each operation containing a referential constraint is verified when it occurs. If a row of a multi-row operation violates the referential constraint, the operation terminates. The rows that were affected prior to the termination cannot be rolled back.

For example, in a multi-row delete of a parent table, if 15 rows are candidates for deletion and the ninth row violates the DELETE RESTRICT rule, then the first eight rows would be deleted and the operation would cease with the ninth row. The integrity of the table would be maintained but the operation would be only partially completed.

Because a unit of work modifying both recoverable and nonrecoverable pools can only ROLLBACK the recoverable pool, referential constraints cannot be created between the two types of pools.

You cannot use the ALTER DBSPACE statement to change the storage pool of a dbspace after you acquire it.

If you do not specify the STORPOOL parameter, a dbspace of the correct size and type will be acquired from any recoverable storage pool.

Storage Device Considerations: The storage pool you select should be chosen to:

- Balance device utilization

- Exploit device characteristics for data in the dbspace.

A table resides on the devices used to support the storage pool to which the table's dbspace is assigned. Consider storing different tables on different devices based on device characteristics and table usage. To do this, you need multiple storage pools and multiple dbspaces.

For example, if you have two tables that are highly active, you can reduce potential device contention by storing them in different dbspaces that are assigned to different storage pools. The dbextents defined for the two storage pools would be on different devices.

You could use a similar technique for storing selected tables on higher or lower speed devices as appropriate.

For more information about storage pools, refer to either the *DB2 Server for VM System Administration* or the *DB2 Server for VSE System Administration* manual.

Determining the Number of Header Pages (NHEADER)

Header pages contain control information on the tables and indexes stored in the dbspace.

The number of header pages required depends on the number of objects to be stored in the dbspace. Generally, taking the default (8 pages) is recommended, as this gives you the most flexibility at nominal cost. However, if you plan to have few tables or indexes in the dbspace, you may allocate fewer. You must allocate at least one.

To estimate your needs, see "Appendix A. Estimating Your Dbspace Requirements" on page 215.

You cannot change the NHEADER parameter with the ALTER DBSPACE statement; after you set it, the only way to change it is to move all the data in the current dbspace to another dbspace having the required NHEADER value (see "Altering the Design of a Dbspace" on page 73).

Determining the Percent Free Space Desired (PCTFREE)

The PCTFREE parameter refers to the percentage of each page that is to be reserved 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 by reducing the PCTFREE value through an ALTER DBSPACE statement.

The PCTFREE value 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 and non-nullable in the tables. Here, you would set PCTFREE 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 high value. 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.

- High Insert/High Update Activity

  In this situation, set PCTFREE to a high value and then lower it. This would allow space for use by both update and insert activities.

Note: Updating refers to the replacement of a row of data into the same location in a page of a dbspace, unless the row can no longer fit because of an increase in the size of one or more columns. The replacement row is placed on the same page of the dbspace if there is still sufficient space available in the area set aside using PCTFREE.

In situations where there is high insert activity, consider using a clustering index. The first index created on a table is always considered the clustering index. A clustering index determines the placement of rows in pages of a dbspace to minimize DASD I/Os when the table rows are accessed in the index sequence. For more information, see "Clustering Rows of a Table on an Index" on page 48.

Note: Clustering refers to the grouping or gathering of items; in the above case, the grouping of table rows is done according to the indexes.

If an updated row no longer fits on its original page, its contents are moved to the next available page with enough room to accommodate it. Continual movement of row contents to new pages as a result of this overflow may lead to a decrease in performance as the database manager must make one additional page reference before locating a row's contents.

The database manager typically reserves **more** space than you specify. For an explanation of free space management design, see the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual. Calculate PCTFREE using the following formula:

```
PCTFREE = (FREEBYTES - AVGROWLEN) / 40
```

where FREEBYTES is the number of bytes you want reserved on each page, and AVGROWLEN is the average row length for tables in the dbspace. If you have modeled the tables to be stored in the dbspace, you can obtain a value for AVGROWLEN for each of the tables from SYSTEM.SYSCATALOG.

For normal processing, set PCTFREE somewhere between:

```
[AVGROWLEN / 40] and [50 - (AVGROWLEN / 40)].
```

Setting it below the lower limit would mean the unused bytes could not be used (the average row would not fit) and the space set aside for updates would be wasted, while setting it greater than the upper limit may restrict you unnecessarily to one row per page.

For more information on how the PCTFREE parameter determines actual reserved bytes, see "Appendix A. Estimating Your Dbspace Requirements" on page 215.

Determining the Percentage for Index Pages (PCTINDEX)

When you acquire a dbspace, you must reserve some portion of it for holding indexes on the tables in the dbspace. PCTINDEX reserves the amount of space in the directory to be formatted for this purpose. Under most circumstances, you should let this value default to 33 percent. With this default, there are

approximately twice as many data pages for holding table rows as there are index pages for holding indexes on the tables. You can create or drop indexes at any time (these functions can be performed online); so do not constrain the potential indexing you might want to do by specifying a lower value for PCTINDEX. There are two cases when you might want to consider overriding the default:

- Read-only data

  Some data is used exclusively, or primarily, for read-only (SELECT) access. You can create a more than one index on such data to improve the performance of a wide variety of user queries. The indexes are created after the data is loaded and are referenced as required by a query. Because the data is not subject to update operations, you do not have to worry about the performance implications of index maintenance. Thus, you should consider specifying a high value for PCTINDEX. To do this, estimate the number of index pages that would be required for various indexes that might be created on these tables in the dbspace. See "Estimating the Number of Index Pages" on page 227.
- Highly tuned operational data

  This is data that is subject to frequent updates, and the performance requirements limit the amount of indexing you want to do on the tables. Determine the set of indexes you require for the data and set the PCTINDEX parameter accordingly.

You establish the PCTINDEX parameter with the ACQUIRE DBSPACE statement. You cannot change the PCTINDEX parameter with the ALTER DBSPACE statement; after you set it, the only way to change it is to move all the data in the current dbspace to another dbspace having the required PCTINDEX value (see "Altering the Design of a Dbspace" on page 73).

Determining the Lock Size (LOCK)

When you acquire a PUBLIC dbspace you can specify three levels of locking: DBSPACE, PAGE, or ROW. You can change the lock size later with the ALTER DBSPACE statement.

The lock size can be set for PUBLIC dbspaces only. (PRIVATE dbspaces are always locked at the DBSPACE level.)

The default lock size is PAGE. Select ROW if the dbspace is to contain a small table that will fit on a small number of pages, and it is expected that this table will be frequently updated by multiple users.

Locking the dbspace at the row level also causes indexes in it to be locked at the key level. (Usually indexes are locked at the page level.) Key-level locking for indexes, like row-level locking for tables, reduces contention but adds overhead.

Retrieving Information about Dbspace Parameters

Information about the dbspace parameters is maintained in the SYSTEM.SYSDBSPACES catalog table.

Example

Use the following query to retrieve information about dbspace MYDB:

```
SELECT DBSPACENO, DBSPACETYPE, POOL, NPAGES,
       NRHEADER, PCTINDX, FREEPCT, LOCKMODE
  FROM SYSTEM.SYSDBSPACES
  WHERE DBSPACENAME = 'MYDB'
```

To see how many header, data, and index pages are being used in a given dbspace, issue the SHOW DBSPACE operator statement from either the database console or from ISQL. (Its format is described in the *DB2 Server for VSE & VM Operation* manual.) This information may be helpful, especially before attempting to load large amounts of data into a dbspace.

Restrictions on the ACQUIRE DBSPACE Statement

To acquire a dbspace, it must have already been added to the database. When you issue the ACQUIRE DBSPACE statement, the database manager searches for a dbspace with the appropriate size (number of PAGES), storage pool assignment, and type (PUBLIC or PRIVATE). If one of the requested size cannot be found, the next **largest** suitable one will be used. (This could result in a very large dbspace being used to contain a small amount of data.) If no existing dbspace satisfies the requirements, then the ACQUIRE DBSPACE statement will fail, and you will have to add additional dbspaces to the database.

The SYSDBSPACES system catalog table contains information about dbspaces. You can issue an ISQL query to retrieve this information.

The following query yields information on the type and size of all available dbspaces (those that have been added but not yet acquired):

```
SELECT DBSPACETYPE, NPAGES
  FROM SYSTEM.SYSDBSPACES
  WHERE DBSPACENAME=''
```

The value of DBSPACETYPE is 1 for PUBLIC dbspaces and 2 for PRIVATE ones.

Creating Tables

Relational databases use tables to store information. This section explains how to create tables and how to define referential and unique constraints in the DB2 Server for VSE & VM environment.

Controlling Who Creates Tables

Designing tables to be used by many applications is a critical task. Although you can add columns and use views to mask certain changes, generally you cannot change the design of a table after it has been implemented without disrupting applications. Table design is difficult because there are many ways to represent the same information, and often you have to decide between the conflicting objectives of logical design and physical design. (One example of such a conflict is normalization, discussed in "Step 7: Normalize Your Tables" on page 9.)

If you have DBA authority, you will probably want to keep the responsibility for creating tables, and then pass the authorization for their use on to the application developers. However, you can grant authority for creating tables to others; or, if some users want to use the application server with minimum assistance or control, you can acquire PRIVATE dbspaces for them and authorize them to create whatever data objects they need, including tables.

How to Create Tables

After designing a table, issue the CREATE TABLE statement. Creating a table involves:

- Naming it
- Naming the columns within it
- Defining the appropriate data type for each column

- Defining primary keys
- Defining the relationships between tables
- Defining unique constraints.

To create a table, the connected user must have the proper authority (see "Chapter 5. Providing Security" on page 87). Whoever issues the CREATE TABLE statement has complete authority over the table.

When you create a table, a definition of it is recorded in the catalog; no application data is stored. (For a description of how to put data into the table, see "Loading Data into Tables" on page 59.)

Figure 17 shows the statement used to create the sample EMPLOYEE table.

```
CREATE TABLE JOHN.EMPLOYEE
  (EMPNO     CHAR(6)     NOT NULL,
   FIRSTNME  VARCHAR(12) NOT NULL,
   MIDINIT   CHAR(1)     NOT NULL
   LASTNAME  VARCHAR(15) NOT NULL,
   WORKDEPT  CHAR(3),
   PHONENO   CHAR(4),
   HIREDATE  DATE,
   JOB       CHAR(8),
   EDLEVEL   SMALLINT    NOT NULL,
   SEX       CHAR(2),
   BIRTHDATE DATE,
   SALARY    DECIMAL(9,2),
   BONUS     DECIMAL(9,2),
   COMM      DECIMAL(9,2),
   PRIMARY KEY (EMPNO),
   FOREIGN KEY EMPFKEY (WORKDEPT)
     REFERENCES DEPARTMENT ON DELETE SET NULL)
 IN PUBLIC.SAMPLE
```

Figure 17. Example of CREATE TABLE. A foreign key cannot be defined unless the corresponding primary key already exists.

This example creates a table called EMPLOYEE, which has 14 columns, by a creator with the ID JOHN. The table uses the column EMPNO as the primary key, and the column WORKDEPT as a foreign key called EMPFKEY, which references WORKDEPT in the DEPARTMENT table. The delete rule is SET NULL, and the table resides in the "PUBLIC".SAMPLE dbspace.

Naming Tables

A table name can be up to 18 characters long (18 bytes). Table names that are not explicitly qualified by the creator name in the CREATE TABLE statement are qualified by the database manager. For example, assume that a user with an ID of SMITH is entering SQL statements interactively. If SMITH creates a table named ABC, with no qualifier, the table name becomes SMITH.ABC. SMITH can own only one table, view, or synonym called ABC. A different user ID, JONES, can create another table, view, or synonym called ABC, which will become JONES.ABC.

If the DBCS option is enabled, you can use DBCS characters in table names (the 18-byte length restriction still applies). Enabling the DBCS option is discussed in the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals.

Choosing Columns

You implement your database design primarily by choosing the columns that make up each table. Almost inevitably, there is some conflict between the theoretical design and the most practical implementation, as described in the following sections.

Considerations for Normalization

In "Step 7: Normalize Your Tables" on page 9 normalization was discussed only from the viewpoint of logical database design, without considering performance. Consider the example there of the column that contains the addresses of warehouses. The column is first shown as part of a table that contains information about parts and warehouses; then, to further normalize the design, it is removed from that table and defined as part of a table that contains information only about warehouses. The other possible design (in which the column is part of both tables) was not considered.

Some applications might require information about both parts and warehouses, including the addresses of warehouses. With normalization, information can be retrieved by joining tables. The problem is that a join operation can be very time-consuming, even for only two tables, and as the number of tables increases the access costs can increase enormously, depending on the size of the tables and the available indexes. If indexes are not available, the join of many large tables can take hours. Furthermore, the number of tables that can be joined is at most 15 and, depending on the complexity of the statement, can be significantly less. Thus, an unnormalized design may be absolutely necessary.

Consider making both tables have a column that contains the addresses of warehouses. If this design makes join operations unnecessary, it could be a worthwhile redundancy. Warehouse addresses do not change often, and if one does change, DB2 Server for VSE & VM makes it easy to update all occurrences.

Considerations for Row Size

Rows are stored within pages. A single row cannot occupy more than one page, and you cannot create a table with a maximum row size that is greater than the page size. One exception is that columns of type LONG VARCHAR or LONG VARGRAPHIC can be longer than one page; therefore, the rows that contain them can occupy more than one page. There is no other absolute limit, but if you ignore row size in favor of implementing a good theoretical design, you may waste storage.

Row Length—Fixed or Varying: Table rows may be of fixed or varying lengths. Two considerations apply:

- The presence of any columns with varying-length data types will result in a varying-length row.
- If the rightmost columns of the row are defined as allowing nulls, and if no values for those columns are supplied when a row is inserted, storage is not allocated for those columns. If those columns in the inserted row are subsequently updated, the row length will be increased to accommodate the non-null column values.

The disadvantage of varying-length rows is that if the row length is increased, the row may have to be repositioned. If the row is repositioned and there is not enough free space on the current page to accommodate the row, then the row will be moved to another page. In this case, whenever that moved row is accessed, an additional page reference is required.

Row Lengths and Pages: Along with the bytes of actual data, each row has:
- A 6-byte prefix
- A 2-byte slot for each row stored in the page
- 1 additional byte for each column that may contain null values
- 1 additional byte for each varying-length column.

In addition, every data page has a 16-byte header.

This overhead affects the amount of data that can be stored on each page in your dbspace. In designing your table, consider your design needs while looking for ways to store your data as efficiently as possible.

Some Space-Wasting Designs: Space is wasted in a dbspace if all its rows are slightly longer than half a page, because then only one row can fit in each page. If you can reduce the row lengths to just under half a page, you will need only half as many pages. Similar considerations apply to rows that are just over a third of a page, a quarter of a page, and so on.

It is particularly important to minimize the number of pages in a dbspace because if an index is not used, the database manager will read every active page of the dbspace.

For example, suppose you design a table to hold a large array of floating-point numbers. If you define each column as FLOAT and use the maximum number of columns (255), the row length is 2048 and only one row fits on each page. If you use 240 columns, two rows could fit on each page, and a page would contain 480 floating-point numbers, rather than only 255.

Specifying Columns

A column contains all occurrences of one of the entities in a table. (You can think of it as a field in a row.) In Figure 17 on page 28, the lines immediately following the table name contain the names of the columns within the table. In the sample EMPLOYEE table, the HIREDATE column contains all the hire dates for all employees represented by EMPNO. You cannot redefine or overlap columns and, after you have implemented the design of your database, you usually cannot change a column definition without disrupting applications. Therefore, consider carefully the decisions you make about column definitions. (However, you can add columns to an existing table. See "Altering the Design of a Table" on page 64.)

For each column, you must specify a name and a data type.

For each column, you may specify:
- A length (of values in the column, not the number of values) and whether null values are permitted. For a column containing character data, you can also specify the subtype. For further information, refer to "Specifying SBCS, Mixed, or Bit Subtypes" on page 34.
- A CCSID for a column with character or graphic data, if you want to override the default CCSID. For further information, refer to "Specifying a CCSID" on page 34.
- Whether you plan to run a user-written exit routine whenever a program enters or retrieves data in the column. This type of routine, called a *field procedure*, can be used, for example, to alter the sorting sequence of values entered in the column. Field procedures are assigned to specific columns when the table is created or altered. For further information on field procedures, see "Specifying a FIELDPROC" on page 35.

Column Names

Column names must be unique within a table, but you can use the same name in different tables. The maximum length is 18 bytes.

If the DBCS option is enabled, you can use DBCS characters in the column names. See the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual.

Nulls

As mentioned under "Step 3: Provide Column Definitions for Tables" on page 4, some columns cannot have a meaningful value in every row.

A special value indicator, called the *null value*, represents an unknown or missing value. It should not be confused with a zero value, a blank, or an empty string: it is a special value interpreted by the database manager to mean that no data has been supplied.

Unless you specify otherwise, any column you define can contain null values, and rows can be created in the table without providing a value for the column. Avoid using nulls for columns that will be used as indexes. To disallow null values, use the NOT NULL clause, and provide a non-null value for that column whenever you store data. Columns that will be referenced in a primary key or unique constraint must be defined as NOT NULL.

If you add a column to an existing table, it contains no data and so cannot be defined as NOT NULL.

In the example in Figure 17 on page 28, nulls are acceptable for certain columns and prohibited for others.

Before you decide whether to allow nulls for unknown values in a column, be aware of how nulls can affect the result of a query.

- Nulls in predicates

  Nulls do not satisfy any condition in an SQL statement other than the special NULL predicate. Null values do not act like other values. For instance, if you try to determine whether a null value is larger or smaller than a given known value, you get an answer of UNKNOWN.
- Nulls in quantified predicates

  If either the left side or the subselect list of a quantified predicate is null, the quantified predicate is residual. Residual predicates require more processing because of the communication between the Relational Data System (RDS) and the Database Storage Subsystem (DBSS). Predicate processing is described in the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.
- Nulls with Field Procedures

  If you allow nulls in a column with a field procedure, that field procedure is not invoked when you access a null value: the database manager returns the null value.

Specifying Data Types

You must give a data type for each column of a table, to specify the type of data the column will contain and the length of the data field.

The first thing you must decide when defining a column is what kind of data the column will contain—string, numeric, or date/time. The decision is often obvious

because only a string column can contain letters or special characters. If the data consists solely of digits, however, you have to decide whether to specify it as string or numeric data. And if the values represent dates, times, or timestamps, you will want to consider the data types DATE, TIME, and TIMESTAMP.

Numeric Data Types

The data types for numbers are shown in Table 6.

Table 6. Numeric Data Types

Data Type	Denotes a column of...
SMALLINT	Small integers. A small integer is an IBM System/370* halfword signed binary integer of 16 bits; the range is -32,768 to +32,767.
INTEGER or INT	Large integers. A large integer is an IBM System/370 fullword signed binary integer of 32 bits; the range is -2,147,483,648 to +2,147,483,647.
REAL or FLOAT(*n*)	Single precision floating-point numbers. *n* must be in the range 1 through 21. There is no default; if you omit *n* when declaring a data type of FLOAT, the column has double precision. A single precision floating-point number is an IBM System/370 short floating-point number of 32 bits.
FLOAT, FLOAT(*n*), or DOUBLE PRECISION	Double precision floating-point numbers. *n* must be in the range 22 through 53; its default is 53. A double precision floating-point number is an IBM System/370 long floating-point number of 64 bits. The range of magnitudes for floating-point numbers of either type is approximately ±5.4E-79 to ±7.2E+75.
DECIMAL(*p*,*s*), DEC(*p*,*s*) or NUMERIC (*p*,*s*)	IBM System/370 packed decimal numbers with precision *p* and scale *s*. The precision *p*, which is the total number of digits, must be greater than 0 and less than 32. The scale *s*, which is the number of digits in the fractional part of the number, must be greater than or equal to 0 and less than or equal to the precision. *s* may be omitted; its default is 0. And if *s* is omitted, *p* may also be omitted; its default is 5. The range of decimal values is 31 digits, and these values can be positive or negative. NUMERIC and DEC are synonymous with DECIMAL.

For integer values, SMALLINT or INTEGER (depending on the range of the values) are preferable to DECIMAL or FLOAT.

For real numbers with a small precision and scale, DECIMAL is preferable to FLOAT.

For numeric data, use numeric rather than string columns for the following reasons:

- They require less space.
- They permit arithmetic operations.
- They are accessed more efficiently. For example, if numbers are represented as strings, when the database manager calculates a range, the optimizer takes into consideration all possible bit patterns and cannot calculate an appropriate filter factor. Because of this, a much higher number of rows is returned. For further information on filter factors, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

String Data Types

The data types for strings are shown in Table 7.

Table 7. String Data Types

Data Type	Denotes a column of...
CHAR(*n*) or CHARACTER(*n*)	Fixed-length character strings with a length of *n* bytes. *n* must be greater than 0 and less than 255.
VARCHAR(*n*)	Varying-length character strings with a maximum length of *n* bytes. *n* must be greater than 0. If *n* is greater than 254, certain restrictions apply to the use of the columns in SQL statements. The upper limit on the value of *n* is 16,383.
LONG VARCHAR	Varying-length character strings with a maximum length of 32,767 bytes. The restrictions that apply to VARCHAR columns where n>254 also apply to LONG VARCHAR columns.
GRAPHIC(*n*)	Fixed-length graphic strings containing *n* double-byte characters. *n* must be greater than 0 and less than 128.
VARGRAPHIC(*n*)	Varying-length graphic strings. The maximum length, *n*, must be greater than 0. If *n* is greater than 127, certain restrictions apply to the use of the column in SQL statements. The upper limit on the value of *n* is 16,383.
LONG VARGRAPHIC	Varying length graphic strings with a maximum length of 16,383 bytes. The restrictions that apply to the use of a VARGRAPHIC column where n>127 also apply to a LONG VARGRAPHIC column.

If you want to use a field procedure with a column, the column must have a short string data type. You can also use string columns to specify binary (bit) data or character data for exchange with other application servers.

Choosing Fixed-Length or Varying-Length Data Types: VARCHAR saves DASD space. The saving is at the cost of a 1-byte overhead for each value and the additional processing required for varying-length rows. Thus, CHAR is preferable to VARCHAR, unless the space saved by the use of VARCHAR is significant. The

saving is not significant if the maximum length is small or the lengths of the values do not have a significant variation.

If you use VARCHAR, do not specify a maximum length greater than necessary. In particular, note the restrictions on columns of strings longer than 254 bytes; for example, they cannot be indexed.

The database manager will not use index-only access to retrieve the data if the index is created on a VARCHAR column. For information on index-only access, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

Do not use LONG VARCHAR unless you really want the maximum row length to be as large as possible, because there is a higher cost associated with accessing long fields.

In most cases, the content of the data intended for a column dictates the data type you choose. For example, the data type selected for the department name (DEPTNAME) of the DEPARTMENT table is VARCHAR(36). Because department names normally vary considerably in length, the choice of a varying-length data type seems appropriate. Choosing a data type of CHAR(36), for example, would result in much wasted space, because all department names, regardless of their length, would be assigned the same amount of space (36 bytes).

The foregoing considerations about CHAR, VARCHAR, and LONG VARCHAR columns apply in the same way to GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC columns. The one exception is that the length (*n*) of a GRAPHIC or VARGRAPHIC column is given as a number of double-byte characters; hence, the length in bytes is twice *n*.

Specifying SBCS, Mixed, or Bit Subtypes: The use of subtypes applies only to character data such as CHAR, VARCHAR, and LONG VARCHAR. A default subtype for character columns is set at installation time. You can override this default for any column in a table when the table is created (or when a column is added to an existing table).

Choose the SBCS subtype when the data in the column is single-byte character data and the default is not.

Choosing FOR MIXED DATA lets you store (and to have the column flagged as storing) both single- and double-byte characters. The database manager ensures the integrity of valid mixed data during truncation.

For columns that contain binary data that should not be modified when moved between different environments (such as from ASCII to EBCDIC), specify `FOR BIT DATA`.

Note: When specifying a subtype, you are also implicitly specifying the CCSID for the subtype.

Specifying a CCSID: Default CCSID values for character and graphic data are specified during installation. To override the CCSID used for a column containing any of these data types, specify one of your own.

Each CCSID is associated with either graphic data or a specific subtype of character data. Query the SYSTEM.SYSCCSIDS system catalog table to determine the CCSID values for each of these.

If you compare data from two columns or move data between two columns having different CCSIDs, and if a conversion selection table exists, the data in one of the columns is converted to ensure a consistent comparison. Query the SYSTEM.SYSSTRINGS catalog table for a list of valid conversion selection tables. (In VM you can also look at the ARISSTR MACRO on the production minidisk for a list of valid conversion selection tables.) Consider your users' environments and needs when specifying a CCSID for a particular column. When you override the default CCSID for a column of data, you can minimize the amount of converting done on tables that are accessed primarily by users requiring different CCSIDs.

Note: Converting from one CCSID to another, then another, and then returning to the original CCSID, can result in the misinterpretation of data if there is not a one-to-one correspondence between the two sets of characters.

See the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual for more information about specifying CCSIDs.

Specifying a FIELDPROC: A field procedure (FIELDPROC) is a user-written exit routine used to encode and decode values in a character string. Field procedures can only be used on short character strings (CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC).

A field procedure can be used to alter the sorting sequences of a short character string column. It is assigned to a column during execution of the CREATE TABLE or ALTER TABLE statement, and is called whenever values in the column are changed, inserted, or retrieved. To specify that a column use a field procedure, use the FIELDPROC option followed by the program name of the procedure and, optionally, a list of parameters.

For example, to specify a field procedure for the column LASTNAME of the EMPLOYEE sample table, change one line of Figure 17 on page 28 to look like this:

```
LASTNAME VARCHAR(15) NOT NULL FIELDPROC MYPROG (4, 3, 7),
```

In the example, the name of the field procedure is chosen as MYPROG. The parameters 4, 3, and 7 are passed to the procedure when it is invoked by the CREATE TABLE or ALTER TABLE statement.

For more information about field procedures, see the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual.

Data Types for Dates, Times, and Timestamps

The data types for dates, times, and timestamps are shown in Table 8.

Table 8. Date, Time, and Timestamp Data Types

Data Type	Denotes a column of...
DATE	Dates. A **date** is a three-part value representing a year, month, and day in the range 0001-01-01 to 9999-12-31.
TIME	Times. A **time** is a three-part value representing a time of day in hours, minutes, and seconds, in the range 00.00.00 to 24.00.00.
TIMESTAMP	Timestamps. A **timestamp** is a seven-part value representing a date and time by year, month, day, hour, minute, second, and microsecond, in the range 0001-01-01-00.00.00.000000 to 9999-12-31-24.00.00.000000.

For a detailed description of Date/Time characteristics, see the *DB2 Server for VSE & VM SQL Reference* manual.

Advantages of Date/Time Data Types

Numbers representing dates and times can, of course, be stored in columns with numeric data types; if they include special characters as separators, they can be stored in string columns. But neither of these options provides the advantages of the DATE, TIME, and TIMESTAMP data types, as described below.

Variable Input and Output Format: Date/time values are stored in a special internal format, which is freely convertible on output or input to or from any of the formats in Table 9.

Table 9. Date Formats

Format Name	Abbreviation	Typical Date	Typical Time
International Standards Organization	ISO	1992-12-25	13.30.05
IBM USA standard	USA	12/25/1992	1:30 PM
IBM European standard	EUR	25.12.1992	13.30.05
Japanese Industrial Standard (Christian Era)	JIS	1992-12-25	13:30:05

You also have the option of supplying an exit routine to make conversions to and from any local standard. For instructions about writing and using a date or time exit routine, see the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual.

When loading date or time values from an outside source, the database manager accepts any of these formats, and convert valid input values to the internal format. For retrieval, there is a default format that you select at the time of installation. You can change the default at any time by updating the SYSOPTIONS catalog; you can override it for every statement in a program by a precompiler option, or for particular instances by the CHAR scalar function. For example, whatever your local default, the following statement displays employees' birth dates in IBM USA standard form:

```
SELECT EMPNO, CHAR(BIRTHDATE, USA) FROM EMPLOYEE
```

Date/Time Arithmetic and Durations

Date/time arithmetic involves intervals of time that are represented by numbers called *durations*. A duration is an interpretation of a number, not a data type.

A *labeled duration* is any number of years, months, days, hours, minutes, seconds, or microseconds. A *date duration* is a number of years, months, and days. A *time duration* is a number of hours, minutes, or seconds. A *timestamp duration* is a number of years, months, days, hours, minutes, seconds, and microseconds. For a further discussion of durations, see "Date/Time Arithmetic" on page 176, or the *DB2 Server for VSE & VM SQL Reference* manual.

The only arithmetic operators that can be applied to date/time values are addition and subtraction. If a date/time value is the operand of addition, the other operand must be a duration.

For example, the following statement lists employees who have been hired after the age of 40:

```
SELECT * FROM EMPLOYEE
  WHERE HIREDATE > BIRTHDATE + 40 YEARS
```

This statement lists employees who have been hired in the last 3 months:

```
SELECT * FROM EMPLOYEE
  WHERE HIREDATE > CURRENT DATE - 3 MONTHS
```

Date/Time Functions: There are functions to extract the years, months, days, hours, minutes, seconds, and microseconds of dates, times, and timestamps. For example, this statement lists all employees who have a service anniversary on June 21:

```
SELECT * FROM EMPLOYEE
  WHERE MONTH(HIREDATE) = 6 AND DAY(HIREDATE) = 21
```

There are also functions to convert dates, times, and timestamps to character or integer representations.

String Representations of Date/Time Values: In the following example, 07/28/1971 is interpreted as a date because it is compared to a date; in other contexts (a SELECT list, for example) 07/28/1971 is merely a character string.

```
SELECT * FROM EMPLOYEE
  WHERE HIREDATE = '07/28/1971'
```

Date/Time Comparisons: All comparison operators are allowed. The statement below lists all employees hired after October 31, 1979. To show another of the recognized date formats, we have arbitrarily chosen to write the date in the IBM European standard.

```
SELECT * FROM EMPLOYEE
  WHERE HIREDATE > '31.10.1979'
```

Comparing Data Types

You can compare values of different types and lengths provided that both values are numeric, both are character strings, or both are graphic strings.

Date and time comparisons **cannot** be made with values of different types: a date can be compared only with a date, a time with a time, and a timestamp with a timestamp (or, in each case, with a valid string representation of a date, time, or timestamp).

If a column uses a field procedure, values to be compared to it are first encoded by the field procedure. If a column with a field procedure is compared to another column, both columns must have the same field procedure and data type.

Columns do not have to have the same CCSID to be compared. When two columns with differing CCSIDs are compared, and a conversion selection table exists, the data in one of the columns is converted to ensure a consistent comparison. For further information, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

Specifying a PRIMARY KEY

The primary key of a table, if one has been created, consists of one or more columns that uniquely identify each row in the table. In the example in Figure 17 on page 28, the employee number is the primary key of the employee table, and the PRIMARY KEY clause identifies the column of employee numbers (EMPNO).

A table that is to be a parent of dependent tables must have a primary key—the foreign keys of the dependent tables refer to it. Otherwise, a primary key is optional. If you are defining referential constraints, read "Considerations for Referential Integrity when Creating Tables" before creating or altering any of the tables involved.

If you specify a primary key, a unique index is automatically defined on the same set of columns, in the same order as those columns. The primary key values must then be unique and cannot be null. Their uniqueness cannot depend upon trailing blanks in columns containing VARCHAR or VARGRAPHIC data. Automatic enforcement of these restrictions can be useful even if the table is not involved in referential constraints. If each row of your table does relate to a unique occurrence of some entity, then consider creating a primary key.

If the primary key is created on a VARCHAR or VARGRAPHIC column, index-only access is not used to retrieve the data. For information on index-only access, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

Specifying a UNIQUE Constraint

The unique constraints on a table ensure the uniqueness of values in columns making up each constraint. Although functionally similar to a unique index, a unique constraint can be defined when the table is created, deactivated, and then reactivated to enforce the uniqueness of values in its key. This simplifies administration when you load data or perform operations that could temporarily violate the unique constraint. For this reason, unique constraints are preferable to unique indexes, which must be individually and explicitly dropped and recreated to suspend or enforce uniqueness.

A unique constraint is also similar to a primary key in that:
- It consists of one or more columns
- The columns are not nullable
- The database manager enforces uniqueness by creating a unique index.

It differs from a primary key in that:
- It cannot be referenced by a foreign key
- You can define more than one on any table
- It can be given a name.

Considerations in Defining Unique Constraints

- The columns in a unique constraint cannot allow null values.
- You cannot duplicate a unique constraint on a table.
- The columns of a unique constraint should not be the same as columns in a primary key. The converse is also true.
- A unique constraint can be added after the table is created through the ALTER TABLE statement.
- Like primary keys and unique indexes, the uniqueness of values in a unique constraint cannot depend upon trailing blanks in columns with VARCHAR or VARGRAPHIC data.

Considerations for Referential Integrity when Creating Tables

For any table, you can define one primary key using the primary key clause, and any number of foreign keys using the referential constraint clauses. In a referential constraint, the table that has the foreign key definition is the dependent table and the table that is referenced by the foreign key is the parent.

The constraint-name identifies the key being specified. It is optional. The database manager generates a constraint-name if one is not provided; however, you should create your own for foreign keys. Constraint-names should be symbolic and indicate the parent and foreign key names, which will make working with the keys much easier. Working with keys is discussed in "Altering Referential and Unique Constraints" on page 65.

A *referential constraint* is defined by creating or altering tables to have a parent/dependent relationship between them. A referential constraint can span dbspaces. A *referential structure* is a set of tables that are related to each other by referential constraints. A dbspace may have more than one referential structure but that is generally not desirable.

Primary Key Index

When a primary key is defined, a unique index is created automatically to enforce its uniqueness. If you have not specified information such as index order and percent free space on the key definition, the index is created using default values.

When a primary key is defined by the CREATE TABLE statement, the CLUSTERING index is the one associated with the primary key. If you want to have this index on columns other than those comprising the primary key, create the table without a primary key, then create an index on the desired columns, and then use the ALTER TABLE statement to add the primary key.

If the primary key is dropped, either implicitly (when the table or dbspace is dropped) or explicitly (with the ALTER TABLE statement), the system-generated index is automatically dropped. You cannot use the DROP INDEX statement to explicitly drop an index that was created to support a primary key.

Use the ALTER TABLE ACTIVATE PRIMARY KEY statement to reorganize the primary key index if the primary key is active, or to recreate the index if the primary key is inactive. For more information about this statement, see "Altering the Design of a Table" on page 64.

Usage Notes:

- The primary key columns must not allow null values, and the primary key clause must not be used more than once.
- Corresponding columns in primary and foreign keys of the same referential constraint must have the same data type.
- The columns in a key must exist in the table, and may not be used more than once.
- If the same referential constraint is defined more than once, a warning is issued, and a new foreign key is added.
- The parent table referenced by a foreign key must already exist. It must not be a view, and it must have an active primary key.
- The delete rule, if specified, must be one of RESTRICT, SET NULL, or CASCADE.
- IF SET NULL is used, at least one foreign key column must be nullable.
- When defining foreign keys, you must have REFERENCES privilege on the parent table and ALTER privilege on the dependent table.
- When defining referential constraints, if a primary key has a field procedure, then the foreign key must have the same field procedure.

Restrictions on Keys and Referential Constraints::

- Keys cannot be added to or dropped from the system catalog tables, and a system catalog cannot be referenced in any referential constraint.
- No table in a referential cycle with two or more tables may be delete-connected to itself. This ensures that the result of a delete from a table does not depend upon the sequence when the database manager accesses the table. In a referential cycle of two tables, neither delete rule can be CASCADE. For a referential cycle of more than two tables, two or more delete rules must not be CASCADE.

  A table is delete-connected to another table if deletion of rows from one table affects the other table. The implications are:

  – A dependent table is always delete-connected to its parents, whatever the delete rule is.
  – A descendent table is delete-connected to a table higher than it in the hierarchy if a delete of rows in the higher-level table can cause a delete of rows in the descendent's parent table.
- For a descendent table to be delete-connected to the same higher-level table through more than one path, all delete rules on each path must be CASCADE, except possibly the delete rule between the descendent and its immediate parent on each path. The delete rules of the descendent with its parent table on each path must be the same and must not be SET NULL. This ensures that the order in which the delete rules are applied has no effect on the result of an operation.

  For further information on tables that are delete-connected through multiple paths, refer to the *DB2 Server for VSE & VM SQL Reference* manual.
- Self-referencing tables are not supported.

For further information on referential integrity, refer to "Elements of Referential Integrity" on page 6.

Integrity Rules for DELETE: There are no rules for the deletion of rows from dependent tables. The deletion rule specified in the referential constraint clause defines what action should be taken by the database manager when a row in the parent table is to be deleted. See "DELETE Rules" on page 7.

Integrity Rules for INSERT: Insert rules always apply when primary and foreign keys are defined. See "INSERT Rules" on page 8.

Integrity Rules for UPDATE: Update rules always apply when primary and foreign keys are defined. See "UPDATE Rules" on page 8.

Note: If a table is a parent in one relationship and a dependent in another, integrity rules for DELETE, INSERT, or UPDATE must be satisfied for both relationships.

To determine the delete rule of an existing foreign key, access the SYSKEYS catalog table as follows:

```
SELECT KEYTYPE, KEYNAME, DELETERULE FROM SYSTEM.SYSKEYS
  WHERE TNAME='table-name'
```

Placing Tables in Dbspaces

When creating a table, you can specify the dbspace in which it is to reside. If you do not, it is put in the creator's PRIVATE dbspace. If the creator does not have a PRIVATE dbspace, then the CREATE TABLE statement fails.

If you specify the name of the dbspace but not the name of the owner, the database manager searches for a PRIVATE dbspace of the specified name that is owned by the creator of the table. If this does not exist, the database manager then looks for a PUBLIC dbspace with the specified name. If that does not exist, then the CREATE TABLE statement fails. Refer to the *DB2 Server for VSE & VM SQL Reference* manual for more information about the CREATE TABLE statement.

Table placement under the various possible default conditions is illustrated in Figure 18.

Connected User Is	Table Specified	Dbspace Specified	Result of the Create Table
DBA named DD	CC.TT	BB.XX	CC.TT in BB.XX
	CC.TT	PUBLIC.XX	CC.TT in PUBLIC.XX
	CC.TT	XX	CC.TT in CC.XX or PUBLIC.XX
	CC.TT	none	CC.TT in CC.ZZ
	TT	BB.XX	DD.TT in BB.XX
	TT	PUBLIC.XX	DD.TT in PUBLIC.XX
	TT	XX	DD.TT in DD.XX or PUBLIC.XX
	TT	none	DD.TT in DD.YY
RR with RESOURCE AUTHORITY	CC.TT	BB.XX	ERROR
	CC.TT	PUBLIC.XX	ERROR
	CC.TT	XX	ERROR
	CC.TT	none	ERROR
	TT	BB.XX	ERROR
	TT	PUBLIC.XX	RR.TT in PUBLIC.XX
	TT	XX	RR.TT in RR.XX or PUBLIC.XX
	TT	none	RR.TT in RR.SS

Figure 18. Default Placement of Tables in Dbspaces

Notes for Figure 18:

- A user with DBA authority can create tables for any user in any dbspace.
- Users with RESOURCE authority can create tables for themselves only, and then only in their own dbspaces or in any PUBLIC dbspaces.
- If the dbspace name is specified but not qualified (just XX), the database manager first looks for a PRIVATE dbspace owned by the creator. If this is not found, then the database manager looks for PUBLIC.XX.
- If the dbspace is defaulted, the required default PRIVATE DBSPACE (CC.ZZ, DD.YY, or RR.SS) must exist.
- If you omit the dbspace name, the database manager will not select a dbspace that resides in a nonrecoverable storage pool. If you want to create a table in a nonrecoverable dbspace, you must specify the dbspace name.

You can easily avoid confusion by fully qualifying both the table name and the dbspace name.

Creating Views

Some of your users may find that no single table contains all the data they need; rather, the data might be scattered among several tables. Or one table might contain more data than they want to see or are authorized to see. For those situations, you can create *views*. A view is an alternative way of describing data that exists in one or more tables.

You can create a view any time after creating the underlying tables. The owner of a set of tables implicitly has the authority to create a view on them, and someone with DBA authority can create a view for any owner on any set of tables.

Use the CREATE VIEW statement to define a view and give it a name. Unless you specifically list different column names after the view name, the column names of the view will be the same as those of the underlying table. (Table 11 on page 43 shows an example of this.) When creating different column names for your view, remember the naming conventions you established when designing the database.

As Table 11 on page 43 illustrates, the information in the view is described by a SELECT statement. This statement can name other views as well as tables, and can use WHERE, WITH CHECK OPTION, GROUP BY, and HAVING clauses. It cannot use ORDER BY, name a host variable, or contain the UNION operator.

By specifying a WHERE clause in the subquery of a view definition, you can limit the rows addressed through a view. If an application (or user) deals with a specific set of rows in a table, you can create a view to limit the rows addressed to only those required. If a view is created using the WHERE and WITH CHECK OPTION clauses, all subsequent UPDATEs and INSERTs will prevent changes to rows that fall outside the set of rows defined by the view. Refer to the *DB2 Server for VSE & VM SQL Reference* manual for more information about creating views.

Reasons for Using Views

Some reasons you might want to use views are:

- To provide a customized table for a specific user

  Some tables may have a large number of columns, not all of which are of interest to all users or are named or ordered appropriately. You can, in effect, create a smaller table for certain users by defining a view that contains only the columns of interest. You can rename columns and reorder the column sequence to tailor the view to the user's needs.

- To limit access to certain kinds of data

  You can create a view containing only selected columns and rows from a table or tables. Users with the SELECT privilege on the view see only the information you describe. For example, a view could be defined that showed only the FIRSTNME, LASTNAME, WORKDEPT, and EDLEVEL columns for employees in Department D11.

- To alter tables without affecting application programs

  For example, a program that uses an INSERT into T1 without a specified list of column names will cause an error after you add a column to table T1. The error is generated because the number of values being inserted into the table is different than the number of columns in the table. If T1 is a view, you will be protected from that error because adding a column to the table does not affect the view definition and, therefore, does not affect the program.

Creating a View on a Table

The example below illustrates creating a view on a single table, the DEPARTMENT table. Of the four columns in the table, only three are required for the view: DEPTNO, DEPTNAME, and MGRNO. The order of the columns in the SELECT clause is the order in which they appear in the view.

```
CREATE VIEW VDEPT3 AS
  SELECT DEPTNO,DEPTNAME,MGRNO
  FROM DEPARTMENT
```

In this example, no column list follows the view name, VDEPT3. Hence, the columns of the view have the same names as those of the table on which it is based (DEPTNO, DEPTNAME, MGRNO). Table 10 shows the result of executing the following SQL statement:

```
SELECT * FROM VDEPT3
```

Table 10. View of a Table

DEPTNO	DEPTNAME	MGRNO
A00	SPIFFY COMPUTER SERVICE DIV.	000010
B01	PLANNING	000020
C01	INFORMATION CENTER	000030
D01	DEVELOPMENT CENTER	?
D11	MANUFACTURING SYSTEMS	000060
D21	ADMINISTRATION SYSTEMS	000070
E01	SUPPORT SERVICES	000050
E11	OPERATIONS	000090
E21	SOFTWARE SUPPORT	000100

Creating a View from Several Tables

Name more than one table in the FROM clause to create a view that combines information from two or more tables. This operation is called a *join*, and is shown in the following example, which includes the manager's name (from the EMPLOYEE table) and information from the DEPARTMENT table.

```
CREATE VIEW SMITH.VDEPTM  AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
  FROM DEPARTMENT, EMPLOYEE
  WHERE EMPLOYEE.EMPNO = DEPARTMENT.MGRNO
```

Table 11 shows the result of executing the following SQL statement:

```
SELECT * FROM SMITH.VDEPTM
```

Table 11. View of Two Tables

DEPTNO	MGRNO	LASTNAME	ADMRDEPT
A00	000010	HAAS	A00
B01	000020	THOMPSON	A00
C01	000030	KWAN	A00
D11	000060	STERN	D01
D21	000070	PULASKI	D01
E01	000050	GEYER	A00
E11	000090	HENDERSON	E01

Table 11. View of Two Tables (continued)

DEPTNO	MGRNO	LASTNAME	ADMRDEPT
E21	000100	SPENSER	E01

Now, suppose you want to create a similar view that includes only the departments that report administratively to Department A00. Suppose also that you want a different set of column names. The appropriate CREATE statement is as follows:

```
CREATE VIEW SMITH.VDEPTMA00
  (DEPT, MGR, NAME, REPORTTO)
  AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
  FROM EMPLOYEE, DEPARTMENT
  WHERE EMPLOYEE.EMPNO = DEPARTMENT.MGRNO
  AND ADMRDEPT = 'A00'
```

Table 12 shows the result of executing the following SQL statement:

```
SELECT * FROM SMITH.VDEPTMA00
```

Table 12. View Created with New Column Names

DEPT	MGR	NAME	REPORTTO
A00	000010	HAAS	A00
B01	000020	THOMPSON	A00
C01	000030	KWAN	A00
E01	000050	GEYER	A00

Things You Cannot Do with a View

When designing views, consider the following restrictions:

- You cannot update, insert, or delete through a view if it involves any of the following:
  - SQL column functions (SUM, MAX, MIN, AVG, COUNT)
  - Elimination of duplicate rows (DISTINCT)
  - Grouping (GROUP BY), or HAVING clause
  - A FROM clause that uses more than one table (that is, a join).

  In the above cases, you can retrieve data from the views by means of the SQL SELECT statement, but you cannot use INSERT, UPDATE, or DELETE statements.
- You cannot insert a row through a view if the view has a column derived from an arithmetic or string expression, a scalar function, or a constant.
- You cannot update a column of a view that is derived from an arithmetic or string expression, a scalar function, or a constant (for example, a column that is defined as 1.6 x SALARY).

For more detailed information about view restrictions, see the *DB2 Server for VSE & VM SQL Reference* manual.

You can make changes to a table through a view when the view does not contain the same number of columns or the same number of rows as the table on which it is based. Table 13 on page 45 summarizes the restrictions on accessing views.

Table 13. Restrictions on View Access

STATEMENT	RESTRICTIONS
UPDATE	You cannot update a view defined as the join of multiple tables. This includes views defined on views defined as the join of multiple tables. You cannot update a view that is defined using DISTINCT, GROUP BY, or column functions. You cannot update rows of a view that is defined using WITH CHECK OPTION, if the updated rows fall outside the set of rows defined by the view. You cannot update virtual columns. (A virtual column is a column on a view that is not derived directly from a column of a stored table. For example, view columns defined by expressions such as MAX(SALARY), SALARY+BONUS, or AVG(PRSTAFF) are all virtual columns).
INSERT	You cannot insert into a view defined as the join of multiple tables. This includes views defined on views defined as the join of multiple tables. You cannot insert into a view that contains a column of the underlying table which allows nulls. You cannot insert into a view that is defined using DISTINCT, GROUP BY, or column functions. You cannot insert into a view that is defined using WITH CHECK OPTION, if the inserted rows fall outside the set of rows defined by the view. You cannot insert into the virtual columns of a view.
DELETE	You cannot delete from a view defined as the join of multiple tables. This includes views defined on views defined as the join of multiple tables. You cannot delete from a view that is defined using DISTINCT, GROUP BY, column functions.
INDEX	You cannot create an index on a view.
ALTER	You cannot alter a view (for example, add columns, or keys).
DBS UNLOAD	Unloading a view will sequence the unloaded rows in an arbitrary order chosen by the database manager. Rows may not be in sequence of any index on an underlying table.
DBS RELOAD	RELOADing through a view will not drop and recreate indexes on the underlying table. You must do this yourself using SQL statements that precede and follow the RELOAD statement. The INSERT restrictions shown above also apply.
DBS DATALOAD, ISQL INPUT, SQL PUT	The INSERT restrictions shown above also apply.

Materializing a View

When designing views, you should be aware of the view-processing techniques used by the database manager and the circumstances in which each is used. Two view-processing techniques are used: view merge and view materialization. This section describes the circumstances in which a view is materialized.

When a view is referenced in an SQL statement, the view definition is merged with the SQL statement and a new statement is created that references only base tables and columns and that contains only added or modified WHERE predicates, and an added or modified GROUP BY clause. The new statement is then processed. This process is *view merge*. Some view-referencing statements cannot be processed using the view merge technique. The database manager uses the view materialization technique to process these statements.

With view materialization, a temporary table is created internally and a view (that could not otherwise be accessed) is materialized into the table at run time. The database manager then performs the statement on the materialized view. A materialized view is read-only, because queries on the view are on a temporary table. Each view that is materialized in an SQL statement is materialized in a temporary dbspace.

Because view merge is more efficient than view materialization, view materialization is used only if view merge cannot be used.

A view is materialized if it is created with:

- A GROUP BY or HAVING clause, and is accessed by a statement that requires the view to be joined or that specifies column functions on the view
- Column functions, and is accessed by a statement that requires the view to be joined
- One or more DISTINCT columns, and is accessed by a statement that requires the view to be joined
- Multiple DISTINCT columns, and is accessed by a statement that specifies column functions on the view
- One DISTINCT column, and is accessed by a statement that specifies multiple column functions on the view
- A column defined with column functions, and that column is accessed by a statement that specifies column functions
- One or more DISTINCT columns, and a DISTINCT column is accessed by a statement that includes arithmetic expressions with column functions
- Multiple DISTINCT columns, and is accessed by a statement that does not specify all columns in the select-list of the SELECT statement
- A column defined with built-in functions, expressions, or literals, and that column is referenced in the GROUP BY or HAVING clause of a SELECT statement accessing the view
- A column function with a DISTINCT specification, and is accessed by a SELECT statement with a DISTINCT specification
- A column defined with a column function, and that column is referenced in the WHERE clause of a statement accessing the view
- A column derived from an expression, function or constant, and that column is accessed by a statement containing a WHERE clause with a LIKE predicate
- A virtual column, and that column is referenced in a DISTINCT column function of statement accessing the view.

Note: If the SELECT list of the view definition statement contains a long field, the view cannot be materialized because of long-field restrictions.

For information on determining if view materialization occurs, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

Creating Indexes

The purpose of nonunique indexes is to provide efficient access to data. Unique indexes have the additional purpose of ensuring that key values are unique.

Even when present, the index is not always used: the database manager selects an access path to the data based on a combination of factors. To see whether an index is used in processing a particular SQL statement, use the EXPLAIN statement. For information on using the EXPLAIN statement and on explanation tables, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

Indexes can improve performance of table access; however, this is at the expense of the DASD storage required for them, and the performance of INSERT, UPDATE, and DELETE operations. Thus, while you will want to create indexes on your tables, some judgement is advised. For information about the storage required by an index, see "Estimating the Number of Index Pages" on page 227.

To create an index on a table, use the SQL CREATE INDEX statement. You must have DBA authority or the INDEX privilege on the table. An index may be defined on 1 to 16 columns.

Index Key

The columns identified in the CREATE INDEX statement build a key. An *index key* is a column or an ordered collection of columns on which an index is defined. A multicolumn key is a key built on two or more columns.

The usefulness of an index depends on its key. Columns that you use frequently in performing selection, join, projection, grouping, and ordering operations are good candidates for use as keys. See "Estimating the Number of Index Pages" on page 227 for information on calculating the size of index keys. For columns with a field procedure, use the number of bytes in the encoded field, not the number in the decoded column.

For information about restrictions on key length, see the description of the CREATE INDEX statement in the *DB2 Server for VSE & VM SQL Reference* manual.

The ordering of the columns specified in the CREATE INDEX statement is important to the definition of the key sequence. The major order determinant columns must be specified first. For example, an index on the PROJ_ACT table, defined over the PROJNO, ACTNO, and ACSTDATE columns sequences activity numbers within project numbers, and estimated activity start date within activity numbers, if the columns are specified in this order for the index.

For each column participating in the key, you can specify whether its order in the key sequence is ascending or descending. The default is ascending. When creating a unique index, the uniqueness of each value in the index key cannot depend upon trailing blanks. The database manager also ignores trailing blanks when sequencing indexes made up of VARCHAR or VARGRAPHIC values.

UNIQUE Indexes

You can enter duplicate values in a key. If you do not want duplicate values, use CREATE UNIQUE INDEX.

For example, in the sample database, it is important that there be no duplicate activity keywords in the ACTIVITY table. Creating a unique index, as in the following example, prevents duplicates.

```
CREATE UNIQUE INDEX XACT1
  ON ACTIVITY (ACTKWD)
```

The index name is XACT1 and the indexed column is ACTKWD.

If you are planning to use referential integrity or unique constraints, described in "Creating Tables" on page 27, it may be unnecessary to explicitly create unique indexes. When using the primary key or unique constraint clause, the database manager automatically creates a unique index on the table. However, you may want additional indexes for other columns and foreign keys.

The PCTFREE Clause

The PCTFREE clause specifies how much space is to be reserved 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 PCTFREE 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 PCTFREE to zero.

Usually, a low PCTFREE value, 5–10 percent, is a good choice when creating an index, as it provides enough room to accommodate a low level of maintenance. It also provides extra room at localized key ranges where high update activity is taking place by splitting a full index page into two half-empty pages when an insertion or update must go into that page.

Clustering Rows of a Table on an Index

A *CLUSTERING index* is used by the database manager to determine placement of rows in pages of a dbspace. The first index created on a table is, by default, the CLUSTERING index. The database manager tries to place rows with the same or similar keys on the same dbspace page.

A *CLUSTERED index* is an index whose sequence of key values corresponds closely to the sequence in which the table rows are actually stored in the database. It can be effectively used to minimize DASD input/output whenever the table rows are accessed in the index sequence of a CLUSTERED index. A CLUSTERING index should always be made a CLUSTERED index. This is done by loading the table rows in the key sequence of the CLUSTERING index.

To establish a CLUSTERING index that also has the property of being a CLUSTERED index, do the following:

1. Load the table in the index sequence (key sequence) of the CLUSTERING index.

   This establishes the initial clustering of rows with similar keys. For the load operation, set PCTFREE for the dbspace to a high enough value to allow space on pages for future clustered insertion of rows.
2. Create the indexes on the table.

After loading the table, create the indexes on the table. The first index you create will be the CLUSTERING index. Any index having an order that matches the load sequence of the rows will be marked as a CLUSTERED index.

The CLUSTERING index will be a CLUSTERED index because you have loaded the table rows in the sequence of this index. In the SYSINDEXES catalog table, the CLUSTER column value for this index is F, indicating that it is the first index created by the table, and that it is currently a CLUSTERED index. If, after many INSERTs of new rows into the table, the order in which the rows are stored in the database no longer closely match the index key sequence, the CLUSTER column value is changed to W (the next time UPDATE STATISTICS is performed). This indicates that the index is the first index created on the table, and it is currently not a CLUSTERED index. You can reorganize the table. Refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual for information on reorganizing tables. The database manager will continue to use this index to decide where new rows should be stored, because it is still the CLUSTERING index for the table.

One or more of the other indexes created on the table may also happen to have an index sequence that closely matches the sequence in which the table rows are stored. Although this is fortuitous and cannot be directly controlled by the user, the database manager will record these indexes as CLUSTERED by setting their CLUSTER column in SYSINDEXES to C. Such indexes can be exploited as efficient access paths by the database manager. When one of these indexes is no longer CLUSTERED, its CLUSTER column is changed to N the next time UPDATE STATISTICS is performed.

3. Reduce the PCTFREE value for the dbspace.

   This is necessary to make the free space reserved during the load operation available for use on normal INSERT activity. On an INSERT or ISQL INPUT, the database manager attempts to place the inserted row on the same page as a row with the same or similar key.

You can define the key ordering of the CLUSTERING index to be any you wish. However, the primary considerations would be frequently used table orderings (that is, frequently used ORDER BYs) and joins.

If you cluster a table on an index that has a key ordering that matches the most common ORDER BY clauses for queries against the table, you can avoid internal sorting of the query results. A related consideration is the size of an ordered query result. Internal sorting of a small query result is not expensive. However, if you have a large, ordered query result (for a batch job or a comprehensive report), the internal sort could be quite time-consuming. You should consider clustering a table to support your most frequent, large sequential access orderings.

If you have a table that is frequently referenced by a join on a particular column (or set of columns), you may want to consider clustering it on an index on the join column(s). For example, between the DEPARTMENT and EMPLOYEE tables there are two likely join candidates (referential constraints are defined): one between the EMPNO column in EMPLOYEE and the MGRNO column in DEPARTMENT, and the other between the DEPTNO column in DEPARTMENT and the WORKDEPT column in EMPLOYEE. In this case you could choose to cluster both tables on either employee numbers or department numbers, depending on which join is expected more frequently.

Note: You can change the clustering that you initially define for a table. Refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual for information on reorganizing tables.

Figure 19 illustrates both clustered and nonclustered indexes.

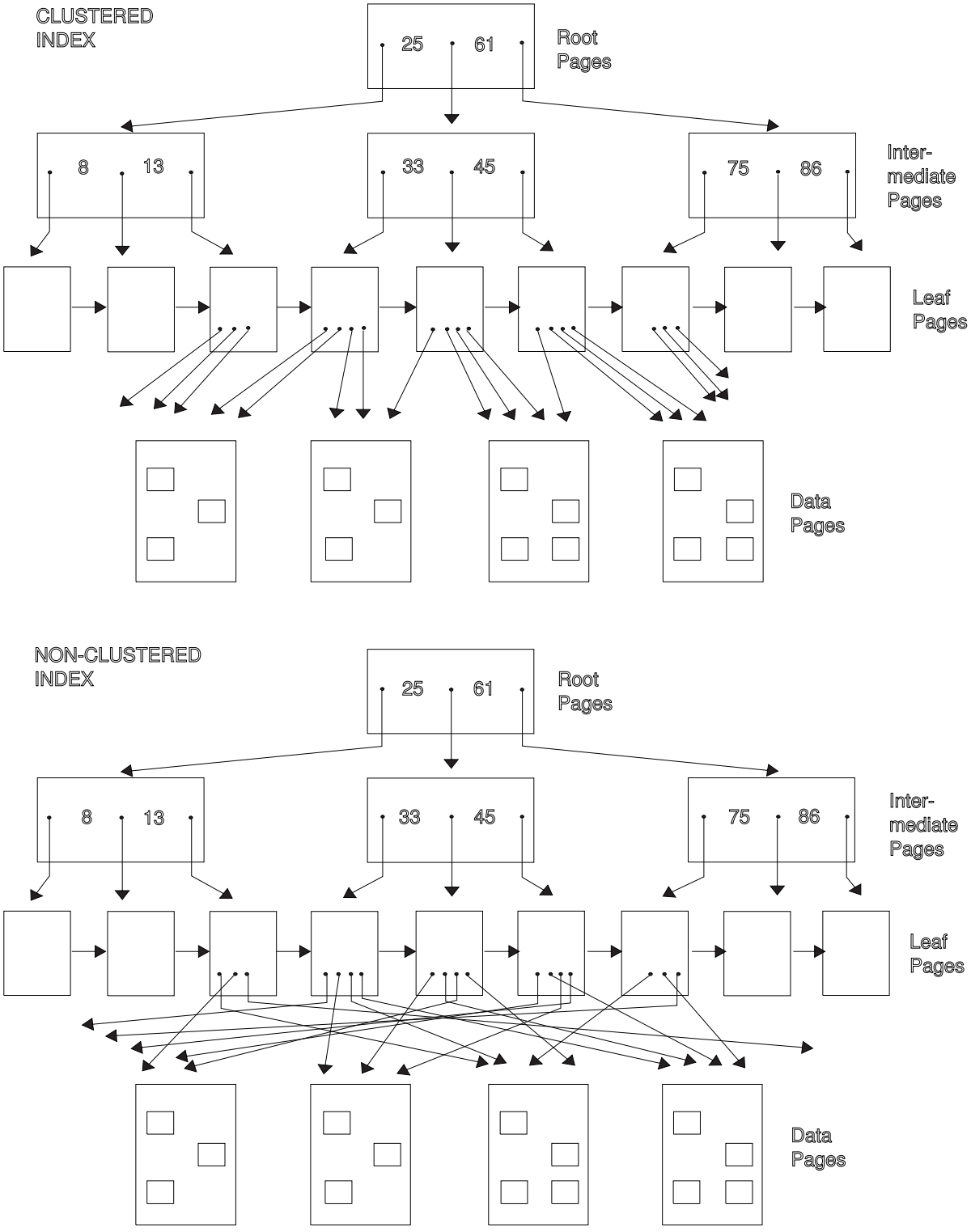


Figure 19. Clustered and Nonclustered Indexes

Some Things to Remember When Defining Keys

Column updates require index updates. Define as few indexes as possible on a column that is updated frequently, because every change must be reflected in each

index. For more information about potential problems with indexes and performance, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

A multicolumn key may be more useful than a key on a single column when the comparison is for equality. A single multicolumn index is more efficient when the comparison is for equality and the initial columns are provided. For example, if an index is composed of columns A, B, and C, a SELECT statement with a WHERE clause of the form WHERE A = *value* AND B = *value* may be processed more efficiently than if there are separate indexes on A and on B. Additional columns may also improve performance by allowing index-only access scanning. Refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual for information on index-only access scanning.

Indexes are important tools for improving performance. Suggestions for using indexes effectively are in the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

An index cannot be defined over multiple tables. Furthermore, an index key cannot include any columns defined as long fields. **Avoid using VARCHAR or VARGRAPHIC columns in an index.** Fixed-length indexes perform better than variable-length indexes. Data pages as well as index pages must be read when VARCHAR or VARGRAPHIC columns are included in an index. The variable-length fields have trailing blanks removed before being put into the index. This may result in the data page values differing from the index page values, and necessitates that both index and data pages be read when using the index as an access path for data retrieved.

Note: Long fields include the following data types: VARCHAR(n) with n>254, VARGRAPHIC(n) with n>127, LONG VARCHAR, or LONG VARGRAPHIC.

General Performance Considerations on the Use of Indexes

It is good practice to create a unique index on the column or set of columns that uniquely define each record in the table (its key). A unique index can easily be created by specifying a primary key or a unique constraint when you create the table. A primary key can be used as an index even if automatic referential integrity is not being used. Using a unique constraint or primary key helps data integrity because the database manager enforces this uniqueness.

Consider creating additional indexes on other columns based on how often you expect the column to be used in search criteria. Once you have identified all the desired indexes, decide which column is apt to be used most often in search criteria. Then load the table in that column's sequence, thus making the column's corresponding index a CLUSTERED index.

If the table is to have a CLUSTERING index, be sure to create that index first after initial table loading. You should do this because the database manager tries to place inserted records so that the physical sequence of the table's records is the same as the sequence defined by the first index created on that table.

It is more efficient to first load a table and then create the indexes on it, rather than the other way around.

Usually, each table should have at least one index. Part of the decision of whether to create an index on a specific column should be based on the trade-off between

the faster access achieved, versus the index maintenance processing that the database manager must do whenever that column is modified. A column is an ideal candidate for being indexed if it is likely to be a frequent search argument on SQL statements, but not likely to be changed. Avoid creating indexes on frequently updated columns.

Indexes can be created and dropped. If high query activity is anticipated, temporarily create indexes on the columns that are likely access paths for those queries.

Migration Considerations for Indexes

The SQL/DS Version 2 Release 2 product introduced a new index structure for nonunique indexes. This format requires more space than in earlier releases, but it allows nonunique indexes to perform almost as well as unique ones.

The new format requires more space. The number of additional bytes required for each nonunique index in the new format is:

```
4 x (number of index pages) - 4
```

If you are migrating from Version 2 Release 1 or earlier, some of your dbspaces may not have room for indexes in the new format. Before deleting the old indexes, determine if there are sufficient index pages available to create the index in the new format. Nonunique indexes created before Version 2 Release 2 can coexist with the new type of nonunique index, so you do not have to drop and re-create indexes.

Note: You cannot migrate from Version 2 directly to Version 7. You must migrate to Version 3 first and then migrate from Version 3 to Version 7.

Using the Catalog in Database Design

The catalog tables contain information that can be helpful in designing your database. The *DB2 Server for VSE & VM SQL Reference* manual lists these tables and what is stored in them.

You can also use the catalog to verify the accuracy of your database definition process. After you have created the objects in your database, display selected information from the catalog to check that there were no errors in your CREATE statements, and to verify that you have the correct tables in each dbspace.

The information in the catalog is vital to normal database system operation. As the following examples show, you can **retrieve** catalog information, **but changing it could have serious consequences.** Thus, you cannot process INSERT or DELETE statements against the catalog, and you can update only a few of the columns in selected catalog tables.

To run the following examples, you need at least the SELECT privilege on the appropriate catalog tables. Be careful with your own examples: querying the catalog can result in a long dbspace scan.

Retrieving Catalog Information about a Table

The SYSTEM.SYSCATALOG table contains a row for each table and view in your database. For each, it tells you whether the object is a table or view, its name, who created it, what dbspace contains it, and other information. It also has a REMARKS

column in which you can store your own information about the table in question. See "The COMMENT ON Statement" on page 54 for information about how to do this.

Enter the following statement to display all the information for the project activity sample table:

```
SELECT *
  FROM SYSTEM.SYSCATALOG
  WHERE TNAME = 'PROJ_ACT'
  AND CREATOR = 'SQLDBA'
```

Retrieving Catalog Information about Columns

The SYSTEM.SYSCOLUMNS table has one row for each column of each table and view. You can query it, for example, if you cannot remember a particular column name.

The following statement retrieves information about columns in the sample department table:

```
SELECT CNAME, TNAME, COLTYPE, LENGTH, NULLS
  FROM SYSTEM.SYSCOLUMNS
  WHERE TNAME='DEPARTMENT'
  AND CREATOR = 'SQLDBA'
```

As shown in Table 14, for each column it displays:

- The column name
- The name of the table that contains it
- Its data type
- Its length attribute
- Whether or not it allows nulls.

Table 14. Retrieving Information about Columns from SYSCOLUMNS

CNAME	TNAME	COLTYPE	LENGTH	NULLS
DEPTNO	DEPARTMENT	CHAR	3	N
DEPTNAME	DEPARTMENT	VARCHAR	36	N
MGRNO	DEPARTMENT	CHAR	6	Y
ADMRDEPT	DEPARTMENT	CHAR	3	N

Retrieving Catalog Information about Indexes

The SYSTEM.SYSINDEXES table contains a row for each index, including indexes on catalog tables.

The following query retrieves information about the index XEMPL2:

```
SELECT *
  FROM SYSTEM.SYSINDEXES
  WHERE INAME = 'XEMPL2'
  AND ICREATOR = 'SQLDBA'
```

A table can have more than one index. The following query retrieves information about all the indexes of a table:

```
SELECT *
  FROM SYSTEM.SYSINDEXES
  WHERE TNAME = 'EMPLOYEE'
  AND CREATOR = 'SQLDBA'
```

Retrieving Catalog Information about Views

The SYSTEM.SYSVIEWS table contains a row for each view.

The following query retrieves information about the view SYSUSERLIST:

```
SELECT *
  FROM SYSTEM.SYSVIEWS
  WHERE VIEWNAME = 'SYSUSERLIST'
  AND VCREATOR = 'SQLDBA'
```

Retrieving Catalog Information about Authorization

The following 4 tables contain information about the privileges held over tables and views:

- SYSCOLAUTH

  Contains information regarding grants of the UPDATE privilege on columns of tables or views.
- SYSPROGAUTH

  Details privileges regarding who can run packages.
- SYSTABAUTH

  Contains information about the privileges held by authorization IDs and packages on tables and views.
- SYSUSERAUTH

  Records special privileges held by authorization IDs (for example, DBA, CONNECT authority).

Only users with DBA authority can access SYSUSERAUTH. Other users can access this information using a view called SYSUSERLIST, which contains all the columns of SYSUSERAUTH except the PASSWORD column.

Query these tables to learn who can access data in your application server. For example, the following query retrieves the names of all users who have been granted access to the SQLDBA.DEPARTMENT table, as well as any views on that table:

```
SELECT GRANTEE
  FROM SYSTEM.SYSTABAUTH
  WHERE TTNAME = 'DEPARTMENT' AND GRANTEETYPE = ' '
  AND TCREATOR = 'SQLDBA'
```

GRANTEE is the name of the column that contains authorization IDs and package names for users of tables. TTNAME and TCREATOR specify the SQLDBA.DEPARTMENT table. The clause GRANTEETYPE = ' ' ensures that you retrieve the names only of users (not packages) that have authority to access the table.

The COMMENT ON Statement

After you create a table or view, you can provide explanatory information about it for future reference—for example, the purpose of the table, who uses it, and anything unusual about it. To do this, use the COMMENT ON statement. You can both store comments about the table or view as a whole, and include one for **each column**. A comment must not exceed 254 bytes.

Comments are especially useful if your names do not clearly indicate the contents of columns or tables.

Below are two examples of COMMENT ON:

```
COMMENT ON TABLE SQLDBA.EMPLOYEE IS
  'Employee table. Each row in this table represents one
   employee of the company.'
COMMENT ON COLUMN SQLDBA.PROJECT.PRSTDATE IS
  'Estimated project start date. The format is DATE.'
```

Retrieving Comments

When you process a COMMENT ON statement, your comments are stored in the REMARKS column of SYSTEM.SYSCATALOG or SYSTEM.SYSCOLUMNS. Any comment already present in the row is replaced by the new one. The following queries retrieve the comments added by the two COMMENT ON statements above:

```
SELECT REMARKS
  FROM SYSTEM.SYSCATALOG
  WHERE TNAME = 'EMPLOYEE'
  AND CREATOR = 'SQLDBA'
SELECT REMARKS
  FROM SYSTEM.SYSCOLUMNS
  WHERE CNAME = 'PRSTDATE' AND TNAME = 'PROJECT'
  AND CREATOR = 'SQLDBA'
```

Chapter 3. Maintaining Your Database

The previous chapter described how to **implement** your database design. This chapter deals with the various maintenance tasks you may need to perform to **maintain** tables and dbspaces. The following tasks are discussed:

Maintaining Tables

- Loading information into tables

  There is considerable flexibility in how data can be entered.
- Copying a table

  When information is being shared, the owner of a table may choose to have other users copy it, so that they can make changes to their own copy of the table without affecting the original.
- Moving tables from one dbspace to another

  You may want to move tables to another dbspace to:
  - Improve concurrent access to tables

    If a table resides in a PRIVATE dbspace and many users need to update that table at the same time, you should move it into a PUBLIC dbspace, which allows concurrent access.
  - Recover dbspaces

    You may want to move a table from a nonrecoverable dbspace to a recoverable one, or a recoverable dbspace to a nonrecoverable one.
  - Get more space for a table

    The amount of information that you can store in a table depends on the size of the dbspace it is in, and the storage requirements of the other tables there. If a table requires more space for data or indexes, you should consider moving it to a larger dbspace.
- Merging data from multiple tables

  It may be necessary to combine all the columns or a subset of the columns from different tables into a new table.
- Altering the design of a table

  You may want to change the design of a table after it has been created: for example, add or delete columns, change the data type of a column, or change the name of the table.
- Altering referential constraints on a table

  You may wish to add referential integrity to tables that do not have it.
- Enforcing referential constraints

  You may want to enforce the referential constraints when your tables are created, or defer enforcement until you have performed other activities.
- Moving data from one application server to another

  The second application server can be a DB2 Server for VSE & VM application server, or another application server supporting IBM's implementation of the Distributed Relational Database Architecture (DRDA) protocol.
- Removing tables

  If tables are no longer required, you can remove them.

Maintaining Dbspaces

- Altering the design of a dbspace

  When you created a dbspace, you specified the following parameters for it: its potential size (in pages), its type (PUBLIC or PRIVATE), its storage pool assignment (STORPOOL), the number of pages for its header (NHEADER), the percentage of each page reserved for updates that cannot be placed in the original location (PCTFREE), the number of pages reserved for indexes (PCTINDEX), and the size of the locks (LOCK).

  As requirements change, you may need to change some of these settings. You can change the PCTFREE and LOCK parameters with the ALTER DBSPACE statement. If any of the other parameters need to be changed, you will have to acquire a new dbspace (which satisfies your new requirements), and move all the tables from the old dbspace to the new one.
- Reorganizing a dbspace to free storage pool pages

  As part of maintaining your dbspaces, you may have to reorganize it to release pages back to a storage pool.
- Removing dbspaces

  If a dbspace is no longer required, you can remove it and its contents by using the DROP DBSPACE statement.
- Using VSAM (VSE only)

  There are VSAM restrictions when managing storage.

Reorganizing Catalog Table Indexes

The catalog tables have indexes to improve the speed of access. Occasionally, you should reorganize these indexes. See “Reorganizing Indexes on the Catalog Tables” on page 76.

Maintaining Tables

After designing and creating a table, you may have to load data into it, copy it, move it from one dbspace to another, move data in it from one application server to another, change an aspect of its design, or remove it from the database.

Loading Data into Tables

This section reviews the possible ways to load data into tables. Many of these methods use the Database Services Utility commands: for more information on these commands, refer to the *DB2 Server for VSE & VM Database Services Utility* manual.

Loading Data in VM Using the DBS Utility

Interactively: You can load data into tables interactively through the DBS Utility. To do this, invoke the utility so the terminal controls file input (SYSIN). You can then either enter multiple INSERT statements, or execute the DBS Utility DATALOAD TABLE command using the INFILE (*) subcommand.

From a CMS File: The DBS Utility DATALOAD TABLE command will accept input data records in a user-created CMS file. One or more tables can be loaded during a single pass of the data records. The existing data in the tables loaded with this method are not affected. Rows are added to a table through the PREPARE, OPEN, PUT, and CLOSE facilities of SQL.

From a Virtual Reader File: The DBS Utility DATALOAD TABLE command will also accept input data records in a CMS virtual reader file with no header. One or more tables can be loaded during a single pass of the data records. The existing data in the tables loaded with this method are not affected. Rows are added to a table through INSERT statements executed using the PREPARE and EXECUTE facilities of SQL.

Refer to the *DB2 Server for VSE & VM Database Services Utility* manual for more information.

Loading Data Using the DBS Utility in VSE/ICCF

To load data into a table from data records entered from a terminal, as an alternative to entering multiple INSERT statements, users can use the DBS Utility under VSE/ICCF in conversational mode. To initiate this, enter the following VSE/ICCF control statements:

```
/LOAD ARIDBS
/OPTION GETVIS=AUTO
/DATA INCON
```

In response to the prompt to ENTER DATA, the appropriate series of SQL statements or DBS Utility DATALOAD TABLE commands must be entered. After a DATALOAD TABLE command the user must enter the INFILE (*) subcommand to initiate input data record processing and the ENDDATA subcommand to end it. An outline of the interactive terminal input is:

```
CONNECT userid IDENTIFIED BY password;
DATALOAD TABLE (table-name)
  column-name1   1-5
  column-name2   6-7

⋮

INFILE (*)
  data record
```

```
    data record
⋮
  ENDDATA
```

These commands are described in the *DB2 Server for VSE & VM Database Services Utility* manual.

Each record (row) is entered in a fixed format as defined by the column specifications in your DATALOAD command. In this example, the user enters column 1 data into typing positions 1–5 of the command line, column 2 data into positions 6–7, and so on.

Do **not** put quotation marks around character data, and do **not** use commas to separate data values. Such punctuation can be used **outside** the data positions of the command line defined by the column specifications of the DATALOAD command.

As an alternative to entering each input data record interactively, the user can embed DBS Utility commands and data records in the VSE/ICCF control statements. An outline of loading a table under VSE/ICCF in a nonconversational manner is:

```
/LOAD ARIDBS
/OPTION GETVIS=AUTO
/DATA
CONNECT userid IDENTIFIED BY password;
DATALOAD TABLE (table-name)
  column-name1   1-5
  column-name2   6-7
⋮
INFILE (*)
  data record
⋮
ENDDATA
```

Loading Data from a Terminal Using ISQL INPUT

The ISQL INPUT statement enables a user to enter multiple rows of data into a table. The table name and (optionally) the column names need to be entered only once. The column names, along with their data types, are then displayed in the order that the data must be entered, and the user can then enter data one row at a time.

For data that is similar, the user can use the PF12 RETRIEVE function. That is, the user can retrieve the previous data row entered, and then type over the fields that are different. This can save keystrokes.

Data entered with an INPUT statement is not stored in the table until the INPUT statement is ended by an END statement. ISQL will issue an INSERT statement for every row entered, using the PREPARE and EXECUTE facilities of SQL. However, before the INPUT statement is ended, the data can be committed or backed out by the statement:

SAVE — Stores all data entered since the last SAVE statement. If no SAVE statement has been issued, it commits all the data since the start of the INPUT statement.

BACKOUT — Deletes all data entered since the last SAVE statement. If no SAVE statement has been issued, it deletes all the data since the start of the INPUT statement.
CANCEL — Performs a BACKOUT and also ends the INPUT statement.

Remember that the AUTOCOMMIT mode affects the processing of the SAVE, BACKOUT, and CANCEL statements. For additional information on the ISQL INPUT, SAVE, and BACKOUT statements, refer to the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual.

Loading Data from Sequential Files in VSE

The DBS Utility DATALOAD TABLE command accepts SYSIPT data records or data records contained in a user-created sequential file. One or more tables can be loaded during a single pass of the data records. The existing data in the tables loaded with this method is not affected. The DATALOAD TABLE processing adds rows to a table through the PREPARE, OPEN, PUT, and CLOSE facilities of SQL.

Loading Data from VSAM Files

A VSAM file can be converted to either of the following:

- a sequential (SAM) file using the VSE/VSAM Access Methods Services REPRO command
- a CMS or tape file through the VM VSE/VSAM Access Methods Services (AMSERV command) using the REPRO control statement.

This sequential file can then be identified as the input data file to DBS Utility DATALOAD TABLE processing.

Note: The VSAM REPRO command should never be used to copy the DB2 database itself.

Loading Data from Other Tables

Data can be copied into a table from other tables by using the following methods:

- An INSERT with Subselect statement executed through ISQL, the DBS Utility, or a user program. An INSERT with Subselect copies one or more rows which are selected or computed from other tables into a table.
- The execution of a DBS UNLOAD and RELOAD command series. This technique allows data to be copied from tables in the same or different databases but only a complete replacement of the data in the target table is possible.
- The execution of a DBS DATAUNLOAD and DATALOAD command series. This technique allows data to be copied from tables in the same or different application servers, and allows more selectivity than the UNLOAD/RELOAD sequence. This is useful when you want to copy only parts of tables.

All of these techniques allow the source of the data to be copied to be identified by a view that is defined on one or more tables. A view can be used to identify the target table if the view definition meets the requirements defined for inserting rows into a view.

If referential constraints are in place on tables in which you wish to load data, you should consider whether you would like to enforce constraints while the data is loading or after it is loaded. See “Enforcing Referential Constraints” on page 68 for more information.

Copying Tables

To make a copy of an existing table, use the DBS Utility UNLOAD and RELOAD commands.

Example

A user with the user ID SMITH has the SELECT privilege on the SQLDBA.EMPLOYEE table. To make a copy of this table, to be called SMITH.EMPLOYEE, in the PRIVATE dbspace SMITHDB, enter the following commands either in a CMS file called CONTROL DBSINPUT A or in the appropriate job control:

```
CONNECT SMITH IDENTIFIED BY SMITHPW;
UNLOAD TABLE (SQLDBA.EMPLOYEE)  OUTFILE(TEMPFIL);
RELOAD TABLE (EMPLOYEE) NEW (SMITHDB)
INTABLE (SQLDBA.EMPLOYEE)  INFILE(TEMPFIL);
```

To execute these commands in VM, invoke the DBS Utility, as follows:

```
FILEDEF TEMPFIL DISK MYDATA MYFILE A4 (RECFM VBS BLOCK 800
SQLDBSU SYSIN(CONTROL DBSINPUT A) SYSPRINT(LIST DBSLIST A)
```

The RELOAD statement creates tables without const raints, losing all referential constraints on the table you are copying. You must reinstate referential constraints later with the ALTER TABLE statement. See "Altering Referential and Unique Constraints" on page 65.

The RELOAD statement with the 'NEW' parameter recreates the table without field procedures. Instead of reloading the table using the 'NEW' parameter, recreate the table to include field procedures and reload the table using the 'PURGE' parameter.

Moving Tables from One Dbspace to Another

To move a table from one dbspace to another, you must first unload it using the DBS UNLOAD command, drop it from the database, then reload it into the new dbspace. When a table is dropped, all indexes, privileges, views, primary and foreign keys, and unique constraints for it are removed, and must be re-established.

As well, if a table has field procedures associated with it, the table should be dropped and recreated to include the field procedures and reloaded using the 'PURGE' parameter.

Example

User SMITH has a table (called SMITH.MYTABLE) that he wishes to move from the SMITH.PERSONAL dbspace to the SMITH.SECRET dbspace.

Enter the following commands in either a CMS file called CONTROL DBSINPUT A, or inside the appropriate job control:

```
CONNECT SMITH IDENTIFIED BY SMITHPW;
UNLOAD  TABLE (SMITH.MYTABLE) OUTFILE(TEMPFIL);
DROP    TABLE SMITH.MYTABLE;
RELOAD  TABLE (SMITH.MYTABLE)
        NEW (SMITH.SECRET) INFILE(TEMPFIL);
```

In VM you run these commands by invoking the DBS Utility, as follows:

```
FILEDEF TEMPFIL DISK MYDATA MYFILE A4 (RECFM VBS BLOCK 800
SQLDBSU SYSIN(CONTROL DBSINPUT A) SYSPRINT(LIST DBSLIST A)
```

Merging Data from Multiple Tables

It may be necessary to combine all columns or a subset of the columns from different tables into a new table. You can do this through ISQL or the DBS Utility using the following procedure:

1. Create the new table with a CREATE TABLE statement.
2. Insert rows into the new table by selecting columns from the source tables with an INSERT with Subselect statement.
3. Execute an UPDATE STATISTICS statement against the new table.
4. Create the required indexes for the new table with CREATE INDEX statements.
5. Create the required views on the new table.
6. Grant the required authorizations on the new table and views.
7. If necessary, redefine the views on the old tables to eliminate access to the columns merged into the new table.

To identify authorizations and views on the old tables, you can query the system catalog with a SELECT statement entered through ISQL or the DBS Utility. The following tables contain information pertinent to this task:

- SYSTEM.SYSUSAGE identifies the base table on which a view is defined
- SYSTEM.SYSVIEWS identifies the view definitions
- SYSTEM.SYSTABAUTH identifies the users who have privileges to access tables and views.

Example

To identify the base tables for the view ORGANIZATION, enter the following query:

```
SELECT BNAME FROM SYSTEM.SYSUSAGE
  WHERE DNAME = 'ORGANIZATION'
```

To identify the view definitions, enter the following query:

```
SELECT VIEWTEXT FROM SYSTEM.SYSVIEWS
  WHERE VIEWNAME = 'ORGANIZATION'
```

To identify the users who have privileges to access the view or its base tables, enter the following query:

```
SELECT GRANTEE, STNAME FROM SYSTEM.SYSTABAUTH
  WHERE TTNAME = 'ORGANIZATION'
```

If a view is defined for all the columns required in the new table, steps 1, 2, and 3 (needed to merge data from multiple tables) can be replaced by the following:

1. Enter the DBS Utility UNLOAD command to unload the view.
2. Enter the DBS Utility RELOAD command to create and load the new table.
3. Process an UPDATE STATISTICS statement for the new table, if necessary. By default, this statement is performed for each table loaded during RELOAD TABLE command processing. For more information, see the *DB2 Server for VSE & VM Database Services Utility* manual.

Examples

In VM: Include the following SQL statements and DBS Utility commands within your DBS Utility control file to perform the above task:

```
CONNECT userid IDENTIFIED BY userpw;
UNLOAD TABLE creator.viewname OUTFILE(DUMPFIL);
RELOAD TABLE creator.newtablename NEW INFILE(DUMPFIL);
```

Invoke the DBS Utility, as usual, to process the above statements and commands.

In VSE: Use the following job control commands, SQL statements, and DBS Utility commands to perform the above task:

```
// JOB MERGE DATA
// EXEC PROC=DBNAME01
// ASSGN SYS005,...
// ASSGN SYS004,...
// TLBL DUMPFIL,...

⋮
CONNECT userid IDENTIFIED BY userpw;
UNLOAD TABLE creator.viewname OUTFILE(DUMPFIL)
RELOAD TABLE creator.newtablename NEW INFILE(DUMPFIL)
/&
```

See the *DB2 Server for VSE & VM Database Services Utility* manual for details.

Altering the Design of a Table

If you want to change the design of a table after it has been created, use the SQL ALTER TABLE statement. This will not change the data in the table; only its specifications. You can:

- Add a column to a table
- Add or drop a primary key, a foreign key, or a unique constraint.

When you alter a table, information in the system catalog about it is also changed. For example, when you add a new column to a table, SYSTEM.SYSCOLUMNS is changed to record it, and the field in there that records the number of columns is increased by one.

Authorization

To alter a table, you must have the ALTER privilege on it, and if the operation involves a primary key you must have the ALTER privilege on all dependent tables as well. If the operation involves a foreign key, you must have the REFERENCES privilege on the parent table.

You can alter any table if you have DBA authority.

You cannot delete a column, change the name of a column, change the data type of a column, or add or change a field procedure for a column for existing tables using the ALTER TABLE statement. To do these operations, you must drop the existing table and re-create it.

Example

There are two ways to change the data type of the DEPTNAME column of the DEPARTMENT table from VARCHAR(36) to VARCHAR(40):

- Create a new table (DEPT) with the required column definitions, and copy data to it.

```
CREATE TABLE DEPT
   (DEPTNO    CHAR(3)          NOT NULL,
    DEPTNAME  VARCHAR(40)      NOT NULL,
    MGRNO     CHAR(6)                  ,
    ADMRDEPT  CHAR(3)          NOT NULL,
    PRIMARY KEY (DEPTNO)               )
INSERT INTO DEPT SELECT * FROM DEPARTMENT
```

Indexes, views, and privileges have to be reestablished for the new table DEPT; only the data is copied from the DEPARTMENT table. Also, all applications that used the original table must be changed to reflect the new table name, then re-preprocessed.

- DATAUNLOAD the contents of the DEPARTMENT table to a flat file, drop the table, re-create it with the new data type definition of the DEPTNAME column, then DATALOAD the contents of the flat file back into the DEPARTMENT table. For details on DATAUNLOAD and DATALOAD, see the *DB2 Server for VSE & VM Database Services Utility* manual.

Adding a New Column

When you add a column to an existing table, it is placed on the far right.

The physical records are not actually changed until users insert values in the new column, so access time to the table is not affected immediately. After values are inserted, however, this could impact performance by forcing rows onto another physical page. To avoid that situation, define enough free space on each page ahead of time.

You cannot define the new column as NOT NULL; it must allow NULL values.

Example

Add a new column to the table DEPARTMENT, containing a location code for the department. The column name is LOCNCODE, and its data type is CHAR (4).

```
ALTER TABLE DEPARTMENT
  ADD LOCNCODE CHAR (4)
```

Table 15 shows part of the original table.

Table 15. Before Adding a New Column to a Table

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00

Table 16 shows the table after adding the new column and updating a location code in the third row.

Table 16. After Adding a New Column to a Table and Updating a Row

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCNCODE
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	?
B01	PLANNING	000020	A00	?
C01	INFORMATION CENTER	000030	A00	B126

Altering Referential and Unique Constraints

If you plan to let the database manager enforce referential integrity in a set of tables, see "Considerations for Referential Integrity when Creating Tables" on page 38 and "Specifying a UNIQUE Constraint" on page 38.

The following terms are used in the discussion of the ALTER TABLE statement:

Inactive Key or Constraint A primary key, a foreign key, or a unique

	constraint that has been made inoperable by the ALTER TABLE ... DEACTIVATE statement. Neither referential nor unique constraints are enforced until the related keys are activated.
Implicitly Inactive Key	A foreign key that is not explicitly inactive, but references a table with an inactive primary key. A referential constraint is not enforced until the related primary key is activated.
Inactive Table	A table that contains an inactive or implicitly inactive key, or contains an active primary key referenced by an inactive foreign key. This limits access to the table to the creator or a DBA, and allows deferred constraint enforcement.
Dependently Inactive Table	A dependent table or foreign key that has been flagged as inactive because the primary key of its parent table has been deactivated.

Table 17 is a summary of the authorization required to alter referential constraints.

Table 17. Authorization Required when Altering Referential Constraints

Statement	Parent Table	Dependent Table
ADD		
Primary Key	A	
Foreign Key	R	A
DROP		
Primary Key	A, R(1)	A
Foreign Key	R	A
ACTIVATE		
Primary Key	A, R(1)	A
Foreign Key	R	A
All	A, R(1)	A
DEACTIVATE		
Primary Key	A, R(1)	A
Foreign Key	R	A
All	A, R(1)	A
Note: ALTER privilege is required when A appears. REFERENCES privilege is required when R appears, and (1) applies when a dependent table exists.		

Considerations When Adding Keys or Constraints

The following restrictions apply when you add a primary key, a foreign key, or a unique constraint to an existing table:

- The columns named for the key being added must exist.
- If adding a primary key, there should be no existing primary key on the table.
- If adding a primary key or a unique constraint there must not be duplicate values in the specified columns.
- When adding a foreign key:
  - The constraint name must not already exist.
  - If the key columns are identical to those of another foreign key that references the same parent table, a warning is issued and the foreign key is created.
- You can use only one FOREIGN KEY clause in each ALTER TABLE statement; if you want to add two foreign keys to a table, you must execute two statements.

- If you add a foreign key, the primary key of the parent table must already exist.
- To add a foreign key, you must have REFERENCES privilege on the parent table and ALTER privilege on the dependent table.
- If adding a foreign key, the foreign key must not cause a table to be delete-connected to another table through multiple paths with different delete rules or with a delete rule of SET NULL.
- A referential cycle with two or more tables must not cause a table to be delete-connected to itself. For further information on delete-connected tables, refer to "Restrictions on Keys and Referential Constraints:" on page 39.

For further information on referential integrity, refer to "Elements of Referential Integrity" on page 6.

Considerations When Dropping a Primary or Foreign Key

The following restrictions apply when you drop a primary key or a foreign key from an existing table:

- When you drop a foreign key, the corresponding referential relationship is also dropped.
- To drop a foreign key, you must have REFERENCES privilege on the parent table and ALTER privilege on the dependent table.
- When you drop a primary key, all the referential relationships in which the table is a parent are also dropped.
- You must have ALTER privilege on any dependent tables.
- The dependent tables no longer have foreign keys.
- The unique index (created to enforce uniqueness in the primary key) is dropped.

In both situations, you should consider carefully the effects on your application programs of dropping keys. The primary key of a table is intended to serve as a permanent, unique identifier of the occurrences of the entities it describes, and quite likely some of your programs depend on that. The foreign key defines a referential relationship and a delete rule, and without it your programs must enforce the constraints.

Considerations When Activating Keys and Constraints

Primary Key: To activate a primary key you must have ALTER privilege on the parent and dependent tables and REFERENCES privilege on all dependent tables.

If any dependent foreign keys were deactivated implicitly when the primary key was made inactive, they will be verified against the primary key. If the primary key index can be created successfully and the dependent foreign key values are found in the parent table's primary key, then the primary key and the dependent foreign keys will be activated. If any of these processes fail, none of the keys will be activated.

Activating the primary key will neither verify nor affect the status of any dependent foreign keys that were deactivated explicitly with the ALTER TABLE *table-name* DEACTIVATE FOREIGN KEY statement.

Foreign Key: To activate a foreign key you must have ALTER privilege on the dependent table and REFERENCES privilege on the parent table.

If a foreign key is already active, attempts to activate it are ignored. If the primary key of the parent table referenced by this foreign key is inactive, the foreign key

cannot be activated. Otherwise, the inactive foreign key will have its values verified against its parent table. If all values can be found in the parent's primary key, the foreign key will be activated.

Unique Constraint: To activate a unique constraint you must have ALTER privilege on the table. The unique constraint will be activated only if all values in its key are unique. If there are duplicate values you must change them to be unique before the constraint can be activated.

All: To activate the primary key, each unique constraint, and each explicitly inactive foreign key in a table, use the ACTIVATE ALL option. You must have the required ALTER and REFERENCES privileges.

Implications of Activating a Primary Key or Unique Constraint: Activating a primary key or unique constraint that is already active causes the unique index associated with the key or constraint to be reorganized. This is more efficient than deactivating the key or constraint (which would drop the underlying index), and then activating the key or constraint (which would re-create the underlying index). For more information on the benefits of reorganizing an index, see the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual.

Considerations When Deactivating Keys and Constraints

Primary Key: Deactivating a primary key drops the primary key index from the parent table and implicitly deactivates all active dependent foreign keys. This limits the access to all inactive dependent tables to the creator or a DBA, and allow deferred constraint enforcement. For information on deferred constraint enforcement see "Enforcing Referential Constraints".

To deactivate a primary key you must have ALTER and REFERENCES privileges on the parent table, and ALTER privilege on all dependent tables.

Foreign Key: To deactivate a referential constraint that is active, you must have ALTER privilege on the dependent table and REFERENCES privilege on the parent table.

If a foreign key has been explicitly deactivated already, attempts to deactivate it again are ignored.

Deactivating a foreign key will make the two tables in the relationship inactive. Access to the inactive table is limited to the creator or a DBA. For information on the effects of deactivating a foreign key, see "Advantages and Disadvantages of Deferred Constraint Enforcement" on page 69.

Unique Constraints: Deactivating a unique constraint drops the unique index associated with the constraint, causing the table to become inactive. This will limit access to the table to its creator or a DBA.

Enforcing Referential Constraints

Two forms of enforcement are possible:

- Immediate Constraint Enforcement.

  After the referential constraints have been defined, the enforcement of the referential constraint is immediate. That is, the insert, update, and delete rules are enforced when the INSERT, UPDATE, and DELETE statements are issued. During immediate constraint enforcement, keys and tables are in the active state.

- Deferred Constraint Enforcement.

  A table can be made inactive by deactivating its primary key, any of its foreign keys, any of its unique constraints, or a dependent foreign key, by using the ALTER TABLE statement. A referential relationship is between two keys in different tables. If either a primary or foreign key is deactivated, **both** tables become inactive.

  When a table is in an inactive state, only the owner or someone with DBA authority can issue Data Manipulation Language (DML) statements against it. No one can issue DML statements (for example, SELECT or UPDATE statements) against **any** table that would result in implicit access of an inactive table to enforce referential constraints.

  When the keys are activated, the constraints will be verified automatically and the tables become active again.

Advantages and Disadvantages of Deferred Constraint Enforcement

You may want to deactivate the enforcement of referential integrity among tables to improve performance when you are loading data into a table.

When referential integrity is active between two tables, each INSERT statement on a dependent table causes a check to be issued against the parent table. This check verifies that the foreign key value being inserted has a matching primary key value in the parent table. When data is being loaded into a dependent table, each inserted row causes a check of the parent table; if many rows are being loaded, the overhead of this checking becomes significant. In this case, you may improve your overall performance of the load by deactivating any referential constraints. When the load completes, you then reactivate them to validate the data.

If referential integrity is in effect at the beginning of an LUW, and the constraints are deactivated, the data loaded and the constraints re-activated all within the same LUW, then referential integrity exists at the end of the LUW as well. However, within that LUW, referential constraints are not enforced. You could load rows into the dependent table that had no parents when loaded. Since the database can be in an inconsistent state during an LUW, but not at its completion, you can use a more flexible sequence of statements within an LUW. At some point you must load parent rows for the dependent rows into the parent table. Otherwise, you would be unable to reactivate the referential constraint. There are some disadvantages to deactivating a referential constraint between tables:

- Only users with DBA authority and the owner of a table can use DML statements on that table, or tables referenced by it through an inactive referential constraint. This is to prevent people from inserting, deleting, or updating data in a table that they may believe to have an active referential constraint.
- When referential constraints are deactivated, any indexes created to enforce the constraints are dropped. Dropping these indexes will invalidate any packages that require the use of the indexes. Three major costs will be incurred on reactivating the referential constraints:
  - The underlying indexes are re-created
  - Any dependent rows are checked against the referential constraints
  - All invalidated packages are automatically re-preprocessed when they are first used.

  If a relatively small number of rows are added to the table by the load process, then the costs of reactivating the referential constraints may exceed the savings realized by deferring referential constraint enforcement on each row loaded.

You should deactivate the referential constraints between tables only when large amounts of data are to be loaded, or when a significant amount of data is to be loaded in an order that violates the referential constraint at some point during the data-loading operation. For example, you can load new rows into a dependent table before loading matching rows into the parent table only while the referential constraint is inactive.

Repairing Rows that Violate Referential Constraints

If you deactivate a referential constraint in order to load data, then receive an error when you try to reactivate it, it could be for one of the following reasons:

- You activated a foreign key that references an inactive primary key. You must first activate the inactive primary key.
- One or more rows in one of the tables violates the referential constraint, and you must fix these rows. This error condition may also arise when you are creating a referential constraint.

Note: When the above error occurs, SQLCODE -667 (SQLSTATE 22519) and the name of the constraint in error are returned as a message token in SQLCA.

Isolating Duplicate Primary Key Values: To find duplicate primary key values, use the statement shown below. In the example, the name of the table is P1, and the primary key is represented by the columns PKCOL1, PKCOL2, and so on, for all columns that form the primary key:

```
SELECT PKCOL1, PKCOL2, ... FROM P1
   GROUP BY PKCOL1, PKCOL2, ...
   HAVING COUNT(*) > 1
```

You could then eliminate the duplicate values with UPDATE and DELETE statements, or move them to a special table if you do not want to eliminate them immediately.

To move the rows to a special table (called an EXCEPTION table in this explanation), create a table with the same column definitions as the original table (but with no key definitions). If there are many duplicate values, you may want to create a nonunique index for the duplicate primary key columns in the EXCEPTION table to improve performance.

Use the statements shown below to copy the rows with duplicate primary key values into the EXCEPTION table (called E1 in this example):

```
INSERT INTO E1
   SELECT * FROM P1 A WHERE EXISTS
      (SELECT PKCOL1, PKCOL2, ...  FROM P1 B
         GROUP BY PKCOL1, PKCOL2, ...
         HAVING COUNT(*) > 1
                AND B.PKCOL1 = A.PKCOL1
                AND B.PKCOL2 = A.PKCOL2
                ... )
```

To remove these rows from P1, use this statement:

```
DELETE FROM P1 A  WHERE EXISTS
   (SELECT 1 FROM E1
       WHERE E1.PKCOL1 = A.PKCOL1
         AND E1.PKCOL2 = A.PKCOL2
         ... )
```

Isolating Nonmatching Foreign Key Values: Foreign key values may not match primary key values because either of them may be wrong. This example shows you how to move the nonmatching foreign keys to a separate table. Then, you can

determine whether the foreign or the primary keys are wrong, and fix them with INSERT, UPDATE, or DELETE statements.

This statement retrieves nonmatching foreign key values. In the example, P1 is the parent table; C1 is the dependent table; PKCOL1, PKCOL2, and so on form the primary key; and FKCOL1, FKCOL2, and so on form the foreign key.

```
SELECT FKCOL1, FKCOL2, ...  FROM C1 A
   WHERE (FKCOL1 IS NOT NULL AND
          FKCOL2 IS NOT NULL AND
           ... )
     AND NOT EXISTS
         (SELECT 1 FROM P1 B
            WHERE  B.PKCOL1 = A.FKCOL1 AND
                   B.PKCOL2 = A.FKCOL2 AND
                   ... )
```

To move the rows to a special table (called an EXCEPTION table in this explanation), create a table with the same column definitions as the dependent table (but with no key definitions). If there are many duplicate values, you may want to create a nonunique index for the foreign key columns in the EXCEPTION table to improve performance. To copy the rows with nonmatching foreign keys to the EXCEPTION table (E1 in this example), use the following statement:

```
INSERT INTO E1
   SELECT * FROM C1 A
      WHERE (FKCOL1 IS NOT NULL AND
             FKCOL2 IS NOT NULL AND
             ... )
       AND NOT EXISTS
            (SELECT 1 FROM P1 B
               WHERE  B.PKCOL1 = A.FKCOL1 AND
                      B.PKCOL2 = A.FKCOL2 AND
                      ... )
```

To remove the rows from C1, use the following statement:

```
DELETE FROM C1 A  WHERE EXISTS
   (SELECT 1 FROM E1
       WHERE E1.FKCOL1 = A.FKCOL1
         AND E1.FKCOL2 = A.FKCOL2
         ... )
```

Moving Data from One Application Server to Another

You can use the DBS Utility to move data from one application server to another.

Moving data from a DB2 Server for VSE & VM application server to a remote DRDA application server requires unloading the data from the DB2 Server for VSE & VM application server using the DBS Utility DATAUNLOAD command and reloading the data into the other application server using the DBS Utility DATALOAD command. Moving data from one DB2 Server for VSE & VM application server to another local DB2 Server for VSE & VM application server can be done as above, or by using the DBS Utility UNLOAD and RELOAD commands.

For more information about DBS Utility commands, refer to the *DB2 Server for VSE & VM Database Services Utility* manual.

Notes:

1. When moving data between two application servers, ensure that the appropriate coded character set identifier (CCSID) conversion is done to maintain the correct interpretation of the data.

For example, an application server uses a CHARNAME value of ENGLISH (or the CCSID equivalent), and another application server uses a CHARNAME value of GERMAN (or the CCSID equivalent). Issue the SQLINIT EXEC (in VM), the transaction DSQU (in CICS), or the VSE batch program ARIRBGUD (JCL: ARISBGUD.Z) and specify a CHARNAME for the application requester corresponding to the CHARNAME of one of the application servers (either ENGLISH or GERMAN). Then, to ensure the integrity of the data when moving it between these two application servers, specify the same CHARNAME value for the application requester for both the DATAUNLOAD (or UNLOAD) and DATALOAD (or RELOAD) operations. If ENGLISH is the CHARNAME value specified for the application requester for the data unload operation, then it must also be set to ENGLISH for the data load operation. You can then perform the data unloading and reloading operations.

For more information on CCSID conversion, see the *DB2 Server for VSE System Administration* manual.

2. If you want to move data from one DB2 Server for VSE application server to a DB2 Server for VM application server (not using Guest Sharing), or vice versa, using the DBS Utility UNLOAD command, you can only do so when using a tape.

Removing Tables

To remove tables from the database, use the DROP TABLE statement. For example, to remove a table called PROJECT, enter:

```
DROP TABLE PROJECT
```

Only the table's creator or a user with DBA authority can remove the table. If you have DBA authority, include the user ID of the owner to remove a table. For example, to remove SMITH's table called PROJECT, enter:

```
DROP TABLE SMITH.PROJECT
```

When a table is dropped, the row in the SYSTEM.SYSCATALOG catalog table that contains information about it is deleted. Any other objects that depend on that table are also dropped. As a result:

- The column names of the table are dropped from SYSTEM.SYSCOLUMNS.
- Any views based on the table are dropped.
- Application plans using the table are invalidated.
- *Synonyms* for the table are dropped from SYSTEM.SYSSYNONYMS.
- Indexes created on any columns of the table are dropped.
- Unique constraints on any columns of the table are dropped.
- Referential constraints that involve the table are dropped. In the case of the PROJECT table, it is no longer a dependent of the DEPARTMENT and EMPLOYEE tables, nor a parent of the PROJ_ACT table.
- Authorization information kept in the authorization tables is updated to reflect the dropping of the table. Users who were previously authorized to use the table, or views on it, no longer have those privileges.

You must commit the DROP statement on a table before you can re-create a table of the same name, or before you can create any new indexes with the same name as an index on the table being dropped.

Maintaining Dbspaces

Altering the Design of a Dbspace

You may need to change the parameters of a dbspace for any of the following reasons:

- Storage capacity (PAGES).

  You may have underestimated the storage required by the tables in the dbspace, and need to increase its potential size (in pages).
- Storage pool assignment (STORPOOL).

  You may want to change the storage pool assignment, which determines whether a dbspace is recoverable or nonrecoverable.
- Type (PUBLIC or PRIVATE).

  If the tables in a PRIVATE dbspace are to be shared by many users, then you should consider making it PUBLIC.
- Header Space (NHEADER).

  At the front of every dbspace are one to eight header pages, which contain control information on the tables and indexes stored there. You may need to increase the number of these pages.
- Index Space (PCTINDEX).

  If your dbspace contains more indexes than expected, you may need to increase the index space to accommodate them.
- Free Space (PCTFREE).

  You may want to change the percentage of each data page reserved for updates of rows resulting in larger rows that cannot be placed in the original locations in the page.
- Lock Size (LOCK).

  For PUBLIC dbspaces, you may change the locking level. A lower lock level allows more users to access the same table at the same time; however, there is a cost because of lock acquisitions, an increased possibility of lock escalations. If lock escalation occurs frequently, you may want to increase the locking level. Refer to the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual. for more information about lock escalations.

For a review of these parameters, see "Acquiring Dbspaces" on page 22.

Changing the PAGES, STORPOOL, DBSPACE Type, NHEADER, or PCTINDEX

There is no statement to change these five parameters of a dbspace. If you need to change any of them, you must move all the data in the current dbspace to another dbspace that has the required characteristics. To do this:

1. UNLOAD the current dbspace.
2. DROP the current dbspace.
3. ACQUIRE a new dbspace with the required characteristics.
4. RELOAD the new dbspace.
5. Drop the table with field procedures, recreate it to include the field procedures, and reload the table using the 'PURGE' parameter.
6. CREATE all indexes for the tables involved.
7. Recreate all referential constraints.
8. GRANT all authorizations for the tables involved.
9. CREATE all views relating to the tables involved.

To identify the tables, views, authorizations, and referential constraints related to the dbspace, query the system catalog.

To identify the tables with field procedures, query the SYSFILEDS and SYSFPARMS tables.

Example: To increase the storage capacity of a PRIVATE dbspace called SMITH.SAMPLE to 2 048 pages with defaults for the other dbspace parameters, use the following SQL statements and DBS Utility commands:

```
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
UNLOAD DBSPACE (SMITH.SAMPLE)  OUTFILE (TEMPFIL);
DROP DBSPACE SMITH.SAMPLE;
ACQUIRE PRIVATE DBSPACE NAMED (SMITH.SAMPLE) (PAGES=2048);
RELOAD DBSPACE (SMITH.SAMPLE)  NEW  INFILE(TEMPFIL);
```

Invoke the DBS Utility, as usual, to process the above statements and commands (see the *DB2 Server for VSE & VM Database Services Utility* manual for details).

Indexes, views, authorizations, and referential constraints must be recreated for all the tables in the dbspace.

Changing the PCTFREE and LOCK Parameters

To change these parameters, use the ALTER DBSPACE statement. You must have DBA authority or (in the case of a PRIVATE dbspace) be the owner of the dbspace.

Example: Change the PCTFREE parameter to 10 for the dbspace called MYDBSPACE. type:

```
ALTER DBSPACE MYDBSPACE (PCTFREE = 10)
```

To change both the PCTFREE and the LOCK parameters at the same time, type:

```
ALTER DBSPACE MYDBSPACE (PCTFREE = 10, LOCK = PAGE)
```

Reorganizing a Dbspace to Free Storage Pool Pages

Reorganizing a dbspace releases pages in it back to its storage pool. There are two reasons why you might want to do this:

- You are unable to drop a table in a dbspace when you issue a DROP TABLE statement and you receive a message that the storage pool is full. This occurs because there are not enough *shadow pages* in the storage pool to allow the database manager to remove all the rows for that table from the dbspace. For information on shadow pages, see the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual.

  After the database has been restarted (with STARTUP=W), there will be a row in the catalog table SYSDROP for the dropped table. Any subsequent DROP TABLE statements will cause SYSDROP to be processed. When the database manager processes the row for the dropped table, it will end and issue a message indicating that the storage pool is full unless you take other steps to provide sufficient pages in the storage pool for shadow pages. You can provide sufficient pages in the storage pool by adding dbextents to the storage pool, or by reorganizing the dbspace where the table resides.

  If reorganizing the dbspace does not provide sufficient shadow pages to allow you to drop the table, then you must add dbextents to the storage pool. For information on adding dbextents, see the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual.
- You want to release unused pages back to the storage pool.

Once a page is allocated to a dbspace, it remains allocated until you drop the dbspace. This can cause the storage pool that contains the dbspace to become short on storage. For example, if a large table occupied a dbspace, and has been dropped, all pages used to store the rows for that table are still allocated to the dbspace. To determine whether many empty pages are allocated to a dbspace, enter the SHOW DBSPACE operator command.

To reorganize a dbspace, follow these steps:

1. Unload all tables in the dbspace, except those that should be dropped.
2. Drop the dbspace (see note 1 below).
3. Reacquire the dbspace.
4. Reload the tables (see note 2 and 4 below).
5. Re-create all indexes and unique constraints for all tables.
6. Grant all authorizations for the tables again.
7. Re-create all referential constraints for tables (see note 3 below).
8. Re-create all views that reference the tables.

Notes:

1. Before dropping the dbspace, obtain the information necessary to perform steps 5, 6, 7, 8, and note 4 below from the catalog tables.
2. The RELOAD TABLE commands create all tables by default with the user ID of the person who enters the commands, usually the DBA. If you want a table to retain the user ID of its original owner, specify this user ID in the table parameter of the RELOAD TABLE command. When performing this procedure, use the NEW option on the RELOAD TABLE and RELOAD DBSPACE commands. See the *DB2 Server for VSE & VM Database Services Utility* manual for more details.
3. If a table has referential constraints, these will be lost when the table is unloaded and reloaded. To re-create any foreign keys, primary keys, unique constraints, or primary keys that have dependent foreign keys in tables that reside in other dbspaces, use the ALTER TABLE statement.
4. If a table has field procedures, they will be lost when the table is reloaded using the 'NEW' option. To include the field procedures, drop the table, recreate it, and reload the table using the 'PURGE' option.

Removing Dbspaces

To drop the contents of a dbspace and return it to the available state, issue the DROP DBSPACE statement. Dbspaces that are available can then be reacquired, using the ACQUIRE DBSPACE statement.

When a dbspace is dropped, all tables in it are also dropped. When a table is dropped, all authorizations, views, referential constraints, unique constraints, and field procedures relating to it are dropped.

If a dbspace contains only one table, it is more efficient to drop and then reacquire the entire dbspace later, than to drop the table.

The DROP DBSPACE statement may be carried out on both PUBLIC and PRIVATE dbspaces. You must have DBA authority to delete a dbspace or (in the case of a PRIVATE dbspace) be the owner. No user, not even one with DBA authority, can delete the dbspace that contains the system catalog.

Example

To remove your own PRIVATE dbspace named MYDBSPACE, type:

```
DROP DBSPACE MYDBSPACE
```

VSAM Restrictions

VSAM defines storage for DB2 Server for VSE databases but it does not manage this storage. VSAM commands such as EXPORT, IMPORT, REPRO, and VERIFY should **never** be used on the DB2 Server for VSE database. If you receive an error message indicating an OPEN error (RC=74), ignore it and **do not run VERIFY**.

Reorganizing Indexes on the Catalog Tables

The catalog indexes need to be reorganized when indexes on the catalog tables become fragmented, and the database manager can no longer insert entries into the catalog dbspace.

Index fragmentation often happens in an application development environment. Application development requires frequent preprocessing; and each time a program is preprocessed, many entries are added to the catalog tables. It may not be possible to plan properly for the range of index keys that might be created.

Index fragmentation can lead to the inefficient use of the index pages of the catalog dbspace (SYS0001). If most of the index pages in your catalog dbspace are occupied, fragmentation is a likely cause. To determine the number of index pages occupied in the catalog dbspace, enter the SHOW DBSPACE command. (The number of this catalog dbspace is 1; so you type SHOW DBSPACE 1.) If there is a high percentage of occupied pages, consider running the catalog index reorganization utility, which optimizes the indexes as they exist on the catalog tables.

To run the catalog index reorganization utility in a VSE environment, start the database in single user mode with STARTUP=I specified. Figure 20 shows an example of the job control statements.

```
// JOB REORG
// EXEC PROC=ARIS71SL
// EXEC PROC=DBNAME01
// EXEC PGM=ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,STARTUP=I,PARMID=name'
/*
/&
```

Figure 20. Example Job Control to Reorganize the Catalog Indexes

Notes:

1. For ARIS71SL, substitute your procedure or job control that identifies the DB2 Server for VSE service libraries. The catalog index reorganization utility uses the ARISCAT source member.
2. For DBNAME01, substitute your procedure or job control that identifies the database whose catalog indexes you wish to reorganize.
3. The initialization parameters SYSMODE=S and STARTUP=I are required. You can also supply any of the following initialization parameters (PARMID is included in the example in Figure 20.):
   PARMID=*name*
   DBPSWD=*password*
   NPAGBUF=*n*
   NDIRBUF=*n*
   NCSCANS=*n*
   LOGMODE=Y | A | L | N
   CHKINTVL=*n*

SLOGCUSH=*n*
ARCHPCT=*n*
SOSLEVEL=*n*
CHARNAME=*name*
DSPLYDEV=L | C | B
DUMPTYPE=P | F | N
TRACDBSS=*nnnnnnnnnn*
TRACRDS=*nnnnnn*
TRACCONV=*n*
TRACDSC=*nn*
TRACBUF=*n*
TRACSTG=*n*
LTIMEOUT=*n*
SYNCPNT=Y | N

See the *DB2 Server for VSE System Administration* manual for a description of initialization parameters.

To avoid the processing involved in switching log modes, use the same LOGMODE that you normally use.

To run the catalog index reorganization utility in a VM environment:

1. Log on to the virtual machine that owns the database.
2. Get read access to the service minidisk (ACCESS 193 V).
3. Invoke the SQLCIREO EXEC. This EXEC resides on the service minidisk. It invokes the DB2 Server for VM application server in single-user mode with STARTUP=I. See "SQLCIREO EXEC" on page 249 for its syntax.

Because the catalog index reorganization utility runs in single user mode, the only way to trace it is with the TRACRDS, TRACDBSS, TRACDSC, and TRACCONV initialization parameters. The TRACE operator command cannot be used in single user mode.

Chapter 4. Supporting Your Users

As the database administrator, you provide the support that users need to gain access to your DB2 Server for VSE & VM application server and the data it manages.

This chapter describes the tasks involved in supporting new users, and removing the data and access of those who have left.

Adding a New User

To a DB2 Server for VSE & VM Application Server:

The following are the steps involved in adding new users to DB2 Server for VSE & VM application servers.

New users need CONNECT authority on the application server (alternatively, this may be granted to ALLUSERS). To add a new user to the system, perform the following tasks:

1. In VM, define the user's virtual machine as a DB2 Server for VM user machine. This involves making VM directory changes and is discussed in the *DB2 Server for VM System Administration* manual.
2. Setup the user as a new ISQL user.
3. Grant the user an appropriate level of authority to access data and use resources.
4. Specify the default application server.
5. If required, load initial tables.
6. Ensure that the new user obtains adequate system training.

To a Non-DB2 Server for VSE & VM Application Server:

To enable a user to access a non-DB2 Server for VSE & VM application server perform the following tasks:

Notes:

1. In VM, if the user is not already a DB2 Server for VM user, define the user's virtual machine as a DB2 Server for VM user machine. This involves making VM directory changes and is discussed in the *DB2 Server for VM System Administration* manual.
2. Arrange system-level sign-on authority with the system administrator of each remote application server.

   Note: A new user ID and password may be required at some of the remote application servers, depending on the LU 6.2 security level that is required for the connection. See the *Distributed Relational Database Connectivity Guide* manual.
3. In VM, setup a new entry in the CMS Communication Directory (COMDIR) for the remote application server (if it has not already been done), and make the COMDIR accessible to the user.

   In VSE, set up a new entry in the DBNAME Directory for the remote application server (if it has not already been done).

4. If the user will be accessing an application server through ISQL, then setup the user as a new ISQL user. Make sure that ISQL has been installed on the remote application server.
5. Grant the user (or arrange to have granted) the appropriate level of authority to access data and use resources at each of the remote application servers.
6. Specify the default application server.
7. If required, load initial tables.
8. Ensure that the new user obtains adequate training on the remote application server and on how to access it from the local DB2 Server for VSE & VM application requester.

Setting Up New ISQL Users

To set up a new ISQL user to access the resources of an application server, run the SQLDBA.ARINEWUS routine supplied by IBM and previously loaded into the SQLDBA.ROUTINE table during database generation.

Note: The ARINEWUS routine is intended for DB2 Server for VSE & VM application servers only. If you need to add a new ISQL user to a non-DB2 Server for VSE & VM application server create your own routine using ARINEWUS as a sample. This routine uses a CONNECT statement containing an IDENTIFIED BY clause, as well as a GRANT CONNECT statement. These statements are unique to DB2 Server for VSE & VM application servers and may not be supported by non-DB2 Server for VSE & VM application servers.

Start ISQL and connect as SQLDBA (or some other user ID with DBA authority), then type:

```
RUN SQLDBA.ARINEWUS (newuser newuserpw)
```

For *newuser*, specify:

- In VM, the CP LOGON user ID (the name of the user's virtual machine) or,
- In VSE, the user's CICS sign-on ID.

For *newuserpw*, specify a password for the new user.

The ARINEWUS routine does the following:

- Issues an ISQL SET RUNMODE CANCEL command.
- Issues an ISQL SET AUTOCOMMIT OFF command.
- Grants CONNECT authority to the new user. The routine parameters *newuser* and *newuserpw* are used on the CONNECT statement.
- Creates a copy of a set of sample tables for the user, and grants him or her full authority on them. These tables are named:
  newuser.DEPARTMENT
  newuser.EMPLOYEE
  newuser.PROJECT
  newuser.ACTIVITY
  newuser.PROJ_ACT
  newuser.EMP_ACT
  newuser.CL_SCHED
  newuser.IN_TRAY

- Copies data from the sample tables owned by user ID SQLDBA into the new user's sample tables. (Only the rows needed to duplicate the examples shown in the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual are copied.)
- Creates indexes on the sample tables.
- Creates and loads an ISQL routine table (*newuser*.ROUTINE), which includes an ISQL PROFILE routine, as follows:

```
NAME      SEQNO   COMMAND          REMARKS
-------   -----   --------------   -------
PROFILE      10   SET VARCHAR 35   NULL
PROFILE      20   SET CASE UPPER   NULL
```

- Creates an index on the routine table.
- Issues an ISQL SET AUTOCOMMIT ON command.

When you run ARINEWUS, you will be prompted to enter either COMMIT or ROLLBACK. If no errors occurred, enter COMMIT; otherwise, enter ROLLBACK.

The ARINEWUS routine sets up the new user only in the application server that you are connected to when you invoke the routine. If a user is to have access to more than one application server, connect to these other application servers and run ARINEWUS again for each one.

Example

A new user with a user ID of ALEX and a password of ALEXPW is defined. Alex does application development work and needs access to two application servers: PROD and TEST.

Do the following in a VM system:

1. Log on to your own user machine, and IPL CMS.
2. Issue SQLINIT DBNAME(PROD). (Assume that the PROD application server is currently being accessed by some database machine in multiple user mode.)
3. Start the ISQL program.
4. Connect to the PROD application server under a user ID with DBA authority. In the example below, the user ID is SQLDBA. The step is optional if you already have DBA authority. Enter:

   ```
   CONNECT SQLDBA IDENTIFIED BY sqldbapw
   ```

   Assume you know the password of the SQLDBA user ID for both application servers
5. Start the ARINEWUS routine:

   ```
   RUN SQLDBA.ARINEWUS (ALEX ALEXPW)
   ```
6. Connect to the TEST application server under a user ID with DBA authority:

   ```
   CONNECT SQLDBA IDENTIFIED BY sqldbapw TO TEST
   ```
7. Start the ARINEWUS routine:

   ```
   RUN SQLDBA.ARINEWUS (ALEX ALEXPW)
   ```
8. Exit from the ISQL program.

Do the following in a VSE system:

1. Ensure that the application servers PROD and TEST have been started with the DLBL and LIBDEF statements required for accessing the application servers. Also ensure that the CICS system has been started and initialized for DB2 Server for VSE on-line access to both PROD and TEST.

2. Start the ISQL program.
3. Connect to the PROD application server under a user ID with DBA authority. In the example below, the user ID is SQLDBA. The step is optional if you already have DBA authority. Enter:

   ```
   CONNECT SQLDBA IDENTIFIED BY sqldbapw TO PROD
   ```

   Assume you know the password of the SQLDBA user ID for both application servers
4. Start the ARINEWUS routine:

   ```
   RUN SQLDBA.ARINEWUS (ALEX ALEXPW)
   ```
5. Connect to the TEST application server as SQLDBA (or with any ID that has DBA authority).

   ```
   CONNECT SQLDBA IDENTIFIED BY sqldbapw TO TEST
   ```
6. Start the ARINEWUS routine:

   ```
   RUN SQLDBA.ARINEWUS (ALEX ALEXPW)
   ```
7. Exit from the ISQL program.

Alex is now set up to use both application servers.

If you want to review the contents of the ARINEWUS routine before you invoke it, issue the following SELECT statement on either application server:

```
SELECT COMMAND FROM SQLDBA.ROUTINE WHERE NAME = 'ARINEWUS'
```

Authorizing Access

Once you have run ARINEWUS, your new user has CONNECT authority to the application server. This is the lowest level of authority. To decide if this is the appropriate level for this user, and to change it if not, see “Chapter 5. Providing Security” on page 87.

After providing new users with CONNECT authority, you can do any of the following:

- Acquire PRIVATE dbspaces for them so that they can create their own tables
- Grant them RESOURCE authority
- Grant them DBA authority
- Ensure that they are granted privileges on other users’ tables and views
- Create new views on tables to restrict their access to data that is appropriate for them to see.

Specifying a Default Application Server in VM

Before VM users can access an application server, a default application server needs to be established. Users must process the SQLINIT EXEC to specify the application server they intend to access. For example, if the user intends to access the TEST application server, he or she must enter:

```
SQLINIT DBNAME(TEST)
```

Users only need to re-process the SQLINIT EXEC if they want to explicitly change the current SQLINIT options. The most current SQLINIT information is stored on each user’s A-disk. For more information, see “SQLINIT EXEC” on page 235.

Loading Initial Tables

New users likely have existing files of data that they want to store in the database. If the files are short, the data they contain can be typed in at the terminal using ISQL statements. This method, however, is not suitable for large files. Here, you can use the DBS Utility to transfer data into a database. For information on how to use the DBS Utility, see "Loading Data into Tables" on page 59.

Training New Users

It is your responsibility to assist new users with the DB2 Server for VSE & VM database manager, and to deal with their questions and problems. Ensuring new users are adequately trained will reduce your problem-solving duties.

Removing Users from an Application Server

When users leave your area, both their access to the application server and any unwanted data should be removed. You should try to get people to remove their own data before they leave; however, you will often have to do so yourself.

The following steps describe how to remove a user's access to an application server. If a user was using multiple application servers, you must perform this process for each server. You must have DBA authority to perform these steps.

If you have DBA authority, you can revoke a user's authority to access the application server at any time by issuing the REVOKE CONNECT statement listing the user(s) affected. For example:

```
REVOKE CONNECT FROM JOHN,KAREN,ALICE
```

Revoking a user's CONNECT authority prevents that user ID from accessing the application server. This action only removes the user IDs from the SYSTEM.SYSUSERAUTH catalog table; it does not affect any objects (for example, tables) in the database which those users may have created, nor does it affect any privileges that may have been granted to them.

Example

An employee whose user ID was SMITH has left the company. To remove SMITH's database objects, do the following:

1. Determine the names of PRIVATE dbspaces owned by SMITH. Type:

   ```
   SELECT DBSPACENAME FROM SYSTEM.SYSDBSPACES
     WHERE OWNER='SMITH'
   ```

2. Determine the names of tables owned by SMITH. Type:

   ```
   SELECT TNAME,DBSPACENAME FROM SYSTEM.SYSCATALOG
     WHERE CREATOR='SMITH'
     AND TABLETYPE='R'
   ```

   This command displays the names of the tables that SMITH created, and the dbspaces where they were created. The TABLETYPE='R' (R stands for real table) indicates that you want to see only the tables at this point; you do not yet want to see any views that SMITH defined. Record those tables that are in PUBLIC dbspaces for later use in step 8.

3. Determine whether any of SMITH's tables participate in a referential structure that is not wholly owned by SMITH.

   ```
   SELECT TNAME, TCREATOR, REFTNAME, REFTCREATOR FROM SYSTEM.SYSKEYS
   WHERE  (TCREATOR ¬= 'SMITH' AND REFTCREATOR = 'SMITH')
   OR (TCREATOR = 'SMITH' AND REFTCREATOR ¬= 'SMITH')
   ```

This command displays tables created by others that reference tables created by SMITH, as well as tables created by SMITH that reference tables created by others. Make note of the tables you want to save.

4. Determine if the PRIVATE dbspace owned by SMITH contains any tables that were created by other users. Remember that when you drop a dbspace, you drop all tables that exist in it, whether they were created by the owner or by other users.

   For each PRIVATE dbspace owned by SMITH, type:

   ```
   SELECT TNAME,CREATOR FROM SYSTEM.SYSCATALOG
     WHERE CREATOR¬='SMITH'
     AND DBSPACENAME='dbspacename'
     AND TABLETYPE='R'
   ```

   This command lists the names of all tables in *dbspacename* that SMITH did not create, along with the names of who created them. The TABLETYPE='R' (R stands for real table) indicates that you want to see only the tables at this point, not views.

5. Based on the information you acquired in the last three steps, transfer any tables that you want to save. If any of these tables participate in referential structures, the referential constraints must be rebuilt to reflect the changed ownership of the tables.

   There are many ways to transfer (copy) tables to another dbspace. One way is to first create a new table with the same format in a different dbspace; then use an INSERT with Subselect statement to retrieve data from the original table and insert it into the new table.

   There are more sophisticated techniques available using the DBS Utility. For information, refer to "Maintaining Tables" on page 59 or to the *DB2 Server for VSE & VM Database Services Utility* manual.

6. Copy any programs that you want to save that currently reside in SMITH's PRIVATE dbspaces into another dbspace.

7. Drop the PRIVATE dbspaces owned by SMITH, which you determined in step 1, by issuing the DROP statement:

   ```
   DROP DBSPACE SMITH.dbspacename
   ```

8. Drop any of SMITH's tables you no longer need, as determined in step 2. All associated indexes and views are also dropped.

   ```
   DROP TABLE SMITH.tablename
   ```

9. Drop any of SMITH's views that were defined on other users' objects in PUBLIC dbspaces or in other users' PRIVATE dbspaces. To get the names of those views from the catalog tables, type:

   ```
   SELECT VIEWNAME FROM SYSTEM.SYSVIEWS
      WHERE VCREATOR='SMITH'

   DROP VIEW SMITH.viewname
   ```

10. Drop any of SMITH's indexes that were defined on other users' objects in PUBLIC dbspaces or in other users' PRIVATE dbspaces. To get the names of those indexes from the catalog tables, type:

    ```
    SELECT INAME FROM SYSTEM.SYSINDEXES
       WHERE ICREATOR='SMITH'

    DROP INDEX SMITH.indexname
    ```

11. Drop any of the packages created by SMITH. To display the names of those packages from the catalog tables, type:

```
SELECT TNAME FROM SYSTEM.SYSACCESS
   WHERE CREATOR='SMITH'

DROP PACKAGE SMITH.packagename
```

12. Delete any synonyms created by SMITH:

    ```
    DELETE FROM SYSTEM.SYSSYNONYMS
       WHERE USERID='SMITH'
    ```

13. Delete any ISQL stored queries created by SMITH.

    To determine these queries, type:

    ```
    SELECT STMTNAME FROM SQLDBA."STORED QUERIES"
       WHERE CREATOR='SMITH'
    ```

    Then issue a single DELETE statement:

    ```
    DELETE FROM SQLDBA."STORED QUERIES"
       WHERE CREATOR='SMITH'
    ```

    It is helpful if departing employees remove their own data from the database. Only someone with DBA authority can delete stored queries in the above manner; others use the ISQL ERASE command. For example, to delete a stored query called MYQUERY, SMITH would start ISQL and type:

    ```
    ERASE MYQUERY
    ```

14. Revoke any privileges granted to SMITH. To get the names of all users who granted privileges to SMITH, type:

    ```
    SELECT * FROM SYSTEM.SYSTABAUTH
       WHERE GRANTEE='SMITH'

    SELECT * FROM SYSTEM.SYSPROGAUTH
       WHERE GRANTEE='SMITH'
    ```

    Contact these users and have them revoke all of SMITH's privileges. Or, if a user is not available, you can explicitly connect with his or her password to revoke them yourself.

15. In VM, remove any IUCV links.

    If SMITH's VM directory contains IUCV entries or the MAXCONN OPTION for the database resources, these entries should be removed, as well as access to the 195 production disk.

16. Remove Access from VSE guests.

    If SMITH accessed a DB2 Server for VM application server from a VSE guest, and used the CICS system, you should remove the transaction IDs used by SMITH in the CICS system. For more information on transaction IDs, see the *DB2 Server for VM System Administration* manual.

Chapter 5. Providing Security

The database manager controls security with *authorities* and *privileges* granted to users (identified by their user IDs). Authorities limit people's use of DB2 resources (for example, whether they can create tables in PUBLIC dbspaces or acquire PRIVATE dbspaces), while privileges provide security for existing objects in the database (tables, views, indexes, and packages).

All privileges and authorities held within an application server are recorded in the catalog tables.

To access and perform SQL requests for an application server, users (ISQL users, DBS Utility users, and application programs) must be allowed to CONNECT to the application server *implicitly* (without a user ID or password), or *explicitly* (with a user ID and its password). With either type of connecting, the user can work with utilities, programs, and the data in the database based on pre-established authorities. Connecting is much the same as logging on to the VM or VSE system.

This chapter discusses the following topics:

1. Authorities.

   This section discusses the four types of authorities and how they can be given (granted) to or taken away (revoked) from users.
2. User Privileges.

   This section describes how privileges can be used to share or restrict access to the data in tables or views.
3. Connecting to an Application Server

   This section discusses how a user can connect to an application server. Users must connect to an application server before they can use it.
4. Restricting Access Using Views.

   This section discusses the use of views to restrict access to tables.
5. Changing User Passwords.

   This section describes how you can change the password of your DB2 Server for VSE & VM users.
6. Securing the Database Catalog Tables.

   This section discusses how you can limit access to the catalog tables.
7. Security Audit Trace.

   This section describes the two ways that you can audit security: by querying the catalog tables or by having the database manager do a security audit trace.

Authorities

When a database is initially generated, there is only one user ID defined for it: SQLDBA. This user ID belongs to the database administrator (DBA). Only a DBA can grant or revoke authorities to other users.

Types of Authorities

There are four types of authority: CONNECT, RESOURCE, SCHEDULE, and DBA.

Authorities are hierarchical, with DBA the highest, RESOURCE and SCHEDULE the next, and finally CONNECT. If you have a higher authority, then you also have the authority below it. For example, if you are given DBA authority, you have RESOURCE, SCHEDULE, and CONNECT authority as well. If you are given RESOURCE authority, you also have CONNECT authority but not SCHEDULE or DBA authority.

CONNECT Authority

This authority enables a user to access a particular application server, and to exercise all privileges that have been granted to PUBLIC. These privileges are discussed in detail in "Privileges" on page 92.

A user with CONNECT authority can access data in one of two ways:

- By owning a PRIVATE dbspace, in which he or she can create tables and load and access them. A user with DBA authority must acquire the dbspace for this user.
- By receiving access privileges (such as SELECT, INSERT, and UPDATE) for tables created by other users. See "Privileges" on page 92.

RESOURCE Authority

Users with this authority can acquire PRIVATE dbspaces for themselves, and create tables both there and in PUBLIC dbspaces.

A DBA automatically possesses RESOURCE authority and the ability to grant it to users. You can give it to just a few users to exercise tight control, or you can extend it to any number. If you want to allow someone to create tables and you must also control how much resources are used, acquire a PRIVATE dbspace for that user rather than granting him or her RESOURCE authority. Because you acquire this dbspace yourself, you control its size and the amount of resources used. This technique is sometimes called "CREATE TABLE authority", but this term is misleading because there is no GRANT CREATE TABLE statement.

SCHEDULE Authority

The function associated with SCHEDULE authority is not available in the SQL statement set. Therefore, DB2 Server for VSE & VM users cannot use it and SCHEDULE authority is of no direct benefit to DB2 Server for VSE & VM users.

SCHEDULE authority is useful only to online resource managers that manage subsystems of multiple second-level users. The only current example is the DB2 Server for VSE online resource adapter that manages secondary users through the CICS subsystem. The CICS subsystem is a first-level user of the database manager. The use of SCHEDULE authority in a CICS subsystem is discussed here.

The online resource adapter resides in each CICS partition. It initializes the communication links between the CICS partition and the local DB2 Server for VSE database manager, or the DB2 Server for VM database manager accessed through guest sharing, when the operator executes the CICS CIRB transaction or the CICS CIRA transaction. It also does a CONNECT on each link, specifying DBDCCICS as the user ID and SQLDBAPW as the password. This user ID and password can be overridden. Refer to "CICS Transaction Environment" on page 101 for details.

The online resource adapter in each CICS partition can connect to many application servers. The DBNAME parameter of the CIRB or CIRA transaction specifies the application server to which you want to connect. If DBNAME is not specified on the CIRB transaction, the default application server is used. Refer to "Establishing a Default Application Server" on page 99 for information on

DBNAME default rules. All online applications in a CICS partition can access the application servers connected with the online resource adapter.

The schedule function comes into play when a CICS transaction uses SQL statements **without** preceding them with a CONNECT statement[1] on a local application server or on a VM application server accessed through guest sharing. When this occurs, the resource adapter sends a schedule request to the database manager. This request travels on the link being used by the transaction. A schedule request is similar to a CONNECT, but it has no password. The resource adapter determines the user ID as described in "CICS Transaction Environment" on page 101.

The schedule function allows dynamic changing of the current user ID on a link to the database manager without requiring a password. For this to occur, the initial user of the link must have SCHEDULE authority. For a CICS session, the initial user of the link is the unique application name (APPLID) assigned to the CICS partition in the DFHSIT table. The default APPLID name is DBDCCICS. This user ID represents the entire CICS subsystem. The database administrator must grant each APPLID SCHEDULE authority on the application server so that the links to the database manager can be shared implicitly by multiple transactions. If a CICS partition is to connect to more than one local application server, the APPLID for the partition must be granted SCHEDULE authority on each application server.

Transactions that do not issue CONNECT statements[1] receive their connection to the database manager implicitly through the CICS subsystem. The assumption is made that the CICS subsystem checked the user's identification and password when the user began the CICS session, so the database manager does not need to do further checking. On the other hand, each transaction is subject to all the other security controls. The user ID received by the database manager with the schedule request is the basis for this transaction user's authorization.

Because CONNECT authority is not needed for CICS transactions, the user IDs that they use need not appear in the SYSTEM.SYSUSERAUTH catalog table. This catalog table does **not necessarily** have an entry for every user. Second-level users can access all PUBLIC data and may be granted access to PRIVATE data as well. Although a user may not be given CONNECT authority explicitly, that user can be granted RESOURCE authority or SCHEDULE authority and will receive CONNECT authority as a result.

Note: This discussion applies only to transactions that do not issue a CONNECT statement. When a transaction does issue a CONNECT statement[1], it appears as an ordinary user, and the schedule function is not used.

A user possessing DBA authority possesses SCHEDULE authority and the ability to grant SCHEDULE authority to other users.

To grant SCHEDULE authority, use a statement such as:

```
GRANT SCHEDULE TO dbdccics IDENTIFIED BY password
```

If the user's password has been entered previously and is not to be changed, you can omit the "IDENTIFIED BY password" portion of the GRANT statement. Refer to "CICS Transaction Environment" on page 101 for details.

1. The CONNECT statement with the following format: CONNECT *userid* IDENTIFIED BY *password*.

DBA Authority

Authorization mechanisms do not apply to users with this authority. They can perform all operations on all tables, can run all programs, and are the only ones who have the following privileges:

- Grant and revoke SCHEDULE, CONNECT, RESOURCE, and DBA authority to/from other users. All DBAs at a site have equal authority, and can grant and revoke DBA authorities to each other. Because no user may revoke his or her own authority, there will always be at least one DBA (not necessarily the original one).
- Acquire a PUBLIC dbspace.
- Alter or drop any PUBLIC dbspace except for system dbspaces (those whose names begin with "SYS").
- Acquire, alter, or drop a PRIVATE dbspace or create, alter, or drop a table, index, synonym or view, in the name of another user.
- Drop a package belonging to another user.
- Lock another user's PRIVATE dbspace or any PUBLIC dbspace (except system dbspaces).
- Lock another user's table (except the catalog tables).
- Issue a COMMENT statement on a table or field owned by another user.
- Create a table in a system dbspace.
- Issue Data Manipulation Language statements directly against an inactive table. See "Altering Referential and Unique Constraints" on page 65.
- Modify the contents of a catalog table with a regular UPDATE statement. Rows cannot be INSERTed or DELETEd. Because all access to the data in the database depends on the correctness of the catalog tables, manual updating of catalog tables should be done only under extraordinary circumstances. Only a small set of catalog table columns can be updated. These are listed in the *DB2 Server for VSE & VM SQL Reference* manual.
- For Extended Dynamic Statements:
  - Drop another user's program (package) or drop a statement from that package.
  - Use PREPARE, DESCRIBE, EXECUTE, or DECLARE CURSOR for a statement residing in another user's package.

No user, including those with DBA authority, can drop a catalog table.

As DBA, you may perform certain operations that are otherwise unauthorized, but may not grant or revoke these operations. For example, you may update a particular table that you do not own explicitly, but you may not grant or revoke this privilege to others.

The functions enabled by DBA authority are potentially quite dangerous to the integrity of the database if applied by an untrained user. Therefore, you should carefully control who receives this authority, as well as being very cautious in the use of this special authority yourself.

Granting Authorities

To grant any authority (SCHEDULE, CONNECT, RESOURCE, or DBA) to other users of an application server, issue the GRANT statement. You must have DBA authority on that application server. For information on the syntax of this statement, see the *DB2 Server for VSE & VM SQL Reference* manual.

Granting someone a higher authority automatically gives them the lower authority as well, regardless of whether these are specified on the GRANT statement. Thus, a user who is granted RESOURCE authority will also have CONNECT authority; one who has DBA authority also has CONNECT, RESOURCE, and SCHEDULE authority.

If you are granting authority to a user at a remote system, the *authorization-name* specified in the GRANT statement must be the authorized user ID of the user on the system where the authority is being granted, not that on the system where the request originates.

Examples

Granting authority to a single user: To give the user ID MIKE CONNECT authority to the application server, enter:

```
GRANT CONNECT TO MIKE IDENTIFIED BY mikespwd
```

If the user MIKE intends to connect to the application server implicitly, you can omit his password:

```
GRANT CONNECT TO MIKE
```

Granting authority to many users: To give the user IDs MIKE and JOHN RESOURCE authority to the application server, enter:

```
GRANT RESOURCE TO MIKE,JOHN IDENTIFIED BY mikespwd,johnspwd
```

If MIKE intends to connect to the application server implicitly, you may omit his password and just enter:

```
GRANT RESOURCE TO JOHN IDENTIFIED BY johnspwd
GRANT RESOURCE TO MIKE
```

Granting CONNECT authority to all users: The following statement enables all users to connect to the application server implicitly:

```
GRANT CONNECT TO ALLUSERS
```

Users who wish to connect explicitly to the application server must be given CONNECT authority with a password. In VM, the ability to communicate with a DB2 Server for VM database manager depends on VM directory statements and is discussed in the *DB2 Server for VM System Administration* manual.

Granting Access to VSE Guests

When VSE/AF runs as a guest operating system under the VM/ESA operating systems, VSE users and programs can optionally access a DB2 Server for VM application server. A VSE guest who wishes to do this must obtain authorization. On the GRANT statement, specify a VM user ID that is authorized to run the VSE subsystem.

The *subsystemid* follows the same general rules for naming data objects as the user ID, and cannot contain lowercase characters, special characters, or DBCS characters.

Example: To give the CICS subsystem MYCICS SCHEDULE authority, enter:

```
GRANT SCHEDULE TO MYCICS IDENTIFIED BY cicspw
```

where *cicspw* is the current password set for the subsystem.

Revoking Authorities

To revoke authorities previously granted to users, issue the REVOKE statement. You must have DBA authority. For information on the syntax of this statement, see the *DB2 Server for VSE & VM SQL Reference* manual.

Revoking a user's CONNECT authority does not automatically cause any objects owned by that user to be dropped, nor does it revoke any privileges the user has on those objects. For information on how to drop objects, see "Removing Users from an Application Server" on page 83.

If a user's CONNECT authority is revoked, all other authorities are lost. For example, if you are a DBA and another DBA revokes your CONNECT authority, then you will lose your RESOURCE, SCHEDULE, and DBA authorities as well.

A user who loses RESOURCE authority will still have CONNECT authority. You cannot revoke RESOURCE authority from a user with DBA authority.

A user who loses SCHEDULE authority will still have CONNECT authority. You cannot revoke SCHEDULE authority from a user with DBA authority.

A user who loses DBA authority will also lose RESOURCE and SCHEDULE authority, but will retain CONNECT authority.

When revoking remote users, the *authorization-name* specified in the REVOKE statement must be the authorized user ID of the user on the remote system where the authority is being revoked, not that on the system where the request originates.

Examples

To revoke JOHN's CONNECT authority, enter:

```
REVOKE CONNECT FROM JOHN
```

To revoke JOHN and ALICE's DBA authority, enter:

```
REVOKE DBA FROM JOHN,ALICE
```

To revoke JOHN and ALICE's SCHEDULE authority, enter:

```
REVOKE SCHEDULE FROM JOHN,ALICE
```

Revoking Access from VSE Guests

Use the REVOKE SCHEDULE statement to revoke remote access by a VSE subsystem.

To revoke the SCHEDULE authority of the CICS subsystem called MYCICS, enter:

```
REVOKE SCHEDULE FROM MYCICS
```

Privileges

The DBA grants authorities to the users of the application server. Within the framework set up by the DBA, individual users can grant to each other the privileges they need to access specific data. To grant or revoke privileges on an object, a user must hold GRANT authority on those privileges, and be connected to the application server where the object resides.

The following are the privileges that can be held on a table (or view) in the database:

SELECT To read from a table

INSERT	To add rows to a table
DELETE	To delete rows from a table
UPDATE	Can apply to individual columns
ALTER	To add new columns, primary keys or foreign keys to a table, or to activate or deactivate existing keys
INDEX	To create or manipulate indexes on a table
REFERENCES	To add, drop, activate, or deactivate a foreign key relationship

The first four privileges in this list apply to views as well as to tables.

The holder of a privilege may exercise it directly through a user mechanism such as ISQL, or by compiling and running programs that entail using it.

Privileges of Ownership

When an object is created, its ownership is established. If the object name is not qualified (for example, EMPLOYEE), the owner is the connected user. If the object is qualified (for example, JESSICA.EMPLOYEE), the owner is the individual whose user ID is specified. The owner of an object automatically has full privileges on it.

Once the ownership of a table or view is established, there is no way to change it or to revoke the privileges that accompany ownership. If either of these is necessary, you must drop the object, which deletes all privileges on it, and then re-create it with a new owner.

Granting Privileges to Other Users

The owner of an object possesses the GRANT option on each privilege, meaning the ability to grant individual privileges, or any combination of them, to other users. When a privilege is granted, the GRANT option (the ability for the recipient to in turn make further grants) may or may not be included.

Privileges can be granted to other users using the GRANT statement described in the *DB2 Server for VSE & VM SQL Reference* manual.

- Issuing GRANT ALL or GRANT ALL PRIVILEGES grants the recipient all the privileges possessed by the grantor on that object (which may of course not include all possible ones). If GRANT ALL is issued on a view, only the privileges on the view, not those on the base tables, are granted.
- Issuing GRANT REFERENCES enables the recipient to reference the parent table when a foreign key is added, dropped, activated, or deactivated through the CREATE TABLE or ALTER TABLE statements.
- Issuing GRANT ALTER enables the recipient to add a new column or to add, drop, activate, or deactivate a primary or foreign key. To alter a primary key, the ALTER privilege is required on the parent table and **all** dependent tables. To alter a foreign key, the ALTER privilege is required on the dependent table, and the REFERENCES privilege is required on the parent.

  Withholding these privileges restricts the ability of the recipient to change the state of referential constraints. If the owner of a parent table grants the REFERENCES privilege on it to another user, and the recipient then creates a foreign key relationship with the parent's primary key but does not grant ALTER privilege on the dependent table back to the owner of the parent table, the owner cannot drop the primary key. (He or she may, of course, drop the entire table.)
- The UPDATE privilege can apply to specific columns. For example, the following statement will allow CINDY to update the address (ADDR) and phone number columns (PHONE) of the EMPDATA table:

```
GRANT UPDATE (ADDR,PHONE) ON EMPDATA TO CINDY
```

If you are granting a user privileges at a remote system, the *authorization-name* specified in the GRANT statement must be the same as the name that the grantee uses to access the database manager system on the remote system.

Revoking Privileges

A user who grants another user a privilege may later revoke it, by issuing the REVOKE statement described in the *DB2 Server for VSE & VM SQL Reference* manual. If a user loses a privilege, all other users to whom that user granted it automatically lose it too by the *cascading* effect, unless they have another independent source for it. Issuing REVOKE ALL or REVOKE ALL PRIVILEGES takes away all privileges that were granted.

If you are revoking a user's privileges at a remote system, the *authorization-name* specified in the REVOKE statement must be the name that the user specifies to access the database manager system on the remote system.

Monitoring Privileges

All the privileges held by users on tables and views are listed in the catalog tables SYSTEM.SYSTABAUTH and SYSTEM.SYSCOLAUTH. Users can check which privileges they hold and which they have granted to others, by querying these tables.

Examples

To determine the privileges that you hold, enter:

```
SELECT * FROM SYSTEM.SYSTABAUTH
         WHERE GRANTEE = user
```

To determine the privileges that you have granted to other users, enter:

```
SELECT * FROM SYSTEM.SYSTABAUTH
         WHERE GRANTOR  = user
           AND GRANTEE <> user
           AND GRANTEETYPE = ' '
```

For descriptions of the catalog tables, see the *DB2 Server for VSE & VM SQL Reference* manual.

Privileges on Application Programs

DB2 Server for VSE & VM application programs must be preprocessed before they are compiled or assembled. In VM and VSE batch environments, successful preprocessing of an application program results in the creation or replacement of a *package* in the database. In VSE, successful preprocessing and/or CBNDing of an application program results in the creation or replacement of a package in the database. The contents of the package are instructions used to satisfy database requests at run time.

When a package is created, a level of EXECUTE privilege is granted to its creator. This level is dependent on several factors, such as the preprocessed SQL statements, the existence and ownership of the referenced objects (tables, indexes, and dbspaces), and the creator's authorization level (DBA, RESOURCE, or CONNECT). The creator's EXECUTE privilege follows rules and conditions that are discussed in the *DB2 Server for VSE & VM Application Programming* manual.

Connecting to an Application Server in VM

A VM user must have CONNECT authority and be connected to an application server in order to perform SQL requests on it.

All VM users must connect to an application server explicitly or implicitly regardless of whether they are accessing it in multiple user mode or single user mode. If a user does not have a DB2 Server for VM authorization ID and password, the user must connect implicitly. A user with a DB2 Server for VM authorization ID and password can connect either implicitly or explicitly.

Establishing a Default Application Server

In order to run a preprocessor, the DBS Utility, any application program, or ISQL, VM users must establish a default application server. This is done by invoking the SQLINIT EXEC, and needs to be done only once.

Example

To establish the SQLDBA application server as the default, enter:

```
SQLINIT DBNAME(SQLDBA)
```

Information about the default application server chosen is stored on the VM user's minidisk (A-disk) in the ARISRMBT module and the LASTING GLOBALV file. If the VM user wants to establish another application server as the default or to change any of the options, he or she would have to re-run the SQLINIT EXEC. For more information see "SQLINIT EXEC" on page 235.

Connecting to the Application Server Implicitly

Connecting to the application server implicitly means to connect to it without providing an authorization ID and password explicitly. If a VM user does not provide a CONNECT statement, then the first time that he or she tries to run an SQL statement, the VM application requester connects to the application server implicitly. The database manager checks its catalog tables to see whether that user's ID, the VM logon ID (established in the CP LOGON procedure), has been granted CONNECT authority. (It does not compare the user's CP LOGON password with the DB2 Server for VM application server password, as it can be assumed that a password that has been verified by the CP LOGON procedure is valid.)

Most VM users will want to connect to the application server implicitly, so when you grant them CONNECT authority, use their CP LOGON user IDs.

The implicit connect support works the same for VM application programs, for ISQL, for the DBS Utility, and for remote application servers; however, each has its own considerations, as discussed below.

Note: When working in an environment that includes several application servers that can be accessed from several different application requesters, there is the need for unique authorization IDs. The database manager does not recognize the same authorization ID from two (or more) different application requesters as being different. It is the administrator's responsibility to ensure that the authorization IDs in this situation are unique.

How Implicit CONNECT Applies to VM Programs

For application programs that contain SQL statements, a distinction is made between the *creator* and the *runner* of the program.

- The creator is the VM user who submits the program to one of the language preprocessors. This individual's authorization ID, which is specified in the USERID parameter passed to the preprocessor, is used to perform all authorization checking for the functions performed against data managed by DB2 Server for VSE & VM, and is the default *owner* of all objects (tables or views) created by the program. This authorization ID automatically has the EXECUTE privilege for the program.

  When not coded explicitly, the authorization ID is derived from the CP LOGON.
- The runner is the VM user who runs (executes) the program. This individual's authorization ID is either that specified in the CONNECT statement run by the program, or is the authorization ID that is connected implicitly. The runner may be the creator, or may be someone to whom the creator has granted the EXECUTE privilege.

  When coded explicitly, the authorization ID and password for the CONNECT statement are derived from host variables in the program. The values for these variables should be acquired at run time from control cards by the executing program. If they are constants fixed in the program, anyone can run the program.

  When not coded explicitly, the authorization ID is derived from the CP LOGON.

Refer to the *DB2 Server for VSE & VM Application Programming* manual for more information about how implicit CONNECT applies to application programs.

How Implicit CONNECT Applies to ISQL (VM)

To start ISQL, a VM user invokes the ISQL EXEC. The database manager always initially does an implicit connect for ISQL users, so this EXEC does not accept an authorization ID. The authorization ID is derived from the ID of the user's virtual machine, as described on page 95.

The user can issue explicit CONNECT statements to override any previous explicit or implicit connection established for the ISQL session.

Refer to the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual for more information.

How Implicit CONNECT Applies to the DBS Utility (VM)

When the DBS Utility begins processing an input control file, it expects a CONNECT statement before any other DBS Utility or SQL statements. If none is supplied, the database manager will use the ID of the user's virtual machine.

If the utility is invoked from an application that has already issued a CONNECT statement (implicitly or explicitly), then another one is not expected. Here, the authorization ID that was in effect when the program first invoked the utility is used.

The user can issue explicit CONNECT statements to override any previous explicit or implicit connection.

Refer to the *DB2 Server for VSE & VM Database Services Utility* manual for more information.

How Implicit CONNECT Applies to Remote Application Servers

When a VM user implicitly connects to a remote application server, the authorization ID passed by the requester or received by the server may be different than the VM logon user ID. It will depend on how the CMS Communication Directory has been set up for the requester, and whether the server performs user ID translation. Refer to the *Distributed Relational Database Connectivity Guide* manual for more information about security levels specified in the CMS Communication Directory when implicitly connecting to a remote application server.

How Implicit CONNECT Applies to TCP/IP

When a VM user implicitly connects to an application server using TCP/IP as the communications protocol, an explicit connect is performed by the resource adapter using the authorization ID and password found in the CMS Communications Directory. There is no implicit connect when TCP/IP is being used.

Connecting to the Application Server Explicitly

VM users may want to connect to an application server other than the default one, switch to another application server, or connect to an application server as a different authorization ID. These situations entail making an explicit connection.

Switching to Another Application Server

After connecting to an application server, a VM user may want to switch to a different one. The user issues an SQL CONNECT statement to switch to this second application server.

Example - Without Specifying an Authorization ID and Password: To switch to the DB01 application server, enter:

```
CONNECT TO DB01
```

Since the authorization ID and password are not specified on the CONNECT statement, they will be taken from the VM communications directory file if it is used and if it contains an entry for the DB01 application server. If the file is not used, if it does not exist, or if it does not contain an entry for the DB01 application server, the VM logon user ID will be used in an implicit connect.

If this statement fails, the VM user will **not** remain connected to the original application server and no other SQL statements will be accepted. The VM user will have to issue a new CONNECT statement.

When the VM user issues the first SQL statement to be processed on the second application server, the database manager will try to implicitly connect him or her to that application server, using the VM logon user ID as the authorization ID. VM users can avoid the implicit connect by connecting as another user (discussed next) while switching application servers.

Example - Specifying an Authorization ID and Password: To switch to the DB01 application server under an authorization ID JOHN with a password of *johnpw*, enter:

```
CONNECT JOHN IDENTIFIED BY johnpw TO DB01
```

Note: CONNECT *userid* IDENTIFIED BY *password* is not supported for the Distributed Relational Database Architecture (DRDA) protocol.

If this statement fails, the VM user will **not** remain connected to the original application server and no other SQL statements will be accepted. The user will have to issue a new CONNECT statement.

Connecting under Another Authorization ID

A VM user connects under another authorization ID to the currently established application server by issuing an SQL CONNECT statement. If the user is not currently connected to an application server, if the previous connection has been released, or if the user switched to a new CMS Work Unit in VM/ESA, then the default application server, established by the SQLINIT EXEC, will be used.

Example: To connect to the currently established application server under the authorization ID JOHN with a password of *johnpw*, enter:

```
CONNECT JOHN IDENTIFIED BY johnpw
```

Note: CONNECT *userid* IDENTIFIED BY *password* is not supported for the DRDA protocol.

If this statement fails, the VM user will remain connected to the application server as the original authorization ID.

A previous connection could be released for the following reasons:

- A COMMIT RELEASE or ROLLBACK RELEASE statement was issued.
- The previous logical unit of work (LUW) was canceled by the user (using SQLHX or ISQL CANCEL) or by the operator (using the FORCE statement). The cancelation releases the connection.
- The previous connection was disabled by the operator (using FORCE DISABLE) or by other errors such as the database machine not being ready or communications problems.
- In the VM environment, the previous connection was disabled by the operator using a FORCE without the DISABLE option.

Determining the Currently Established Application Server

If a user issues an SQL CONNECT statement without any parameters, the database manager will return the following information:

- The currently connected user ID
- The application server name
- The product ID which can be 'ARI ' or 'ARI7010' depending on when the CONNECT was issued.

Refer to the *DB2 Server for VSE & VM SQL Reference* manual for more information about the CONNECT statement.

Connecting to an Application Server in VSE

To control access to the data managed by the database manager, it is necessary to:

- Tell the database manager the users that are authorized to use the DB2 Server for VSE database and protect their access by means of passwords.
- Inform the database manager when a particular user wants to begin accessing the DB2 Server for VSE database.

The authority to use a DB2 Server for VSE database is established by granting a user CONNECT authority. The CONNECT authority carries with it a DB2 Server for VSE password, which is that user's key to the application server. After a user has received CONNECT authority (been assigned an authorization ID and password), the user can begin to use an application server through the CONNECT function. After users have received their authorization IDs and passwords, they can change their own passwords at any time.

All VSE users must connect to an application server explicitly or implicitly:

- Batch users must have a DB2 Server for VSE authorization ID and password, and must connect explicitly.
- Online users who do not have a DB2 Server for VSE authorization ID and password must connect implicitly. Other online users can use either method.

The CONNECT function can also be used either directly (through the CONNECT statement with the "*userid* IDENTIFIED BY *password*" clause) or indirectly (through a subsystem logon procedure). The procedure for connecting to an application server is slightly different for each user environment. The following sections describe these situations.

Establishing a Default Application Server

You may identify the desired application server by specifying the DBNAME parameter at system startup, on the CICS CIRB or CICS CIRC transaction, on the CONNECT statement, when preprocessing, or when CBNDing. If you do not specify a server name, these DBNAME default rules apply:

- The partition default DBNAME is used if it is specified in the PARTDEF field of the DBNAME Directory.
- If a partition default is not specified, the system default DBNAME is used if it is specified in the SYSDEF field of the DBNAME Directory.
- If neither a partition nor a system default is specified, the default DBNAME is SQLDS and the default APPLID is SYSARI00.

  Note: SQLDS must still be identified in the DBNAME Directory.

For further information on the DBNAME Directory, refer to the *DB2 Server for VSE System Administration* manual.

Connecting to the Application Server in Different VSE Environments

DB2 Server for VSE users can connect to the application server in the following environments:

CICS/VSE Online Environment

In a CICS/VSE online environment, online users can connect to the application server implicitly and explicitly. If online users do not explicitly issue a CONNECT statement specifying the authorization ID and the password, then the first time they try to process an SQL statement, the CICS/VSE user is connected to the application server implicitly.

A CONNECT...TO statement is supported in this environment and can be used to switch to a different application server between logical units of work. For further information on switching, refer to "Switching to Another Application Server" on page 103.

If the first SQL statement in a CICS/VSE application is not a CONNECT statement with the TO clause, the default application server is connected. On subsequent CONNECTs performed by that application, if the TO parameter is not specified, then the connection to the previously connected server will be maintained. For further information on default application servers, refer to "Establishing a Default Application Server".

Batch/Interactive Environment

In a VSE batch or VSE/ICCF environment, an explicit CONNECT must be the first statement entered by the batch user to access the application server. This statement is described in the *DB2 Server for VSE & VM Application Programming* manual. Explicit connection is required for all user programs. This connection identifies the authorization ID, and optionally the name of the application server on which the program will run.

A CONNECT..TO statement is supported in this environment and can be used to switch to a different application server between logical units of work. For further information on switching, refer to "Switching to Another Application Server" on page 103.

If the first SQL statement in an application is a CONNECT statement in which the TO *server_name* clause is not specified, or if this clause is not specified as part of the CONNECT statement following a COMMIT RELEASE or ROLLBACK RELEASE statement, the default application server is connected. If the TO *server_name* clause is not specified as part of the CONNECT statement following a COMMIT or ROLLBACK statement, the connection to the previously connected server will be maintained. For further information on default application servers, refer to "Establishing a Default Application Server" on page 99.

In this environment, there is a distinction between the user who preprocesses a program that contains SQL statements, and the user who later runs that program.

- The *creator* of a program is the VSE user who submits the program to one of the language preprocessors. The user ID specified in the USERID= parameter passed to the preprocessor is the basis for all authorization checking for the functions performed against data managed by the system as well as the default *owner* of all objects (tables or views) created by the program. This user ID receives RUN authority when the program is successfully preprocessed.
- The *runner* of a program is the VSE user who runs (executes) a program that contains SQL statements. The user ID specified in the CONNECT statement run by the program must be either the creator or a user ID to whom the creator has granted the RUN privilege for this program.

  The user ID and password for the CONNECT statement are derived from host variables in the program. Their values should be acquired at run time from control cards by the executing program. If they are constants fixed in the program, anyone can run the program.

  The runner of a program gets the privilege of accessing the application server from the creator of the program.

ISQL Environment

When CICS users start ISQL, they are prompted for a user ID, password, and target database. If the user enters the user ID and password only, ISQL does an explicit CONNECT to the default target database for the user. If the user does not enter a user ID, password or target database, ISQL does a CONNECT to the default target database as a default user ID for the user; this defaulting is called an implicit CONNECT. If the ISQL user enters a target database only, a CONNECT would be made to that target database using a default user ID. If the user enters the user ID, password and target database, ISQL does an explicit CONNECT to the target database.

In the ISQL environment, you can access any of the application servers connected with the online resource adapter. If the online resource adapter is not connected to an application server, you cannot access the ISQL environment.

Note: The ISQL environment is a specific case of the CICS transaction environment, which is discussed in the next section. An ISQL user can enter explicit CONNECT statements to change the connection and override any previous explicit or implicit connection established for the ISQL terminal session.

Refer to the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* or the *DB2 Server for VSE & VM SQL Reference* manual for additional details.

CICS Transaction Environment

Online transactions need not enter a CONNECT command to establish the user ID within the database manager. If a CONNECT command[2] is not entered, the online support establishes the authorization ID for the transaction. The implicit CONNECT is carried out by a SCHEDULE call in the case where the online transaction is connecting to a local application server.

This implicit CONNECT capability is useful if your installation requires terminal users to sign on CICS. For many transactions, your installation might consider the sign-on verification sufficient. It may also be useful if your installation has just installed the database manager, and finds it convenient to have all users identified by one name (for example, TESTUSER).

The online support establishes a user ID for CICS transactions connecting to a local DB2 Server for VSE application server as follows:

1. If the local transaction issues a CONNECT command[2] the user ID is established explicitly for the application.
2. If the transaction does not issue a CONNECT command,[2] the online support establishes the user ID as follows:
   a. If the transaction had a user ID established from a previous local logical unit of work (LUW) and that LUW did not specify the RELEASE option for COMMIT WORK or ROLLBACK WORK, that user ID is used. The CICS communication link to the application server is freed every time an LUW ends, and a new link is established for each LUW in the transaction.
   b. If the transaction has a valid CICS sign on *userid* and is associated with a terminal, the CICS signon *userid* is used for the user ID.
   c. If a user ID was specified as an input parameter to the CIRB or CIRA transaction that established connections to the application server, that user ID is used. The person who invoked CIRB or CIRA will know what the user ID is.
   d. If a user ID was not specified in the CIRB or CIRA transaction that established connections to the application server, the default user ID CICSUSER is established for your transaction.

After the user ID is determined as described above for cases b, c, and d, one more requirement must be met to successfully complete the connection to the application server: CONNECT authority must be granted to either the specific authorization ID or “ALLUSERS”. ALLUSERS is a special authorization ID that permits any user ID to be implicitly connected without having been specifically granted CONNECT authority, and can be used by the database administrator to turn on or turn off the implicit CONNECT capability. During database generation, ALLUSERS is granted CONNECT authority by default.

2. The CONNECT statement with the following format: CONNECT *userid* IDENTIFIED BY *password*.

At many installations, the CICS user need not be aware of DB2 Server for VSE authorization ID or authorization capabilities. Here, the CICS implicit connect support can be very useful.

Suppose you code a transaction called STAT that displays the inventory status of a given part. Banes and Smith are to be the users of the application.

You define Banes and Smith to the CICS signon process.

You must then authorize BANES and SMITH to run your program. Of course, you must have the RUN privilege with the GRANT option on your program. For this example, assume that the program was preprocessed with the name INVSTAT:

```
GRANT RUN ON INVSTAT TO BANES, SMITH
```

Note: BANES and SMITH do **not** need CONNECT authority. It is connected through internal mechanisms of the DB2 Server for VSE online support.

You must also establish the security key when you define the inventory program to CICS.

To use the STAT transaction, Banes and Smith merely sign on to the CICS subsystem by entering, for example:

```
CESN BANES, XXXX
```

After signed on, they need only enter the transaction identifier STAT, which causes the INVSTAT program to be loaded and invoked. Since there is no CONNECT statement in the program, the user ID established is the signed-on user ID (BANES). Because BANES was granted RUN authority on INVSTAT, the database manager allows the program to process.

Online applications can access any of the application servers connected with the online resource adapter. The online resource adapter can connect to many application servers using the CIRA or CIRB transactions.

Refer to the *DB2 Server for VSE & VM Application Programming* manual for additional information on this environment.

User IDs for Remote CICS/VSE Transactions

For online DB2 Server for VSE transactions which are accessing a **remote** server and which issued an SQL CONNECT statement with the "*userid* IDENTIFIED BY *password*" clause to establish the user ID within the database manager, the user ID is established explicitly for the transaction.

For online DB2 Server for VSE transactions which are accessing a **remote** server and which did not issue an SQL CONNECT statement with the "*userid* IDENTIFIED BY *password*" clause to establish the user ID within the database manager, the Online Resource Adapter will attempt to establish the user ID for the transaction implicitly as follows:

1. If the transaction had a user ID established for a previous remote logical unit of work, and the previous logical unit of work did not specify the RELEASE option for COMMIT WORK or ROLLBACK WORK, and the transaction did not switch to another application server, that user ID and its corresponding password are used. (Remember that every time a logical unit of work ends with RELEASE or the transaction switched to another application server, and you enter another SQL statement, you are implicitly connected as the CICS

signon userid. Therefore, the user ID has to be re-established if the transaction has more than one logical unit of work ending with RELEASE or if the transaction is switching application servers.)

2. The user ID returned by the CICS ASSIGN command is used for the user ID.

Connecting to an Application Server in Special Circumstances

VSE users can connect to an application server other than the default one, or connect to an application server as a different authorization ID. VSE batch users can switch from an application server to another. These situations require making an explicit connection. VSE online users can also switch from an application server to another, by issuing an SQL CONNECT statement with the TO parameter, provided that the online resource adapter has established connections to the application server.

Switching to Another Application Server

After connecting to an application server, a VSE user can switch to another one by issuing an SQL CONNECT statement. The switch occurs between logical units of work.

Example - Without Specifying an Authorization ID and Password

To switch to the DB01 application server, enter:

```
CONNECT TO DB01
```

Because the user ID and password are not specified on the CONNECT statement, the user ID and password used is determined according to the rules described in "CICS Transaction Environment" on page 101 and "User IDs for Remote CICS/VSE Transactions" on page 102. For VSE batch users, the user ID and password used in the previous LUW are used if the LUW ends with a COMMIT WORK or ROLLBACK WORK statement. However, if the LUW ends with a COMMIT RELEASE or ROLLBACK RELEASE statement, the next SQL statement after the CONNECT statement is unsuccessful.

If the CONNECT statement is not successful, the VSE batch user does **not** remain connected to the original application server, and no other SQL statements are accepted. The batch user has to enter a new CONNECT statement.

When the VSE user enters the first SQL statement to be processed on the second application server, the batch resource adapter or the online resource adapter connects the user to that application server using user ID and password previously established. A VSE user can switch to another application server as different ID by connecting as another user, as discussed in the next section.

Example - Specifying an Authorization ID and Password

To switch to the DB01 application server under an authorization ID JOHN with a password of *johnpw*, enter:

```
CONNECT JOHN IDENTIFIED BY johnpw TO DB01
```

If this statement is not successful, the VSE batch user does **not** remain connected to the original application server and no other SQL statements are accepted. The batch user will have to enter a new CONNECT statement.

Connecting under Another Authorization ID

A VSE user connects under another authorization ID to the established application server by issuing an SQL CONNECT statement. If the user is not connected to an application server, the default application server is accessed. For batch users, if the previous connection has been released, the default application server is accessed.

Example

To connect to the currently established application server under the authorization ID JOHN with a password of *johnpw*, enter:

```
CONNECT JOHN IDENTIFIED BY johnpw
```

If this statement fails, the VSE user will remain connected to the application server as the original authorization ID.

A previous connection could be released for the following reasons:

- A COMMIT RELEASE or ROLLBACK RELEASE statement is entered.
- The previous LUW is canceled by a local user entering the FORCE DISABLE statement, or a remote user entering the FORCE statement. Canceling an LUW in this manner releases the connection.
- The previous connection is ended by the operator (using FORCE DISABLE) or by other errors, for example, communications problems.
- The CICS transaction switched from a local to a remote server, from a remote to a local server, from one remote server to another remote server.

Determining the Current User ID and Application Server

If a user enters an SQL CONNECT statement without any parameters, or after the successful execution of a CONNECT statement, the database manager returns the following information in the SQLCA:

- Currently connected user ID
- Application server name
- Product ID, which can be 'ARI ' or 'ARI7010' depending on when the CONNECT was issued.

Refer to the *DB2 Server for VSE & VM SQL Reference* manual for more information about the CONNECT statement.

Resolving Remote Server Name to Target Database (CICS)

1. If the CICS/VSE transaction issues an SQL CONNECT statement with the "TO *server name*" clause, the server name is established explicitly for the transaction and the Online Resource Adapter will use the DBNAME Directory to resolve the server name to the target database.

   If the specified application server is remote and the Communication Protocol specified by the connected user is not TCP/IP, the AR will issue a GDS ALLOCATE command to acquire a session to the remote system where the server runs. The SYSID used in this ALLOCATE command will be the SYSID value from the matching DBNAME Directory entry (and must match a CEDA DEF CONNECTION definition). The AR will then issue a GDS CONNECT PROCESS command to initiate an APPC basic conversation with the remote server. The PROCNAME used in this CONNECT PROCESS command will be the REMTPN value from the matching DBNAME Directory entry.

   If the specified application server is remote and the Communication Protocol specified by the user is TCP/IP, the AR will acquire a TCP/IP socket. Then the AR will use this socket to originate a connection request to initiate a TCP/IP communication with the remote server. In this case, the TCPPORT and the TCPHOST or the IPADDR from the matching DBNAME Directory entry are required for issuing the connect request.
2. If the CICS/VSE transaction did not issue an SQL CONNECT statement with the "TO *server name*" clause, the Online Resource Adapter will attempt to connect to the *default* application server.

If the *default* application server is remote and the Communication Protocol specifed by the connected user is not TCP/IP, the AR will issue a GDS ALLOCATE command to acquire a session to the remote system where the *default* application server runs. The SYSID used in this ALLOCATE command will be the SYSID value of the *default* server (and must match a CEDA DEF CONNECTION definition). The AR will then issue a GDS CONNECT PROCESS command to initiate an APPC basic conversation with the remote server. The PROCNAME used in this CONNECT PROCESS command will be the REMTPN value of the *default* server.

If the *default* application server is a remote server and the Communication Protocol specified by the user is TCP/IP, the AR will acquire a TCP/IP socket. Then the AR will use this socket to originate a connection request to initiate a TCP/IP communication with the remote server. In this case, the TCPPORT and the TCPHOST or the IPADDR from the *default* server's DBNAME Directory entry are required for issuing the connect request.

The *default* application server is determined when the CIRB transaction was invoked and can be changed subsequently by a CIRC transaction. For more information on establishing a *default* application server, see Establishing a Default Application Server.

Resolving Remote Server Name to Target Database (VSE Batch)

If the Batch application issues an SQL CONNECT statement with the "TO server name" clause, the server name is established explicitly for the transaction and the Batch Resource Adapter uses the DBNAME Directory to resolve the server name to the target database.

If the specified application server is a local or host VM (Guest Sharing) server, communications is done using XPCC as it is currently done. If the application server is remote and TCP/IP information is present in the matching DBNAME Directory entry, communications is done using TCP/IP. If TCP/IP information is not present, an error is returned in the SQLCA: SQLCODE -841, SQLSTATE 57040, with a reason code in SQLERRD2.

If the Batch Application issues an SQL CONNECT statement without the "TO server name" clause, the actions taken by the Batch Resource Adapter depend on the previous connection state. If the previous state was established with a COMMIT or ROLLBACK, then the Batch Resource Adapter connects back to the previous Server name. If the previous state was established with a COMMIT or ROLLBACK with the RELEASE option, then the Batch Resource Adapter attempts to connect to the *default* application server.

If the *default* application server is a local or host VM (Guest Sharing) server, communications is done using XPCC as it is currently done. If the application server is remote and TCP/IP information is present in the matching DBNAME Directory entry, communications is done using TCP/IP. If TCP/IP information is not present, an error is returned in the SQLCA: SQLCODE -841, SQLSTATE 57040, with a reason code in SQLERRD2.

The *default* application server is determined from the DBNAME Directory as is currently done. For more information on establishing a *default* application server, see "Establishing a Default Application Server" on page 99. Note that Batch applications cannot access a Remote server via SNA, only via TCP/IP.

Restricting Access Using Views

Views control who has access to what data. They can be set up to allow access to a subset of the columns or the rows of a table.

Example

To show how a view can be used to restrict access to information, consider the information presented in Table 18.

Table 18. Employee Information (EMP_INFO) Table

NAME	DEPT	SALARY	PHONENO
SMITH	100	25750	3978
BANES	200	15051	3476
ADAMSON	105	33075	4738
PARKER	200	26250	6789
KWAN	100	22260	7831
WALKER	105	23840	5498

Many different people may require access to information in this table for different reasons.

Examples

1. The personnel department needs to be able to update and look at the entire table.

   This requirement is met by granting users in the personnel department SELECT and UPDATE privileges on this table, as follows:

   ```
   GRANT SELECT,UPDATE ON EMP_INFO TO PERSONNL
   ```

2. Individual department managers need to look at the salary information for their employees.

   This requirement is met by creating a view for each manager. For example, the following view (called EMP100) can be created for JANE, the manager of department 100:

   ```
   CREATE VIEW EMP100
     AS SELECT NAME,SALARY,PHONENO
     FROM EMP_INFO
     WHERE DEPT=100

   GRANT SELECT ON EMP100 TO JANE
   ```

   JANE (and any others who have SELECT privilege on this view) would query it as they would an ordinary table. It would appear as the following:

   Table 19. EMP100 View

NAME	SALARY	PHONENO
SMITH	25750	3978
KWAN	22260	7831

3. All users require access to telephone number information.

   This requirement is met by creating a view (called PHONE) on the NAME and PHONENO columns:

```
CREATE VIEW PHONE
  AS SELECT NAME,PHONENO
  FROM EMP_INFO

GRANT SELECT ON PHONE TO PUBLIC
```

The keyword PUBLIC grants the privileges on the PHONE view to all users. Users who access it will see the following table:

Table 20. PHONE View

NAME	PHONENO
SMITH	3978
BANES	3476
ADAMSON	4738
PARKER	6789
KWAN	7831
WALKER	5498

Changing User Passwords

All users' passwords are recorded in the SYSTEM.SYSUSERAUTH catalog table. As a DBA, you can change any user's password at any time. To do this, use a GRANT CONNECT statement.

Example

```
GRANT CONNECT TO JOHN IDENTIFIED BY xyzabc
```

Users can also change their own passwords at any time, by issuing a GRANT CONNECT statement to themselves. To change a user's password verified by the CICS subsystem, or some other subsystem, follow the procedure for that subsystem.

You should change all passwords on a periodic basis; for example, every four months.

Securing the Database Catalog Tables

During database generation, the SELECT privilege is granted to PUBLIC on the catalog tables. In most cases this presents no security problem, but for very sensitive data it may be undesirable. These tables describe every object in the database, thus, while users would not know what specific items of data are stored, they would be able to tell what **kind** of data existed. Conceivably, a malicious individual could make destructive use of this knowledge.

Before revoking general access to the tables, however, you must weigh the advantages of securing the information in them against the disadvantages of users being unable to retrieve the information they require. The catalog tables are an active dictionary facility, and help to maintain definitions, control information, and general information on data. For example, users can query them to find out what tables they have created, the names and data types of the columns in each of those tables, and any synonyms they have defined.

You might consider revoking PUBLIC access to only the SYSCOLSTATS table, which records the first- and second-most frequent values in the first column used by every index on every table in the database.

If you do decide to secure all the catalog tables, the easiest way to do this is to revoke the SELECT privilege from PUBLIC on them. You must be connected as user ID SQLDBA and have DBA authority. You can then grant authority on specific tables to specific users.

Example 1

To revoke the SELECT privilege from PUBLIC on SYSTEM.SYSCATALOG, enter:

```
REVOKE SELECT ON SYSTEM.SYSCATALOG FROM PUBLIC
```

Before you revoke SELECT privileges from PUBLIC, you should also consider what impact there might be on existing applications. In particular, some applications may need to read a catalog table, so will fail if this authority is revoked. Naturally, in these cases you must grant the SELECT privilege to the creator of the program.

Note also that if the creator (the person who preprocessed the program) is not its sole runner, you must also specify the WITH GRANT OPTION clause for this person, in order to enable him or her to grant authority to other users to run the program.

Example 2

User JULIE has created a program that accesses SYSTEM.SYSCATALOG, and she grants RUN authority to KATHY and BILL. If you revoke the SELECT privilege from PUBLIC, you can preserve KATHY's and BILL's authority to run JULIE's program by issuing:

```
GRANT SELECT ON SYSTEM.SYSCATALOG TO JULIE WITH GRANT OPTION
```

If you revoke the SELECT privilege from PUBLIC on a catalog table, and later wish to completely restore it, you should also specify the WITH GRANT OPTION clause.

Example 3

To restore authority to PUBLIC on SYSTEM.SYSACCESS, enter:

```
GRANT SELECT ON SYSTEM.SYSACCESS TO PUBLIC WITH GRANT OPTION
```

Refer to the *DB2 Server for VSE & VM SQL Reference* manual for a description of the catalog tables.

Security Auditing

There are two ways to audit security: by querying the catalog tables, or by having the database manager do a security audit trace.

If you simply want to know what security structures exist, the first method is sufficient. The catalog tables maintain a record of authorization privileges: who has what authority and from whom they received it. But they do not record information about the **use** of these privileges: for example, the number of unsuccessful attempts to access a resource, the number of accesses based strictly on DBA authority, or similar authorization use information. For this type of information, you must use a security audit trace.

Both ways of auditing security are discussed below.

Auditing Security Using the Catalog Tables

The following are examples of queries you might enter against the catalog tables in security auditing:

1. What users are permitted to connect directly to the DB2 Server for VSE & VM application server? (DBA authority is required for this query.)

   ```
   SELECT NAME FROM SYSTEM.SYSUSERAUTH
     WHERE AUTHOR=' '
   ```

   The WHERE clause serves to eliminate any entries in SYSTEM.SYSUSERAUTH for program dependencies from the query result.

2. How many users have been granted RUN authority on WALTERS.PAYROLL by user BENNETT? (User WALTERS is the creator of the program; the creator is determined by the USERID parameter when the program is preprocessed.)

   ```
   SELECT COUNT(*) FROM SYSTEM.SYSPROGAUTH
     WHERE CREATOR = 'WALTERS'
       AND PROGNAME = 'PAYROLL'
       AND GRANTOR = 'BENNETT'
   ```

   This query only counts user BENNETT's first-level grantees (those who received their authority directly from user BENNETT).

3. Who are all the users who have received RUN authority on PAYROLL from someone other than WALTERS?

   ```
   SELECT COUNT(*) FROM SYSTEM.SYSPROGAUTH
     WHERE CREATOR = 'WALTERS'
       AND PROGNAME = 'PAYROLL'
       AND GRANTOR <> 'WALTERS'
   ```

4. How many users have RESOURCE authority but not DBA authority?

   ```
   SELECT COUNT(*) FROM SQLDBA.SYSUSERLIST
     WHERE RESOURCEAUTH = 'Y'
       AND DBAAUTH <> 'Y'
       AND AUTHOR = ' '
   ```

5. How many secondary authorizations (those that originated from other than the creator) exist for the JOHNSON.EMPLOYEE table created by user JOHNSON?

   ```
   SELECT COUNT(*) FROM SYSTEM.SYSTABAUTH
     WHERE TCREATOR = 'JOHNSON'
       AND TTNAME = 'EMPLOYEE'
       AND GRANTOR <> 'JONES'
       AND GRANTEETYPE = ' '
   ```

   Here, the GRANTEETYPE = ' ' portion of the WHERE clause eliminates entries for programs.

6. Which users have been granted SELECT authority on the PERSONNL.EMPLOYEE table by user LAPIS?

   ```
   SELECT * FROM SYSTEM.SYSTABAUTH
     WHERE TCREATOR = 'PERSONNL'
       AND TTNAME = 'EMPLOYEE'
       AND SELECTAUTH = 'Y'
       AND GRANTEETYPE = ' '
       AND GRANTOR = 'LAPIS'
     ORDER BY TIMESTAMP
   ```

Auditing Security Using Tracing

Security audit tracing is one of the functions that can be performed using the trace facility. A security audit trace is unique in that it is not necessarily done for

problem determination. Start a trace of the security audit function of the RDS component by using the TRACRDS initialization parameter. Alternatively, you can start it by issuing the TRACE command from the operator's console after the application server has been started.

For descriptions of the TRACRDS parameter, the TRACE operator command, the trace output records, and the utility that formats these records into readable output, see the *DB2 Server for VSE & VM Operation* manual.

In VM, you can direct the trace output to tape, to a CMS file, or to a memory area known as a trace buffer. However, if your installation uses the security audit trace frequently, you may want to direct the output to a CMS file. To do this, you must enter a CMS FILEDEF command before starting the application server, and supply particular responses to the prompts that come up when tracing is started. For descriptions of the FILEDEF command and the appropriate message responses, see the *DB2 Server for VSE & VM Operation* manual.

As with other traces, you can get two levels of information. Level 1 traces and records the following information:
- All unsuccessful attempts to obtain access to a resource
- Access that is based strictly on DBA authority
- All CONNECTs to the application server
- All grants of special privileges (DBA, CONNECT, SCHEDULE, or RESOURCE authorities)
- All grants of RUN authority.

Level 2 keeps track of all DB2 Server for VSE & VM authorization checks.

Table 21 shows each type of authorization verification that the database manager does, and which results are traced.

Table 21. Information Recorded by a Security Audit Trace

Type of Authorization Check	**Result Traced at Level 2**	**Result Traced at Level 1**
CONNECT	Y,I,N	Y,I,N
RUN	G,Y,D,N,P	D,N
SELECT, INSERT, UPDATE, DELETE, ALTER, and INDEX	G,Y,D,N,P	D,N
RESOURCE	Y,N	N
REFERENCES	Y,N	D,N
DBA	D,N	D,N
Grants of Special Privileges (DBA, CONNECT, RESOURCE, and SCHEDULE)	D,N	G,N
Grants of RUN Authority	G,N	

Y Yes, the user is authorized.
N No, the user is not authorized.
G Yes, the user is authorized to use and grant this privilege.
P The resource is PUBLIC, and thus all users are authorized.
D The user is authorized based only on DBA authority (that is, does not have specific privileges).
I CONNECT on special link without password verification (scheduled).

For each result of an authorization check that is traced, the database manager creates a trace record in the same format as other kinds of trace records. These records are identical in format for all levels and types of authorization, and are written to the same (VSE) trace output file, or (VM) trace tape (or CMS file).

If a value does not apply for a specific occurrence, the database manager sets it to blanks. For example, a trace record for CONNECT does not contain the name of a resource (that is, a table name).

Each trace record contains (where applicable):

- Date and time of verification.
- The user ID for which the verification is being done.
- Resource 1 (for example, the name of a table to be accessed or the name of a program to be run).
- Resource 2 (for example, the name of a particular column to be updated).
- The creator of the resource.
- The type of authorization requested (as listed in Table 21 on page 110).
- The result of the authorization check (Y, N, G, P, D, I).
- The external logical unit of work identifier (EXTLUWID) of the connection, which uniquely identifies an LU6.2 conversation. Its value is *netid.luname.instance_number.sequence_number*, where *netid* and *luname* are up to 8 characters long, *instance_number* is 12 characters long, and *sequence_number* is 4 characters long. The EXTLUWID is only used for conversations that use the DRDA protocol.

The Resource 2 field shows the column (where applicable) on checks of UPDATE authority. It can also contain a description of the reason that the database manager is checking a certain authority. For example, it might contain "ALTER PUB DBSPACE" on a check for the DBA authority needed to alter a PUBLIC dbspace. In this case, DBA would be the type of authorization being checked, while the Resource 2 field provides more information about why this authority is required.

When analyzing trace records, remember that many operations on views are restricted. These restrictions are reflected in the trace records generated during CREATE VIEW processing. When the database manager creates a view, it checks the user's authority on the base tables to determine what authority to give that user on the view. It also checks the view itself to see what operations cannot be performed on it. For example, because deletions are not allowed in views that involve a join, the authorization check for DELETE would return an N. The N shows that deletions are not allowed against the view; it does not necessarily imply that the creator is not authorized to delete from the base table.

Authorization checks during CREATE VIEW processing are traced, but only at level 2. The result field of the trace record indicates whether an authorization check is a result of CREATE VIEW processing. The CREATE VIEW indicator is the letter V following the usual result indicator. For example, a successful verification of SELECT authority on a base table produces a result value of YV — yes during view creation. You can use this indicator to distinguish between normal authorization checks and those done during view creation.

Note: Tracing occurs during preprocessing and execution of programs, and during the dynamic execution of statements in ISQL or DBS Utility.

Authorization traces for data manipulation operations in programs occur during preprocessing, not during execution.

Loading Security Audit Information into Tables

You can use the DBS Utility to load security audit trace records into a table. When the trace information is in a table, you can use SQL statements to answer questions such as:

- Who was denied access to a resource?
- Who used DBA authority to access a resource?
- When was RUN authority on a particular program granted to additional users?

Figure 21 on page 113 shows a DB2 Server for VSE example DBS Utility job to create a security table and load trace records into it. In the example, the trace output file is on tape.

```
// JOB DATALOAD SECURITY AUDIT TRACE
// EXEC=PROC=DBNAME01
// EXEC=PROC=ARIS71PL
// TLBL ARITRAC
// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,LOGMODE=Y,PROGNAME=ARIDBS'

COMMENT '                                    '
COMMENT ' ********************************   '
COMMENT ' * DATALOAD SECURITY AUDIT TRACE *   '
COMMENT ' ********************************   '
COMMENT '                                    '
COMMENT '  ACQUIRE A DBSPACE(PRIVATE)        '
COMMENT '  NAMED SECURITY                    '
COMMENT '                                    '
ACQUIRE PRIVATE DBSPACE NAMED SECURITY;

COMMENT '                                    '
COMMENT ' CREATE A TABLE IN THE PRIVATE DBSPACE '
COMMENT '                                    '

CREATE TABLE AUDIT_TAB(TRPOINT   SMALLINT,
                       YEAR      CHAR(2),
                       MONTH     CHAR(2),
                       DAY       CHAR(2),
                       TIME      CHAR(8),
                       USERID    CHAR(8),
                       GRANTEE   CHAR(8),
                       RESOURCE1 CHAR(18),
                       RESOURCE2 CHAR(18),
                       OWNER     CHAR(8),
                       AUTHTYPE  CHAR(8),
                       RESULT    CHAR(2),
                       EXTLUWID  CHAR(35))
                  IN SECURITY;

COMMENT '                                    '
COMMENT ' LOAD DATA - (NOTE _ YOU MAY        '
COMMENT '   WISH TO INTERCHANGE DAY/MONTH)   '
COMMENT '                                    '

DATALOAD TABLE (AUDIT_TAB) IF POS (11-14)=-220659706
           TRPOINT             7-8            FIXED
           EXTLUWID           41-75           CHAR
           YEAR              124-125          CHAR
           MONTH             118-119          CHAR
           DAY               121-122          CHAR
           TIME              143-150          CHAR
           USERID            168-175          CHAR
           GRANTEE           193-200          CHAR
           RESOURCE1         218-235          CHAR
           RESOURCE2         253-270          CHAR
           OWNER             288-295          CHAR
           AUTHTYPE          313-320          CHAR
           RESULT            338-339          CHAR
  INFILE(ARITRAC PDEV(TAPE) BLKSZ(4096) RECFM(VB) RECSZ(384))
```

Figure 21. Loading Security Audit Records into a Table - DB2 Server for VSE

Figure 22 on page 114 shows a DB2 Server for VM example of running the DBS Utility. The utility reads a CMS file (SECTAB DATA A), which contains statements to create a security audit table and load trace records into it. Before invoking the utility, ensure that the appropriate trace tape is mounted on virtual device 182.

Command to Invoke the DBS Utility:

```
FILEDEF TRACE1 TAP2 SL (RECFM VB BLOCK 4096 LRECL 384
EXEC SQLDBSU ID(SQLDBA) IN(SECTAB DATA A) PR(TERMINAL)
```

SECTAB DATA A Contains:

```
  CONNECT user IDENTIFIED BY password;
  COMMENT '                                      '
  COMMENT ' ********************************     '
  COMMENT ' * DATALOAD SECURITY AUDIT TRACE *     '
  COMMENT ' ********************************     '
  COMMENT '                                      '
  COMMENT '  ACQUIRE A DBSPACE(PRIVATE)          '
  COMMENT '  NAMED SECURITY                      '
  COMMENT '                                      '
  ACQUIRE PRIVATE DBSPACE NAMED SECURITY;
  COMMENT '                                      '
  COMMENT ' CREATE A TABLE IN THE PRIVATE DBSPACE '
  COMMENT '                                      '
  CREATE TABLE AUDIT_TAB(TRPOINT   SMALLINT,
                         YEAR      CHAR(2),
                         MONTH     CHAR(2),
                         DAY       CHAR(2),
                         TIME      CHAR(8),
                         USERID    CHAR(8),
                         GRANTEE   CHAR(8),
                         RESOURCE1 CHAR(18),
                         RESOURCE2 CHAR(18),
                         OWNER     CHAR(8),
                         AUTHTYPE  CHAR(8),
                         RESULT    CHAR(2),
                         EXTLUWID  CHAR(35))
                   IN SECURITY;
  COMMENT '                                      '
  COMMENT ' LOAD DATA - (NOTE _ YOU MAY          '
  COMMENT '   WISH TO INTERCHANGE DAY/MONTH)     '
  COMMENT '                                      '
  DATALOAD TABLE (AUDIT_TAB) IF POS (11-14) = -220659706
           TRPOINT             7-8             FIXED
           EXTLUWID           41-75            CHAR
           YEAR              124-125           CHAR
           MONTH             118-119           CHAR
           DAY               121-122           CHAR
           TIME              143-150           CHAR
           USERID            168-175           CHAR
           GRANTEE           193-200           CHAR
           RESOURCE1         218-235           CHAR
           RESOURCE2         253-270           CHAR
           OWNER             288-295           CHAR
           AUTHTYPE          313-320           CHAR
           RESULT            338-339           CHAR
   INFILE(TRACE1)
```

Figure 22. Loading Security Audit Records into a Table - DB2 Server for VM

Note: The external logical unit of work identifier (EXTLUWID) is only used for conversations that use the DRDA protocol.

If you have other trace functions active while you are tracing a security audit, include an input-record-id clause (IF POS (11-14) = -220659706) on the DATALOAD command to identify that only security audit trace records are to be loaded. This is necessary because the trace records from other functions are interspersed with those of the security audit trace.

When doing a security audit trace, it is usually to your advantage to trace the parser component at the same time. When you trace this component at level 1, the resultant trace records describe the SQL statement entered into the database manager. By using the timestamp in the trace records, you can correlate the input to the security audit trace records produced.

If you plan to load the security audit trace records into a table, you may want to print the parser trace records by using the trace formatter. If you are printing the security audit records, you may want to also print the parser records by specifying both the parser and security audit components for the trace formatter. An example producing such a listing is shown in Table 22 on page 117 and Table 23 on page 117.

In VM, if you directed the trace output to a CMS file (by issuing a CMS FILEDEF command), you can still use the DBS Utility to load the trace data into tables. To do this, enter the following CMS FILEDEF command before invoking the SQLDBSU EXEC:

```
FILEDEF ddname DISK filename filetype filemode (RECFM VB LRECL 384 BLOCK 4096
```

Notes:

1. The ddname on the FILEDEF command must match that used in the DBS Utility INFILE parameter of the DATALOAD command.
2. You must enter the RECFM, LRECL, and BLOCK values shown.

In VSE, if, when starting the application server, you directed the trace output to disk, you must change the INFILE statement to:

```
INFILE(ARITRAC PDEV(DASD) BLKSZ(4088) RECFM(VB) RECSZ(384))
```

In addition, you must change the job control to identify the DASD SAM trace output file. For example:

- For a DASD file that is **not** managed by the VSE/VSAM Space Management for SAM Feature, you might specify:

  ```
  // DLBL ARITRAC,'TRACE.FILE1'
  // EXTENT ,VSER01,1,0,301,120
  // ASSGN SYS006,195
  ```

- For one that is, you might specify:

  ```
  // DLBL ARITRAC,'TRACE.FILE1',0,VSAM,DISP=(,DELETE)
  ```

When DISP=(,DELETE), the VSAM file is deleted after it is read. If you do not want the file to be deleted, specify DISP=(,KEEP) or omit the DISP parameter.

The above examples would replace the TLBL statement in Figure 21 on page 113.

Once you have loaded the security audit trace records into a table, you can enter SQL statements against them. This method may make viewing the records easier, but has a disadvantage in that any user who has DBA authority can change the table, and any tampering may make the data incorrect. You should always print the trace records and protect the trace tape to ensure that there is always a valid copy.

Figure 23 on page 116 shows examples of typical security audit queries. Some of the records traced appear only at level 2; level 2 can generate a significant number of trace records. These queries are shown as they might appear as input to the DBS Utility.

```
COMMENT '****************************************'
COMMENT ' SELECT ALL RECORDS FROM AUDIT TABLE      '
COMMENT '        WHERE AUTHORIZATION WAS DENIED    '
COMMENT '****************************************'
SELECT * FROM AUDIT_TAB WHERE RESULT='N';
COMMENT '****************************************'
COMMENT ' SELECT RECORDS FROM AUDIT TABLE          '
COMMENT ' RECORDED BETWEEN 8 A.M. AND 12:30 P.M.   '
COMMENT ' ON JUNE 29                               '
COMMENT '****************************************'
SELECT * FROM AUDIT_TAB WHERE MONTH = '06'
   AND DAY = '29' AND TIME BETWEEN '08:00:00' AND '12:30:00';
COMMENT '****************************************'
COMMENT ' SELECT RECORDS FROM AUDIT TABLE          '
COMMENT ' RECORDED BETWEEN 12:30 P.M. AND 4:00 P.M.'
COMMENT ' ON JUNE 29 AND AUTHORIZED DUE TO DBAAUTH.'
COMMENT '****************************************'
SELECT * FROM AUDIT_TAB WHERE MONTH = '06'
   AND DAY = '29' AND TIME BETWEEN '12:30:00'
   AND '16:00:00' AND         RESULT = 'D';
COMMENT '****************************************'
COMMENT ' SELECT CHECKS OF UPDATE AUTHORITY        '
COMMENT ' AGAINST TABLE USER1.TAB1                 '
COMMENT ' RECORDED BETWEEN 08:00 P.M. AND 4:00 P.M.'
COMMENT ' ON JUNE 29 NOT DUE TO VIEW CREATION.     '
COMMENT '  (update checks traced at level 2)       '
COMMENT '****************************************'
SELECT * FROM AUDIT_TAB WHERE MONTH = '06'
   AND DAY = '29' AND TIME BETWEEN '08:00:00'
   AND '16:00:00' AND OWNER = 'USER1'
   AND RESOURCE1 = 'TAB1' AND AUTHTYPE = 'UPDATE'
   AND RESULT NOT LIKE '%V';
COMMENT '****************************************'
COMMENT ' SELECT CHECKS OF UPDATE AUTHORITY        '
COMMENT ' AGAINST TABLE USER1.TAB1                 '
COMMENT ' RECORDED BETWEEN 08:00 P.M. AND 4:00 P.M.'
COMMENT ' ON JUNE 29 DUE TO DBAAUTH                '
COMMENT '  (DBA activity traced at level 1 or 2)   '
COMMENT '****************************************'
SELECT * FROM AUDIT_TAB WHERE MONTH = '06' AND
   DAY = '29' AND TIME BETWEEN '08:00:00'
   AND '16:00:00' AND OWNER = 'USER1'
   AND RESOURCE1 = 'TAB1' AND AUTHTYPE = 'UPDATE'
   AND RESULT = 'D';
COMMENT '****************************************'
COMMENT ' SELECT ALL GRANTS OF RUN AUTH ON         '
COMMENT ' PROGRAMS USER1.DBD1 AND USER1.DBD3.      '
COMMENT '****************************************'
SELECT * FROM AUDIT_TAB WHERE AUTHTYPE = 'RUN' AND
   OWNER = 'USER1' AND RESOURCE1 = 'DBD1' OR
           RESOURCE1 = 'DBD3';
```

Figure 23. Example Security Audit Queries

Printing Security Audit Information from the Trace File

A security audit trace, especially a level 2 one, can generate a large amount of information, and even more information is generated if you are tracing other components or functions at the same time. All of these records are placed in a single trace file. To print them selectively, you need to use the **trace formatting utility**.

This utility accepts control statements, which in VM, it reads from a CMS file. As it does not access the database manager, the latter does not have to be running for the trace formatter to work.

Table 22 shows an example of invoking the DB2 Server for VSE trace formatter. The control statements print out all security audit trace records and all parser trace records. The example also restricts the output by date and time and is only for USER1.

Table 22. Printing Security Audit Records from the Trace File

```
// JOB RUN TRACE FORMATTER
// TLBL ARITRAC,file-id    <-- File-id of trace tape (optional  )
// ASSGN SYS004,cuu        <-- Address of tape unit
// EXEC ARIMTRA,SIZE=AUTO
SUBCOMP AU PA
USERID USER1
DATE 06/29/85
TIME 12:00:00 23:00:00
/*
/&
```

Notes:

1. The tape should be mounted on the physical device specified by **cuu** before running the job.
2. The tape file-id must be the same file-id as was specified on the TLBL statement when the tape was created.
3. The *DB2 Server for VSE & VM Operation* manual contains examples of running the trace formatter to process a trace file that resides on DASD.

Table 23 shows an example of invoking the DB2 Server for VM trace formatter. The interactive SQLTRFMT EXEC supplied by IBM resides on the production minidisk (Q-disk) and invokes XEDIT to edit a CMS file called SQLTRFMT TRACE A. Use this exec to type in the control statements. When you file SQLTRFMT SQLTRACE A, the SQLTRFMT EXEC then asks where you want its output directed.

The control statements shown in Table 23 print all security audit (AU) trace records and all parser (PA) trace records. They also restrict the output to those records generated for a specific date (06/29/85), time (12:00:00 to 23:00:00), and user (USER1).

Table 23. Printing Security Audit Records from the Trace File

Invoking the Trace Formatter:

```
SQLTRFMT
```

Example Control Statements

```
SUBCOMP AU PA
USERID USER1
DATE 06/29/85
TIME 12:00:00 23:00:00
```

If you are directing your trace output to tape, then before invoking the trace formatter, ensure that the appropriate tape is mounted on virtual device 182. If you are directing it to a CMS file, you must enter a CMS FILEDEF command for the file before invoking SQLTRFMT. Use the same FILEDEF that you issued before you

invoked SQLSTART (and initiated the trace). See the *DB2 Server for VSE & VM Operation* manual for the command format.

Complete instructions for using the utility are in the *DB2 Server for VSE & VM Operation* manual.

Chapter 6. Recovering from Failures

A variety of problems can occur in a relational database management system, leading to inaccuracies or loss of data. A power failure can bring the computer to a halt; the disk used to store information could become damaged; users can make errors such as dropping the wrong table or dbspace. Database recovery refers to the processing needed to correct the data when something goes wrong.

The problems that can occur fall into the following categories. This chapter explains how to recover from those that fall into the first two categories. For information on how to recover from the other types, see the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual.

- Application Failure

  A single application (for example, an ISQL command or routine, or a DBS Utility command) fails to complete successfully.
- User Logic Errors

  The system or application does the requested function, but the request itself was in error: that is, the user (or application program) did not specify the correct function.
- System Failure

  The operating system, CICS subsystem, or the database manager can end abnormally because of error conditions or a power failure.
- DASD Failure

  The system may be unable to read data from or write it to the DASD device on which it is stored because the storage medium is unreadable or damaged. Such an error could occur on the log or the storage pool.
- Subsystem Failures (VM Only)

  With VSE Guest Sharing, whereby users on VSE are accessing a DB2 Server for VM application server, any of the subsystems involved (the database manager, VM/ESA operating system, VSE, or the CICS subsystem) may end abnormally.

Overview of Recovery Concepts

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 processed within an LUW. If no LUW exists when a statement is issued, then the database manager creates one.

CMS Work Units

VM users 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.

Atomic Operations

An operation is atomic if within a logical unit of work (LUW), it can succeed or fail on its own; that is, it does not affect other operations as long as they do not depend on it. The DB2 Server for VSE & VM database manager considers all operations are atomic except those that occur in dbspaces residing in nonrecoverable storage pools, and those that occur when LOGMODE=N (running with the no-log option).

Example:

Suppose you have an application program that performs the following operations within one LUW: a DELETE, an UPDATE, and an INSERT statement. Assume the DELETE statement will process successfully; then, the UPDATE statement will change the values in the table as specified. If, at the end of statement processing, any duplicates exist in the primary key, the UPDATE operation is rolled back. Because the failure of the UPDATE statement does not affect the DELETE statement (both operations are atomic), you can let the program proceed and perform the INSERT. Alternatively, you could COMMIT the successful DELETE or ROLLBACK the LUW.

For a further discussion of atomic operations, see “Backouts Initiated by Application Programs” on page 129.

Dynamic Application Backout

This process reverses the changes made by a logical unit of work (LUW) that ends abnormally. It is performed while the system is online and processing other applications. It is supported for the following:

- DB2 Server for VM
- DB2 Server for VSE
- ISQL
- DBS Utility
- preprocessor operations
- Batch
- VSE/ICCF, and
- the CICS subsystem

The DB2 Server for VSE dynamic application backout facilities are also coordinated with the Dynamic Transaction Backout facilities of the CICS subsystem. A backout initiated by the DB2 Server for VSE database manager initiates a CICS transaction

backout for the affected transaction. Similarly, a transaction backout initiated by the CICS subsystem initiates a DB2 Server for VSE backout, if the transaction was doing any SQL processing.

Restart Processing

If the system or the database manager ends abnormally, this process reverses any database changes made by applications that were in progress within an LUW at the time of the failure. It also ensures commitment of all changes made by those applications that completed successfully.

Recovery from Application Failures

To take advantage of the DB2 Server for VSE & VM recovery support, applications should be designed so that all SQL requests that constitute one logical change to the database are properly grouped into logical units of work (LUWs). For example, if an application transfers funds from one account to another, which entails an update to two different rows in the database, the updates should be done in the same LUW. Thus, if the application should fail, the database would be left in one of two consistent states: either the transfer was done completely (both rows updated), or it was not done at all (neither row updated). If the updates were in different LUWs, an application failure could result in only half of the transfer being performed (only one row updated).

Designing an application properly requires an understanding of when an LUW begins and ends. When it ends, the changes made within the LUW are either committed to the database, or backed out. Figure 24 on page 122 shows the general rules for DB2 Server for VSE LUWs. Table 24 shows the general rules for VM users and Table 25 on page 123 shows the general rules for VSE guest users accessing an application server on a VM/ESA system. There are, however, variations and special considerations that depend on the application environment and application implementation techniques. These variations are discussed in the following sections.

	WHEN A LOGICAL UNIT OF WORK BEGINS	WHEN A LOGICAL UNIT OF WORK ENDS	
		COMMITTED	BACKED OUT
Programs			
-Batch/ICCF	First SQL statement	-COMMIT WORK -Normal end	-ROLLBACK WORK -Abnormal end -Implicit rollback -Statement error
-CICS	First SQL statement	-COMMIT WORK -SYNCPOINT -Normal end	-ROLLBACK WORK -SYNCPOINT ROLLBACK -Abnormal end -Implicit rollback
ISQL Sessions			
-AUTOCOMMIT ON	Command entry	-After successful command process-ing -COMMIT WORK for multiple row update	-CANCEL -Command Error -ROLLBACK WORK -Implicit rollback
-AUTOCOMMIT OFF	First Command	-COMMIT WORK	-ROLLBACK WORK -Abnormal end -CANCEL -Command Error -Implicit rollback
DBS Jobs			
-AUTOCOMMIT ON	Command entry	-After successful command process-ing	-Command error -Abnormal end
-AUTOCOMMIT OFF	First Command	-COMMIT WORK -Normal end of program	-Command error -ROLLBACK WORK -Abnormal end of program
Preprocessor Jobs	Start of job setp	-Normal end of job step	-Abnormal end of job step

Figure 24. General Rules for DB2 Server for VSE Logical Units of Work

Notes to Figure 24:

- Note that DBS ERRORMODE processing may change the DBS AUTOCOMMIT mode. Refer to the *DB2 Server for VSE & VM Database Services Utility* manual for details.
- When AUTOCOMMIT is on, ISQL issues a COMMIT WORK when the statement completes successfully. The exception is for UPDATE, DELETE, and INSERT statements that affect more than one row. For that case, you are prompted before ISQL issues a COMMIT WORK.

Table 24. General Rules for DB2 Server for VM Logical Units of Work

LOGICAL UNIT OF WORK	**All Programs Under CMS**	**ISQL Sessions**		**DBS Utility**		**Pre-processors**
		AUTOCOMMIT ON	**AUTOCOMMIT OFF**	**AUTOCOMMIT ON**	**AUTOCOMMIT OFF**	
BEGINS	First SQL statement	Each SQL statement entry	First SQL statement	Command entry	First command	Start of CMS command
ENDS COMMITTED	COMMIT WORK Normal end of CMS command	After successful SQL statement processing, COMMIT WORK for multi-row updates	COMMIT WORK	After successful command processing	COMMIT WORK Normal end of CMS command	Normal end of CMS command Implicit rollback SQLHX

Table 24. General Rules for DB2 Server for VM Logical Units of Work (continued)

LOGICAL UNIT OF WORK	All Programs Under CMS	ISQL Sessions		DBS Utility		Pre-processors
		AUTOCOMMIT ON	AUTOCOMMIT OFF	AUTOCOMMIT ON	AUTOCOMMIT OFF	
ENDS BACKED OUT	ROLLBACK WORK Abnormal end of CMS command Implicit rollback SQLHX Statement error	CANCEL Statement error ROLLBACK WORK Implicit rollback	ROLLBACK WORK Abnormal end CANCEL Implicit rollback	Abnormal end Command error SQLHX Implicit rollback	Command error ROLLBACK WORK Abnormal end of CMS command Implicit rollback SQLHX	Abnormal end of CMS command

Notes to Table 24 on page 122:

1. If a DB2 application program (including preprocessors and utilities) is not invoked from an EXEC, it is considered to be a command, and the COMMIT and ROLLBACK rules apply. If the program is issued from an EXEC (as is almost always the case), then it is considered to be a subcommand. For EXECs, end-of-command COMMIT and ROLLBACK processing does not occur until the EXEC completes.
2. DBS ERRORMODE processing may change the DBS AUTOCOMMIT mode. Refer to the *DB2 Server for VSE & VM Database Services Utility* manual for details.
3. When AUTOCOMMIT is on, ISQL issues a COMMIT WORK when the statement completes successfully (as shown in Table 24 on page 122). The exception is for UPDATE, DELETE, and INSERT statements that affect more than one row. For that case, you are prompted before ISQL issues a COMMIT WORK.
4. For the normal end situation, the database manager will attempt to commit LUWs. The commit may fail if a deadlock occurs, a log full condition is encountered, or some other system condition occurs that causes the program to end.

Table 25. General Rules for DB2 Server for VM Logical Units of Work from VSE Guests

LOGICAL UNIT OF WORK	PROGRAMS		Preprocessor Jobs
	CICS	Batch/ICCF	
BEGINS	First SQL Statement	First SQL Statement	Start of job step
ENDS COMMITTED	COMMIT WORK Normal end	COMMIT WORK SYNCPOINT Normal end	Normal end of job step

Table 25. General Rules for DB2 Server for VM Logical Units of Work from VSE Guests (continued)

LOGICAL UNIT OF WORK	PROGRAMS		Preprocessor Jobs
	CICS	Batch/ICCF	
ENDS BACKED OUT	ROLLBACK WORK Abnormal end Implicit rollback Statement error	ROLLBACK WORK SYNCPOINT ROLLBACK Abnormal end Implicit rollback	Abnormal end of job step

Application Program Recovery in VM

An application is considered to have ended normally when it returns to CMS. In single user mode, an application ends normally when it returns to the DB2 Server for VM calling routine. All other types of termination (such as HX, CMS abend, program check, or any user machine termination) are considered abnormal.

Note: In single user mode, an application's Register 15 return code protocol is not part of the definition of termination, and is not used by the application server to determine whether it should proceed with normal or abnormal termination processing. The application server establishes a CMS ABNEXIT exit in the database machine. The exit attempts recovery and dumps important diagnostic information when the recovery attempt is not successful. If a single user mode application establishes an abnormal end exit (for example, by way of ABNEXIT, STAE, SPIE, STXIT), the DB2 Server for VM abend exit is overridden.

Some compilers provide a mechanism that handles program interrupts during the execution of a program and before control returns to CMS. Consequently, the application server may not be aware that the program termination is abnormal, and will perform an implicit COMMIT rather than an implicit ROLLBACK. See the *DB2 Server for VSE & VM Application Programming* manual for more information about program interrupts.

Users should be aware of how CMS handles multiple abnormal end exits, and should clear any that have been set by the application program before returning to the DB2 Server for VM application server, or else unpredictable results may occur when later CMS commands are issued. Also, the user should reset the abnormal exit before returning to the CMS abnormal termination routine after handling an abnormal end condition.

Dropping the DB2 Server for VM Resource Adapter Code

When users switch from one program to another, the SQLRMEND EXEC enables application programs to free the storage used by the resource adapter code. This EXEC can also be used to perform COMMIT/ROLLBACK processing on outstanding work before running the next program.

For more information, see "SQLRMEND EXEC" on page 251.

Batch and VSE/ICCF Application Recovery

If a batch application executing in multiple user mode ends without freeing its link to the DB2 Server for VSE & VM application server, the operating system informs the application server whether the application ended normally or abnormally. The

indication is normal if the application ends with the EOJ macro and the high-order bit of general purpose register 15 is set to 0. Other conditions indicate an abnormal end. The database manager automatically commits updates if the termination is normal, or does a rollback if it is abnormal.

The DB2 Server for VSE application server establishes an STXIT AB exit in the database partition. The exit attempts recovery and dumps important diagnostic information if the recovery attempt is not successful. If an application is running with the TRAP(ON) run-time option of LE/VSE and it did not issue an STXIT AB MACRO, LE/VSE and DB2 Server for VSE will keep track of calls to and returns from DB2 Server for VSE. If an abend occurrs while the application is running, the LE/VSE condition manager is informed whether the problem occurred in the application or in DB2 Server for VSE. If the abend occurs in DB2 Server for VSE, the LE/VSE condition handler passes the condition back to DB2 Server for VSE. For information on condition handling with LE/VSE see the *DB2 Server for VSE & VM Application Programming* manual. Furthermore, if a single user mode application issues an STXIT AB macro, the DB2 Server for VSE abend exit is overridden. Similarly, if the application issues an STXIT PC, then the DB2 Server for VSE abend exit is overridden for program check conditions. Other abend conditions are still processed by the application server.

Online Application Recovery

DB2 Server for VSE & VM recovery from failures of online (CICS) transaction is coordinated with CICS recovery processing.

Consistency among multiple application servers is ensured at CICS *synchronization points*, when related data across multiple application servers is kept in a consistent state. Synchronization points (*syncpoints*) are points, during the processing of a transaction, at which updates or modifications to the transaction's resources are logically complete and error-free. To take advantage of the CICS syncpoints, the database manager online support runs as a CICS resource adapter, using the CICS Application Program interface and User Exit interface. For more information, refer to the *CICS/VSE Customization Guide.*

Syncpoints occur during the execution of an application under any of the following circumstances:

- An application explicitly issues a request for a syncpoint: either the statement EXEC CICS SYNCPOINT to request a COMMIT of all updates, or EXEC CICS SYNCPOINT ROLLBACK to request a ROLLBACK. For further information, refer to the *CICS/VSE Application Programming Reference*.
- Any termination of a CICS transaction calls the CICS syncpoint manager. Normal termination results in COMMIT. Abnormal termination results in ROLLBACK.
- The SQL COMMIT WORK statement causes the DB2 Server for VSE online support to issue a CICS SYNCPOINT (COMMIT). The SQL ROLLBACK WORK statement causes a CICS SYNCPOINT (ROLLBACK).

  Additionally, when the DB2 Server for VSE online support detects an internal ROLLBACK of a unit of work, it issues CICS SYNCPOINT (ROLLBACK). (Such an internal rollback could happen, for example, if the system operator entered the FORCE command to ROLLBACK an LUW).

As a performance note, it is more efficient for applications to use a CICS syncpoint. The SQL COMMIT or ROLLBACK calls are less efficient, because they result in a

longer path. A CICS syncpoint is also easier to understand : when it is time to commit, the application program calls the global synchronization function (CICS SYNCPOINT [ROLLBACK]).

The assumptions are that individual application programs do not plan to do their own recovery, and that updates are not to be committed unless normal termination occurs or the application program explicitly requests a commit.

Notes:

An installation must explicitly request the CICS subsystem to start the syncpoint protocol by:

1. Generating CICS System Initialization Table (DFHSIT) with DBP=YES. If this is not done, the CICS process at synchronization points attempts to commit all updates. Alternatively, DBP=xx may be specified if a suffixed version of the CICS Dynamic Transaction Backout Program is being used.
2. Ensuring that each online application program that accesses an application server has Dynamic Transaction Backout set to YES. Do this by specifying INDOUBT=BACKOUT when defining the transaction.

ISQL Sessions

If an ISQL session ends abnormally, the database manager attempts to notify the user about the abnormal condition, and leaves the database in a consistent state. In VM, the database manager issues a ROLLBACK WORK and the session ends. Control returns to CMS. In VSE or in a VSE Guest Sharing environment, the CICS syncpoint manager issues a ROLLBACK WORK. All CICS temporary storage for routines is deleted, and both the ISQL transaction and the CISQ transaction are terminated, if possible. If the CICS syncpoint manager is in control when the CISQ transaction abnormal termination occurs, the ISQL transaction abends with the abend code GCBE. For more information on GCBE, see the *DB2 Server for VSE Messages and Codes* manual.

DBS Utility Processing

If the DBS Utility fails to complete the processing of all commands supplied in the command input, or if it terminates with a return code equal to or greater than 8, then before the Utility can be restarted the DBS message file listing must be analyzed to determine the commands that were processed and the error that occurred. If there are no error messages here that describe the reason for the failure, then the database machine console messages must be analyzed. After the error has been corrected, restart the Utility as described below:

- If the DBS command input that failed was processing without any of the following commands:
  SET AUTOCOMMIT ON
  SET ERRORMODE OFF
  SET ERRORMODE CONTINUE
  COMMITCOUNT parameter on an INFILE subcommand
  COMMITCOUNT parameter on an INMOD subcommand (VSE Only)
  SQL COMMIT WORK statements,

  just restart the Utility.
- If it was processing with any of the above commands:
  1. Correct any command syntax errors.
  2. Remove all commands that were successfully processed and committed.
  3. Restart the Utility.

If the Utility ends with a return code of 4, this means that all the commands supplied in the command input were processed successfully but a DBS program termination error occurred. The Utility does not need to be rerun.

For full descriptions of DBS Utility return codes and error processing, see the *DB2 Server for VSE & VM Database Services Utility* manual.

Preprocessor

If the preprocessor fails to complete the processing of all source statements supplied as input, or if it terminates with a return code equal to or greater than 8, then before running it again you must analyze the source statement listing produced to determine the errors that occurred. If there are no error messages there that describe the error condition(s), look at the console messages. After all source statements and any other errors are corrected, rerun the preprocessor from the beginning.

If the preprocessor ends with a return code of 1 while the program is being preprocessed with the BLOCK option, this means that one or more SQL statements are disqualified for blocking. For further information on blocking, refer to the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

If it ends with a return code of 4, then one or more preprocessor warning messages are contained in the source statement listing. The preprocessor does not have to be rerun; however, the source statement listing should be checked to insure that the warning conditions involve objects known to be nonexistent at the time the preprocessor was run.

If it ends with a return code of 0 and no package was created, then the source statements read by the preprocessor contained no SQL statements. Here, the preprocessor must be rerun if the incorrect input source statements were supplied as input.

Recovery from User Logic Errors

User logic errors are those where the application server carries out the functions as requested, but the user (or program) determines that the change(s) requested should not have been made — for example, the wrong table or dbspace may have been dropped.

Recovery from a user logic error depends on when the error is detected. If it is detected before the changes have been committed, the application server supports user (or program) invoked dynamic application backout. A user or program can take certain actions to back out these changes, depending on the way in which the application server is being used. ISQL users accomplish this by issuing either the SQL ROLLBACK WORK statement or the ISQL CANCEL command, or by responding to ISQL prompts for CANCEL or ROLLBACK. The error handling logic in application programs can accomplish this by issuing a ROLLBACK WORK statement. In addition, in VM the invoker of the program can enter either the HX or SQLHX immediate command (HX causes a rollback and ends the CMS command; SQLHX causes a rollback, but does not end the CMS command.) If you have coded your own interactive program to process SQL statements dynamically, you can also code a cancel exit. This would allow a user of your program to perform a function similar to the ISQL CANCEL command.

For more information on cancel exits, refer to the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual. For user errors that are detected after changes have been committed, the user has three choices:

1. Manually reverse the effects of the changes.

   This involves issuing the INSERTs, PUTs, UPDATEs, and DELETEs necessary to cancel all changes. If the committed changes involved definitional change statements (CREATE, DROP, or ALTER), these too must be manually backed out, which can be quite a chore since definitional statements do not always have straightforward cancellation operations. For example, a DROP TABLE statement would have dropped views and authorizations along with the table; thus, to reverse its effects would include re-creating the views and regranting the authorizations.

2. Reset the data and reenter valid changes.

   If a back-up copy of the data exists, it may be simplest to just revert to this version and then reenter any valid changes made to the data since the copy was made. Reentering the valid changes can, of course, be as involved as the effort to back out invalid ones; however, it has the advantage in that it can be done by reexecuting applications.

   The DBS Utility UNLOAD facilities can be used to create back-up copies of data, and the RELOAD facilities can be used to reset data to a previous state. The DB2 Server for VSE & VM database archiving support can also be used to create back-up copies of the entire database and reset it.

3. Use filtered log recovery to bypass the changes.

   Filtered log recovery lets you rollback a committed logical unit of work (LUW). It sounds like an easy solution, but it must be exercised with extreme care. When you undo past errors, other database changes may be altered as well: rows that users thought they had deleted may unexpectedly reappear; the values in updated rows may change.

   If you are using referential integrity, then on completion of the filtered log recovery you should deactivate and activate your primary and foreign keys to have the database manager automatically recheck the referential constraints. See “Altering Referential and Unique Constraints” on page 65.

   Filtered log recovery can be used to bypass the operations recorded in the log. The smallest set of operations you can bypass is all the work done in a single LUW. You tell the application server which logical units of work to bypass by supplying EXTEND input file commands. Because you want to bypass work that has already been committed, you would use the ROLLBACK COMMITTED WORK command. All the EXTEND input file commands are described in the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual.

Dynamic Recovery from User Errors

To dynamically recover from user errors, users should take advantage of the facilities that are provided for detecting error situations and for backing out changes that should not have been committed.

Backing Out Data During an ISQL Session

When using ISQL, there will be times when you will want to backout an invalid action: for example, if AUTOCOMMIT is OFF and you entered an SQL statement that resulted in a negative SQLCODE, or changes to a table that proved to be incorrect.

Note: You cannot backout changes in a nonrecoverable storage pool.

If you detect an error before a change is committed, you can backout the change. How many changes you can backout depends on whether AUTOCOMMIT is ON or OFF.

If it is ON, every statement is its own logical unit of work (LUW), and ISQL will immediately issue a COMMIT WORK after processing the statement. The only exception is for INSERT, UPDATE, and DELETE statements that affect more than one row: in that case, ISQL displays a message that gives you the option of backing out. For all other statements, you can backout the changes before the statement completes its processing, by:

- Answering CANCEL, ROLLBACK, or NO (based on the reply prompt) to an ISQL message requesting a reply
- Entering the ISQL CANCEL command when you are informed that the terminal is free (VSE Only)
- Entering CANCEL if you are prompted to clear the screen or enter CANCEL. (VSE Only)
- Entering CANCEL or SQLHX if you are prompted to clear the screen (clear the screen after entering CANCEL). (VM Only)

When using the INPUT command, you can enter the BACKOUT command after an invalid data row is entered. This deletes all data rows entered since INPUT was issued, or since the last SAVE command was entered.

If AUTOCOMMIT is OFF, you have control over what is an LUW and when changes are to be made. When you backout a change, this undoes all changes made since the beginning of the LUW. You can backout a change by any of these methods:

- Entering a CANCEL or an SQL ROLLBACK WORK statement
- Answering CANCEL to any ISQL message requesting a reply (and then answering YES to message ARI7041D)
- Entering the ISQL CANCEL command when you are informed that the terminal is free (VSE Only)
- Entering CANCEL if you are prompted to clear the screen or enter CANCEL. (VSE Only)
- Entering CANCEL or SQLHX if you are prompted to clear the screen (clear the screen after entering CANCEL). (VM Only)

Note: In VM, when you enter a CANCEL command, ISQL does ROLLBACK WORK RELEASE processing. Any explicit connection you have made will be released. You should reissue the CONNECT statement if you want to explicitly connect to ISQL again.

Backouts Initiated by Application Programs

An application program may begin a backout if the application server shows that there is an error, or if the program detects something wrong internally. To detect and handle errors, the program should have the WHENEVER statement coded into it. It can then determine whether to continue or to stop execution when an error occurs.

All operations against recoverable storage pools are atomic, except in SUM NOLOG mode. That is, either the operation will be completed successfully, or any changes made by the operation will be reversed automatically. Changes made by previous operations in the same LUW are not affected. The application is free to either continue working within the same LUW, to COMMIT the changes made so

far, or to ROLLBACK the LUW. Some errors, such as deadlock, still require the entire LUW to be rolled back. The status of the LUW is indicated in SQLWARN6 in the SQLCA.

When running with LOGMODE=N, atomicity of operations is enforced by rolling back the current LUW to avoid partial completion of an operation. For operations on data in nonrecoverable storage pools, there is no support for atomicity of operations.

Note: When blocking, the database manager does not insert rows into the database until the block is full and it does not notify your program of an insert error until the PUT that fills a block is run. To determine when (or if) rows are actually inserted into the database, your program should examine SQLERRD(3) in the SQLCA when doing PUTs.

To rollback work when an SQL error is encountered, code a ROLLBACK WORK statement in the program, and use a WHENEVER SQLERROR GO TO statement to cause a branch to the ROLLBACK statement when there is an SQL error. After the program issues a ROLLBACK WORK, it may continue processing more SQL statements without the previous error affecting their outcome.

If the application programmers do not wish to worry about setting up error-recovery logic in their programs, they can enable them to stop executing when an SQL error is detected. This is done by coding WHENEVER SQLERROR STOP (COBOL, COBOL II, PL/I) or WHENEVER SQLERROR GOTO. When this is coded, the database manager will issue either a CANCEL (in VSE) or a CMS DMSABN macro (in VM) for the application when any command results in a negative SQLCODE, which results in a ROLLBACK WORK for any outstanding LUW within the application program. Alternatively, the application programmer could code a WHENEVER SQLERROR GOTO and branch to a label or routine to perform the ROLLBACK WORK and end the program.

If the program detects an internal error and wishes to discontinue processing, it is probably best to issue a ROLLBACK WORK (if possible) before terminating it. This can be done by coding a ROLLBACK WORK statement in the application and branching to it when an internal program error is detected. After the ROLLBACK WORK statement is run, the program can stop, or continue if desired.

In VM, once a program is running, you can stop it by using the immediate commands HX or SQLHX, both of which cause a ROLLBACK WORK for the current LUW. You might want to do this if, for example, you start the program and then realize you have provided the wrong inputs. The difference between the two commands is that HX causes an end to the CMS command, while SQLHX does not. Thus, the choice of command is a matter of convenience. For example, issuing HX from ISQL both rolls back the current LUW and ends the ISQL session, so the user must reinitialize ISQL to continue processing; issuing SQLHX causes the LUW to be rolled back but the ISQL session continues.

Note: The ISQL CANCEL command and the more general SQLHX command have equivalent functions. The CANCEL command, however, does not work for user programs. In addition, CANCEL, SQLHX, and HX do not work if you have processed the SQLINIT command with the SYNCHRONOUS(YES) option.

Selective Recovery from User Data Errors

It is a good idea to maintain backup copies of specific tables or dbspaces, so that they can be reset in case of major errors.

Periodic Backup of Critical Data

Individual tables or entire dbspaces should be periodically unloaded to either a SAM tape or DASD file (in VSE), or to a tape or CMS minidisk file (in VM) with the DBS Utility UNLOAD command.

Multiple UNLOAD commands can be put in a single DBS SYSIPT (VSE), or SYSIN (VM), input file. You might establish one such job stream for periodic back-up of users' PRIVATE dbspaces, and others for periodic back-up of selected application production data. Different types of data would typically have different back-up schedules. For example, production data would probably be backed up more frequently than query user data. Some DB2 Server for VSE data, such as certain data extracted from DL/I, would not require back-up; that is, the DL/I copy of the data is sufficient back-up.

Note: You cannot use the DBS Utility UNLOAD facilities to back up data in the system dbspaces (SYS000n). The catalog tables and packages cannot be reset by DBS RELOAD processing.

Resetting Data Using DBS RELOAD Processing

When data is backed up, you can recall the backup copy if necessary. Data that was backed up with the UNLOAD TABLE command is recalled with the RELOAD TABLE command; data that was backed up with the UNLOAD DBSPACE command can be recalled with either RELOAD DBSPACE (to reset the entire dbspace) or with RELOAD TABLE (to recall selected tables in the dbspace). Often, user data errors that have been introduced into the database are isolated to just a few tables; thus, even if the data had been unloaded with an UNLOAD DBSPACE command, you would use RELOAD TABLE to reset it.

When a table is RELOADed with the NEW option, a new table is created and data reloaded. None of the primary keys, indexes, unique constraints, referential constraints or field procedures are reproduced in the new table.

When you use the RELOAD command with the PURGE option to replace the contents of a table, the DBS Utility does the following to the table being replaced:

1. Drops the CLUSTERING index (if one exists).
2. Deactivates the active primary key (if one exists).
3. Deactivates all active foreign keys.
4. Deactivates all unique constraints.
5. Drops all other indexes.
6. Deletes all rows from the table.
7. Reloads data.
8. Recreates the CLUSTERING index previously dropped.
9. Activates the primary key previously de-activated.
10. Activates the unique constraints previously de-activated.
11. Recreates any remaining indexes previously dropped.

As a result, the CLUSTERING index will be preserved, as well as the primary key, foreign keys, unique constraints, and indexes existing on the table at the time of the RELOAD/PURGE command. If no CLUSTERING index exists, then the primary key becomes the CLUSTERING index. There is no requirement to order the reloading of tables, because all referential constraints are inactive while the data is inserted.

Consider running the DBS Utility in single user mode with LOGMODE=N when resetting data through RELOAD DBSPACE or RELOAD TABLE processing. This will eliminate any log overflow conditions that result from the table row deletes and inserts performed by RELOAD processing with the PURGE option. If you use log archiving, however, remember that switching the log mode disrupts the continuity of the log.

Running the DBS Utility with LOGMODE=N is shown in Figure 25 and Figure 26. If the data resides in a nonrecoverable storage pool, there is no need to use LOGMODE=N, because logging is automatically suppressed for nonrecoverable data.

```
// JOB RESTORE DBSPACE
// EXEC PROC=DBNAME01
// EXEC PROC=ARIS71PL
// TLBL DUMPTAP,.........
// ASSGN SYS004,.........
// EXEC ARISQLDS,SIZE=AUTO,PARM='STARTUP=L,SYSMODE=S,LOGMODE=N,DUALLOG=Y'
// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,LOGMODE=N,PROGNAME=ARIDBS'
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
RELOAD DBSPACE (SQLDBA.EXAMPLE) PURGE INFILE(DUMPTAP PDEV(TAPE)
/&
```

Figure 25. Resetting a DB2 Server for VSE DBSPACE from a Back-up Copy

Notes:

1. The job control here assumes that the DB2 Server for VSE database was last shut down with the ARCHIVE, UARCHIVE, or LARCHIVE option (depending on whether you use LOGMODE=A or LOGMODE=L).
2. The first execution of the ARISQLDS exec starts the DB2 Server for VSE system in single user mode (SYSMODE=S), and does a COLDLOG (STARTUP=L) to redefine the log data sets. This step switches from LOGMODE=A or LOGMODE=L to LOGMODE=N, and is not needed unless you run with LOGMODE A or L. Omit the parameter DUALLOG=Y if you are not using dual logging.
3. The second execution of the ARISQLDS exec runs the DBS Utility with the input shown. This step RELOADs all the table data into the DBSPACE named SQLDBA.EXAMPLE from a tape file (filename=DUMPTAP) created by the DBS Utility UNLOAD DBSPACE command.

   For further information about switching log modes, see the *DB2 Server for VSE System Administration* manual.

```
EXEC SQLLOG DB(dbname)
FILEDEF DUMPTAP TAPn (RECFM VBS BLOCK 800
EXEC SQLDBSU DB(dbname) IN(TERM) LOGMODE(N)
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
RELOAD DBSPACE (SQLDBA.EXAMPLE) PURGE INFILE(DUMPTAP)
COMMIT WORK RELEASE;
```

Figure 26. Resetting a DB2 Server for VM Dbspace from a Back-up Copy

Notes:

1. The CMS commands here assume that the application server was last shut down with the ARCHIVE, UARCHIVE, or LARCHIVE option (depending on whether you use LOGMODE=A or LOGMODE=L). This ensures that you will be able to restore the database if a DASD fails.

2. The first run of the DB2 Server for VM program (by way of the SQLLOG EXEC) does a COLDLOG, which is necessary to switch from LOGMODE=A or L to LOGMODE=N. If you do not run with LOGMODE=A or L, you do not need to run SQLLOG to do a COLDLOG.
3. Respond `N` for NO to message ARI0688D, which asks whether you want to FORMAT and RESERVE the log minidisk(s).
4. The second run processes the DBS Utility with the input shown, to RELOAD all the table data in the dbspace named SQLDBA.EXAMPLE from a tape file (ddname=DUMPTAP) created by DBS Utility UNLOAD DBSPACE command processing. CMS FILEDEF commands direct the DBS input to the terminal and DUMPTAP to the tape.
5. After reloading the table, switch back to LOGMODE=A or L and create another database archive.

Database Recovery from User Logic Errors

To protect the entire database from user logic errors, use the archiving and COLDLOG facilities of the database manager. These facilities are required to protect the system catalog tables and the package dbspaces. Backup copies of the system dbspaces (SYS000*n*) made by DBS Utility UNLOAD command cannot be used to reset catalog tables or packages to a previous state.

Creating a Proper Back-up Copy of the Database

The back-up copy of the database can be either a database archive or a database archive and subsequent log archives.

You can create the database archive by using a variety of facilities. You must, however, create the archive when no user is accessing the database. Create the archive by using either the SQLEND ARCHIVE or SQLEND UARCHIVE command. Because no user is accessing the database when the database archive is taken, no incomplete changes are recorded in the database archive.

If you use log archiving, you can think of the last back-up copy as being the last database archive plus all subsequent log archives. Log archives do not record changes by incomplete logical units of work.

Note: If you are using the CICS subsystem and it ends abnormally, or the connections from the online resource adapter to the application server are ended by a CIRR QUICK, or the online adapter is ended by a CIRT QUICK or CIRR QUICK command, an exception can occur: that is, incomplete changes can be in the archive copy of the database if there are CICS transactions that are left in-doubt when the SQLEND archive is taken. To avoid this condition, enter a SHOW ACTIVE command to see if there are any LUWs that are marked as being in-doubt. If there are, enter the necessary FORCE commands to complete them before you enter the SQLEND ARCHIVE command.

You can create a proper back-up copy even if you have been running the database manager with LOGMODE=Y. However, if you create a database archive by using SQLEND parameters when LOGMODE=Y, you must follow the steps outlined in the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual to restart the database manager with LOGMODE=Y, because the log mode will automatically change to A when taking the database archive.

Resetting the Database to a Previous Copy

If you are restoring from a database archive without using subsequent log archives, you can reset the database to any previous database archive copy, not just the latest one.

To reset a DB2 Server for VSE database to a previous copy generated by an SQLEND command, run COLDLOG before restoring the database from the archive copy. This reformats the log so that changes since the archive was taken are not applied again.

To reset a DB2 Server for VM database, run the SQLLOG EXEC (omitting the LOG1 and LOG2 parameters) to reformat the log with a COLDLOG. Respond "NO" to message ARI0688D (for single logging) or ARI6129D (for dual logging). When you respond NO, the database manager reformats the log such that changes since the archive was taken are not applied again.

If you are restoring from a database archive and subsequent log archives, no COLDLOG is required. When the database is restored, the logs are restored in sequence. You are prompted to continue the log restore before processing each log archive. You can end the restore process at any log archive by responding "END RESTORE" to the appropriate prompt.

When resetting the database to a back-level copy, even if you are using subsequent log archives, you should be aware of the following:

- The database archive copy includes a copy of the database directory, but the database manager does not recognize any ADD DBSPACE and ADD DBEXTENT operations which were done after the database archive. To reestablish these dbspaces and dbextents in a VSE system, you must rerun the appropriate ADD DBSPACE and ADD DBEXTENT operations. You can determine how many dbextents exist in the restored back-level database by using the SHOW DBEXTENT operator command. (Add the values in the NO._OF_EXTENTS column.) You can determine the numbers of public and private dbspaces in the restored database by querying the SYSTEM.SYSDBSPACES catalog table. For more information on the catalog tables, see the *DB2 Server for VSE System Administration* manual or the *DB2 Server for VM System Administration* manual. In VM, if you want to reestablish the dbspaces added after the database archive was created, you must rerun the SQLADBSP EXEC. For more information, see the *DB2 Server for VM System Administration* manual.

  Any dbextents added to the database (by an ADD DBEXTENT operation) after the database archive was created do not exist in the archive copy of the database. In VM, the CP LINK and CMS FILEDEF commands for these dbextents are present in the *resid* SQLFDEF file (on the DB2 Server for VM production minidisk) for the database. To redefine these dbextents in the DB2 Server for VM database, perform the following procedure:

  1. Create an ADD DBEXTENT card image input CMS file with a line entry for each added dbextent. Each entry should contain the dbextent number and the storage pool number for the dbextent.

     Note: The SHOW DBEXTENT operator command tells you how many dbextents are defined in the database.
  2. Enter a CMS FILEDEF command with *ddname* SYSIN for the CMS input file:

     ```
     FILEDEF SYSIN DISK fn ft fm
     ```
  3. Run the SQLSTART EXEC with PARM(SYSMODE=S,STARTUP=E...) to redefine the dbextents in the database.

- The database archive of the directory shows the DUALLOG value in effect when the database archive was created. The database archive also shows the size of the logs when the archive was taken. You can reset the DUALLOG value and the size of the logs by doing a COLDLOG operation to reformat the logs after the database is restored.

  In VSE, do a COLDLOG by specifying STARTUP=L and the DUALLOG value that you want. For more information about DUALLOG, see the *DB2 Server for VSE System Administration* manual.

  In VM, do a COLDLOG by running the SQLLOG EXEC without the LOG1 and LOG2 parameters. In this situation, the log minidisk is already reserved and formatted; only the directory needs to be updated. Respond "NO" to message ARI0688D (for single logging) or ARI6129D (for dual logging), which prompts you to FORMAT and RESERVE the log minidisk (or minidisks).

This final consideration applies when you restore a database archive without applying subsequent log archives:

- All data (including the catalog table information) is reset to the database archive copy. Any preprocessing, data definitions, grants, revokes, and stored queries or routines established after the database archive was created are lost. The database may not be consistent with other facilities on your system. In particular, it may not be consistent with your CICS or DL/I data, and the packages may not reflect the SQL application programs you installed on your system after the database archive was created.

Resetting the Database without Reformatting the DB2 Server for VSE Data Sets

A database restore (STARTUP=R) reformats the VSAM database data sets before the data is reloaded. Reformatting the data sets is necessary after a data set is replaced (for example, when restoring because of a media failure or database reconfiguration). Reformatting the data sets is not necessary when none of the database data sets is being replaced.

When restoring the database to a previous level to recover from a user logic error, you usually do not change the data sets. To save processing time, use STARTUP=F (fast restore) when you have not replaced any of the database data sets. The STARTUP=F processing does not format the VSAM data sets: it loads the data. Eliminating the formatting of the data sets significantly reduces the restore time.

Chapter 7. Customizing the HELP Text and Messages Text

The DB2 Server for VSE & VM messages and HELP texts are stored in tables, meaning that they can be retrieved and manipulated just like any other data. You can modify the information to suit local needs in the following ways:

- Adding or deleting topics
- Changing the information in existing topics
- Adding HELP text supplied by IBM in supported languages
- Adding your own HELP text in supported languages.

Note: A HELP command causes ISQL to issue a SELECT statement on the tables.

Figure 27 shows the relationships between the tables used by the application server for HELP text support.

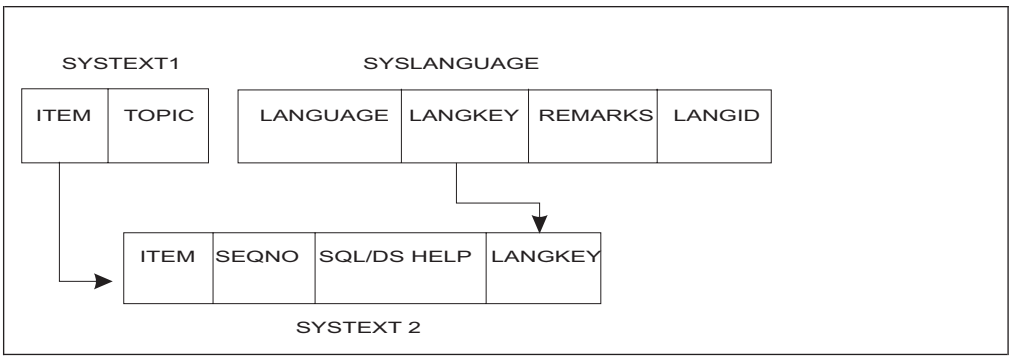


Figure 27. Relationships between SYSLANGUAGE, SYSTEXT1, and SYSTEXT2

The relationships between the tables are maintained through the following sets of matching columns:

- LANGKEY in both SYSLANGUAGE and SYSTEXT2
- ITEM in both SYSTEXT1 and SYSTEXT2.

These tables are explained in more detail below.

The SYSLANGUAGE Table

HELP and messages texts are provided in several national languages. During installation, one language is established as the default; it can be changed after installation. In addition, you can make more than one language available to ISQL users.

The SYSLANGUAGE table is created as part of the installation process. It lists all national languages that are currently supported on the application server, meaning that both HELP text and messages text are available in these languages. Its primary purpose is for use with the message repository, which is a mandatory part of the product installation. Installing the HELP text is optional.

Each entry in this table has the following fields:

1. LANGUAGE — the name of the language. There can be more than one entry to describe the same language: for example, FRANCAIS, FRENCH, and FR can all be LANGUAGE field values for French.

2. LANGKEY — the language key. This is a four-character code that uniquely identifies each language, regardless of what name labels it in the LANGUAGE field.
3. REMARKS — a description of the entry.
4. LANGID — the language identifier.

To view all the columns of the SYSLANGUAGE table, enter the following query:

```
SELECT * FROM SQLDBA.SYSLANGUAGE
```

Figure 28 shows a sample SYSLANGUAGE table.

```
LANGUAGE        LANGKEY  REMARKS                                  LANGID
--------------  -------  ---------------------------------------  --------
ENGLISH            S001  AMERICAN ENGLISH VERSION OF HELP TEXT     AMENG
ENGLISH            S002  ENGLISH UPPER CASE VERSION OF HELP TEXT   UCENG
FRENCH             S003  FRENCH VERSION OF HELP TEXT               FRANC
FRANCAIS           S003  TEXTE D'AIDE FRANCAIS                     FRANC
```

Figure 28. Sample SQLDBA.SYSLANGUAGE Table

The language key (LANGKEY) can be one of those listed in Table 26.

Table 26. Language Keys

Language Key	Description	Language ID
S001	American mixed case	AMENG
S002	English upper case	UCENG
S003	French	FRANC
S004	German	GER
D001	Japanese	KANJI
D003	Simplified Chinese	HANZI

Note: IBM has reserved the following language key ranges for use by future languages supplied by IBM:

- S000 to S500 for single-byte character set (SBCS) or EBCDIC languages
- D000 to D500 for double-byte character set (DBCS) languages.

In VM, the default language is established by the language currently set in CMS. If this language is not supported by the application server, then the default language defined during installation is used.

In VSE, the default language is established by the following:

- ISQL — from either a parameter in the CIRB transaction (LANGID), or a language supplied by IBM.
- DBSU — link-edited with the default language (messages only)
- PREP — link-edited with the default language (messages only).
- DSQG, DSQU, DSQQ and DSQD — from a parameter in the CIRB transaction (LANGID).
- CBND — from a parameter in the CIRB transaction (LANGID).
- SQLGLOB File Batch Update/Query Program — link-edited with the default language (messages only).

The SYSTEXT1 and SYSTEXT2 Tables

The HELP text tables are normally loaded during the installation process. The DBS Utility accomplishes this task by creating the HELP text tables SYSTEXT1 and SYSTEXT2 for the user SQLDBA, loading data into both tables (through DATALOAD), and creating an index on each.

Figure 29 shows the formats of these tables, but not the actual tables.

```
SYSTEXT1                    SYSTEXT2

TOPIC        ITEM           ITEM  SEQNO  "SQL/DS HELP"              LANGKEY
-----------  ----           ----  -----  ----------------------     -------
VIEW         5260           5260     10  TOPIC NAME: CREATE VIEW       S001
VIEW         5330            •        •              •                  •
VIEW         5920           5260    110  CREATE VIEW is an SQL ...     S001
VIEWS        5260           5260    120  more tables. You can ...      S001
VIEWS        5330            •        •              •                  •
VIEWS        5920            •        •              •                  •
    •          •            5260   1070  DELIVERY_TIME was less ...    S001
CREATE VIEW  5260           5260   1080                                S001
    •                       5260     10  RUBRIQUE :  CREATE VIEW       S003
DROP VIEW    5330            •        •              •                  •
    •          •             •        •              •                  •
    •          •            5260    100  CREATE VIEW est une ...       S003
    •          •            5260    110  d'une ou plusieurs ...        S003
                             •        •              •                  •
                             •        •              •                  •
                            5260   1110  FAB, ART et JOURS) ...        S003
                            5260   1120                                S003
                            5330     10  TOPIC NAME: DROP VIEW         S001
                             •        •              •                  •
                             •        •              •                  •
                            5330     90  DROP VIEW is an SQL ...       S001
                            5330    100  SQL/DS also automatically ... S001
                             •        •              •                  •
                             •        •              •                  •
```

Figure 29. Formats of the Tables SYSTEXT1 and SYSTEXT2

The following SQL statements are used during the loading process to create SYSTEXT1 and SYSTEXT2:

```
CREATE TABLE SQLDBA.SYSTEXT1 (TOPIC CHAR(20) FOR BIT DATA NOT NULL,
                              ITEM  SMALLINT NOT NULL)
                              IN "PUBLIC"."HELPTEXT"


CREATE TABLE SQLDBA.SYSTEXT2 (ITEM  SMALLINT NOT NULL,
                              SEQNO SMALLINT NOT NULL,
                              "SQL/DS HELP" CHAR(60) FOR BIT DATA NOT NULL,
                              LANGKEY CHAR(4) NOT NULL)
                              IN "PUBLIC"."HELPTEXT"
```

When a user enters a HELP command, a query like this is processed:

```
SELECT "SQL/DS HELP"
  FROM SQLDBA.SYSTEXT1, SQLDBA.SYSTEXT2

  WHERE TOPIC = 'topicname'

    AND SQLDBA.SYSTEXT1.ITEM = SQLDBA.SYSTEXT2.ITEM
    AND LANGKEY = 'XXXX'
```

where XXXX is the four-character language key that indicates a specific HELP text language from among those currently installed on the DB2 Server for VSE & VM application server.

When the support for languages is installed, HELP text may or may not be available depending on your site's requirements. Each ISQL user can select from among the languages currently installed on the application server. To view which languages are currently installed, a user enters the following query:

```
SELECT LANGUAGE FROM SQLDBA.SYSLANGUAGE
```

The user can then change the default language with the ISQL SET LANGUAGE command.

The topic that the user supplies is substituted in *topicname*. An ORDER BY clause is not used in the query because these indexes are defined on the tables:

```
CREATE INDEX SQLDBA.SYSTEXT1INDEX
                ON SQLDBA.SYSTEXT1(TOPIC,ITEM)

CREATE INDEX SQLDBA.SYSTEXT2INDEX
                ON SQLDBA.SYSTEXT2(ITEM,SEQNO,LANGKEY)

CREATE INDEX SQLDBA.SYSLANGINDEX
                ON SQLDBA.SYSLANGUAGE(LANGUAGE)

CREATE INDEX SQLDBA.SYSLANGINDEX
                ON SQLDBA.SYSLANGUAGE(LANGID)
```

A HELP command uses SYSTEXT1 as a pointer to SYSTEXT2. Suppose an ISQL user enters:

```
help 'view'
```

The parameter 'view' is converted to uppercase. The database manager finds all occurrences of the character string 'VIEW' in the TOPIC column of SYSTEXT1 for the HELP text of the current language. See Figure 30.

```
          SYSTEXT1

          TOPIC        ITEM
          -----------  ----
------->  VIEW         5260
------->  VIEW         5330
------->  VIEW         5920
          CREATE VIEW  5260
          CREATE V     5260
          DROP VIEW    5330
          VIEW QUERY   5030
          VIEW MODS    5040
              •          •
              •          •
              •          •
```

Figure 30. Use of the SYSTEXT1 Table

Figure 30 shows three occurrences of the string *VIEW*. Each has an item number associated with it (5260, 5330, 5920). These numbers and the language key are used as pointers (through the query join) to the ITEM numbers and language key in table SYSTEXT2. The rows in SYSTEXT2 having those ITEM numbers and

language key are retrieved in order, primarily by ITEM number and the language key, and secondarily by sequence number (SEQNO). Thus, three unique topics are returned when HELP 'VIEW' is entered.

Note that other rows in SYSTEXT1 have identical ITEM numbers but different names (TOPIC). These rows enable retrieval of each of the four topics separately. For example, the command HELP 'CREATE VIEW' retrieves only the topic having ITEM number 5260. Similarly, the 'CREATE V' entry in table SYSTEXT1 is an alias for 'CREATE VIEW'; it also points to ITEM 5260.

This cross-referencing scheme has three forms:

- Duplicate topic names pointing to more than one actual topic (for example, HELP 'VIEW').
- Multiple topic names pointing to the same topic (for example, HELP 'CREATE VIEW' and HELP 'CREATE V').
- A unique topic name pointing to one topic (for example, HELP 'VIEW MODS').

Adding Topics to HELP Text Tables

You can add new topics to the HELP text tables supplied by IBM, or create your own HELP text table. As modifying the HELP text supplied by IBM greatly increases the amount of administrative work required if you must later reinstall the HELP text, a much better method is to set up your own independent HELP text tables in some other PUBLIC dbspace. This method is described in "Creating Your Own HELP Text Tables" on page 142.

Adding a HELP Topic to the HELP Text Supplied by IBM

Parts of this task require DBA authority (or at least INSERT authority on the SYSTEXT1 and SYSTEXT2 tables). If you plan to add much new material to the HELP text, see "Making the HELPTEXT Dbspace Larger" on page 143 and "Moving the HELP Text to Another Dbspace" on page 145.

To add your own topic to the tables, follow these steps:

1. Pick a TOPIC name, up to a maximum of 20 characters. This name must be unique among all TOPIC names in table SYSTEXT1. An easy way to check this is to enter the query:

   ```
   SELECT * FROM SQLDBA.SYSTEXT1 WHERE TOPIC = 'candidate name'
   ```

   If rows are returned, that TOPIC name already exists, and you must choose and test another.
2. Choose an ITEM number less than 5 000 for the new topic. Numbers of 5 000 and above are reserved for topics supplied by IBM.
3. Insert a row into SYSTEXT1 for the new TOPIC name and its ITEM number. For example:

   ```
   INSERT INTO SQLDBA.SYSTEXT1 VALUES ('HOURS',1000)
   ```
4. Insert rows into SYSTEXT2 for the information to be displayed when a user requests HELP on this new topic. This information must include the values to be used in the four columns of table SYSTEXT2. For example:

   ```
   INSERT INTO SQLDBA.SYSTEXT2
       VALUES(1000,10,'HOURS OF USE:','S001')
   INSERT INTO SQLDBA.SYSTEXT2
       VALUES(1000,20,'8 AM TO 6 PM','S001')
   ```

where “S001” is the English language key. You can repeat this type of INSERT for every other language.

Note: The “SQL/DS HELP” column has a length of 60 characters.

When adding HELP text to the SYSTEXT2 table, a language key must be specified. A list of valid language keys is found in Table 26 on page 138. You should use installation procedures supplied by IBM.

Creating Your Own HELP Text Tables

You should consider using the SYSTEXT1 and SYSTEXT2 tables as the basis for creating your own HELP text tables. SYSLANGUAGE must still exist for the HELP command to work, unless you establish HELP text tables and query those tables as shown in Figure 31.

Figure 31 shows example SQL commands to set up your own local HELP text.

```
CREATE TABLE SQLDBA.LTEXT1 (TOPIC CHAR(20) FOR BIT DATA NOT NULL,
                            ITEM SMALLINT NOT NULL)
                            IN "PUBLIC".LOCAL

CREATE TABLE SQLDBA.LTEXT2 (ITEM SMALLINT NOT NULL,
                            SEQNO SMALLINT NOT NULL,
                            "LOCAL HELP" CHAR(60) FOR BIT DATA NOT NULL,
                            LANGKEY CHAR(4) NOT NULL)
                            IN "PUBLIC".LOCAL

CREATE INDEX SQLDBA.LTEXT1INDEX
                 ON SQLDBA.LTEXT1(TOPIC,ITEM)

CREATE INDEX SQLDBA.LTEXT2INDEX
                 ON SQLDBA.LTEXT2(ITEM,SEQNO,LANGKEY)
...
SELECT "LOCAL HELP"
  FROM SQLDBA.LTEXT1, SQLDBA.LTEXT2

  WHERE TOPIC = 'topicname'

    AND SQLDBA.LTEXT1.ITEM = SQLDBA.LTEXT2.ITEM
    AND LANGKEY='XXXX'
```

Figure 31. Implementing Your Own HELP Text Tables

XXXX in the LANGKEY column represents the language key.

In this example, two tables, SQLDBA.LTEXT1 and SQLDBA.LTEXT2, are created in a PUBLIC dbspace called LOCAL. Appropriate indexes are also defined. Once the tables are created, you can add topics in a way similar to that described previously for the tables of HELP text supplied by IBM. Replace the names supplied by IBM for the HELP text tables, dbspace, and column names with your own names.

Users can then access the new HELP text with an ISQL routine that contains a SELECT statement (see the example in Figure 31). The ISQL stored routines supplied by IBM for accessing the original HELP text may not work for the new tables, so it may be necessary to set up new ones. The SELECT authority must be granted to all users on the table containing the routine and on the HELP text tables.

Making the HELPTEXT Dbspace Larger

The size of the original HELPTEXT dbspace is 8192 pages, which is sufficient to hold the HELP text supplied by IBM and four or five languages. If you plan to add extensively to the text or to add more than five languages, it may be necessary to increase the size of this dbspace.

To see how many pages are currently active in the HELPTEXT dbspace, issue the following query through ISQL or the DBS Utility:

```
SELECT DBSPACENAME,NACTIVE
  FROM SYSTEM.SYSDBSPACES
  WHERE DBSPACENAME='HELPTEXT'
```

If the NACTIVE (number of active data pages) value is close to 4646 (8192 minus the index pages allowance), consider making the HELPTEXT dbspace larger. To estimate how many pages are needed in the dbspace for the modified HELP text, see "Appendix A. Estimating Your Dbspace Requirements" on page 215.

If the estimated number of pages (for both current and future estimated usage) is greater than or close to 8192, increase the size of the dbspace. To do this, you must drop and re-create the dbspace, as follows:

1. UNLOAD the "PUBLIC"."HELPTEXT" dbspace using the DBS Utility.
2. DROP the "PUBLIC"."HELPTEXT" dbspace.
3. ACQUIRE a new "PUBLIC"."HELPTEXT" dbspace with the new required number of pages.
4. RELOAD the dbspace using the DBS Utility.
5. Reinstate the required indexes and authorities.
6. Reinstate any user-defined indexes, views, or authorities.
7. Proceed with the updates to the HELP text.

Figure 32 and Figure 33 show examples of increasing the size of the "PUBLIC"."HELPTEXT" dbspace to 8448 pages. A tape is used in this example to temporarily hold the HELP information that is on your database.

```
// JOB UNLOAD HELP TEXT
// EXEC PROC=DBNAME01
// EXEC PROC=ARIS71PL
// TLBL HELPTAP,.......
// ASSGN SYS005,.......
// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S,LOGMODE=N,PROGNAME=ARIDBS'


CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
UNLOAD DBSPACE ("PUBLIC"."HELPTEXT") OUTFILE(HELPTAP);
DROP DBSPACE "PUBLIC"."HELPTEXT";
ACQUIRE PUBLIC DBSPACE NAMED "HELPTEXT" (PAGES=8448);
RELOAD DBSPACE ("PUBLIC"."HELPTEXT") NEW INFILE(HELPTAP);
CREATE INDEX SQLDBA.SYSTEXT1INDEX ON SQLDBA.SYSTEXT1 (TOPIC,ITEM);
CREATE INDEX SQLDBA.SYSTEXT2INDEX ON SQLDBA.SYSTEXT2 (ITEM,SEQNO,LANGKEY);
GRANT SELECT ON SQLDBA.SYSTEXT1 TO PUBLIC;
GRANT SELECT ON SQLDBA.SYSTEXT2 TO PUBLIC;

COMMENT ' ** OPTIONALLY ADD SQL STATEMENTS TO GRANT AUTHORIZATIONS **

          ** OR CREATE ANY VIEWS REQUIRED FOR THE NEW DATA BASE. **';
CREATE VIEW ........;

:
GRANT ..............;

:

/&
```

Figure 32. Unloading and Reloading the HELP Text in VSE

```
FILEDEF HELPTAP TAPn...
SQLDBSU DB(DBNAME01) IN(TERM)
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
UNLOAD DBSPACE ("PUBLIC"."HELPTEXT") OUTFILE(HELPTAP);
DROP DBSPACE "PUBLIC"."HELPTEXT";
ACQUIRE PUBLIC DBSPACE NAMED "HELPTEXT" (PAGES=8448);
RELOAD DBSPACE ("PUBLIC"."HELPTEXT") NEW INFILE(HELPTAP);
CREATE INDEX SQLDBA.SYSTEXT1INDEX ON SQLDBA.SYSTEXT1 (TOPIC,ITEM);
CREATE INDEX SQLDBA.SYSTEXT2INDEX ON SQLDBA.SYSTEXT2 (ITEM,SEQNO,LANGKEY);
GRANT SELECT ON SQLDBA.SYSTEXT1 TO PUBLIC;
GRANT SELECT ON SQLDBA.SYSTEXT2 TO PUBLIC;

COMMENT ' ** OPTIONALLY ADD SQL STATEMENTS TO GRANT AUTHORIZATIONS **

          ** OR CREATE ANY VIEWS REQUIRED FOR THE NEW DATA BASE. **';
CREATE VIEW ........;

:
GRANT ..............;

:
COMMIT WORK RELEASE;
```

Figure 33. Unloading and Reloading the HELP Text in VM

Moving the HELP Text to Another Dbspace

The HELP text can also be moved to another dbspace if more space is needed for additional user documentation or if it needs to be moved for other administrative reasons. The current size of the HELPTEXT dbspace is 8192 pages. The dbspace to which the HELP text is being moved must be at least that size. To accomplish the move:

1. UNLOAD the "PUBLIC"." HELPTEXT" dbspace using the DBS Utility.
2. DROP the "PUBLIC"."HELPTEXT" dbspace.
3. ACQUIRE a new dbspace with the desired number of pages.
4. RELOAD the new dbspace using the DBS Utility.
5. Reinstate any user-defined indexes, views, or authorities.
6. Proceed with updates to the HELP text (if updates are being done).

Note: The names of the tables, columns, and indexes cannot be changed. In addition, the owner name cannot change. Future reinstallations of the HELP text will assume that the original names exist in the database.

Printing the HELP Text Using the DBS Utility

Use the DBS Utility to produce hardcopy output of the HELP topics. Because the softcopy is stored in tables, you need only code a SELECT statement that retrieves the desired topics, and execute this statement through the DBS Utility control file input. The DBS Utility formats the output of the SELECT statement for you.

The broad categories of HELP topics and their ranges of ITEM numbers are as follows:

Text for	**Appears in ITEMs**
Commands (SQL and ISQL)	5000 - 9999
Messages	10000 - 19999
SQLCODES	20000 - 29999
Copyright Notice	30000

By using these ranges, you can code queries to retrieve various subsets of the HELP topics. For example, the following query retrieves all the SQL statements and ISQL commands (which were extracted from the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual):

```
SELECT * FROM SQLDBA.SYSTEXT2
  WHERE ITEM BETWEEN 5000 AND 9999 OR ITEM = 30000
  AND LANGKEY='S001'
  ORDER BY 1, 2;
```

Item 30,000 (the copyright notice) must be retrieved and printed whenever you print IBM machine-readable information.

To print a copy of all messages and codes for a language, you can use a query like:

```
SELECT * FROM SQLDBA.SYSTEXT2
  WHERE ITEM BETWEEN 10000 AND 30000
    AND LANGKEY='XXXX'
 ORDER BY 1, 2;
```

where "XXXX" represents a selected language key.

Printing the HELP Text Using ISQL

You can also enter the SQL statement described above through an ISQL terminal. When the desired HELP topics are displayed on the screen, enter a PRINT command to obtain a hardcopy. Perform the desired formatting before entering the PRINT command.

The class and number of copies desired can be specified on the PRINT command; otherwise, the defaults are used. See the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual for detailed information.

Because the HELP topics contain both upper- and lowercase characters, a print class that prints both characters should be specified. This depends on which HELP text language you select: in the case of English, for example, the HELP text contains both upper- and lowercase characters, so you should specify an appropriate print class.

Chapter 8. Application Design Considerations

This chapter describes the facilities that are available for designing and implementing applications, and discusses some considerations that application developers and the database administrator (DBA) should take into account.

Application Implementation Capabilities

This section discusses the application implementation alternatives that developers can consider. For each environment, several different ways of implementing application functions are identified and discussed.

The following broad categories of applications are also discussed:

- Query Capabilities
- Report Writing Capabilities
- Programmed Application Capabilities
- Execs that use DB2 Server for VM Facilities

Batch/Interactive Capabilities

For batch/interactive DB2 Server for VSE usage environments, there are two alternatives for application implementations:

- Assembler and High-Level-Language Programs

  These are programs written in FORTRAN, PL/I, COBOL, or assembler that would run as VSE batch jobs or interactive (VSE/ICCF) applications.

  In addition, you can use SQL Extended Dynamic Statements to code your own preprocessor in assembler language. Although the data requests must be made in SQL, you can code your preprocessor to translate some other data manipulation language to SQL statements. General concepts for coding a preprocessor are in the *DB2 Server for VSE & VM Application Programming* manual.
- DBS File Maintenance and Reporting

  These are executions of the Database Services (DBS) Utility, which supports execution of SQL statements and DBS commands.

High-Level-Language Programs

The primary vehicle for implementing application functions would be high-level-language programs written for execution as batch jobs in either VSE partitions or interactive partitions.

VSE Batch Partitions: VSE batch jobs support unit record devices or VSE/POWER spool files and VSE files for input and output processing. The DB2 Server for VSE system can be used to support data sharing, data recovery, and data function requirements of the applications.

VSE/ICCF Applications: Application programs written for execution under VSE/ICCF can support some level of interaction with a terminal user. The VSE/ICCF environment supports invocation of an application from a user terminal, and can be effectively used to manage user input (SYSIPT) and output (SYSLST) files as VSE/ICCF files.

DBS File Maintenance and Reporting

On batch/interactive systems with sufficient real storage available for dynamic SQL processing, the DBS Utility can be effectively used for file maintenance and reporting operations. For detailed information on maintenance, see "Chapter 3. Maintaining Your Database" on page 57.

Briefly, the DBS Utility can be used to:

- Load DB2 Server for VSE database tables from sequential files on DASD, magnetic tape, or SYSIPT
- Unload DB2 Server for VSE database tables to sequential files on DASD, magnetic tape, or SYSIPT
- Load DB2 Server for VSE tables from terminal users from bulk input files developed using the VSE/ICCF editor
- Unload data from DB2 Server for VSE tables to VSE/ICCF Files
- Implement SQL procedures

  Application functions that do not require procedural logic or variable information can be carried out as a sequence of DBS SQL statements in a SYSIPT (or VSE/ICCF) input file.
- Produce DBS reports, which may contain formatted listings of the selected data from an SQL SELECT statement
- Reorganize data in the database

  You can unload the data from the database using a DBS UNLOAD command, and reload it with a different structure using the DBS RELOAD command.
- Convert data for interchange with non-DB2 Server for VSE products

  For example, the utility can be used to load and unload data in a zoned-decimal format. Zoned-decimal is not a DB2 Server for VSE data type, so the DBS Utility converts the data as needed.
- Support interactive DBS processing from a VSE/ICCF terminal.

  If you include the VSE/ICCF `/DATA INCON` job control statement in the DBS job control, the utility takes its input (SYSIPT records) from the terminal input, and displays its output (SYSLST) at the terminal. You can tailor the DBS output to your terminal, by using the DBS `SET LINEWIDTH` command to specify the number of characters to be displayed on an output line.

Online (CICS) Transaction Processing Capabilities

The basic capabilities available in the batch/interactive environment are extended with the addition of the CICS subsystem and the online support. The following sections describe these additional capabilities.

The main method for implementing online transactions is through CICS high-level-language transaction processing programs. These programs can access the DB2 Server for VSE system and exploit the unique facilities of the CICS subsystem. Your transactions must be coded in COBOL, assembler, PL/I, or some other language that the CICS subsystem supports and for which you have coded a preprocessor. The CICS subsystem does not support FORTRAN.

The CICS transaction processing environment supports terminal-driven, fast-response-time application processing. It also provides facilities for interconnecting systems and distributing application processing in a network of systems.

In addition to supporting CICS transaction access to the DB2 Server for VSE system, data stored by the DB2 Server for VSE system can be shared with batch and VSE/ICCF application programs.

Query Capabilities

Data can be queried through either application programs, the DBS Utility, or ISQL. Which facility should be used depends on the complexity of the query and whether it will be used repeatedly.

One-Time Queries

Query functions can be coded as application programs or processed through ISQL. ISQL enables end users to formulate SQL queries on data and view and format the results.

You can satisfy many of your end users' data retrieval requirements by making the ISQL facilities available to them.

Periodic Queries

There are several ways to design queries that will be used repeatedly:

Stored Queries: Queries can be developed under ISQL and stored for future, repetitive use. End users can develop and store their own, or as DBA, you may choose to create a specific set for distribution. When you develop a stored query you can also save information about how its display should be formatted, so that when users invoke the query, the display will be automatically formatted for them. In addition, if you must later change a stored query, you can also change the formatting information. When possible, ISQL saves existing formatting information so that you do not have to re-enter it when there is a minor change to a stored query.

Stored queries cannot be shared among users, so a separate copy must be stored for each of them. The developer of a set of queries could use an ISQL routine (see "ISQL Routines") to enter and store them; then, any user who needed access to those queries would simply run the routine to obtain a copy of them. The user must have SELECT authority on the developer's ROUTINE table.

One of the advantages of stored queries is that users simply START them; they need not be familiar with SQL. With parameters, stored queries can be developed that are general in nature—that is, they can support variable input for the same basic function.

Note: If you develop stored queries and routines for use by multiple users, you may want to consider devising your own HELP tables to provide information on them. See "Creating Your Own HELP Text Tables" on page 142.

ISQL Routines: For more complex application functions that involve multiple SQL functions, consider using ISQL routines. Routines have the following advantages over stored queries:

- They can hold multiple SQL statements
- They support ISQL statements
- They can be shared.

Like stored queries, routines can be parameterized to provide variability in the function provided, and users need not understand the details of the underlying statements.

Because stored routines may support complex functions, you may want to account for possible error conditions, by using the ISQL SET RUNMODE and SET AUTOCOMMIT commands to provide for error handling in routines.

Figure 34 illustrates a routine that updates the SALARY value in the sample EMPLOYEE table. It also displays a report showing the old and new values, and prints the report of the transaction.

```
SALUPD   0010   COMMIT WORK
SALUPD   0020   SET AUTOCOMMIT OFF
SALUPD   0030   SET RUNMODE CANCEL
SALUPD   0040   UPDATE SQLDBA.EMPLOYEE -
SALUPD   0050     SET SALARY = SALARY + &3 -
SALUPD   0060     WHERE EMPNO = &1 AND JOB = &2;
SALUPD   0070   SELECT EMPNO,JOB,SALARY-&3,SALARY -
SALUPD   0080     FROM SQLDBA.EMPLOYEE -
SALUPD   0090     WHERE EMPNO = &1 AND JOB = &2 -
SALUPD   0100   UNION -
SALUPD   0110   SELECT EMPNO,JOB,SALARY -
SALUPD   0120     FROM SQLDBA.EMPLOYEE -
SALUPD   0130     WHERE EMPNO ¬= &1 AND JOB = &2 -
SALUPD   0140   ORDER BY 1
SALUPD   0150   FORMAT COL 3 NAME 'OLD SALARY'
SALUPD   0160   FORMAT COL 4 NAME 'NEW SALARY'
SALUPD   0170   FORMAT TOTAL (3 4)
SALUPD   0180   FORMAT TTITLE 'UPDATED THE SALARY OF EMPLOYEE &1 BY $ &3'
SALUPD   0190   DISPLAY
SALUPD   0200   PRINT COPIES 3
SALUPD   0210   END
SALUPD   0220   COMMIT WORK
```

Figure 34. Example ISQL Routine for the EMPLOYEE Table Update

To execute the ISQL routine in Figure 34, a user would enter (during an ISQL session):

```
RUN SALUPD (empno job change)
```

The routine is designed to update one row of the EMPLOYEE table based on parameter input specified on the RUN command, and run a query that displays the results of the update. All salaries for the job are displayed, not just the updated salary. After reviewing the display, the user enters an END command to have three copies of the display printed; then, the routine commits the transaction to the database. If the displayed results are not correct, the user can cancel the update by issuing the CANCEL command in place of the END command.

ISQL EXECs: In VM, ISQL and SQL statements can be stacked by an EXEC for execution by ISQL. Such EXECs can be created using either EXEC 2 or the System Product Interpreter, and can be written for execution either during or outside of an ISQL session. (You cannot write one that will run both ways.)

To process an ISQL EXEC that is designed to run during an ISQL session, the user enters “CMS” to get into Subset mode, and then types in the name of the EXEC.

The EXEC must place a RETURN CMS command as the **first** entry on the stack, in order to cause control to be returned to ISQL for processing of the rest of the statements on the stack.

CAUTION: EXECs that are processed from CMS Subset should not run ISQL, the DBS Utility, or SQL applications, as the results will be unpredictable.

Figure 35 shows an example of an EXEC called UPDSAL.

```
/*    UPDSAL EXEC                                          6/10/90   */

/* THIS EXEC PROGRAM ALLOWS A USER TO PERFORM UPDATES ON           */
/* THE SALARY COLUMN OF THE EMPLOYEE TABLE.  IT IS DESIGNED        */
/* TO BE STARTED WHILE IN A CMS SUBSET, AND IT AUTOMATICALLY       */
/* RETURNS TO ISQL TO EXECUTE THE UPDATE COMMANDS WHICH HAVE BEEN  */
/* PLACED ON THE PROGRAM STACK; THEN THE TABLE IS DISPLAYED AND    */
/* PRINTED TO SHOW THE CHANGES MADE.                               */

PARTLIST = ""              /* WILL CONTAIN LIST OF EMPNO'S WHOSE   */
                           /* TOTAL SALARY HAS CHANGED             */
DO COUNT = 1
   DO FOREVER
     SAY "ENTER EMPNO                      (ENTER 'END' WHEN DONE)"
     PULL E
     IF E = 'END' THEN LEAVE COUNT
     SAY ENTER JOB               /* USER ENTERS UPDATE INFORMATION */
     PULL J                      /* (DATA IS CHECKED FOR VALID     */
     SAY ENTER CHANGE TO SALARY  /*  TYPES)                        */
     PULL CTS
     IF DATATYPE(E,W) & DATATYPE(J,A) & DATATYPE(CTS,N) THEN LEAVE
     ELSE SAY "DATA ENTERED INCORRECTLY--TRY AGAIN"
   END
   UPD.COUNT = "UPDATE EMPLOYEE SET SALARY = SALARY + "CTS,
        " WHERE EMPNO = "E" AND JOB = "J    /* UPDATE COMMANDS ARE */
   EMPLIST = EMPLIST", "E                   /* HELD IN AN ARRAY    */
END

QUEUE RETURN  /* TO ISQL */
IF COUNT = 1 THEN EXIT                       /* NO UPDATES?        */
 ELSE EMPLIST = SUBSTR(EMPLIST,3)            /* REMOVE FIRST COMMA */
QUEUE COMMIT WORK
QUEUE SET AUTOCOMMIT OFF
DO N = 1 TO COUNT-1                        /* PLACE UPDATE COMMANDS */
   QUEUE UPD.N                             /* ON PROGRAM STACK      */
END
QUEUE "SELECT JOB, EMPNO, SALARY FROM EMPLOYEE -"
IF COUNT > 2
 THEN QUEUE "WHERE EMPNO IN ("EMPLIST") -"
 ELSE QUEUE "WHERE EMPNO = "EMPLIST" -"      /* QUERY, FORMATTING, */
QUEUE "ORDER BY JOB, EMPNO"                  /* AND PRINT COMMANDS */
QUEUE FORMAT GROUP JOB
QUEUE FORMAT SUBTOTAL SALARY
QUEUE "FORMAT TTITLE 'SUMMARY OF CHANGES IN SALARY TOTALS'"
QUEUE DISPLAY
QUEUE PRINT COPIES 3
QUEUE END
QUEUE COMMIT WORK
```

Figure 35. ISQL EXEC for Updating the EMPLOYEE Table During an ISQL Session

Here, an ISQL EXEC stacks SQL UPDATE statements that are defined based on the user's responses to prompts for information. The prompts and stacking of updates are done in a loop, so that multiple EMPLOYEE rows can be updated with one execution of the EXEC.

After the user has entered all the updates, the EXEC stacks a query that will display and print the results of the updates.

To process an ISQL EXEC that is designed to run outside of an ISQL session, the user simply enters the name of the EXEC—that is, the user initiates the EXEC while using CMS, without having to start or even know about ISQL.

This type of EXEC would **not** include a RETURN command, because there is no ISQL session to return to. Instead, it must include an ISQL EXIT command as the **last** entry on the stack, and must start ISQL (EXEC ISQL) after the stack entries have been completed.

Figure 36 illustrates an ISQL EXEC designed to be run outside of an ISQL session, which carries out the same function as the one shown in Figure 35 on page 151.

```
/*    XUPDSAL EXEC                                              */

/* THIS EXEC PROGRAM ALLOWS A USER TO PERFORM UPDATES ON         */
/* THE SALARY COLUMN OF THE EMPLOYEE TABLE.  IT IS DESIGNED      */
/* TO BE STARTED WHILE IN CMS (WITHOUT AN ISQL SESSION).  IT     */
/* AUTOMATICALLY EXECUTES ISQL AFTER QUEUING UPDATE COMMANDS AND */
/* AN ISQL EXIT COMMAND ON THE PROGRAM STACK.  COMMANDS ARE ALSO */
/* STACKED TO DISPLAY AND PRINT THE CHANGES MADE.                */

EMPLIST = ""                /* WILL CONTAIN LIST OF EMPNO'S WHOSE */
                            /* TOTAL SALARY HAS CHANGED          */
DO COUNT = 1
   DO FOREVER
     SAY "ENTER EMPNO                      (ENTER 'END' WHEN DONE)"
     PULL E
     IF E = 'END' THEN LEAVE COUNT
     SAY ENTER JOB              /* USER ENTERS UPDATE INFORMATION*/
     PULL J                     /* (DATA IS CHECKED FOR VALID    */
     SAY ENTER CHANGE TO SALARY /*  NUMBERS)                     */
     PULL CTS
     IF DATATYPE(E,W) & DATATYPE(J,A) & DATATYPE(CTS,N) THEN LEAVE
     ELSE SAY "DATA ENTERED INCORRECTLY--TRY AGAIN"
   END
   UPD.COUNT = "UPDATE EMPLOYEE SET SALARY = SALARY + "CTS,
        " WHERE EMPNO = "E" AND JOB = "J    /* UPDATE COMMANDS ARE */
   EMPLIST = EMPLIST", "E                   /* HELD IN AN ARRAY    */
END
IF COUNT = 1 THEN EXIT                      /* NO UPDATES?        */
 ELSE EMPLIST = SUBSTR(EMPLIST,3)           /* REMOVE FIRST COMMA */
QUEUE COMMIT WORK
QUEUE SET AUTOCOMMIT OFF
DO N = 1 TO COUNT-1                        /* PLACE UPDATE COMMANDS */
   QUEUE UPD.N                             /* ON PROGRAM STACK      */
END
QUEUE "SELECT EMPNO, JOB, SALARY FROM EMPLOYEE -"
IF COUNT > 2
 THEN QUEUE "WHERE EMPNO IN ("EMPLIST") -"
 ELSE QUEUE "WHERE EMPNO = "EMPLIST" -"      /* QUERY, FORMATTING, */
QUEUE "ORDER BY JOB, EMPNO"                  /* AND PRINT COMMANDS */
QUEUE FORMAT GROUP JOB
QUEUE FORMAT SUBTOTAL SALARY
QUEUE "FORMAT TTITLE 'SUMMARY OF CHANGES IN SALARY TOTALS'"
QUEUE DISPLAY
QUEUE PRINT COPIES 3
QUEUE END
QUEUE COMMIT WORK
QUEUE EXIT            /* THE ISQL EXECUTION                     */
EXEC ISQL             /* TO EXECUTE THE COMMANDS THAT WERE STACKED */
```

Figure 36. ISQL EXEC for Updating the EMPLOYEE Table Outside of an ISQL Session

Avoid using commands that would result in ISQL issuing a message that requires a response. For example, SET AUTOCOMMIT OFF will cause message ARI7602D to be issued when the EXIT command is entered, and this message requires a

response of either COMMIT or ROLLBACK. Because of the interactive design of ISQL, the response must be entered by the user, and will not be accepted from the command stack.

Programmed Query Functions: If neither stored queries nor ISQL routines are appropriate, you can program query functions. Their primary advantage is application tailoring of the end user interface—that is, the application controls the user syntax for requesting data and the output format for displaying results. Program a query function if an application-specific interface must be provided to end users.

Another advantage is their ability to apply procedural logic. Unlike stored queries which support only a single SQL statement, or ISQL routines which support a fixed sequence of statements, programmed query functions can run different statements or statement sequences based on the results of previous statements or function input.

When designing a programmed query function, you may want to consider using the SQL Dynamic Statement support. With this, the program can translate queries in an application-specific syntax to SQL statements, which are then dynamically compiled and processed. Such a program can provide many query functions with minimal coding.

For even more sophisticated applications, you can use extended dynamic statements to code preprocessors for programming languages that are not supported by the application server. See the *DB2 Server for VSE & VM Application Programming* manual for information.

Report Writing Capabilities

Reports can be produced through ISQL, the DBS Utility, or an application program.

Report Writing Using ISQL

When ISQL terminal users obtain query results through a SELECT statement, they can create reports from them using the FORMAT command. This command provides the following:

- Titles

  Both top and bottom titles can be created. If no top title is specified, a default is provided that consists of the first 100 characters of the SELECT statement that provided the query results. The bottom title defaults to blanks.
- Totals

  Both subtotals and totals can be provided for desired columns.
- Column Separation

  The characters to be used to separate columns can be specified.
- Outlining

  If outlining is specified, successive duplicate values for a desired column are not repeated unless they start a new screen (or a new page for printed reports).
- Column Characteristics

  Users can control such things as:
  - The number of decimal places for numeric columns
  - The width of a column
  - Whether leading zeros are displayed
  - The column heading
  - The inclusion and exclusion of columns.

For more information on formatting reports, see the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual.

To obtain copies of a report, a user enters an ISQL PRINT command. This command allows you to specify the number of copies desired and the output printer class to be used. The printed reports are dated and the pages numbered.

In VSE, if you enter a PRINT command, but the printer is busy, ISQL will send you a message. Along with this message, ISQL will give you the option of:
- Retrying
- Ending the attempt to print
- Queueing your request until the printer is available.

When the printer is free, ISQL displays the message "THE PRINT IS IN PROGRESS".

Note: Some terminals support a copy key that, when pressed, causes a screen image to be printed on the CICS local printer. Such support does not follow any queue protocol and, if you use it while ISQL PRINT is in progress, the screen image may be interspersed with the ISQL PRINT output.

In VM, the results of an ISQL PRINT command are sent to the user's virtual printer. The default print location will be wherever the user normally has his or her output printed. However, the user can change the destination of the files by going into CMS Subset mode, entering the SPOOL and TAG commands to route the output elsewhere, then returning to ISQL (by entering the RETURN command) and entering the PRINT command.

Example: The following command, entered in CMS, will cause the printer output to be sent to the user's virtual reader:

```
CP SP PRT TO *
```

A reader file can read into a CMS file for inclusion in the text report.

Routines can be used to generate reports automatically. This is especially helpful for daily or weekly reports. A routine could issue a SELECT statement, format the output into the desired report, and print the report.

For more information on ISQL report writing, see the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manual.

Report Writing Using the DBS Utility

The DBS Utility provides a limited report-writing capability through its support for SQL SELECT statement processing. The DBS SELECT processing writes the results of a query to the DBS Utility message file (SYSLST print file), with a default of 120 print positions per print line and 60 print lines per print page. These defaults can be changed through the DBS SET command. Refer to the *DB2 Server for VSE & VM Database Services Utility* manual for more information.

Programmed Reports

If an application requires special handling that is not supported by ISQL or the DBS Utility, it may be necessary to write a program to generate a report. For example, an application may need to generate output on special forms in a special format.

You can vary the contents of a programmed report with program variables. Using the Dynamic Statement support in SQL, you could even vary the tables being

reported. When using the Dynamic Statement support, you would use the SQL DESCRIBE statement to obtain information on the data being accessed (for example, column names and column data types).

Programmed Application Capabilities

For complex application requirements that cannot be met by ISQL or DBS Utility facilities, you must code a program using DB2 Server for VSE & VM facilities.

In addition, you can use DB2 Server for VSE & VM extended dynamic statements to code your own preprocessor in assembler language to support other languages that can be mapped to SQL. Extended dynamic statements are explained in the *DB2 Server for VSE & VM Application Programming* manual.

EXECs that Use DB2 Server for VM Facilities

Some application functions can be implemented using a combination of VM EXEC and DB2 Server for VM EXEC facilities. Several useful examples are discussed in the following sections.

Note: If you are using EXECs to invoke applications or to invoke other EXECs that access the application server, refer to "SQLRMEND EXEC" on page 251.

Editing Private Tables

The DBS Utility provides facilities for unloading tables to and loading them from CMS files. While in a CMS file, data can be manipulated by an editor (for example, XEDIT). Users can take advantage of the combination of these capabilities for editing data in their tables.

CAUTION: The following technique is **not** recommended for tables for which multiple users have UPDATE, INSERT, or DELETE privileges. It assumes that only the user doing the editing has update capabilities.

Figure 37 on page 156 shows an EXEC that can be used to edit private tables.

```
/*    EDITTAB EXEC                                              */

/* THIS EXEC PROGRAM USES THE SQLDBSU EXEC TO UNLOAD A USER'S TABLE*/
/* INTO A CMS FILE FOR EDITING WITH XEDIT.  AFTER EDITING, THE USER*/
/* HAS THE OPTION OF REPLACING THE TABLE WITH THE EDITED CMS FILE, */
/* AND THEN MAY HAVE THE TABLE DISPLAYED BY ISQL.  TWO CMS FILES   */
/* MUST PREVIOUSLY HAVE BEEN CREATED WHICH CONTAIN COMMANDS TO     */
/* SQLDBSU;  THEIR FILENAMES MUST BE THE SAME AS THE NAME OF THE   */
/* TABLE, TRUNCATED TO 8 CHARACTERS, AND THE FILETYPES MUST BE     */
/* 'UNLD' AND 'REPL' (EXAMINE CLOSELY THE EXAMPLE GIVEN FOR THE    */
/* EMPLOYEE TABLE.)                                              */

SIGNAL ON ERROR
SAY WHICH TABLE WOULD YOU LIKE TO EDIT?
PULL TNAME
FN = STRIP(LEFT(TNAME,8))
"STATE" FN "UNLD"                        /* VERIFIES EXISTENCE OF */
"STATE" FN "REPL"                        /* DBSU CONTROL FILES    */

"FILEDEF WORKFILE DISK" FN "TABLE (LRECL 80 RECFM FBA"
"EXEC SQLDBSU IN("FN "UNLD) PR(PRINTER)"          /* UNLOAD TABLE */
"XEDIT" FN "TABLE"                               /* FOR EDITING  */

SAY "DO YOU WANT TO REPLACE THE" TNAME "TABLE? (Y OR N)"
PULL ANSWER1
IF ABBREV(NO,ANSWER1,1) THEN EXIT
"EXEC SQLDBSU IN("FN "REPL) PR(PRINTER)"      /* TABLE IS REPLACED */

SAY WOULD YOU LIKE TO DISPLAY THE NEW TABLE? (Y OR N)
PULL ANSWER2
IF ABBREV(NO,ANSWER2,1) THEN EXIT

QUEUE "SELECT * FROM" TNAME         /* MOVE ISQL COMMANDS INTO THE */
QUEUE DISPLAY                       /* PROGRAM STACK               */
QUEUE END
QUEUE EXIT
EXEC ISQL
EXIT  /* END OF PROGRAM */

ERROR:                                        /* ERROR HANDLING  */
SAY "UNEXPECTED EDITTAB TERMINATION      RETURN CODE:" RC,
        "   LINE:" SIGL
```

Figure 37. Example EXEC for Editing a Private Table

Here, the EDITTAB EXEC prompts the user for the name of the table to be edited (only simple names are accepted), then uses that name to verify the existence of DBS command files needed to support editing of the table. The work file used for the CMS file version of the table is then defined (in the FILEDEF WORKFILE command). The DBS Utility (SQLDBSU) is then initiated to unload the table to the work file. Once the table has been unloaded, XEDIT is initiated to edit the work file.

On completion of the XEDIT session, the user is asked if the table is to be replaced in the database by its edited version. If the answer is yes, the DBS Utility is initiated to perform the REPLACE operation.

Finally, the EXEC asks the user if the new version of the table should be displayed. If the answer is yes, the table is displayed using ISQL.

For the EDITTAB EXEC to work, two DBS command files must be established for each table that is to be supported. Figure 38 and Figure 39 show the DBS command files needed to edit a user's version of the EMPLOYEE sample table.

Note: These examples assume that the user's version of the EMPLOYEE table, *userid*.EMPLOYEE, has already been created.

The command file in Figure 38 unloads the EMPLOYEE table to a file that has been defined as WORKFILE. (This is the file defined in EDITTAB as a CMS file with a file name using the first eight characters of the table name, and a file mode of TABLE.) The information following the SELECT statement identifies the location in the output file (WORKFILE) where the data for the columns in the select-list should be placed.

```
COMMENT 'EMPLYEE UNLD A'
COMMENT 'DATAUNLOAD JOB FOR EDITING A USERS EMPLOYEE TABLE'
DATAUNLOAD
  SELECT * FROM EMPLOYEE ORDER BY EMPNO;
    EMPNO  5-10 CHAR
    FIRSTNME 12-23 CHAR
    MIDINIT 25 CHAR
    LASTNAME 27-41 CHAR
    WORKDEPT 43-45 CHAR
    PHONENO 47-50 CHAR
    HIREDATE 52-61 CHAR
    JOB 63-70 CHAR
    EDLEVEL 72-73 CHAR
    SEX 75 CHAR
    BIRTHDATE 77-86 CHAR
    SALARY 88-96 CHAR
    BONUS 98-106 CHAR
    COMM 108-116 CHAR
OUTFILE (WORKFILE)
```

Figure 38. DBS Unload Command File for Editing EMPLOYEE Table

```
COMMENT 'EMPLYEE REPL A'
COMMENT 'DATALOAD JOB FOR REPLACING AN EDITED EMPLOYEE TABLE'
DELETE FROM EMPLOYEE;
DATALOAD TABLE (EMPLOYEE)
    EMPNO  5-10 CHAR
    FIRSTNME 12-23 CHAR
    MIDINIT 25 CHAR
    LASTNAME 27-41 CHAR
    WORKDEPT 43-45 CHAR
    PHONENO 47-50 CHAR
    HIREDATE 52-61 CHAR
    JOB 63-70 CHAR
    EDLEVEL 72-73 CHAR
    SEX 75 CHAR
    BIRTHDATE 77-86 CHAR
    SALARY 88-96 CHAR
    BONUS 98-106 CHAR
    COMM 108-116 CHAR
INFILE (WORKFILE)
COMMIT WORK;
```

Figure 39. DBS Command File for Replacing Edited EMPLOYEE Table

The command file in Figure 39 deletes the existing rows of the user's EMPLOYEE table, and loads the edited WORKFILE version of the table into it. The information

between the DATALOAD table statement and the INFILE statement identifies the columns in the table to be loaded with the data from the input file at the specified locations. All the EMPLOYEE table columns here will be loaded with data from the WORKFILE input file. For example, data in positions 5 to 10 of the file will be loaded into the EMPNO column.

Editing Routines

Another variation of the EDITTAB EXEC would be an EXEC that edited only a portion of the user's table. To do this, the DATAUNLOAD and DATALOAD command files must be selective about which rows are unloaded and replaced. This can be done using a subquery on the DATAUNLOAD, and a WHERE clause on the DELETE statement.

Figure 40 shows an example where the EXEC unloads an ISQL routine from the user's ROUTINE table, invokes XEDIT on the unloaded rows, and gives the user the option of reloading the edited routine back into the ROUTINE table.

```
/* EDITROUT EXEC                                                    */
SIGNAL ON ERROR
SAY WHICH ROUTINE WOULD YOU LIKE TO EDIT?
PULL RNAME
"STATE" RNAME "UNLD"
"STATE" RNAME "REPL"
"FILEDEF WORKFILE DISK" RNAME "TABLE (LRECL 100 RECFM FBA"
"EXEC SQLDBSU IN("RNAME "UNLD) PR(PRINTER)
"XEDIT" RNAME "TABLE"
SAY "DO YOU WANT TO REPLACE THE" RNAME "ROUTINE? (Y OR N)"
PULL ANSWER1
IF ABBREV(NO,ANSWER1,1) THEN EXIT
"EXEC SQLDBSU IN("RNAME "REPL) PR(PRINTER)"
SAY WOULD YOU LIKE TO DISPLAY THE ROUTINE? (Y OR N)
PULL ANSWER2
IF ABBREV(NO,ANSWER2,1) THEN EXIT
QUEUE "SELECT * FROM ROUTINES WHERE NAME='"RNAME"' -"
QUEUE "ORDER BY SEQNO"
QUEUE DISPLAY
QUEUE END
QUEUE EXIT
EXEC ISQL
EXIT
ERROR:
SAY "UNEXPECTED TERMINATION OF ROUTINE EDIT     RETURN CODE:" RC,
      "     LINE:"  SIGL
```

Figure 40. Example EXEC for Editing Routines

```
  COMMENT 'XMPLROUT UNLD A'
  DATAUNLOAD
    SELECT NAME, SEQNO, COMMAND FROM ROUTINE
      WHERE NAME = 'XMPLROUT';
      NAME    1-8    CHAR
      SEQNO   10-15  CHAR
      COMMAND 17-95  CHAR
    OUTFILE (WORKFILE)
```

Figure 41. Example DBS DATAUNLOAD Command File for Editing Routine XMPLROUT

```
COMMENT 'XMPLROUT REPL A'
DELETE FROM ROUTINE
  WHERE NAME = 'XMPLROUT';
DATALOAD TABLE (ROUTINE)
  NAME    1-8   CHAR
  SEQNO   10-15 CHAR
  COMMAND 17-95 CHAR
INFILE (WORKFILE)
COMMIT WORK;
```

Figure 42. Example DBS Command File for Replacing Routine XMPLROUT

Application Development Capabilities

Complex applications that are coded as programs (as opposed to DBS Utility input files or ISQL sessions) could involve many programs that operate on many tables. This section discusses how to make large-scale application development easier.

Data Prototyping

The DB2 Server for VSE & VM system can be used by application developers to prototype data designs and implement them during the application development process. In particular, in DB2 Server for VSE, the ability to dynamically CREATE, ALTER, and DROP tables from an online, interactive environment allows a developer to experiment with different design alternatives. A developer can then exploit the DB2 Server for VSE & VM catalog tables and explanation tables for documentation and analysis of data designs.

The DB2 Server for VSE & VM facilities that should be considered for data prototyping activities are identified in the following sections.

Modeling Data Designs: ISQL or the DBS Utility can be used to enter table, view and index definitions for validating and testing data design. The interactive definition through ISQL gives the developer direct feedback on definitional errors. This feedback not only addresses syntax errors, but also addresses data mapping errors in view definitions.

Furthermore, if SQL definitional commands are entered through ISQL, these commands may be saved as stored queries. By saving the definitional commands, they can be recalled, modified and rerun as needed.

If you are developing your system under VSE/ICCF, you can save definitional statements by storing them in VSE/ICCF files that are used as input (SYSIPT) to the DBS Utility.

If you are developing your system under CMS, you can save definitional statements by storing them in CMS files that are used as input (SYSIN) to the DBS Utility.

Generation/Loading of Test Data: Tables created for data design purposes can be loaded with test data using any one of several facilities, depending on the source of test data and the availability of machine readable versions of the data.

If data exists on a sequential file, or can be put into a sequential file, test data can be loaded using the DBS DATALOAD command.

If the data does not exist in machine readable form, or cannot be readily converted to a sequential format, it may be necessary to enter the data by hand. This could be done using the ISQL INPUT command or by building a file for input to the

DBS Utility. In DB2 Server for VM the input file to the DBS Utility is a CMS file. In DB2 Server for VSE the input file to the DBS Utility is a VSE/ICCF file.

If the data can be found in existing DB2 Server for VSE & VM tables in the application development database, then data can be copied using the SQL INSERT statement.

If the data can be found in existing tables in another database, the data can be moved using either DBS UNLOAD and RELOAD or DBS DATAUNLOAD and DATALOAD commands. The UNLOAD/RELOAD commands allow easy movement of data on a table or dbspace level. DATAUNLOAD/DATALOAD lets you be more selective in what you want to unload, and where you want to load it. That is, rather than move an entire table, you can use DATAUNLOAD/DATALOAD to move only certain columns of certain rows of a table.

Design Documentation and Analysis: The catalog tables form a base for design documentation in as much as the catalog tables can be queried and used to generate reports. In addition to containing the base information from the SQL definitional commands, the catalog tables contain useful statistical information and dependency information.

You can analyze how a given design will perform by using the explanation tables and the SQL EXPLAIN statement. The EXPLAIN statement can be issued in a program, from an ISQL terminal, by way of the DBS Utility, or through an application program. It lets you get information about the structure and execution performance of other SQL statements (especially the SELECT statement). Naturally, execution performance is affected by the data design.

In addition to the EXPLAIN statement, you can get an idea of how well a given SELECT statement performs by using the ISQL query cost estimate. The query cost estimate is displayed before the result of a SELECT statement is displayed. It is also displayed at the end of every SELECT result in ISQL.

The query cost estimate is a relative number (not expressed in real units) that represents an estimate of the resources used to process the statement. The query cost estimate displayed in ISQL is not the same number that can be obtained by using EXPLAIN. The cost estimate displayed by EXPLAIN is the number that is used internally, while the number displayed by ISQL is the internal number divided by 1000. This makes the query cost estimate more significant to a terminal user.

Prototyping Application Function

Application function can be prototyped using ISQL or DBS facilities for testing and debugging SQL statements to be used by the application.

Using Stored Queries to Test SQL Statements: The stored queries support can be effectively used to develop SQL statements to be used in an application. The stored queries could be developed for a test database. By using parameterized stored queries, you can simulate the use of program variables and test the results of your SQL statement against various input cases.

Using ISQL Routines to Test SQL Functions: You can develop logical sequences of SQL statements by using the ISQL routine support. Different routines would be developed for different paths through the application logic. Again, parameterized

stored routines can be effectively used to simulate program variables and test the functional results of the application path against various input cases.

You can use the ISQL SET RUNMODE command to aid in testing (and, perhaps, correcting) the application logic in routines. SET RUNMODE can be coded in the routine or issued from the terminal. It lets you stop or continue the processing of an ISQL routine when an error is encountered.

Using the DBS Utility to Test SQL Functions: You can use the DBS Utility to try out SQL statements or sequences of SQL statements. Note however, you cannot use it to process parameterized SQL statements or SQL statements having host variables.

Using the DBS Utility to test SQL statements in VSE has the advantage of keeping the SQL statements in a VSE/ICCF file that can be modified. When testing is complete, you can include the VSE/ICCF file in a source code file. If you are using the DBS Utility under VSE/ICCF, use the VSE/ICCF editor to modify the command sequence for each test run.

Using the DBS Utility to test SQL statements in VM has the advantage of keeping the SQL statements in a CMS file that can be modified. When testing is complete, the CMS file can be included in a source code file. Using the DBS Utility under CMS, you would have to modify the command sequence using a CMS editor (XEDIT, for example) for each test run.

As in ISQL, the DBS Utility also provides error handling. Issue the command SET ERRORMODE to tell the utility how (or if) it is to process SQL and DBS commands after an error has occurred.

Code Development

For development of application code, VSE/ICCF and CMS provides an interactive environment in which to build source code files, to run the DB2 Server for VSE & VM preprocessors, to run the high-level-language compilers, and to test batch (or VSE/ICCF) applications. In VSE, tests of CICS transactions must be run under the CICS subsystem.

Building Source Code Files: Using either the VSE/ICCF editor or a CMS editor, developers can interactively build and edit source statements for their programs. Developers can use the DBS Utility to test SQL statements. They can copy tested statements into the source file, and modify them for the appropriate programming language syntax.

If many applications are to use the same host variables or the same SQL statements, the developers should consider using the SQL INCLUDE statement, which causes the preprocessors to include source lines from other source members (VSE) or CMS files (VM). For example, a developer can place a lengthy SELECT statement here and use that query in many programs by coding SQL INCLUDE statements.

Preprocessing Programs under Development: The DB2 Server for VSE preprocessors can be run under VSE/ICCF, using VSE/ICCF files for source code input, or under CMS, using CMS files for source code input. The printed output from the preprocessors, SYSLST for VSE and SYSPRINT for VM, can be directed under VSE/ICCF to VSE/ICCF files, or under CMS, to CMS files for developer review from the terminal. Similarly, the punch output, SYSPCH for VSE and SYSPUNCH for VM, from the preprocessors can be directed under VSE/ICCF to

VSE/ICCF files or under CMS, to CMS files, for input to the appropriate compiler. The preprocessors integrate any external source lines from INCLUDE statements. Use of the INCLUDE statement does not cause more compilation steps.

When developing a program with embedded SQL statements, run the preprocessors with a CHECK option. Under this option the preprocessor produces diagnostics on the SQL in the program, but does not create a package or compiler input. Therefore initial code development and debugging can be done on just a skeleton of the final program.

When preprocessing programs under development, application developers can back up packages that they create with the DBS Utility UNLOAD PROGRAM command. For information on this command, see the *DB2 Server for VSE & VM Database Services Utility* manual.

Testing Application Code under VSE/ICCF: To test user SQL programs the developer must preprocess, compile, and link-edit the program as a multiple user mode batch application program. This can either be done under VSE/ICCF, or as a normal VSE batch job.

Once an application has been preprocessed, compiled, and link-edited, normal VSE/ICCF procedures for application execution can be used. The only DB2 Server for VSE requirement for program execution is that the VSE/ICCF control statement "/OPTION GETVIS=AUTO" must follow the "/LOAD" statement. The program only needs to be re-preprocessed if the SQL statements that the program runs are modified.

Testing Application Code Under CMS: To test user SQL programs under CMS, the application developer would preprocess, compile, and link-edit the program as a multiple user mode application program, using the usual CMS commands.

Following this, CMS commands for application execution can be entered (for example, START and RUN).

CMS Subset Considerations

If you develop a DB2 Server for VM application that invokes CMS Subset, be sure to tell users not to invoke any commands, programs, or EXECs that access the application server while in CMS Subset mode. (The results would be unpredictable and error conditions could be generated.) This also applies if they invoke CMS Subset from ISQL.

Application Database Considerations

The following sections discuss the implications that the various types of application implementations have on database design.

Database Support for Application Development

When applications are being developed, not all of the data is already predefined. You will therefore need to set up your database to support both data that exists in a predefined state, and data that is still under development. For the latter, you should consider establishing both PUBLIC and PRIVATE dbspaces specifically defined for application development purposes.

PRIVATE Dbspaces in Application Development

Application developers and database designers will need their own PRIVATE dbspaces for prototyping data designs and functions on various data designs.

These are better than PUBLIC dbspaces for this activity, because they provide an environment of less concurrency and no deadlocks. They also have the advantage of being user-controlled, so application developers need not worry about others altering the data in their test tables.

Such dbspaces must only be large enough to support sample data in the tables. Because they are used primarily for functional feasibility testing, they do not have to support large versions of the tables.

PUBLIC Dbspaces in Application Development

PUBLIC dbspaces are required to model the final database implementation: the final testing stages and performance testing of applications and data organizations. They would probably represent the actual production environment, as they allow a greater concurrency and are DBA-controlled. Tables stored here would hold a larger, more representative sampling of the data.

Database Support for Query/Report Writing

Queries and report writing also have unique database design requirements. In particular, the needs of query users for private storage of their data, queries, and routines must be considered.

Private Query User Data

Many query users will want to be able to store their own private data. To support this, you need to set up space in the database: how you do so will depend on how you want to control space usage and table creation. Some variations on this are described below.

User Control of Own Data: You can enable query users to define and control their own data by giving them RESOURCE authority, which lets them create tables in the database. For more information see "Granting Authorities" on page 90. They will also need PRIVATE dbspaces to hold their tables. Users with RESOURCE authority can issue their own ACQUIRE DBSPACE statements; however, you will probably prefer to do this for them. (See "Identifying Dbspace Requirements" on page 20.)

Having One User Control Data for a Group: If you do not want to give all query users RESOURCE authority, you could set up a PUBLIC dbspace that would support the data requirements of a whole group, and give just one user RESOURCE authority to handle the data requirements of the group.

Having the DBA Control All Data: If you need to tightly control or centralize control of database usage, you (or someone with DBA authority) can establish PRIVATE dbspaces for individual users and PUBLIC dbspaces for common data requirements, but create all tables yourself and restrict access to those tables to certain users only.

Giving Users Their Own Dbspaces: Users who do not have RESOURCE authority can still create tables in PRIVATE dbspaces that the DBA has acquired for them. This still allows the DBA to control how much space each user has in the database, but gives users the freedom to create whatever tables they choose within that space. This technique is sometimes called "create table authority."

Storage of ISQL Routines

Query users who want to develop their own ISQL routines will need to have a ROUTINE table somewhere in the database. Creating this table is typically done by the DBA when enrolling a new query user on the system. (See "Adding a New User" on page 79.)

Users who have their own PRIVATE dbspaces can create their own ROUTINE tables there. If a user's routines are to be shared by others, then this table should be created in a PUBLIC dbspace instead, and access to it established through views (rather than duplicating the table or having the other users qualify the name of the routine by a user ID).

If users are invoking ISQL to access a non-DB2 Server for VSE & VM application server and you require a master ISQL profile routine, then you must create a table called SQLDBA.ROUTINE and store the master routine in this table. See the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals for details on setting up a routine table.

Note: Access to an application server using the DRDA protocol is only possible if the Distributed Relational Database Architecture (DRDA) facility has been installed on the application requester and if the application server supports IBM's implementation of the DRDA protocol.

System Dbspace Considerations

A final requirement for supporting a query/report writing environment is to define data for three dbspaces: "PUBLIC".ISQL, "PUBLIC".HELPTEXT, and "PUBLIC".SAMPLE.

"PUBLIC".ISQL Dbspace: This dbspace contains the SQLDBA."STORED QUERIES" table, which holds the queries stored by ISQL users. Because stored queries cannot be shared, users must have their own copy of any stored query they need to run. One or more standard stored queries may be established for each new user; in addition, some users may also have application programmer-developed stored queries established for them. Thus, this table may contain redundancy.

If an installation has many stored queries, the "PUBLIC".ISQL dbspace may run out of space. If this happens:

1. Unload the dbspace using the DBS Utility, thus saving its stored queries and routines in an external file.
2. Drop the dbspace and acquire a bigger one with the same name.
3. Reload the new dbspace with the data that was unloaded from the previous version.

"PUBLIC"."HELPTEXT" Dbspace: If the HELP text has been installed, this dbspace contains the SQLDBA.SYSTEXT1, SQLDBA.SYSTEXT2, and SQLDBA.SYSLANGUAGE tables, which hold the information displayed in response to an ISQL HELP command. If you plan to expand the text or topics covered in the HELP tables, you will need to increase the size of this dbspace. See "Making the HELPTEXT Dbspace Larger" on page 143.

"PUBLIC"."SAMPLE" Dbspace: This dbspace contains the sample tables provided with the DB2 Server for VSE & VM product: SQLDBA.EMPLOYEE, SQLDBA.DEPARTMENT, SQLDBA.PROJECT, SQLDBA.ACTIVITY, SQLDBA.EMP_ACT, SQLDBA.PROJ_ACT, SQLDBA.CL_SCHED and SQLDBA.IN_TRAY.

Application Implementation Considerations

The considerations regarding the implementation of applications are discussed here.

VSE Batch/Interactive Application Considerations

The considerations pertinent to user-written application programs for the batch/ICCF environment and to DBS Utility applications are security, recovery and error handling.

Batch/ICCF Application Security

DB2 Server for VSE data protection applies to programs written for batch and VSE/ICCF execution (and to the data itself). In particular, when a user preprocesses a program, the database manager checks the authority and privilege of that user on tables and views used in the program. When a user runs a preprocessed program, the database manager checks only for RUN privilege of that user on the program.

When a program is preprocessed, users with the appropriate data authority for the program functions must supply a user ID and password. Users are verified by the supplied password, and their authority is checked for each SQL request embedded in the program. On successful completion of the preprocessor job, a user becomes the owner of the application program, and can control who else can run it by issuing a GRANT RUN statement.

Note: In a batch/interactive environment, a GRANT statement is typically entered through a DBS Utility execution. For program execution, the batch and VSE/ICCF applications must be written to establish a user connection to the DB2 Server for VSE application server through the SQL CONNECT statement. This statement establishes the user of the program and checks that user's authority to run the program and to perform any interpretive SQL functions in the program.

For example, security-sensitive applications can be written to require that the user ID and password of the program runner be supplied through control statements or terminal input at execution time. The programs should be written to read this information into the host program variables referenced in a DB2 Server for VSE CONNECT statement.

You can bypass this security facility by writing the application so that it supplies the user ID and password independent of the actual user of the application. If you do so, you must code the CONNECT statement in the application and grant RUN authority to the user IDs to be generated.

In general, you can use program authorization as a means of controlling access to data. If the end user of the application has access to the data only through specific application programs, the user can do only what the application is programmed to do.

Batch/ICCF Recovery

All batch/ICCF programs should explicitly issue COMMIT WORK and ROLLBACK WORK statements as required, rather than relying on the implicit COMMIT and ROLLBACK functions of the database manager. The rules determining whether to do an implicit COMMIT WORK or an implicit ROLLBACK WORK are rather complex. By coding explicit COMMIT WORK and ROLLBACK WORK statements, you can determine what work is done by a batch application from the last statement completed.

Batch/ICCF Error Handling

In general, if a batch/ICCF application is terminated abnormally, the database manager backs out all uncommitted changes. If it terminates normally, the

database manager commits changes not explicitly committed by the application. An application program can be coded to handle negative SQLCODEs by using SQL WHENEVER statements, as described in the *DB2 Server for VSE & VM Application Programming* manual.

DBS Utility Application Security

The DBS Utility input must include an SQL CONNECT statement before any other SQL statement or UNLOAD, RELOAD or DATALOAD commands are issued. The only exception to this rule is when the DBS Utility is called by a user application that has already issued an SQL CONNECT statement.

A DBS input (SYSIPT) file can contain multiple SQL CONNECT statements. This capability can be used to write one DBS input file that performs operations for multiple users. The operations to be performed for any one user are preceded by an SQL CONNECT statement.

DBS Utility Application Recovery

The DBS Utility applications can control commit processing through the appropriate use of the DBS SET AUTOCOMMIT command and SQL COMMIT WORK and ROLLBACK WORK statements. By setting AUTOCOMMIT ON, DBS will automatically issue an SQL COMMIT WORK after each SQL or DBS command (except for certain statements that imply commit processing is not appropriate, like ROLLBACK WORK). By setting AUTOCOMMIT OFF, no commit processing will be done unless explicitly requested by an SQL COMMIT WORK statement or when the input command file is exhausted.

For batch processing, you typically run with AUTOCOMMIT set OFF, so commit points are explicitly identified by SQL COMMIT WORK statements. This is the default for DBS processing.

When running DBS in an interactive fashion (under VSE/ICCF with /DATA INCON specified), you should run with AUTOCOMMIT set ON. If you run with AUTOCOMMIT set OFF, shared data is not available to other users while you are thinking about command responses or entering commands, unless the other users are using isolation level UR.

DBS Utility Application Error Handling

DBS terminates execution of commands in the input (SYSIPT) file and performs a ROLLBACK WORK if it encounters an error on any of the commands. However, DBS will read all the input records and provide diagnostics for the remaining commands. In addition, you can include SET ERRORMODE OFF commands to cause DBS to stop processing the input in error mode (that is, resume execution of commands in the input file).

The SET ERRORMODE OFF capability is useful for execution of independent command sequences in the same input file. Each independent sequence of commands would be preceded by a SET ERRORMODE OFF command.

Another use of the SET ERRORMODE command is when running DBS under VSE/ICCF in conversational mode (/DATA INCON). In this case, you should use the SET ERRORMODE CONTINUE command. If a normal SQL error is encountered on any command entered, the DBS utility processes subsequent commands from the terminal user. It goes into error mode processing only if the error is fatal. This saves the terminal user from having to enter SET ERRORMODE OFF every time a minor mistake is made.

Online CICS/VSE Transaction Considerations

Online Application Security

Security design in an online (CICS) transaction processing environment should consider the facilities of the CICS subsystem as well as those offered by the DB2 Server for VSE application server. In particular, the CICS subsystem provides facilities for performing user verification (signon), and for controlling user authority to run CICS transactions. SQL programs written for execution in the CICS programming environment can be designed to take advantage of these facilities.

User Identification and Verification: User identification and verification for CICS SQL transactions can be handled in one of the following ways:

- CICS Signon

  The CICS SQL transactions do not have to contain SQL CONNECT statements. If a CONNECT statement is not present in the transaction, the DB2 Server for VSE online support attempts to obtain the CICS signon userid using the EXEC CICS ASSIGN command. The user ID of the program user is assumed to be this signon ID.

- DB2 Server for VSE CONNECT

  If your CICS users do not go through a signon process for access to CICS transactions, user identification and verification can still be accomplished through the SQL CONNECT statement in the individual transaction programs. However, the transaction would have to obtain a user ID and password in order to issue a CONNECT statement.

  Note: This does not apply in a DRDA environment.

- No user identification/verification

  For the online (CICS) environment, you can choose to run without requiring any user identification or verification, by treating all CICS users as though they had the same user ID and authority. To do this, you would identify the default CICS user ID when starting the DB2 online support (see "CICS Transaction Environment" on page 101).

  Note: This does not apply in a DRDA environment.

You can design each CICS transaction to handle user verification and identification differently. Some may require the user to sign on to the CICS subsystem, others may issue SQL CONNECT statement, and yet others may assume the default user ID for CICS users.

Note: In cases where the CICS subsystem does user verification or the user runs under the default user ID, it is not necessary to have the user defined to the DB2 Server for VSE application server through the GRANT statement (CONNECT authority).

Online Application Recovery

Application recovery processing for CICS SQL transactions is coordinated between the CICS subsystem and the DB2 Server for VSE database manager. In particular, a CICS SYNCPOINT request causes an SQL COMMIT WORK to be issued to commit table information as well as CICS information. Similarly, a CICS SYNCPOINT ROLLBACK request causes an SQL ROLLBACK WORK to be issued. The reverse is also true: a COMMIT WORK causes a CICS SYNCPOINT to be issued and an SQL ROLLBACK WORK causes a CICS SYNCPOINT ROLLBACK to be issued.

In addition, CICS end-of-task processing is coordinated with the DB2 Server for VSE database manager to assure that transaction processing is properly committed or rolled back, depending on the conditions under which the transaction ended.

For supporting recovery processing for CICS SQL transactions, the CICS subsystem **must** be generated with the Dynamic Transaction Backout Program (DBP parameter), and individual transactions **must** be installed with the CICS DTB=YES option.

Pseudoconversational Transactions

Pseudoconversational transactions must not be run on the same terminal with ISQL while ISQL has "timed out".

Application Development Considerations

Loading Data into Test Dbspaces

You can load test data in any of these ways:

- DBS Utility UNLOAD/RELOAD

  Live data can be unloaded and then reloaded back into the system, but directed at the test dbspace. The tables can be created new (using the NEW option of the RELOAD command); or, if the tables already exist, then all rows can be deleted and the unloaded data inserted using the PURGE option of the RELOAD command. If an application development environment exists where the test data is on a separate test database, then DBS UNLOAD/RELOAD can be used to load data from one database to another.

- INSERT with subselect

  Live data can also be loaded into a table by using the INSERT with subselect:

  ```
  INSERT INTO TESTTABLE
      SELECT * FROM USERID.LIVETABLE
      WHERE ...
  ```

  This approach to loading test data has the advantage of using a WHERE clause for defining a sample from the live data rather than the entire table. An INSERT with subselect can be entered through ISQL, the DBS Utility, or an application program. An INSERT with subselect can be used to convert data from one data type to another; the specific limits on data type conversion depend on the number of conversions and the data types involved. Refer to the *DB2 Server for VSE & VM Application Programming* manual for the general restrictions on data type conversions.

- DBS Utility DATAUNLOAD/DATALOAD

  The DBS Utility DATALOAD command may be used to input test data for new tables. The input to the DATALOAD command is from SYSIPT or sequential (SAM) files in VSE, from SYSIN or sequential (CMS) files in VM. A subset of all the data on the input sequential file can be loaded by using the "IF POS" clause of the DATALOAD command. For example, suppose that on an input sequential file containing customer information, the telephone number data is in positions 28-39 and positions 28-30 contain the area code. You could then load just the 555 area codes into the table TESTCUST by specifying the following DATALOAD command format:

  ```
  DATALOAD TABLE (TESTCUST)      IF POS (28-30)='555'
  ```

  ⋮

The test table would contain a subset of the actual data that the application will use.

You can get even more selectivity by using the DATAUNLOAD command to create the sequential file. This is especially useful if the data exists in tables, but not input files.

As the DATAUNLOAD command incorporates an SQL SELECT statement, you can be highly selective about what data you wish to unload. Furthermore, because the DATALOAD command can be used to reload the data, you can significantly restructure the data when you load it. That is, DATAUNLOAD/DATALOAD is not restricted to a table-to-table or dbspace-to-dbspace data movement. For example, you can unload data generated from a subquery, and then load only a portion of the result into a completely different table. This facility is useful for rapidly getting data into a new design prototype.

Use of Synonyms in Application Development

To simplify coding and testing of SQL statements that will eventually reference the live data, a developer may use the SYNONYM capability. Under the user ID established for a developer, synonyms would be defined so table references would translate to test tables when preprocessed under the developer's user ID.

For example, an application needs to be written that will access the PAYROLL table. The fully qualified table name is LOCALDBA.PAYROLL, having been created for the user ID LOCALDBA. A developer, with *userid* = DEV, has a temporary version of the payroll table called DEV.TESTPAY. Because the SQL statements refer to the table name PAYROLL, the developer creates the name PAYROLL as the synonym for TESTPAY:

```
CREATE SYNONYM PAYROLL FOR TESTPAY
```

Now all references to PAYROLL made from userid DEV translate to TESTPAY. When it comes time to switch to the live data, the program will be preprocessed under the userid of the creator of the PAYROLL table (LOCALDBA in this case), so that the program will access the PAYROLL table no matter who runs it.

Note: An exception to the above commands are applications that require Dynamic Statement Support (including Extended Dynamic Statements). For those dynamic statements, table references are translated based on the userid of the user that runs the program (the userid specified in the CONNECT statement). Execution of any test program against live data can be prevented by not granting run authority to anyone who does not have the appropriate synonyms defined.

Testing SQL Statements

Using ISQL and Stored Queries

Before actually coding an application, the programmer may test/develop SQL statements to be embedded in the program by using ISQL against test data. The programmer would develop a set of SQL statements using the stored query facilities of ISQL. As each statement is formed, it would be run against the test data to verify expected results. Syntax and execution errors will be caught and error messages returned. The HELP facility of ISQL could be used to obtain detailed error descriptions and SQL statement descriptive information. For user

logic errors on non-query statements (such as INSERT or UPDATE), the programmer can issue SELECT statements to inspect the effects of the tested statements.

Maintaining Database Consistency Under ISQL

To maintain a consistent state of the test database when using non-query statements, the programmer will want to issue SET AUTOCOMMIT OFF from the ISQL terminal, so that any changes that the test statements may make to the test database can be undone with a ROLLBACK WORK.

Using ISQL Stored Queries for Testing SQL Statements

To place an SQL command in the stored queries table without executing it first, the programmer should use the ISQL HOLD and STORE commands under AUTOCOMMIT ON mode. For example:

```
HOLD DELETE FROM PAYROLL WHERE NAME = 'SMITH'
STORE DELETE1
```

The HOLD command will place the command in the SQL command buffer of ISQL, but will not run it. Then the STORE command will place the contents of the SQL command buffer into the "STORED QUERIES" table. Once the command is in the "STORED QUERIES" table, the programmer can run it while controlling his own logical unit of work (under SET AUTOCOMMIT OFF), so that the changes done by the command can be rolled back.

Using ISQL Routines to Test SQL Statements

As each command is corrected and verified, it can be stored away as parameterized stored SQL command in a ROUTINE table. The programmer would use stored command parameters where the program will have program variables. The commands can be placed in the ROUTINE table in the same logical order that they will be run in the application program. In this manner a prototype will be created that will demonstrate sample application usage. End users can then see the proposed system in operation before it is coded. Design flaws can be more easily corrected at this early phase.

Note: Stored queries and synonyms cannot be shared, but routines can be shared. You can run another user's routine if you have obtained the SELECT privilege (through a GRANT command) on that user's ROUTINE table. Care must be taken in running another user's routines however, because any stored SQL commands or synonyms used in a routine will not be recognized unless you have also defined them yourself.

Checking Application Code

Using the Preprocessor CHECK Option

After debugging and testing the SQL commands on ISQL, the application programmer would then code the application. Having developed the source program with embedded SQL commands, the next step is to run the program through the appropriate DB2 Server for VSE & VM preprocessor. If the programmer is unsure of the SQL commands embedded in the program, he can run the preprocessor with the CHECK option. The SQL commands will be preprocessed and error messages will be output to SYSLST in VSE and SYSPRINT in VM, but a package is not created and no modified source will be produced. Running the preprocessor without the CHECK option will generate a package and the modified source to be used as input to the desired compiler.

Use of ROLLBACK WORK During Application Execution

After the program has been preprocessed and compiled, the final step in the testing cycle would be execution against the test data. To ensure a consistent test database the application programmer should place a ROLLBACK WORK statement in his application that will undo any changes that the program may make during execution before the program terminates. This ROLLBACK WORK statement may be left in the application for the first few runs on the live data. Once the program is operating correctly on the live data, the ROLLBACK WORK statement can be removed (or replaced with a COMMIT WORK statement).

Query/Report Writing Considerations

User Identifiers (Userids) for Query Users

Each query user should be given a unique user identifier and CONNECT authority on the DB2 Server for VSE & VM application server using GRANT statements, as described under "Adding a New User" on page 79. Multiple users can use the same DB2 Server for VSE & VM userid, but this can result in conflicts between the users' access to the system and to data.

ISQL users should be careful if there is more than one user using the same userid. In particular:

- Stored queries should be stored with names that identify the owner. This can be done by using the owner's initials as a prefix to the name.
- Multiple users with the same userid will experience severe contention if they try to update (insert or delete) data in a PRIVATE dbspace owned by their common userid. In such cases, PRIVATE dbspaces should not be used unless the access to the data is read only.
- When multiple users use the same userid, the DB2 Server for VSE & VM security facilities cannot distinguish the individual users. All users using a common userid will share the same access privileges to the database.
- Only one PROFILE ROUTINE can exist for each unique userid. If multiple users (using the same userid) require different profile routines, they can create unique routines (again, perhaps appending their initials to the routine name). This unique routine can then be run either as part of ISQL signon or at any time after signon.

Application Independence with CMS Work Units

Applications that use multiple CMS Work Units can:

- Start a logical unit of work.
- Invoke other application programs in new CMS Work Units.
- Run these application programs independently of one another. When one program commits or rolls back work, it does not affect the work in other CMS Work Units.
- Access a different application server in each CMS work unit. The logical unit of work can be on the same application server or on different application servers. The application server can be a DB2 Server for VM or non-DB2 Server for VM.

For more information on CMS Work Units, see the *DB2 Server for VSE & VM Application Programming* manual.

Note: Access to an application server that is not DB2 Server for VM is only possible if the Distributed Relational Database Architecture (DRDA) facility

has been installed on the application requester and if the non-DB2 Server for VM application server supports IBM's implementation of the DRDA protocol.

Application Maintenance Considerations

DB2 Server for VSE & VM users and programs are independent of the physical storage of data. This means that procedures and programs need not be changed when their information is updated or reorganized, and logical changes can be made to the data without requiring expensive rewrites, retraining, or reorganization of the supporting application system.

This data independence improves productivity, by enabling users and programs to concentrate on the application instead of on details such as how data is stored, which users share it, or what changes have been made to it. It also means that new applications may be written with little initial regard for performance considerations: much of the optimization is handled automatically, and user-directed optimization can be done later without significant effect on the applications using the data. In addition, one user may change the format or organization of some data with minimal effect on other users who share it.

Data Administration Support

The DB2 Server for VSE & VM product supports a powerful query capability, as well as an easy-to-learn interactive support system (ISQL). The DBA can use these functions to scan stored data to determine when reorganization of data is appropriate, decide how to logically organize the data, audit its consistency and accuracy, and assess the impact of changes.

Another way to examine data is through the catalog tables, which are internally updated as a result of many SQL statements. For example, a CREATE TABLE statement causes a new entry in the SYSTEM.SYSCATALOG table; each column in the new table results in an entry in SYSTEM.SYSCOLUMNS. Because the catalog tables are regular tables (with appropriate security protection), the SQL language examines them. DBAs can look at these tables to determine table sizes and statistics, what programs use particular tables or columns, the current data types of columns in a particular table, various security information, and many other things required for understanding the status and dependencies of the database.

Refer to the *DB2 Server for VSE & VM SQL Reference* manual for more information about the catalog tables along with examples of their use.

To see if a data design is meeting performance requirements, the DBA can use the EXPLAIN statement to analyze the structure and performance of frequently used SQL statements, and to determine whether any statements or the data they access should be redesigned. See the *DB2 Server for VSE & VM Application Programming* manual for a description of the EXPLAIN statement.

Data Independence Support

Data Type Changes

A wide range of conversions from one data type to another is supported. This means that, within reason, the data type or the size of a column may be changed without requiring changes to the accessing programs. Data is converted on input and output if the data types used in program variables do not match those defined for the stored data. Data conversions and their restrictions are shown below;

explanatory notes follow.

Source Data Type	Target Data Type						
	CHAR	**DATE**	**DECIMAL**	**FLOAT-DOUBLE**	**FLOAT-SINGLE**	**GRAPHIC**	**INTEGER**
CHAR	YES[3]	YES[6]	NO	NO	NO	NO	NO
DATE	YES[7]	YES	NO	NO	NO	NO	NO
DECIMAL	NO	NO	YES[1,4]	YES[13]	YES[12,13]	NO	YES[1,2]
FLOAT-DOUBLE	NO	NO	YES[1,4,5]	YES	YES[11]	NO	YES[1,2]
FLOAT-SINGLE	NO	NO	YES[1,4,5]	YES[10]	YES	NO	YES[1,2]
GRAPHIC	NO	NO	NO	NO	NO	YES[3]	NO
INTEGER	NO	NO	YES[1]	YES	YES[12]	NO	YES
LONG VARCHAR	YES[3]	NO	NO	NO	NO	NO	NO
LONG VARGRAPHIC	NO	NO	NO	NO	NO	YES[3]	NO
SMALLINT	NO	NO	YES[1]	YES	YES[12]	NO	YES
TIME	YES[7]	NO	NO	NO	NO	NO	NO
TIMESTAMP	YES[7]	NO	NO	NO	NO	NO	NO
VARCHAR[8]	YES[3]	YES[6]	NO	NO	NO	NO	NO
VARGRAPHIC[9]	NO	NO	NO	NO	NO	YES[3]	NO

Figure 43. Data Conversion Chart (Part 1 of 2)

Source Data Type	Target Data Type						
	LONG VARCHAR	**LONG VAR-GRAPHIC**	**SMALL-INT**	**TIME**	**TIME-STAMP**	**VAR-CHAR[8]**	**VAR-GRAPHIC[9]**
CHAR	YES	NO	NO	YES[6]	YES[6]	YES[3]	NO
DATE	NO	NO	NO	NO	NO	YES	NO
DECIMAL	NO	NO	YES[1,2]	NO	NO	NO	NO
FLOAT-DOUBLE	NO	NO	YES[1,2]	NO	NO	NO	NO
FLOAT-SINGLE	NO	NO	YES[1,2]	NO	NO	NO	NO
GRAPHIC	NO	YES	NO	NO	NO	NO	YES[3]
INTEGER	NO	NO	YES[1]	NO	NO	NO	NO
LONG VARCHAR	YES	NO	NO	NO	NO	YES[3]	NO
LONG VARGRAPHIC	NO	YES	NO	NO	NO	NO	YES[3]
SMALLINT	NO	NO	YES	NO	NO	NO	NO
TIME	NO	NO	NO	YES	NO	YES[7]	NO
TIMESTAMP	NO	NO	NO	NO	YES	YES[7]	NO
VARCHAR[8]	YES	NO	NO	YES[6]	YES[6]	YES[3]	NO
VARGRAPHIC[9]	NO	YES	NO	NO	NO	NO	YES[3]

Figure 43. Data Conversion Chart (Part 2 of 2)

Notes to Figure 43:

1. An overflow error may result.
2. The fractional part of the value is dropped.
3. On output, if the length of the target is smaller than the length of the source, truncation occurs. On input, an error occurs.
4. The database manager automatically aligns the decimal point. Overflow of the integer part may result. The fractional part may be truncated.
5. The database manager attempts to create the best possible result in converting from System/370 floating point to scaled fixed point decimal.
6. The character string must contain a valid representation of a date, time, or timestamp value. However, you cannot transfer data from a CHAR or VARCHAR column into a host variable defined as a date, time, or timestamp type.
7. On output, when the source is a datetime data type and the corresponding target is a character data type, certain truncation occurs for time and timestamp. On input, an error occurs.
8. This applies to VARCHAR fields less than or equal to 254. VARCHAR fields greater than 254 are treated like LONG VARCHAR in data conversion.
9. This applies to VARGRAPHIC fields less than or equal to 127. VARGRAPHIC fields greater than 127 are treated like LONG VARGRAPHIC in data conversion.
10. The single-precision data is padded with eight hex zeros.
11. The double-precision data is converted and rounded up on the seventh hex digit.
12. Conversion is first done in double precision and then rounded to single precision.
13. Some accuracy may be lost when converting DECIMAL data type numbers to single- or double-precision floating point numbers.

Arithmetic Operations

The following sections define the rules for arithmetic operations with the data types that are supported. Note the conditions under which overflow errors can occur.

Decimal Arithmetic Operations

A decimal number has a fixed number of places in total, and a fixed number of places in its fractional part (to the right of the decimal point). The total number of places is often called the *precision*, and the number of places in the fractional part *scale*. A decimal column is defined in a CREATE TABLE or ALTER TABLE statement as: DECIMAL (*precision*,*scale*).

The precision and scale of the decimal number resulting from an arithmetic operation on two numbers (operands) are determined by the following rules:

- If one operand is a binary integer and the other is a decimal number, the operation is performed in decimal. A temporary copy of the binary integer that has been converted to decimal is used. Binary integers defined as SMALLINT will be converted to DECIMAL(5,0), while those defined as INTEGER will be converted to DECIMAL(11,0). Integer constants will always be converted to DECIMAL (11,0). The result is a decimal number as specified below.
- If both operands are decimal numbers, the result is a decimal number.
- The precision and scale of the result depend on the arithmetic operation, and on the precision and scale of the operands.

- Precision and scale can be influenced by decimal constants with leading or trailing zeros. See the *DB2 Server for VSE & VM Application Programming* manual for more information.
- If the operation is addition or subtraction, and the operands do not have the same scale, the operation is performed with a temporary copy of one of the operands. The copy is extended with trailing zeros so that its fractional part has the same number of places as the fractional part of the other operand.
- The result of an addition, subtraction, or multiplication operation is derived from a temporary result that has a maximum precision of 31. If the precision of the temporary result is not greater than 31, the final result is the same as the temporary result.
- When the scale of the temporary result is greater than that of the result (see Table 27), then the fractional part of the temporary result will be truncated so that the scales are the same.
- For an integer literal, the precision will be the number of digits, and the scale will be 0. For example, 100 will be set to precision 3 and scale 0.
- The precision and scale of a result are determined as shown in Table 27.

Table 27. Precision and Scale of Decimal Results

Assume the following notation: PA = Precision of the first operand. SA = Scale of the first operand. PB = Precision of the second operand. SB = Scale of the second operand.	
Operation	**Characteristics of the Result**
Addition and Subtraction	Precision: MIN(31,MAX(PA-SA,PB-SB)+MAX(SA,SB)) Scale: MAX(SA,SB)
Multiplication	Precision: MIN(31,PA+PB) Scale: MIN(31,SA+SB)
Division	Precision: 31 Scale: 31-PA+SA-SB (Scale must not be negative)

Binary Arithmetic Operations

If both operands are binary integers, the operation is performed in fixed binary. The result is in the INTEGER data type.

The result of a division operation is truncated. The result of a fixed binary operation must be within the range of the INTEGER data type. See "Specifying Columns" on page 30 for the ranges of data types.

Floating Point Arithmetic Operations

If either operand is a floating point number, both operands are converted to double-precision floating point numbers. The result depends on the data type of the target column or host variable. In the case of decimals, some accuracy may be lost.

If the target data type or host variable is single-precision floating point, the result is converted to single-precision floating point; otherwise, it is converted to double-precision floating point.

The result of a floating point operation must be within the range of the FLOAT data type. See "Specifying Columns" on page 30 for the ranges of data types.

Date/Time Arithmetic

Durations: Date/time arithmetic involves intervals of time that are represented by numbers called *durations*. A duration is an interpretation of a number, not a new data type. The number may be a constant, a column name, a host variable, a function, or an expression. Numbers are interpreted as durations, only in certain contexts as defined below.

The duration types are:

1. Labeled Durations

   A labeled duration is any number of years, months, days, hours, minutes, seconds, or microseconds. It is used in an expression that involves a date/time value, and consists of a numeric expression followed by one of YEAR(S), MONTH(S), DAY(S), HOUR(S), MINUTE(S), SECOND(S), or MICROSECOND(S). For example, in the expression START_DATE + 120 DAYS, the labeled duration is 120 DAYS. Fractional durations will be truncated to whole numbers (for example, 2.9 DAYS = 2 DAYS).
2. Date Durations

   A date duration represents a number of years, months, and days, expressed as a DEC(8,0) number. It has the format *yyyymmdd*, where *yyyy* is the number of years, *mm* the number of months, and *dd* the number of days. An example of a date duration is the result of D1-D2, where D1 and D2 are dates.
3. Time Durations

   A time duration represents a number of hours, minutes, and seconds, expressed as a DEC(6,0) number. It has the format *hhmmss*, where *hh* is the number of hours, *mm* the number of minutes, and *ss* the number of seconds. An example of a time duration is the result of T1-T2, where T1 and T2 are times.
4. Timestamp Durations

   A timestamp duration represents a number of years, months, days, hours, minutes, seconds, and microseconds, expressed as a DEC(20,6) number. It has the format yyyy-xx-dd-hh.mm.ss.zzzzzz, where *yyyy*, *xx*, *dd*, *hh*, *mm*, *ss*, and *zzzzzz* represent, respectively, the number of years, months, days, hours, minutes, seconds, and microseconds. An example of a timestamp duration is the result of TS1-TS2, where TS1 and TS2 are timestamps.

Rules for Date/Time Arithmetic: The only arithmetic operators that can be applied to date/time values are addition and subtraction. If a date/time value is the operand of addition, the other operand must be a duration.

A labeled duration can only be used as the operand of an arithmetic operator such that the other operand is a date/time value. For example, if D is a date and N and M are numbers, D + N DAYS + M MONTHS is a valid expression, but D + (N DAYS + M MONTHS) is not.

No automatic data conversion is provided among date/time data types. If an arithmetic operation is to be performed among different date/time values, the scalar functions should be used to convert them into the same data type. For example, if A is a TIMESTAMP column and B is a DATE column, the difference between the two in date duration can be obtained by DATE(A) - B. If you specify just A - B, an error will occur indicating incompatible types.

The specific rules for the use of the addition operator on date/time values are as follows:

1. If one operand is a date, the other must be a date duration or a labeled duration of years, months, or days.
2. If one operand is a time, the other must be a time duration or a labeled duration of hours, minutes, or seconds.
3. If one operand is a timestamp, the other may be any kind of duration.
4. Neither operand can be a parameter marker "?".

The rules for the use of the subtraction operator on date/time values are not the same as for addition: first, because a date/time value cannot be subtracted from a duration, and second, because the operation of subtracting two date/time values is not the same as that of subtracting a duration from a date/time value. The rules are as follows:

1. If the first operand is a date, the second one must be either a DATE, date duration, string representation of a date, or labeled duration of years, months, or days.
2. If the second operand is a date, the first one must be a date or string representation of a date.
3. If the first operand is a time, the second one must be either a time, time duration, string representation of a time, or labeled duration of hours, minutes, or seconds.
4. If the second operand is a time, the first one must be a time or string representation of a time.
5. If the first operand is a timestamp, the second one must be either a timestamp, a string representation of a timestamp, or a duration.
6. If the second operand is a timestamp, the first one must be a timestamp or a string representation of a timestamp.
7. Neither operand can be a parameter marker "?".

The semantic rules for date, time, and timestamp arithmetic are discussed below. Since there is no established standard for date/time arithmetic, some of the operations are defined procedurally. These procedural definitions use some of the scalar functions.

Date Arithmetic: Dates can be incremented, decremented, and subtracted. The operation of incrementing or decrementing a date by some number of days is well defined and can be verified by a calendar. The other operations are subject to peculiarities because not all months have the same number of days.

Subtracting Dates: When two dates are subtracted, the result is a date duration that gives the number of years, months, and days between those dates. The data type of the result is DECIMAL(8,0).

In the following procedural description of the operation, the term "subtrahend" refers to the number to be subtracted, and "minuend" is the number that the subtrahend is subtracted from.

If DAY(subtrahend) is not greater than DAY(minuend), the day part of the result is equal to DAY(minuend) - DAY(subtrahend).

If DAY(subtrahend) is greater than DAY(minuend), the day part of the result is equal to N + DAY(minuend) - DAY(subtrahend), where N is the last day of

MONTH(subtrahend). (For example, if MONTH(subtrahend) is 1, N is 31.) MONTH(subtrahend) is incremented by one.

If MONTH(subtrahend) is not greater than MONTH(minuend), the month part of the result is equal to MONTH(minuend) - MONTH(subtrahend).

If MONTH(subtrahend) is greater than MONTH(minuend), the month part of the result is equal to 12 + MONTH(minuend) - MONTH(subtrahend). YEAR(subtrahend) is incremented by one.

The year part of the result is equal to YEAR(minuend) - YEAR(subtrahend).

For example, the result of DATE('3/15/2000') - '12/31/1999' is 00000215 (a duration of 0 years, 2 months, and 15 days).

Incrementing and Decrementing Dates: The result of adding a duration or subtracting it from a date is a date. The result must be within the range of dates.

When a labeled duration of years is added to or subtracted from a date, the result is a date (that is, the specified number of years before or after the date in the operation). Only years are counted. The month of the result is always the same as the month of the date in the operation. The day of the result is also the same as the day of the date in the operation, unless the result would be February 29 of a non-leap year, in which case the day part of the result is 28 and SQLWARN7 is set to W.

When a labeled duration of months is added to or subtracted from a date, the result is a date (that is, the specified number of months before or after the date in the operation). Only months (calendar pages) and years (if necessary) are counted. The day of the result is the same as the day of the date in the operation, unless the result would be an incorrect date, in which case the day part of the result is the last day of the month and SQLWARN7 is set to W.

When a labeled duration of days is added to or subtracted from a date, the result is a date (that is, the specified number of days before or after the date in the operation).

When a positive date duration is added to a date or a negative duration subtracted from it, the result is a date (that is, *y* years, *m* months, and *d* days after the date in the operation, where *y*, *m*, and *d* are the year, month, and day parts of the date duration). When a positive date duration is subtracted from a date or a negative duration added to it, the result is a date (that is, *y* years, *m* months, and *d* days before the date in the operation). The arithmetic is performed using the rules defined above, including the setting of SQLWARN7 whenever an end-of-month adjustment is performed. The date duration must be DEC(8,0).

```
Let D1 be the DATE 1984-02-29, a leap year.
                                                          SQLWARN7

 D1 + 1 DAY    = 1984-03-01                                 ' '
 D1 + 2 MONTHS = 1984-04-29                                 ' '
 D1 + 1 YEAR   = 1985-02-28                                 'W'
 D1 + 4 YEARS  = 1988-02-29                                 ' '

Let N be DEC(8,0) and set to 00010203.
 D1 + N
               = 1984-02-29 + 1 YEAR + 2 MONTHS + 3 DAYS
               = 1985-02-28 + 2 MONTHS + 3 DAYS             'W'
               = 1985-04-28 + 3 DAYS
               = 1985-05-01

Let D2 be the DATE 1985-03-31.
                                                          SQLWARN7

 D2 + 1 MONTH  = 1985-04-30                                 'W'
 D2 + 2 MONTHS = 1985-05-31                                 ' '
```

Figure 44. Setting SQLWARN7 During Date Arithmetic. When incrementing or decrementing dates, SQLWARN7 is set when the resulting date is an incorrect date because of leap year or month difference, and a valid date is derived.

Peculiarities of Date Arithmetic: What does it mean to add a month to a given date? The rules defined above are based on the assumption that the result should be the same day of the next month. Thus, one month after January 1 is February 1, and one month after February 1 is March 1. But what is one month after January 31? This difficulty, which is the reason why certain contracts are always dated the first of the month, is resolved by the further assumption that the result should be the last day of February.

Thus, adding a month to a given date gives the same day of the next month unless the next month does not have such a day, in which case the result is the last day of that month. Similarly, one month from the last day of a month is not necessarily the last day of the next month. For example, one month from the last day of February is not the last day of March. In sum, “a date + a labeled duration of months - a labeled duration of months” is not necessarily equal to the original date.

The definition of the month does not permit a consistent system of date arithmetic. If this is a problem, you can avoid it by using days rather than months. For example, to increment the date “DATE3” by the difference between the dates “DATE1” and “DATE2”, the expression “DATE (DAYS(DATE1) - DAYS(DATE2) + DAYS(DATE3))” will give an accurate result, whereas “DATE1 - DATE2 + DATE3” may not.

Time Arithmetic: Times can be incremented, decremented, and subtracted. The only peculiarity is the modules of 24 hours. For example, adding any multiple of 24 hours to a time gives the same time. The exception is 00:00:00, where adding 24:00:00 becomes 24:00:00.

Subtracting Times: When two times are subtracted, the result is a time duration that gives the number of hours, minutes, and seconds between the two times. The data type of the result is DECIMAL(6,0).

In the following procedural description of the operation, the term “subtrahend” refers to the number to be subtracted, and “minuend” is the number that the subtrahend is subtracted from.

If SECOND(subtrahend) is not greater than SECOND(minuend), the seconds part of the result is equal to SECOND(minuend) - SECOND(subtrahend).

If SECOND(subtrahend) is greater than SECOND(minuend), the seconds part of the result is equal to 60 + SECOND(minuend) - SECOND(subtrahend). MINUTE(subtrahend) is incremented by one.

If MINUTE(subtrahend) is not greater than MINUTE(minuend), the minute part of the result is equal to MINUTE(minuend) - MINUTE(subtrahend).

If MINUTE(subtrahend) is greater than MINUTE(minuend), the minute part of the result is equal to 60 + MINUTE(minuend) - MINUTE(subtrahend). HOUR(subtrahend) is incremented by one.

The hour part of the result is equal to HOUR(minuend) - HOUR(subtrahend).

Incrementing and Decrementing Times: The result of adding a duration to a time or subtracting a duration from it is a time. In each of the following cases, any overflow or underflow of hours is discarded. Thus, the result is always within the range of a time.

When a labeled duration of hours is added to or subtracted from a time, the result is a time (that is, the specified number of hours before or after the time in the operation). Only hours are counted. Thus, the minute and second of the result are the same as the minute and second of the time in the operation.

When a labeled duration of minutes is added to or subtracted from a time, the result is a time (that is, the specified number of minutes before or after the time in the operation). Only minutes and hours (if necessary) are counted. Thus, the second of the result is the same as the second of the time in the operation.

When a labeled duration of seconds is added to or subtracted from a time, the result is a time (that is, the specified number of seconds before or after the time in the operation).

When a time duration is added to or subtracted from a time, the result is a time (that is *h* hours, *m* minutes, and *s* seconds before or after the time in the operation, where *h*, *m*, and *s* are the hour, minute, and second parts of the time duration). The time duration must be a DEC(6,0) value.

Timestamp Arithmetic: Timestamps can be incremented, decremented, and subtracted. The operations are a combination of the date arithmetic and time arithmetic defined above, except that any overflow or underflow of hours is reflected in the date part of the result.

Subtracting Timestamps: When two timestamps are subtracted, the result is a timestamp duration that gives the number of years, months, days, hours, minutes, and seconds between the two timestamps. The data type of the result is DECIMAL(20,6).

In the following procedural description of the operation, the term “subtrahend” refers to the number to be subtracted, and “minuend” is the number that the subtrahend is subtracted from.

If MICROSECOND(subtrahend) is not greater than MICROSECOND(minuend), the microseconds part of the result is equal to MICROSECOND(minuend) - MICROSECOND(subtrahend).

If MICROSECOND(subtrahend) is greater than MICROSECOND(minuend), the seconds part of the result is equal to 1000000 + MICROSECOND(minuend) - MICROSECOND(subtrahend). SECOND(subtrahend) is incremented by one.

Second and minute are subtracted as specified in the rules for "Subtracting Times" on page 179.

If HOUR(subtrahend) is not greater than HOUR(minuend), the hour part of the result is equal to HOUR(minuend) - HOUR(subtrahend).

If HOUR(subtrahend) is greater than HOUR(minuend), the hour part of the result is equal to 24 + HOUR(minuend) - HOUR(subtrahend). DAY(subtrahend) is incremented by one.

Day, month, and year are subtracted as specified in the rules for "Subtracting Dates" on page 177.

Incrementing and Decrementing Timestamps: The result of adding a duration to or subtracting it from a timestamp is a timestamp. In each of the following cases, date and time arithmetic are performed as defined above, except that an overflow or underflow of hours is carried into the date part of the result, which must be within the range of dates.

When a labeled duration of years is added to or subtracted from a timestamp, the result is a timestamp (that is, the specified number of years from the timestamp).

When a labeled duration of months is added to or subtracted from a timestamp, the result is a timestamp (that is, the specified number of months from the timestamp).

When a labeled duration of days is added to or subtracted from a timestamp, the result is a timestamp (that is, the specified number of days from the timestamp).

When a labeled duration of hours is added to or subtracted from a timestamp, the result is a timestamp (that is, the specified number of hours from the timestamp).

When a labeled duration of minutes is added to or subtracted from a timestamp, the result is a timestamp (that is, the specified number of minutes from the timestamp).

When a labeled duration of seconds is added to or subtracted from a timestamp, the result is a timestamp (that is, the specified number of seconds from the timestamp).

When a labeled duration of microseconds is added to or subtracted from a timestamp, the result is a timestamp (that is, the specified number of microseconds from the timestamp).

When a date duration is added to or subtracted from a timestamp, the result is a timestamp. The year, month, and day parts are the result of the arithmetic operation performed using the rules defined for incrementing or decrementing a

date by a date duration. The hour, minute, second, and microsecond parts are the same as those of the timestamp in the operation.

When a time duration is added to or subtracted from a timestamp, the result is a timestamp. The time part is the result of the arithmetic operation performed using the rules defined above for incrementing or decrementing a time by a time duration, except that any overflow or underflow of hours is carried into the date part of the result. The microsecond part of the result is the same as the microsecond part of the timestamp in the operation.

When a timestamp duration is added to or subtracted from a timestamp, the result is a timestamp (that is, *y* years, *x* months, *d* days, *h* hours, *m* minutes, *s* seconds, and *z* microseconds before or after the time in the operation, where these values are the year, month, date, hour, minute, second and microsecond parts of the timestamp duration). Date and time arithmetic are performed as previously defined, except that an overflow or underflow of hours is carried into the date part of the result. Microseconds overflow into seconds. The timestamp duration must be DEC(20,6).

Figure 45 and Figure 46 on page 183 summarize date/time addition and subtraction, respectively. The STRING column in both tables mean a character string in a valid date/time format.

DATE/TIME ADDITION = OPERAND + OPERAND

	LEFT OR RIGHT OPERAND														
					DURATIONS										
								SIMPLE							
LEFT OR RIGHT OPERAND ↓	DATE	TIME	TIMESTAMP	STRING	DATE	TIME	TIMESTAMP	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND	MICRO SECONDS	RESULT DATA TYPE ↓
DATE					X			X	X	X					DATE
TIME						X					X	X	X		TIME
TIME STAMP					X	X	X	X	X	X	X	X	X	X	TIME STAMP

Figure 45. Date/Time Addition

An X denotes a valid date/time addition operation.

DATE/TIME SUBTRACTION = MINUEND - SUBTRAHEND

MINUEND ↓	SUBTRAHEND														RESULT DATA TYPE ↓
					DURATIONS										
								SIMPLE							
	DATE	TIME	TIMESTAMP	STRING	DATE	TIME	TIMESTAMP	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND	MICRO SECONDS	
DATE	1			1	2			2	2	2					1=(8,0) 2=DATE
TIME		1		1		2					2	2	2		1=(6,0) 2=TIME
TIME STAMP			1	1	2	2	2	2	2	2	2	2	2	2	1=(20,6) 2=TIME STAMP

Figure 46. Date/Time Subtraction

Both 1 and 2 denote a valid date/time subtraction operation. 1 means a result data type of DECIMAL(8,0), DECIMAL(6,0), or DECIMAL(20,6) that is deemed as a date duration, time duration, or timestamp duration, respectively. 2 means a result data type of date, time, or timestamp.

Data Access Changes

Users do not have to specify **how** data is to be accessed; only **what** data is to be accessed. Access path selection is done by the database manager, which determines which strategy will minimize the cost of processing a query. Cost is based on estimates of processor and I/O requirements. Users are not only free of such matters, they are not allowed to use any knowledge of such details. This allows the program to continue to operate when the underlying storage structures are changed.

Data Structure Changes

Both logical and physical structural changes can be made to data without significant effect on users or their programs. The database manager permits flexibility in the binding of programs' data references to the data objects in the database. This significantly reduces the impact of changes.

The following sections note a few important considerations to reduce the effect of data restructuring.

Program Reference Flexibility: When a program is preprocessed, references to nonexistent tables, views, or columns, or the use of statements that require a level of authority that has not yet been granted, **do not prevent** a package from being created; these conditions only cause warning messages. If the required authority or object exists when the referencing statement is processed, execution will proceed normally.

Although this design is very useful, it exacts a performance penalty. Preprocessing of such a program should be done again before the program is used extensively. By repeating the preprocessing step after acquiring the authority or having the required objects created, you avoid implicit, dynamic preprocessing of those statements that had unresolved objects or authority at the time of the original preprocessing.

Adding New Columns to Existing Tables (ALTER TABLE): When you add new columns to existing tables, referencing programs are normally not affected.

With SELECT statements, when selected columns are specifically named (rather than specifying SELECT *) or when a view is used, there is no effect on the program.

With INSERT statements, the effect of added columns on existing programs can be eliminated by specifically naming the target fields or by using a view, when the new columns permit NULL values. If the fields are not named, and if affected or new columns do not permit NULLS, the program must be changed and preprocessed again.

With the UPDATE statement, there is no effect because of changes, because individual fields are always specifically named.

With DELETE statements, the action applies to the row as a whole, so adding fields has no direct effect on existing programs.

When you add new fields, you may have to rewrite old programs to pick up the new function associated with those fields. However, with the above described restrictions, there need be no effect on old programs for existing function. Programs can continue to work normally through such changes until it is really necessary to update them.

Dropping Columns and Tables (DROP TABLE): To drop a column from a table, drop the table (DROP TABLE) and recreate it (CREATE TABLE) without that column. If the column dropped is not used by an existing program, dropping it in this manner does not functionally affect that program. The program is automatically re-preprocessed when it is next used.

Note: If the table has data in it, before dropping it, save the data. Either use the DBS Utility to unload the table to a tape or DASD SAM file; or create a new temporary table, and use an INSERT with Subselect statement to copy the data into it. Later, you can either use the DBS Utility to reload the data into the newly created table; or use an INSERT with Subselect statement to copy the data from the temporary table to the newly created table, then drop the temporary table.

When you drop a table, all keys, indexes, and privileges are lost.

When a program is preprocessed, all of its dependencies (such as tables needed) are recorded in the SYSTEM.SYSUSAGE table. Then, whenever one of these objects is dropped, the SYSTEM.SYSUSAGE table is searched to check for dependencies; if the program depends on the object just dropped, it marks the entry invalid in the SYSTEM.SYSACCESS table against the package for that program, and marks any loaded copies of the package (in the cache) unusable.

The next time the package is invoked, it will automatically be re-preprocessed. If referenced objects have been reestablished properly, the preprocessing will succeed. The user will not be aware of the activity, except for a longer than usual time delay when the package is first invoked after the change.

Of course, when a program requires a field that is dropped, it can no longer function properly until it is brought current with the change. Because the program references the dropped field, if it is submitted for execution without being changed, the automatic re-preprocessing will fail and the submitter will be notified.

Adding Indexes to Tables: Adding an index to a table has no effect on users or programs that use the table. However, to make it possible to take advantage of potential performance improvements offered by the index, programs using it should be preprocessed again. (You should apply the UPDATE STATISTICS statement for the table after adding the index and before preprocessing the program again.) The preprocessing step enables the database manager to re-examine the possible access strategies, and possibly take advantage of the new index.

Dropping an Index for a Table: Indexes for tables used in programs are recorded in SYSTEM.SYSUSAGE in the same manner described above for tables. When an index is dropped, the same automatic re-preprocessing occurs for dependent programs, allowing adjustment of the access strategy to reflect the lost index. For a dropped index, there is no need for further action by the programmers who create the using programs, because the automatic re-preprocess activity handles required adjustments.

Changing Data Relationships: Data relationships are handled by keeping the data structures simple (see “Step 7: Normalize Your Tables” on page 9) and expressing the relationships in the accessing statements. If this is done properly, new relationships can be accomplished without changing existing programs or users. For example, new tables may be associated with old ones by way of joins; predicates may use fields from many tables, and new views may be added.

Changing Referential Integrity Relationships: There is considerable flexibility allowed in adding or dropping referential constraints. If the structure of your data changes, you can drop the primary key of a table, and create a new primary key for it. For information on primary keys, see “Step 4: Identify One or More Columns as a Primary Key” on page 4. You can also add new foreign keys to accommodate changes in the structure of your data, and drop old ones when they are no longer used. For more information, see “Step 6: Plan for Referential Integrity” on page 6.

When referential constraints are changed or if keys are inactivated, application programs that access the affected tables will automatically be re-preprocessed and compiled.

Changing Unique Constraints: Unique constraints are similar to primary keys, and are useful when uniqueness on more than one column is desired. You can drop the unique constraint of a table and create a new one, or add additional ones. For information on unique constraints, see "Step 4: Identify One or More Columns as a Primary Key" on page 4.

When unique constraints are dropped or inactivated, application programs that access the affected tables will automatically be re-preprocessed and compiled.

Data Authorization Changes

When new authorization is added, old programs are completely unaware of the change.

When the preprocessor encounters program dependencies on specific authorizations, these dependencies are recorded in the SYSTEM.SYSTABAUTH table as described above for dependent objects. When a program-dependent authorization is removed, the package associated with the program is marked invalid, and the automatic re-preprocessing occurs as described before. If proper authorization is re-acquired before the automatic re-preprocessing, the package is re-preprocessed successfully; otherwise, the invoker is notified of the problem and the re-preprocessing fails.

The Preprocessor KEEP Option for RUN Authority

The preprocessor has an option called KEEP|REVOKE, which allows for either keeping or revoking previously granted RUN authority. It pertains to the version of the package that is produced by the new preprocessing step.

This design simplifies the effect of changes that require repeating the preprocessing step, by not having to repeat the associated authorization procedures. When an automatic re-preprocessing occurs, the KEEP option is implicitly in effect.

Changing the Users of Data

Because users are not affected by data sharing, new users can be added, data can be employed by different people in different ways, and previous uses can be discontinued without effect on current users or their programs.

Hypothetical Change Support

The recovery facilities also offer a significant benefit for managing changes to applications. With these facilities, changes can be applied, examined, tested and then everything can be backed out with a ROLLBACK statement, and it will appear as if nothing happened. With ISQL, you must run with AUTOCOMMIT off.

The DBA may ask hypothetical questions without disrupting live data. Answers for questions such as: "What if I change the supplier of part ZT33592 to improve the delivery time?" or "What is the effect on overall product cost?" may be very valuable and, because they impose no permanent changes on the database, may be made safely.

Chapter 9. DB2 Server for VM Database Configurations

This chapter provides an overview of some of the many possible configurations. (For detailed information on how to establish any particular configuration, see the *DB2 Server for VM System Administration* manual.) It also contains information on VSE Guest Sharing, and on the VM/ESA operating environment.

DB2 Server for VM Concepts

The following terms are used in the descriptions of the DB2 Server for VM configurations that follow:

Database
A collection of CMS minidisks that store both user and system information. The latter includes data used to secure and manage the database, such as a list of valid users.

Database Manager
A program that provides database management services. This program executes in its own virtual machine, referred to as the *database machine*.

The database manager controls any updates or deletes made to the database, and maintains its security and integrity.

To protect the integrity of the database, users do not have direct access to it. Rather, their requests are sent to the database manager, which processes them and returns the results to the users.

Service Machine
A virtual machine required by any processor that does not have its own DB2 Server for VM database machine, and has users who want to access a DB2 Server for VM database in a *collection*.

Notes:

1. A collection is a group of VM processors that are connected together using channel-to-channel, binary synchronous lines, or local area networks.
2. The term *local* applies to either resources or users. The service machine provides essential DB2 Server for VM support to users by allowing access to the production minidisk.

User Machine
A virtual machine that runs either ISQL, DBS Utility, or a user-written application program that uses SQL.

User machines cannot make changes directly to the database. They must send SQL statements to the database manager.

Resource Adapter
The DB2 Server for VM code used by ISQL, the DBS Utility, and application programs to communicate with the database machine. It enables users to communicate with the database manager. Users need not be aware of it.

Operating Modes for the Database Machine

The database machine can run in two modes of operation: multiple user or single user. The DB2 Server for VM operator (the person logged onto the database machine) selects the mode when he or she starts up the database machine.

In multiple user mode, the most common mode of operation, one or more users or applications concurrently access the same database. The database manager runs in its own virtual machine, while one or more DB2 Server for VM applications run in other virtual machines.

In single user mode, the database manager and an application program run in a single VM machine, and no other users are allowed access. Some maintenance tasks, such as adding auxiliary storage to the database, require this mode.

It is also possible to operate more than one DB2 Server for VM database machine in multiple user mode: that is, multiple databases are being accessed by many users concurrently. This is called "multiple database mode".

Example Configurations

The following configurations all assume that the database machines are operating in multiple user mode.

One Database Machine with One Database

In the simplest configuration, there is one database machine and one database. (This environment is created by the installation process.)

Figure 47 shows an example. Here, all virtual machines reside on the same processor.

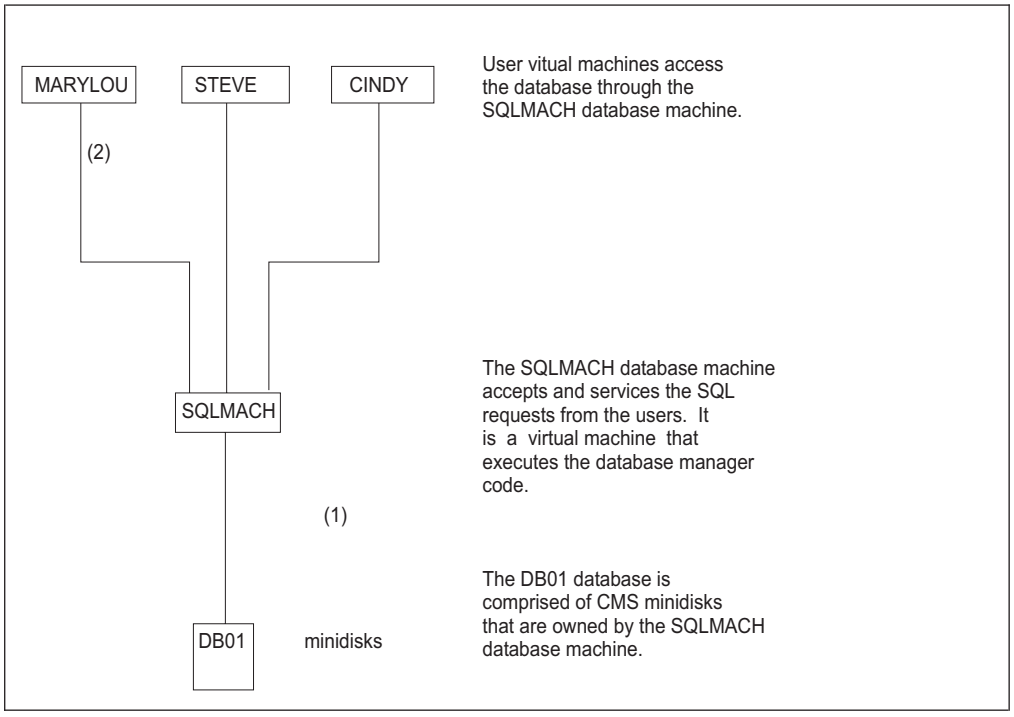


Figure 47. Example of One Database Machine Running One Database

Points (1) and (2) in the figure are as follows:

1. The SQLMACH database machine was set up to use the DB01 database.

   The operator selects a database when the database machine is started.

2. Three user virtual machines (MARYLOU, STEVE, and CINDY) communicate with the SQLMACH database machine to access the DB01 database. They must enter:

   ```
   SQLINIT DBNAME(DB01)
   ```

   to specify DB01 as the default database. Then, when they invoke ISQL, the DBS Utility, or an application program, this default database will be accessed. See "SQLINIT EXEC" on page 235 for information on the SQLINIT EXEC.

One Database Machine with Two Databases

In Figure 48, all virtual machines reside on the same processor.

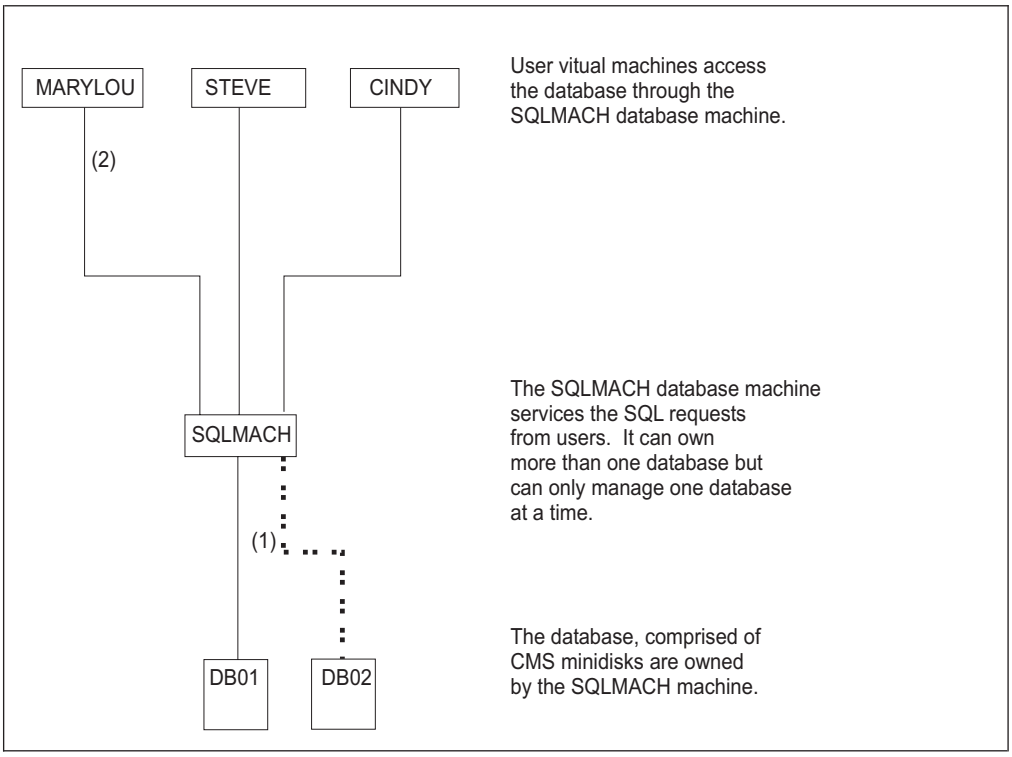


Figure 48. One Database Machine that Owns More than One Database

Points (1) and (2) in the figure are as follows:

1. The database machine was set up to use the DB01 database.

   The operator selects a database when the database machine is started. In this example, the operator had the choice of selecting the DB01 or DB02 database. The DB01 database was chosen.

   Note that a database machine can only access one database at a time. To access the DB02 database, the operator must restart the SQLMACH database machine, specifying the DB02 database.

2. Three user virtual machines (MARYLOU, STEVE, and CINDY) communicate with the database machine to access the DB01 database. They must enter:

   ```
   SQLINIT DBNAME(DB01)
   ```

to specify DB01 as the default database. Then, when they invoke ISQL, the DBS Utility, or an application program, this default database will be accessed. (See "SQLINIT EXEC" on page 235 for information on the SQLINIT EXEC.)

Here, users cannot access the DB02 database. If they entered "SQLINIT DBNAME(DB02)" and then tried to access DB02 (using ISQL, the DBS Utility, or an application program), an SQL error would occur.

If users need to access DB02, the operator will have to restart the SQLMACH database machine, specifying that DB02 is to be accessed. DB01 must be stopped before DB02 can be restarted.

Several Database Machines with Many Databases

In Figure 49, all virtual machines reside on the same processor.

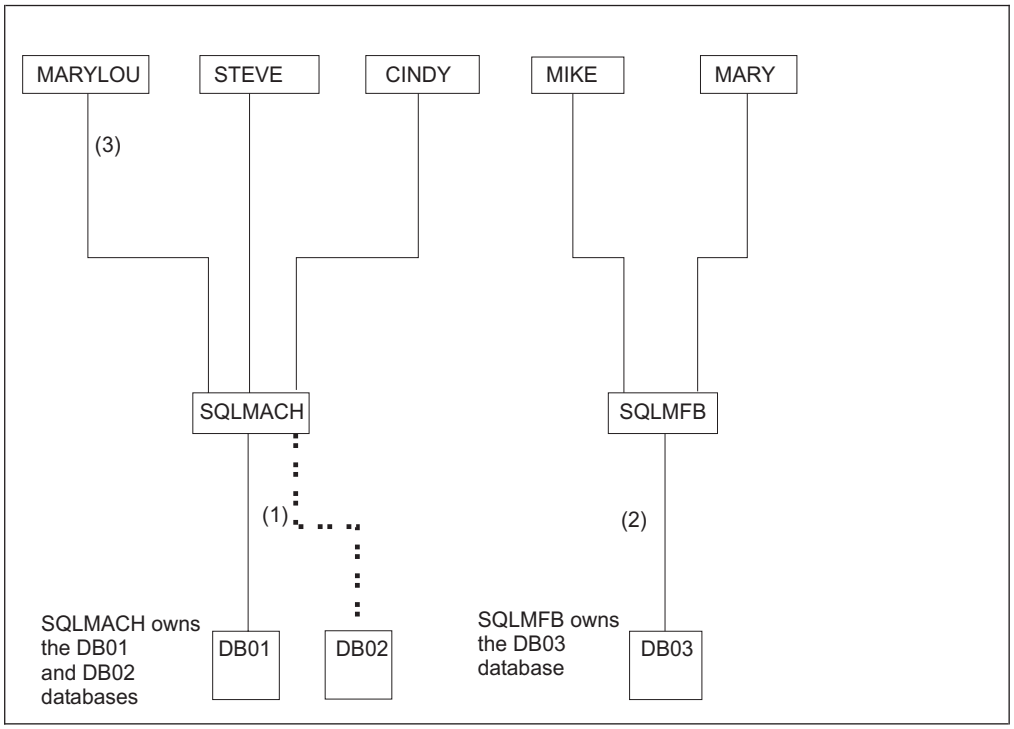


Figure 49. Two Database Machines with Three Databases

Points (1), (2), and (3) of the figure are as follows:

1. The SQLMACH database machine was set up to use the DB01 database.

   The SQLMACH operator selects a database when the database machine is started. In this example, the operator had the choice of selecting the DB01 or DB02 database. The DB01 database was chosen.

   Note that a database machine can only access one database at a time. To access the DB02 database, the operator must restart the SQLMACH database machine, specifying the DB02 database.
2. The SQLMFB database machine was set up to use the DB03 database.

   Note that it is possible for one database machine to access a database "owned" by another database machine, as long as the virtual machines reside on the same processor. For example, the SQLMFB database machine could access the DB02 database, provided that the SQLMFB operator knows the minidisk passwords for the DB02 database minidisks.

Note: A database machine "owns" a database if its virtual machine directory contains the MDISK statements for the database minidisks.

3. Five user virtual machines (MARYLOU, STEVE, CINDY, MIKE and MARY) communicate with the database machines. MARYLOU, STEVE, and CINDY must enter:

   ```
   SQLINIT DBNAME(DB01)
   ```

   to specify DB01 as their default database, while MIKE and MARY must enter:

   ```
   SQLINIT DBNAME(DB03)
   ```

   to specify DB03 as their default database. (See "SQLINIT EXEC" on page 235 for information on the SQLINIT EXEC.)

   Here, users cannot access the DB02 database. If they entered "SQLINIT DBNAME(DB02)" and then tried to access DB02 (using ISQL, the DBS Utility, or an application program), an SQL error would occur.

   If users need to access DB02, the SQLMACH database machine operator will have to restart the SQLMACH database machine, specifying that DB02 is to be accessed. Restarting the SQLMACH database machine to access DB02 will stop users from accessing DB01.

   Users can change the database they are accessing in two ways:

   a. Using the SQLINIT EXEC to specify a new default.
   b. Using the CONNECT statement to switch databases. This can be done from within an application (ISQL, the DBS Utility, or an application program).

      For example, suppose MARYLOU is accessing DB01 using ISQL. She can switch to DB03 by entering the following SQL statement:

      ```
      CONNECT TO DB03
      ```

      See "Connecting to an Application Server in VM" on page 95 for more information.

Multiple Database Machines on Different Processors

Users can access a database that resides on another processor, provided both processors are running on VM/ESA systems, and are connected in TSAF, SNA or TCP/IP network. (Refer to the *DB2 Server for VM System Administration* manual for information about network configurations.)

Figure 50 shows an example of accessing a database located on another processor.

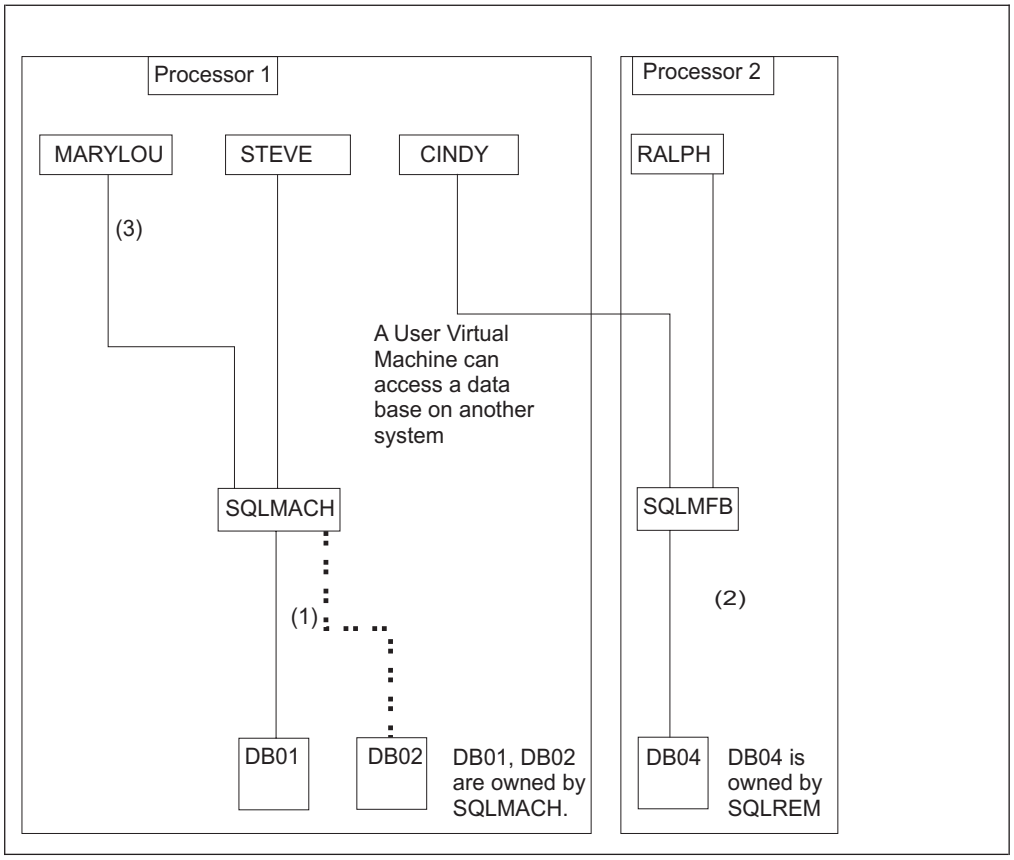


Figure 50. User Accessing a Database on Another Processor

Points (1), (2), and (3) of the figure are as follows:

1. The SQLMACH database machine was set up to use the DB01 database.

   The SQLMACH operator selects a database when the database machine is started. In this example, the operator had the choice of selecting the DB01 or DB02 database. The DB01 database was chosen.

   Note that a database machine can only access one database at a time. To access the DB02 database, the operator must restart the SQLMACH database machine, specifying the DB02 database.
2. The SQLREM database machine was set up to use the DB04 database.

   The database is established as a *global resource*.

   Note: Databases can be classified as either local or global. A local database can only be accessed by users located on the same processor as itself, while a global one can also be accessed by users located on other processors within the collection.

   The SQLREM operator specified the DB04 database at startup.

   It is possible for a database machine to access a database "owned" by another database machine, provided the virtual machines reside on the same processor. The SQLREM database machine cannot access the DB01 database (owned by SQLMACH), because SQLMACH and DB01 are on a different processor.
3. Four user virtual machines (MARYLOU, STEVE, CINDY and RALPH) communicate with the database machines.

   MARYLOU and STEVE enter:

```
SQLINIT DBNAME(DB01)
```

to specify DB01 as their default database, while CINDY and RALPH enter:

```
SQLINIT DBNAME(DB04)
```

to specify DB04 as their default database.

Note that although CINDY is on a different processor from RALPH, both access the DB04 database in the same way, and CINDY is able to specify DB04 as her default database.

Users can change the database they are accessing in two ways:

a. Using the SQLINIT EXEC to specify a new default database.

   After establishing a new default database, the user could then access the database using ISQL, the DBS Utility, or application programs.

b. Using the SQL CONNECT statement.

   This can be done from within an application (ISQL, the DBS Utility or application program).

   For example, suppose MARYLOU is accessing DB01 using ISQL. She can switch DB04 by entering the following SQL statement:

   ```
   CONNECT TO DB04
   ```

   Refer to “Connecting to an Application Server in VM” on page 95 for more information.

Accessing a Database from a Processor that Does Not Have One

Users on processors that do not have a database machine or a database can access a database on another processor. (The processors must be running on VM/ESA systems, and all user IDs must be unique between processors.) Figure 51 shows an example.

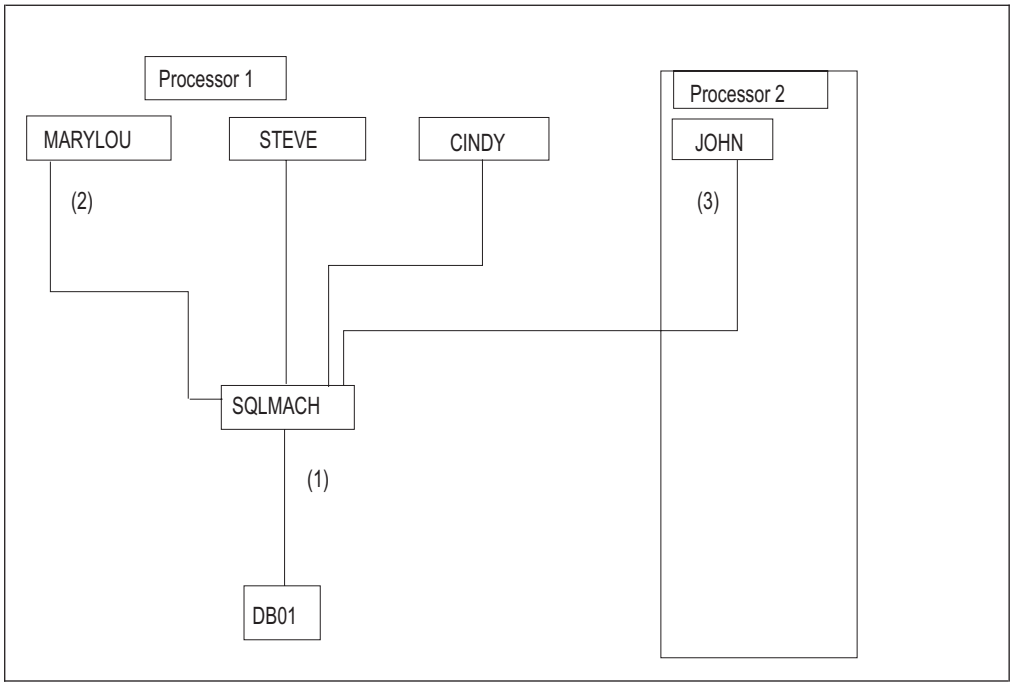


Figure 51. Accessing a Database from Another Processor

Points (1), (2), and (3) of the figure are as follows:

1. The SQLMACH database machine was set up to use the DB01 database.

   The SQLMACH operator selects a database when the database machine is started. This database is established as a global resource.
2. Four user virtual machines (MARYLOU, STEVE, CINDY, and JOHN) communicate with the database machine (SQLMACH) to access DB01. MARYLOU, STEVE, CINDY, and JOHN enter:

   ```
   SQLINIT DBNAME(DB01)
   ```

   to specify DB01 as their default database.

   When MARYLOU, STEVE, CINDY, or JOHN invoke ISQL, the DBS Utility, or an application program, DB01 will be accessed.
3. JOHN is on Processor 2 which does not have a database machine or database.

   JOHN must have a link to the service machine disk in order to access the database on Processor 1. This service machine must be installed on Processor 2.

Performance Considerations with Multiple Databases

Most of the processing time for any transaction is spent in the database machine and not in an application program. It is possible to have users on one processor accessing data stored on another processor if the operating systems are VM/ESA systems. However, because of the processor overhead of inter-processor communication, a database should be placed on a processor that is closest to its greatest number of users, preferably on the same one.

If you plan to have users on one processor accessing a database on another processor, you should consider the overhead of inter-processor communication. Most message traffic between the user and the database machine should flow

within a single processor. Only a small percentage of messages should flow between processors, as happens when users infrequently access data located on other processors.

Although the end user does not need to know the physical location of a database, you must have a good understanding of user-group requirements in a particular business environment. Users should first be grouped according to the business tasks they perform. Databases and tables can then be arranged so that they are always accessed by most users in a particular group. A good understanding of the business environment and the needs of various user groups can aid in determining whether users on one processor are allowed to access databases on other processors.

When planning your database configuration, you should generally avoid:
- Database configurations with central databases
- Databases with randomly distributed tables.

Database configurations should be designed so that:
- Groups of tables are situated in the database closest to their greatest number of users
- Tables in databases on other processors are only occasionally accessed.

For more information on performance considerations, see the *DB2 Server for VM System Administration* manual.

VSE Guest Sharing (On VM/ESA Systems Only)

If you have VSE/AF 4.1.0 or higher running as a guest under a VM/ESA system, VSE users can access DB2 Server for VM databases through VSE Guest Sharing. The database accessed can be on the same VM/ESA system as the one that supports the VSE guest, or on another VM/ESA system in the same TSAF collection, SNA or TCP/IP network. A VSE guest user can use database switching to target another database. An example of a DB2 Server for VM database with VSE Guest Sharing support is shown in Figure 52 on page 196.

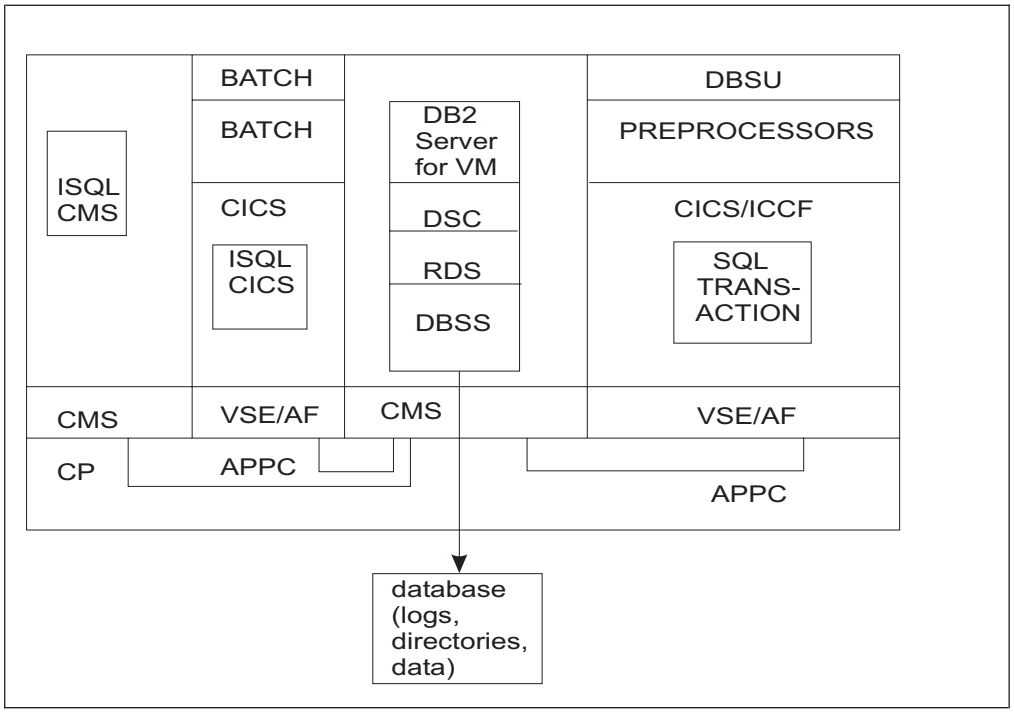


Figure 52. VSE Guest Sharing Configuration

In a VSE Guest Sharing environment, VM/ESA users and applications can use the same functions they use in a VM/ESA environment without being aware of guest sharing.

For more information on VSE Guest Sharing, and on the DB2 Server for VM configurations that you can implement, see the *DB2 Server for VM System Administration* manual.

Chapter 10. Usage Environments in VSE

The hardware and software needed to run the DB2 Server for VSE system varies depending on the usage of the DB2 Server for VSE system.

Depending on your requirements, your DB2 Server for VSE environment may be set-up in different ways. The purpose of this chapter is to give you an idea of the various different usage environments that can be set up. Further information about how to set up various DB2 Server for VSE environments can be found in the *DB2 Server for VSE System Administration* manual.

This chapter describes five different DB2 Server for VSE usage environments. You will need to consider each of these environments and choose the one appropriate for your processing requirements. It also describes the recommended and required options of the associated program products.

Batch/Interactive Application Processing

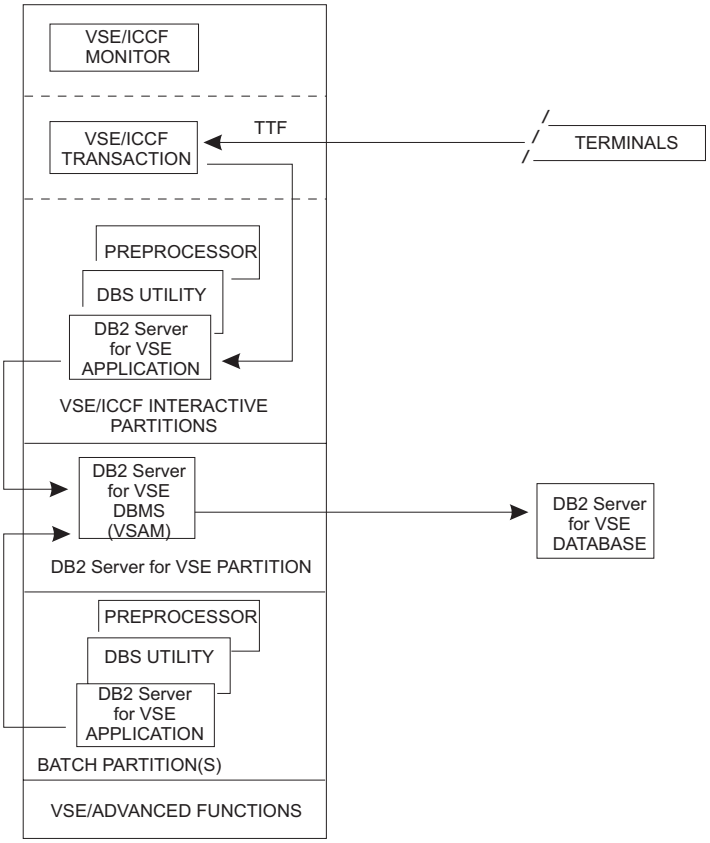


Figure 53. Batch/Interactive Configuration

Batch/interactive processing (Figure 53) is mostly execution of compiled PL/I, COBOL, C, FORTRAN or assembler programs which are batch (or VSE/ICCF) SQL applications. Some dynamic SQL processing may occur in the form of application preprocessing and compilation, and DBS Utility executions.

Operation of this type of system places minimal demands on system resources (real storage and processor power). However, there is a corresponding loss of function, because such a system cannot support the query/report writing facilities (ISQL) of the DB2 Server for VSE or online (CICS) transaction processing with DB2 Server for VSE.

There are no special prerequisites beyond the base DB2 Server for VSE prerequisites for VSE/Advanced Functions and VSE/VSAM except that one of the supported programming languages (PL/I, COBOL, C, FORTRAN, or assembler) is required. VSE/ICCF is not required, but it can be used for terminal access and invocation of the SQL applications, and for data administration activities.

A batch/interactive system should be considered for automation of fixed business applications that do not require end user access to the system. Such a system is typically developed on a larger system that is configured to support application development.

Online (CICS) Transaction Processing

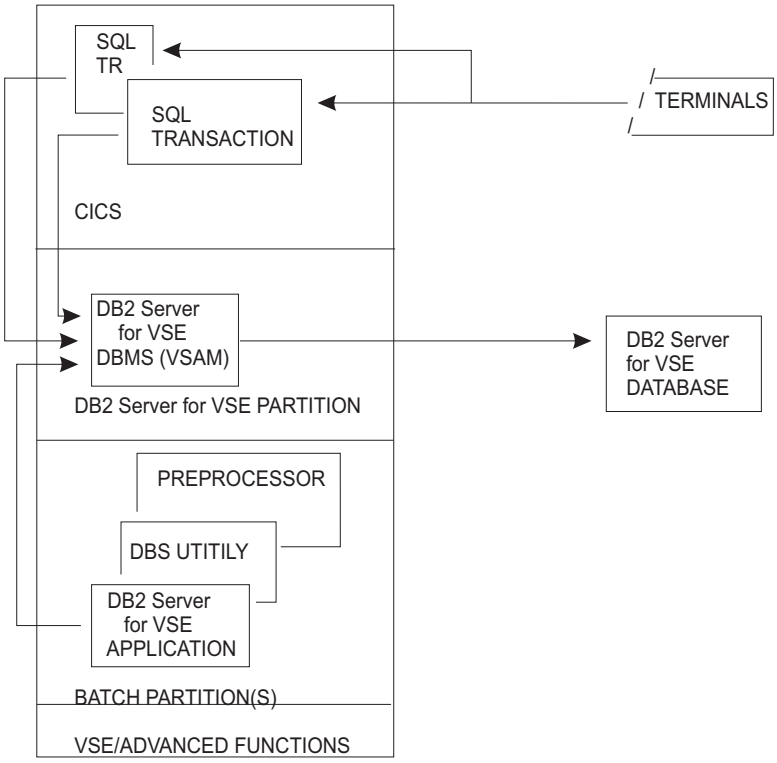


Figure 54. Online Transaction Processing Configuration

Online transaction processing usage of the DB2 Server for VSE system (Figure 54) is mostly preplanned CICS SQL transactions. Some dynamic SQL processing may occur in the form of SQL/DS preprocessor and DBS Utility jobs, which do not run under the CICS subsystem.

Like the batch/interactive application processing usage, the online transaction processing usage demands fewer real storage and processor resources than dynamic SQL usage demands.

Online transaction processing with the DB2 Server for VSE system requires installation of CICS Release 2.3, or an equivalent transaction processing product. The DB2 Server for VSE online support must also be installed. The CICS subsystem provides the terminal management and transaction processing environment. Programs may be written in PL/I, COBOL, C, or assembler, but not FORTRAN as the CICS subsystem does not support FORTRAN. You can, however, use FORTRAN for batch programs.

An online transaction processing system should be considered for preplanned business applications where end user access to the system is managed through CICS transactions programmed for specific end user tasks.

ISQL might be installed, but its use would be limited to data administration functions.

For the online transaction processing environment, the system should be configured as follows:

- CICS Options:
  - The Dynamic Transaction Backout Program (DBP) is required for proper coordination and recovery with the DB2 Server for VSE database manager.
  - The Exec Interface Program (EIP) is required to support transaction access to the DB2 Server for VSE application server.
  - The CICS User Exit Interface is also required for transaction access to the DB2 Server for VSE application server.
  - The CICS Monitoring Facility is optional, but should be used so that the DB2 Server for VSE database manager participates in the monitoring by providing performance class information.
  - The CICS Restart Resynchronization facility is required to support task-related user exit resynchronization and to use the SQL/DS accounting facility in an online environment.
- The VSE/POWER program is required for the system printer or remote workstation printer report-writing support in ISQL. It is not required for report writing to CICS terminal printers. Only the VSE/POWER program provides multiple copy capability.

Application Development

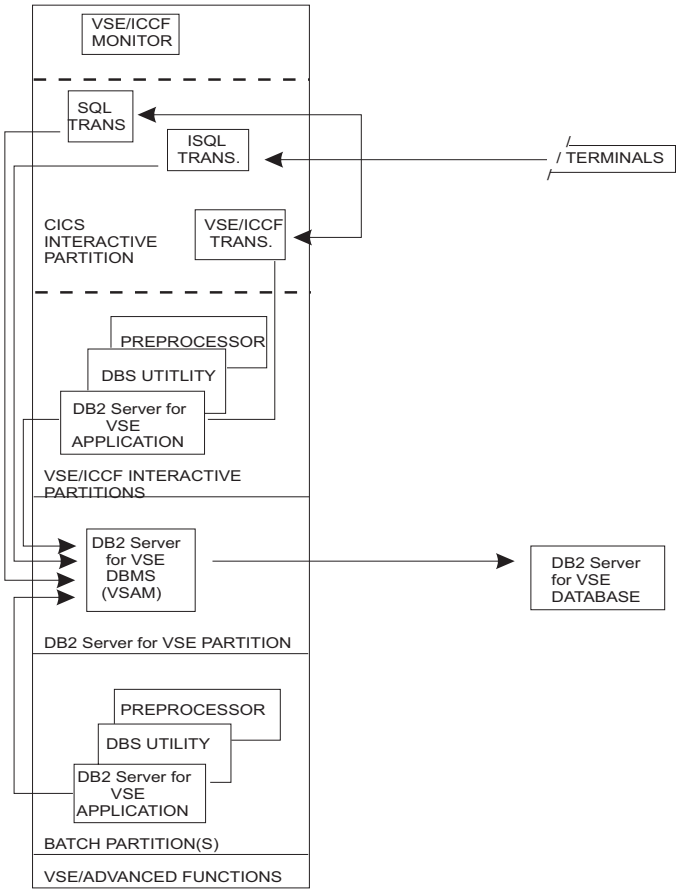


Figure 55. Application Development Configuration

Application development usage of the DB2 Server for VSE system (Figure 55) includes a large amount of data design, application coding, and testing. Such activities typically involve a modest level of dynamic SQL activity in the form of data definition, catalog queries, and program preprocessing. Correspondingly, there is more demand for real storage and processor resources, than that demanded by application or transaction processing.

The program products required to support application development on the DB2 Server for VSE system vary depending on the type of application being developed and the DB2 Server for VSE facilities to be used. The optional program products and their options are discussed below in terms of their value to application development usage of the DB2 Server for VSE system.

1. Programming Languages

   For development of programmed applications you would need one or more of the PL/I, COBOL, C, FORTRAN, or Assembler language products.

2. VSE/ICCF

   You can install VSE/ICCF (or an equivalent) to support an interactive application development capability.

3. The CICS subsystem

The CICS subsystem is required to support development of CICS transactions or use of the ISQL facilities for application development. It should be generated with the dynamic transaction backout program (DBP), the EXEC interface program (EIP), and the CICS user exit interface. Optionally, you can generate it with the Monitoring Facility and Restart Resynchronization. Restart resynchronization is required if you want to (1) use the SQL/DS accounting facility in an online environment, and (2) support task-related user exits.

4. VSE/POWER

   The VSE/POWER program is required for the system printer or remote workstation printer report-writing support in ISQL. It is not required for report writing to CICS terminal printers. Only the VSE/POWER program provides multiple copy capability.

Query/Report Writing

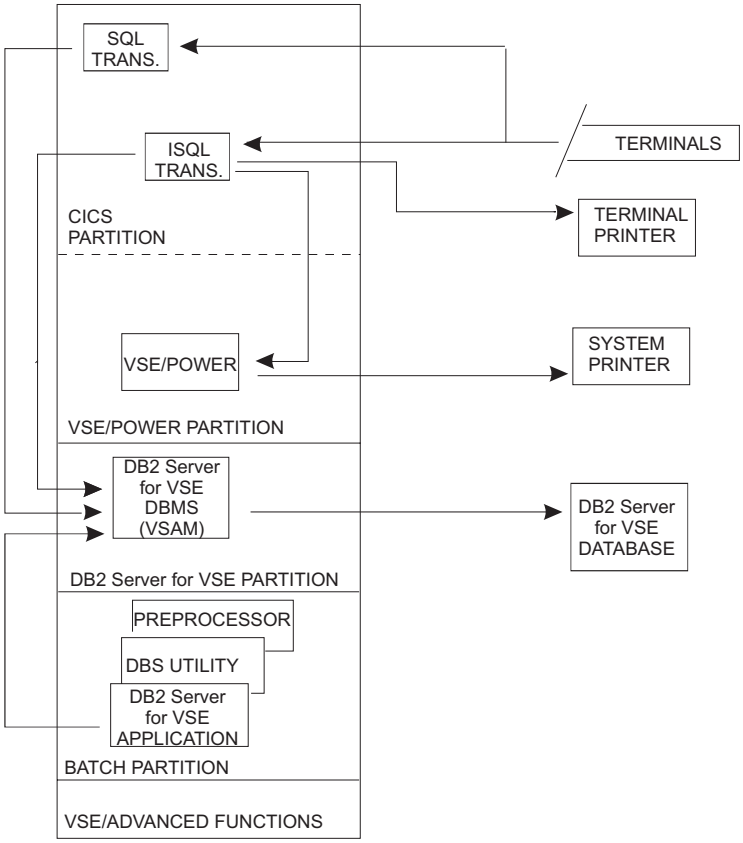


Figure 56. Query/Report Writing Configuration

The query/report-writing usage environment supports dynamic SQL query and report writing by end users. Due to the dynamic interpretation of user requests and the fact that data requests are unconstrained, this usage environment places a relatively high demand on system resources. The internal sort capability of the DB2 Server for VSE database manager is likely to be used frequently.

The program products and options required to support this environment are as follows:

1. The CICS subsystem

ISQL operates as a set of CICS transactions. The CICS subsystem should be generated with the following options:

- Dynamic Backout Program (DBP)
- Exec Interface Program (EIP)
- CICS User Exit Interface
- CICS Monitoring Facility (optional)
- CICS Restart Resynchronization, if you want to use either the SQL/DS accounting facility in an online environment or support task-related user exits.

2. VSE/POWER

   The VSE/POWER program is required for the system printer or remote workstation printer report-writing support in ISQL. It is not required for report writing to CICS terminal printers. Only the VSE/POWER program provides multiple copy capability.

Chapter 11. Stored Procedures

Stored Procedure Concepts

A stored procedure is a user-written application program that is compiled and stored at the server. When the database manager is running in multiple user mode, local applications or remote DRDA applications can invoke the stored procedure. Since the SQL statements issued by a stored procedure are local to the server, 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 the stored procedure.

There are several other benefits that can be gained through the use of stored procedures, including:

- In many applications, the integrity of the host variables used in SQL statements is critical to the business function provided by the application. For example, a debit/credit application might need to guarantee that the host variable values do not change between debit and credit operations. In these applications, the application designer would like to guarantee that sophisticated users cannot employ online debugging tools to manipulate the content of SQL statements or host variables used by the SQL application. By using stored procedures, the application designer can encapsulate the application's SQL statements into a single message to the server, which moves the sensitive processing beyond the reach of even the most sophisticated workstation user.
- Stored procedures can be used to hide the details of the database design from client applications. In addition to simplifying the writing of client applications, this means that if the database design is changed, only the stored procedure needs to be modified. The more client applications that use the stored procedure, the greater the benefit.
- Stored procedures can be used to hide sensitive data from application programs.
- Business logic can be encapsulated at the server, rather than being included in numerous application programs.
- It is easier to maintain an environment in which applications are kept at the server rather than spread across a number of requesters.

Stored Procedure Servers

The Stored Procedure Server

In DB2 Server for VSE & VM, all stored procedures are fenced, which means that they are separated from the database manager with respect to execution and memory usage. This is necessary to ensure that a stored procedure does not

- inadvertently use storage that is allocated to the database manager
- monopolize processing in the database machine or partition, which would effectively hang the database

A fenced implementation is achieved through the use of stored procedure servers. An stored procedure server is an application requester that is local to the database manager and is used to execute the stored procedure. A fenced implementation is achieved as follows:

- In VM, the stored procedure server is a separate virtual machine that is local to the DB2 Server for VSE & VM server, and uses the 'private resource' facility of VM.
- In VSE, the stored procedure server is a separate static or dynamic partition.

Note that a stored procedure server must be dedicated to a single database server.

The Stored Procedure Handler

The stored procedure handler is a DB2 Server for VSE & VM supplied utility, called ARISPRC, that interfaces between the database manager and the stored procedure. It runs in the stored procedure server and does the following:

- Initializes the runtime environment. This is done when the stored procedure server is started. For the steps involved, see "The START PSERVER Command" in the *DB2 Server for VSE & VM Operation* manual.
- Waits to receive an SQL CALL request from the database manager.
- Invokes the specified stored procedure, which will send requests and receive replies directly to and from the database manager.
- Returns the output parameters and result set information to the database after the stored procedure has terminated.
- Waits for another SQL CALL request from the database manager.

Stored Procedure Server Groups

The group clause of the CREATE PSERVER statement makes it possible to define groups of stored procedure servers. This is useful if

- certain stored procedures must always have a server available. In this case, a stored procedure server group could be dedicated to the procedure, and other procedures could share other groups.
- certain procedures have special requirements, for example, the need for unusually large amounts of virtual storage.
- certain procedures must access resources that are note required by most procedures.

The group option gives the database administrator flexibility in defining the enviroment and is useful for system tuning.

Setting up a Stored Procedure Server

The SERVGROUP column in SYSTEM.SYSROUTINES is cross-referenced with the SERVGROUP column in SYSTEM.SYSPSERVERS to establish the server that is to be used for a stored procedure.

In **DB2 Server for VM**, the following requirements exist:

- The PSERVER column in SYSTEM.SYSPSERVERS specifies the name of the stored procedure server.
- The VM machine name (user ID) of the stored procedure server must equal the value in the PSERVER column in SYSTEM.SYSPSERVERS. The CP directory of the database machine must contain the following statement:
  - IUCV *userid*

  where *userid* is the VM ID of the stored procedure server virtual machine. This enables the database manager to request the services of the stored procedure server. Note that this can also be enabled by putting an IUCV ALLOW statement in the CP directory of the stored procedure server virtual machine. However, the first method requires the database machine to have explicit access to the stored procedure server machine, and the second method allows any machine to

connect to the stored procedure server machine. Since the stored procedure server must be dedicated to a single database machine, the first method is preferred.

- The CP directory of the stored procedure server VM machine must contain the following statements:
  1. IPL CMS

     This directory control statement causes CP to start CMS in the stored procedure server virtual machine.
  2. OPTION MAXCONN *nnnn*

     This directory control statement indicates the number of IUCV and APPC/VM connections allowed for the virtual machine. Unless a stored procedure that runs on the server does work that requires additional connections, setting *nnnn* to 1 is sufficient.
- The PROFILE EXEC for the stored procedure server VM machine must have the following CMS commands:
  - SET SERVER ON
  - SET FULLSCREEN OFF
  - SET AUTOREAD OFF

  The following is a sample profile:

```
'GLOBALV INIT'  /* The following three lines have to be in PSERVER's PROFILE EXEC
'SET SERVER ON'
'SET FULLSCREEN OFF'
'SET AUTOREAD OFF'

'SET CMSTYPE HT'
'SET IMSG OFF'
'SET LANGUAGE AMENG (ADD ARI USER'

'CP SET RUN ON'                     /* This prevents CP READ upon    */
                                    /* RECONNECTing to userid.       */
'CP TERM MODE VM'                   /* Accept CMS commands.          */

'GLOBAL LOADLIB SCEERUN'            /* LAODLIB FOR LE PROGRAMS       */

'CP LINK SQLMACH 195 195 RR'        /*Link to database product disk  */
'ACCESS 195 Q'                      /*Access as Q disk               */
'EXEC SQLINIT DB(SQLMACH)'          /*Initialize as normal AR        */
IF RC <> 0 THEN DO
  SAY 'SQLINIT FAILED WITH RETURN CODE = ' RC
  'TELL SQLMACH SQLINIT FAILED IN PSERVER'
  '#CP LOGOFF'
END
ELSE
  SAY 'PSERVER INITIALIZATION COMPLETED.'
```

  Note: the PROFILE EXEC should not contain any commands that require console input, or put the ID into VM READ.
- A $SERVER$ NAMES file, which controls who can connect to the VM machine and what module to invoke when the machine is started, must exist. An example of a $SERVER$ NAMES file entry is:

```
:nick.SQLSVR01 :module.ARISPRC
               :list.SQLMACH
```

The fields in the $SERVER$ NAMES files represent the following:

– **nick**

The name of the private resource. This is VM machine name of the stored procedure server virtual machine.

– **list**

The user IDs of the users that are authorized to access the private resource. This is the VM machine name of the database server virtual machine. Since stored procedure servers must be dedicated to a single database, only one name can be specified here. The stored procedure handler will not start if more than one name is specifed.

– **module**

The name of the stored procedure handler, ARISPRC.

For complete details on setting up the machine, see "Managing Private Resources: in the *VM/ESA Connectivity Planning, Administration, and Operation* manual".

- Ensure that the Stored Procedure handler module (ARISPRC) can be accessed by the stored procedure server. The DB2 Server for VSE & VM installation process creates this module on the database machine's production disk.
- Ensure that the Stored Procedure Server has been defined to the database by using the CREATE PSERVER command. See DB2 for VSE & VM SQL Reference for more information on this command.
- All stored procedures must be LE-compliant and their load modules must reside in a disk accessible to the stored procedure servers that will invoke them.

In **DB2 Server for VSE**, the following requirements exist:

- The JCL used to start the database must contain a statement that defines logical device 098 (for example, 'ASSGN SYS098,cuu'), to enable POWER to find and execute the JCL that starts the stored procedure handler. An example is illustrated below:

```
    // ASSGN SYS098,SYSPCH
```

The following is a sample JCL to start up the DB2 Server for VSE 7.1.0 database:

```
// JOB Start DB2 for VSE in multiple user mode with Stored Procedure Support
// EXEC PROC=ARIS71PL
// EXEC PROC=ARIS71DB
// ASSGN SYS098,SYSPCH
// EXEC PGM=ARISQLDS,SIZE=AUTO,PARM='DBNAME=SAMPLE_DB'
```

Note: You will need to customize ARIS71PL, ARIS71DB to work with your local VSE/ESA environment.

- The PSERVER column in SYSTEM.SYSPSERVERS specifies the name of the stored procedure server. This name must be of datatype CHAR(8).
- The name in the PSERVER column is also the name by which the stored procedure handler identifies itself to VSE for XPCC communications.

- There must be a JCL for each stored procedure server you define. The JCL is used to start up the server's partition. The file name of the JCL and the jobname in the JOB card should both be the stored procedure server's name (as defined in the PSERVER column of SYSTEM.SYSPSERVERS). The JCL must be catalogued into a library, in the search path defined in your database startup JCL, as an A-TYPE member. The following is a sample JCL for stored procedure server SQLSVR01:

```
. $$ PUN CLASS=6,DISP=I,JNM=SQLSVR01
* $$ LST CLASS=V,DISP=D,DEST=(,SYSID01)
// JOB SQLSVR01
// OPTION NODUMP,NOSYSDUMP
// ASSGN SYS098,SYSPCH
// LIBDEF *,SEARCH=(CMPLR22.SCEEBASE,CMPLR22.LEVSEBC,
               CMPLR22.SCEECICS,PRD2.DB2710)
   ON $RC > 0 GOTO END
// EXEC PGM=ARISPRC,SIZE=1M
/.END
/*
/&
```

  Note: For the PUNCH card, you must use . $$ PUN instead of * $$ PUN. Also, you must make sure that all your stored procedure phases are in the search path defined in this JCL so that stored procedure server handler phase, ARISPRC, to be able to find and load the stored procedure.
- Stored procedure server support is currently treated as an optional feature. Hence, you will need to build the stored procedure server handler phase manually. The name of the phase is ARISPRC. The following is a sample JCL used to linkedit the ARISPRC phase. The phase should reside in your production library together with other DB2 for VSE phases.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*    L I N K    PSERVER component                                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// OPTION CATAL
   INCLUDE ARISLKHZ
// EXEC LNKEDT,PARM='MSHP'
/*
```

  Note: ARISLKHZ is a the linkbook used to linkedit the stored procedure server phase, and it is a member of your production library.
- All stored procedure will have to be compiled using LE enabled compiler. The phases of these stored procedures have to be in the search path defined in stored procedure server's startup JCL.

Managing Stored Procedure Servers

Stored Procedure Server Allocation

When a stored procedure is running, the stored procedure server in which it is executing is dedicated to it. Since no other stored procedure can execute in that server until the current one finishes, it is not possible to execute multiple stored procedures concurrently with a single stored procedure server. However, multiple stored procedures can execute concurrently if multiple stored procedure servers are defined.

The CREATE PROCEDURE statement must be used to define a stored procedure before it can be used. The CREATE PROCEDURE puts the definition of the procedure into the catalog tables SYSTEM.SYSROUTINES and SYSTEM.SYSPARMS. See *DB2 Server for VSE & VM SQL Reference* for the definitions of these tables as well as more details on the CREATE PROCEDURE statement. A third catalog table, SYSTEM.SYSPSERVERS, is used to identify the stored procedure servers. See DB2 for VSE & VM SQL Reference for the definition of this table. When the database manager is started, DB2 Server for VSE & VM loads the contents of these tables into cached structures. The cached structures contain the information from the catalog tables, as well as information about the run time status of stored procedures and stored procedure servers. When the database manager is started, the status of a stored procedure defaults to STARTED, and the status of a stored procedure server defaults to STOPPED. The database manager then issues a START PSERVER command for any stored procedure server for which the AUTOSTART value of the corresponding row in SYSTEM.SYSPSERVERS is Y. When the START PSERVER command completes, the status of that stored procedure server is STARTING. When an SQL CALL statement is issued, the database manager uses the cached information to determine the server at which this stored procedure will run. To determine the server, DB2 Server for VSE & VM does the following. Note that when these steps make reference to the catalog tables, it is the cached information from the tables that they are referring to.

1. Gets the value of the SERVGROUP column from the row in SYSTEM.SYSROUTINES for the procedure
2. Looks for the first row in SYSTEM.SYSPSERVERS in which the value of the column SERVGROUP matches the SERVGROUP value retrieved from SYSTEM.SYSROUTINES, and which is not currently running a stored procedure. If one is found, the action taken by the database manager depends on the status of the procedure server:
   - If its status is STARTED, the database manager sends a command to that server, to cause it to invoke the stored procedure.
   - If its status is STARTING, the database manager completes the START PSERVER processing, by establishing a connection to the stored procedure server and invoking the stored procedure handler. It then changes the status of the stored procedure server to STARTED and sends a command to the server, to cause it to invoke the stored procedure.
   - If its status is STOPPING, the database manager ignores that server.
   - If its status is STOPPED, the database manager checks whether that server can be started implicitly. If it can be started implicitly, DB2 Server for VSE & VM issues the START PSERVER command, and when the command completes, that server is used to run the stored procedure. For more information on starting a stored procedure server implicitly, refer to the description of the IMPLICIT/NOIMPLICIT option of the STOP PSERVER command in the *DB2 Server for VSE & VM Operation* manual.

If none of the servers in the group can be used, and the group checked was not the default group, DB2 Server for VSE & VM checks whether the stored procedure can run in the default group. Servers in the default group are indicated by a value of NULL for the SERVGROUP column in SYSTEM.SYSPSERVERS. If the DEFSERV column in SYSTEM.SYSROUTINES contains a 'Y' or is NULL, then that procedure can run in a server in the default group. If no servers are available in SERVGROUP, or if SERVGROUP in SYSTEM.SYSROUTINES is blank, and the procedure can run in the default group, DB2 Server for VSE & VM will attempt to

run the stored procedure in one of the default servers. The process it uses to find a server in the default group is the same as the one it used to look for a server in a specific group, as described earlier.

If none of the servers at which it can run are available, the stored procedure waits for a free server. If more than one stored procedure is waiting for the same server, the one that has been waiting the longest will be invoked when the server is available.

Figure 57 shows how DB2 Server for VSE & VM resolves the server name. Note that in order to illustrate the process used, the figure shows cached data. Not all of the columns shown are in the corresponding catalog table.
The following steps are involved in Figure 57:

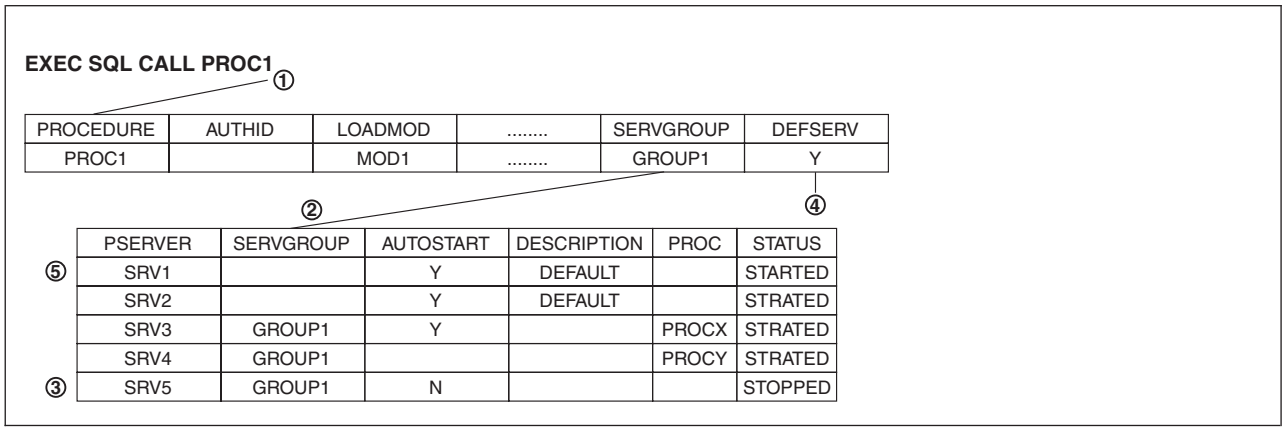


Figure 57. How DB2 Server for VSE & VM Determines the Stored Procedure Server to Use

1. SQL CALL PROC1 is issued. The database manager retrieves the SERVGROUP value from the row for PROC1 in the cached information from SYSTEM.SYSROUTINES.
2. The database manager looks for the first row in the cached information from SYSTEM.SYSPSERVERS in which the SERVGROUP column matches the SERVGROUP value retrieved from the cached information for SYSTEM.SYSROUTINES and the PROC information is blank.
3. The row for PSERVER SRV5 is found, but it is stopped and cannot be started implicitly.
4. Since there are no other servers in GROUP1 that are not already running a stored procedure, the database manager checks the column DEFSERV to see if PROC1 can run in the default server group. A value of 'Y' or NULL in this column indicates that it can run in the default group.
5. The SYSTEM.SYSPSERVERS data contains a value of NULL in the SERVGROUP column for any server in the default group. In this case, SRV1 and SRV2 are in the default server group and both are available. The database manager will use SRV1 since it is the first one found. It will update the cached information from SYSTEM.SYSPSERVERS to indicate that SRV1 is being used for PROC1, and send a command to SRV1 to cause it to invoke the stored procedure PROC1.

States of a Stored Procedure Server

A stored procedure server can be in several different states: STARTED, STARTING, STOPPING or STOPPED. Each state reflects/indicates the availability of the server and whether or not its definition can be altered or dropped. The START and STOP PSERVER operator commands can be used to change the state of a stored

procedure server. An incoming SQL CALL request can also change the state of a stored procedure server, but only in certain situations, as will be described later. Following is a description of how each state is achieved and the functions that can be performed on the server for each of these states.

STOPPED

This is the default state for all stored procedure servers when the database manager is started. It is also achieved when a procedure server is stopped using the STOP PSERVER operator command. This state has two conditions that determine whether certain functions, such as an SQL CALL implicitly starting the server, are allowed. These two conditions are IMPLICIT and NOIMPLICIT. At startup, all stored procedure servers are STOPPED with IMPLICIT. Here is a summary of the differences between the two:

- STOPPED with IMPLICT: in this state, the stored procedure server is considered to be available as it can be implicitly started by the database manager if an SQL CALL request requires it to execute. When the IMPLICIT condition is true, the stored procedure server de finition cannot be altered or dropped. A stored procedure server achieves this state when the STOP PSERVER IMPLICIT operator command is issued.
- STOPPED with NOIMPLICIT: in this state the stored procedure server is not available to execute any SQL CALL requests, and the database manager cannot implicitly start the stored procedure server. Since the server is not available, its definition can be altered or dropped. This is the only state that allows changes to the server's definition. To achieve this status issue the STOP PSERVER NOIMPLICIT operator command on the desired server. Also, if the database manager's attempt to start the server (i.e. establish a physical connection) times out, the stored procedure server will be STOPPED NOIMPLICIT by the database manager. This is done as a time-out might indicate a problem with the stored procedure server handler. See the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual for more information.

A stored procedure server will always be stopped if a severe error is encountered during the execution of a stored procedure. The IMPLICIT/NOIMPLICIT condition is not changed, unless a connection time-out has occurred, as mentioned above.

STOPPING

A server can only achieve this state if it was executing a stored procedure at the time the STOP PSERVER command was issued. When the state is ″stopping″, the database manager know that once the current SQL CALL requests concludes, the stored procedure server must be stopped. The stored procedure server will remain available as long as the IMPLICIT condition is true.

STARTING

In this state, the server is ready to start executing an SQL CALL request. It has been allocated a communication block but it is not yet connected. The database manager will establish the connection once an SQL CALL statement requests to use the server. If the connection is successful, the status will be changed to started. If it fails, the server will be stopped. The IMPLICIT/NOIMPLICIT condition will remain unchanged. If the connection times out, as mentioned earlier, the server will be stopped with the NOIMPLICIT condition true. A server in this state can also be stopped by issuing the STOP PSERVER operator command.

STARTED

In this state, the stored procedure server is connected to the database server and is either executing an SQL CALL statement, or is ready to execute an SQL CALL. This state can only be reached if an SQL CALL statement requested to use the

server. The START PSERVER operator command will not promote a stored procedure server to this state as it does not establish the physical connection with the database server. The STOP PSERVER command demotes the server to a status of STOPPED and severs its connection if it is not currently executing an SQL CALL request. If the server is in use, the STOP PSERVER command will demote it to a status of STOPPING. The database server will complete the ″stopping″ sequence once the SQL CALL request concludes, by severing the stored procedure server's connection, freeing its communication block and changing the status to STOPPED. The IMPLICIT/NOIMPLICIT conditions will be dictated by the last STOP PSERVER operator command issued against the server.

For more information on the STOP and START PSERVER commands see the *DB2 Server for VSE & VM Operation* manual. For more information on the SQL CALL statement see the following manuals:

- *DB2 Server for VSE & VM Application Programming*
- *DB2 Server for VSE & VM SQL Reference*.

Altering or Dropping a Stored Procedure Server Definition

The following describes how to remove or alter a stored procedure server:

- Issue the STOP PSERVER command for that stored procedure server, specifying the NOIMPLICIT option.
- Use the SQL DROP or ALTER PSERVER statement to delete or alter the row for that server from SYSTEM.SYSPSERVERS. Note that the row can not be deleted from SYSTEM.SYSPSERVERS if it is the only server in its group, and procedures exist that must run in that group.
- Optionally take the steps necessary to remove the stored procedure virtual machine (in VM) or redefine the partition (in VSE).

To remove a stored procedure server group, follow the steps above for each stored procedure server in the group. It is possible to remove all the servers in a group only if as there are no stored procedures defined that can use that server group. If this is the case, it is not be possible to remove the last procedure server in the group. Any stored procedures that run in the group must be moved to another group (by using the ALTER PROCEDURE satatement and specifying the SERVER GROUP clause) or dropped before the last stored procedure server in the group can be dropped.

Stored Procedures

Preparing a Stored Procedure to Run

Before a stored procedure can be invoked, it must be

- Preprocessed by the DB2 Server for VSE & VM precompiler to create a package in the database.
- Compiled by the appropriate high level, LE compliant, language compiler, or assembled.
- In VM, Linkedited and GENMODed to create a load module, which must be placed on a disk that can be accessed by the stored procedure server. In VSE, linkedited to create a phase, which must be catalogued into a library identified in the LIBDEF statement of the JCL that starts the stored procedure server.
- Defined to the SYSTEM.SYSROUTINES catalog table by the database administrator, by issuing the SQL CREATE PROCEDURE statement.
- Started, by issuing the START PROC command

If a stored procedure load module or phase is modified, a STOP PROC command must be issued, followed by a START PROC command, so that the database manager will cause the stored procedure server to load the new copy of the stored procedure load module or phase into memory.

After these steps are complete, the user that created the package associated with the stored procedure is able to GRANT RUN authority on the package to other users, allowing them to issue the SQL CALL statement to run the stored procedure.

Dropping or Altering a Stored Procedure

The following describes how to remove or alter a stored procedure:

- Issue the STOP PROC command for that procedure, specifying the REJECT option.
- Use the SQL DROP PROCEDURE statement to delete the rows for the procedure from SYSTEM.SYSROUTINES and SYSTEM.SYSPARMS, or the SQL ALTER PROCEDURE statement to change the definition in SYSTEM.SYSROUTINES.
- Optionally delete the load module or phase

Note: There is a special case where issuing the STOP PROC ACTION REJECT command will not suffice to allow altering or dropping a stored procedure definition. If the procedure is still running in a stored procedure server, you will not be allowed to alter or drop its definition, even if the status is STOP-REJ. The execution of an SQL ALTER or DROP PROCEDURE command will return with SQLCODE -15000. You can use the SHOW PSERVER and SHOW PROC operator commands to monitor the procedure's progress before you try to alter or drop its definition.

Initialization Parameters Affecting Stored Procedure Execution

PTIMEOUT Parameter

This parameter serves two purposes:

1. The number of seconds before DB2 Server for VSE & VM ceases to wait for an SQL CALL to be assigned to a stored procedure server. If the PTIMEOUT interval expires, the SQL statement fails, and SQLCODE -913 is returned with SQLSTATE 40001.
2. The number of seconds before DB2 Server for VSE & VM ceases to wait for the stored procedure server connection request to be established. If the PTIMEOUT interval expires, message ARI4168I is displayed and the connection attempt terminates. DB2 Server for VSE & VM will then try to use the next available stored procedure server, thus the SQL CALL request will not be terminated.

A value of 0 means that no PTIMEOUT is in effect. The default for PTIMEOUT is 180.

PROCMXAB Parameter

Specify the number of times a stored procedure is allowed to terminate abnormally, after which a STOP PROC ACTION REJECT is performed against the procedure and all subsequent SQL CALL statements are rejected. Note that a time-out that occurs while waiting for a stored procedure server to be assigned for an SQL CALL statement is not included in this count. The default, 0, means that the first abend of a stored procedure causes SQL CALLs to that procedure to be rejected. For production systems, you should accept the default.

Summary of Environment Interactions

Figure 58 shows the interactions between the database manager, the stored procedure server, the stored procedure handler, and the stored procedure itself. The figure does not show the definition of the stored procedure server or of the stored procedure itself; it is assumed that this has already been done.

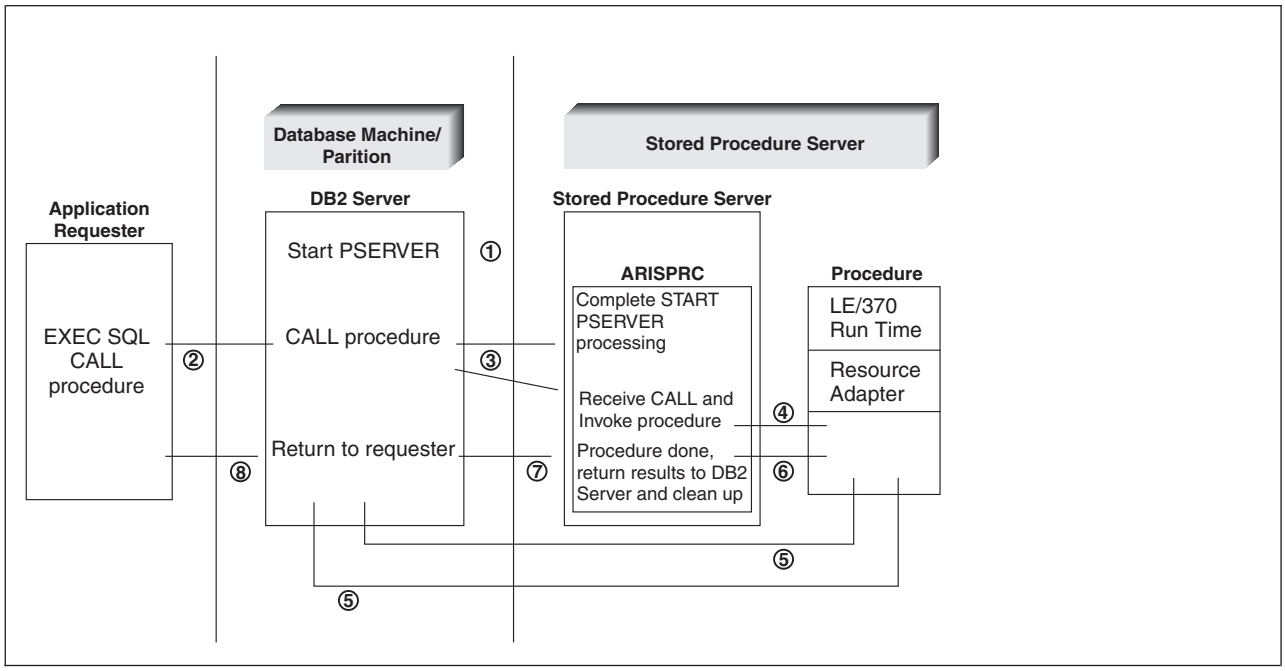


Figure 58. Stored Procedure Environment

1. If the AUTOSTART value in the cached information from SYSTEM.SYSPSERVERS is Y, the database manager starts the stored procedure server during SQLSTART processing. If the AUTOSTART value is N, then the operator issues the START PSERVER command to start the stored procedure server. The START PSERVER command changes the status of the stored procedure server to STARTING. In VM, it also allocates a pseudoagent, and in VSE, it allocates a new XPCC block, both to be used for the connection between the database manager and the stored procedure server. See "The START PSERVER Command" in the *DB2 Server for VSE & VM SQL Reference* for more details.
2. The user application at the application requester executes an EXEC SQL CALL statement.
3. An SQL application can contain one or more SQL CALL statements. The SQL CALL statement is stored in a package in the DB2 Server for VSE & VM database using the DB2 Server for VSE & VM preprocessor. When the SQL CALL statement is received, the database manager consults the cached information from SYSTEM.SYSROUTINES and SYSTEM.SYSPARMS to:
   - Determine the load module or phase associated with the stored procedure.
   - Determine the programming language used to implement the stored procedure.
   - Determine the run time options for the procedure.
   - Validate the parameter list supplied.
   - Perform any necessary data conversion between the parameters provided by the requester and the arguments required by the stored procedure.

- Determine the stored procedure server to use. See ″Stored Procedure Server Allocation″ on page 207 for more information on how DB2 Server for VSE & VM resolves this. If the stored procedure server that is found has a status of STARTING, the database manager must complete the START PSERVER processing before sending the SQL CALL statement to the stored procedure server. In this case, the database manager establishes a connection with the stored procedure server, invokes the stored procedure server, and sets the status of the stored procedure server to STARTED. When the stored procedure handler starts, it initializes the communications and run-time environments and waits for instructions from the database manager concerning which procedure to run.

The database manager hooks the agent used by the requester that issued the SQL CALL to the selected stored procedure server. The database manager must maintain the CONNECT information for the original requester as well, in order to return the result of the SQL CALL statement.

The database manager saves its environment in preparation for receiving and processing requests from the stored procedure. Finally it sends a request to the stored procedure handler to invoke the stored procedure.

4. The stored procedure handler (ARISPRC) receives the request, and does the following:
   - Sets up the parameters the stored procedure expects, using the parameters sent by the database manager.
   - Loads the procedure that is to run, if it is not already loaded from a previous execution.
   - Initializes the resource adapter environment, to ensure that no residual data is inherited from a prior execution.
   - Passes control to the stored procedure.
5. The stored procedure server effectively becomes a local requester and uses the connection that exists to communicate directly with the database manager. It uses private flows to execute the stored procedure. The database manager receives and processes the requests, and sends replies to the resource adapter. This continues until the stored procedure finishes. Note that the resource adapter is responsible for ensuring that disallowed statements are detected. For more details on the disallowed statements see ″The SQL CALL″ in the *DB2 Server for VSE & VM SQL Reference*.
6. When the stored procedure terminates, control returns to ARISPRC.
7. ARISPRC packages any output parameters and sends them to the database manager. Any cursors that were declared with the WITH RETURN option, and are left open when the stored procedure terminates, define result sets that can be fetched by the requester that issued the SQL CALL. Result sets are returned by the database server in the order the cursors were opened in the stored procedure. After sending the results to the database manager, ARISPRC cleans up the resource adapter environment and waits for the next request to invoke a stored procedure.
8. When the database manager receives the stored procedure results from ARISPRC, it hooks the agent structure back to the original requester, and passes the stored procedure results back, using DRDA flows if necessary.

Appendix A. Estimating Your Dbspace Requirements

This appendix describes procedures and calculations you can follow to determine the amount of storage to allocate to your dbspaces. You must determine:

1. The required size of each permanent dbspace, described in "Estimating Dbspace Size"
2. The storage required to hold a working set of the data, described "Estimating Internal Dbspace Size and DASD Needs for Sort Operations" on page 230.

Estimating Dbspace Size

You need to estimate data storage requirements to establish dbspace sizes. The required size of a dbspace depends on:

- The total number of tables and indexes to be stored in the dbspace
- The size of tables
- The size of indexes
- The amount of free space
- The allowance made for unused pages.

Using the above estimated values, you can calculate the required size of the dbspace by determining and adding the number of pages required for:

- The sum of the storage requirements of each table

  Refer to "Estimating Storage for a Table" on page 216. For each table the storage requirement is further described in:
  - "Estimating the Number of Header Pages" on page 218
  - "Estimating the Number of Data Pages" on page 219
  - "Estimating the Number of Index Pages" on page 227.
- An allowance for unused pages and free space.

  Setting allowances and using defaults rather than estimating the number of header and index pages is discussed in "General Guidelines" on page 215.

The following formula shows how these values are used to calculate the number of dbspace pages needed for a set of tables:

```
DBSPACE PAGES = HEADER PAGES + DATA PAGES + INDEX PAGES + ALLOWANCE
```

General Guidelines

For most dbspaces, it is sufficient to use the default value of 8 for the number of HEADER PAGES. You should also use the default of 33 percent for PCTINDEX, rather than estimate the number of index pages needed, unless you anticipate doing extensive indexing. This default reserves approximately one third of the total space for indexes. If you assume both defaults, 8 HEADER PAGES and 33 percent for PCINDEX, the above formula becomes:

```
DBSPACE PAGES = 8 + 1.50 x (DATA PAGES + ALLOWANCE)
```

The DBSPACE PAGES number derived must be rounded up to a multiple of 128. That is,

```
REQUIRED DBSPACE PAGES = TRUNC [ (DBSPACE PAGES + 127 ) / 128 ] x 128
```

The TRUNC function, for truncate operation, indicated here means to compute the value between the brackets [ ] and then use only the integer part of that value. For example, if the value calculated is 27.8, use 27.

You should allow from 50 to 200 percent for ALLOWANCE, depending on the nature of the tables to be stored. If the number of rows are relatively stable and you do not anticipate adding columns to tables (or even adding tables), adding an ALLOWANCE of 50 percent is safe. To allow all forms of growth (inserting rows, adding columns, and adding tables), you should consider an ALLOWANCE of 200 percent (2 x data pages).

Notes:

1. The ALLOWANCE in the above formulas correspond to reserved unused pages. As such, they will not use real space in the storage pool. Dbspace allocations should be substantially greater than what appears to be necessary by the above algorithm. Because dbspace pages do not occupy storage pool slots until they are loaded, there is little to be gained in making a table a tight fit in a dbspace.
2. When a dbspace can no longer contain a table, you can change the parameters of the dbspace. See "Altering the Design of a Dbspace" on page 73 for more information.

Estimating Storage for a Table

To estimate the amount of storage required for a table, consider:

- Amount (or average amount) of storage required for a row of the table
- Storage for long-field columns (average for each column)
- Number of rows the table is likely to have.

Note: For tables with variable length rows, data page requirements will depend upon the placement of the rows of different length on the pages. Some orderings will require more data pages than others. When estimating storage for these tables, refer to "Estimating Data Pages for a Table with Variable Length Rows" on page 224.

There are no guidelines for estimating the number of rows your table will have. However, for the purpose of determining the dbspace requirements, it is wise to look ahead to potential growth of the table in the foreseeable future. Consider the estimated size of the table 2 or 3 years from now, rather than its current size.

The length of a stored row can be estimated using Table 28. To complete the calculations of Table 28, you must know the type and length of all columns in your tables. If long-field columns are involved, you should first calculate the average length of long-field columns using Table 29 on page 218.

Table 28. Form for Calculating the Average Row Length of a Stored Row

COLUMN OVERHEAD	
The number of columns supporting nulls	___
The number of VARCHAR(n) columns with n≤254	___
The number of VARGRAPHIC(n) columns with n≤127	___
The number (N) of LONG FIELDS[1] N x 6	___
SUM OF COLUMN OVERHEAD FACTORS	___

Table 28. Form for Calculating the Average Row Length of a Stored Row (continued)

COLUMN DATA STORAGE FACTORS[2] INTEGER: 4 SMALLINT: 2 DECIMAL: TRUNC [PRECISION/2 + 1] FLOAT: 8 [3] FLOAT: 4 [3] CHAR(n): n GRAPHIC(n): n x 2 [4] DATE: 4 TIME: 3 TIMESTAMP: 10 VARCHAR(n) n≤254: average length VARGRAPHIC(n) n≤127: average length x 2 [4] LONG FIELDS : calculated separately (See Table 29 on page 218.) SUM OF COLUMN DATA STORAGE FACTORS	 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
ROW OVERHEAD FACTOR	8
AVERAGE LENGTH OF EACH STORED ROW (AVGROWLEN)	___

Notes:

1. The following data types are long fields: VARCHAR(n) with n>254, VARGRAPHIC(n) with n>127, LONG VARCHAR, and LONG VARGRAPHIC.
2. The factors indicated in the COLUMN DATA STORAGE FACTORS area are to be used for each column of the type specified. The sum of those factors goes on the line at the right. For example, if a table consists of 4 columns of DECIMAL data, calculate the factor for each DECIMAL column, add up those factors, and enter the sum on the line opposite DECIMAL.
3. The value 8 for FLOAT is for double-precision floating point columns (FLOAT(n) where 22≤n≤53, or n is not specified). The value 4 for FLOAT is for single-precision floating point columns (FLOAT(n) where 1≤n≤21).
4. Each graphic character occupies 2 bytes of storage. When you determine the average length of a GRAPHIC or VARGRAPHIC column in characters, multiply that number by 2 to get the number of bytes.

Column Overhead refers to descriptive information stored with an instance of the column. The overhead depends upon the characteristics of the column, as follows:

- If it is allowed to be NULL, each value has a 1-byte prefix for indication of a null entry.
- If it is a varying length character string (that is VARCHAR(n) with n≤254), each value has a 1-byte length indicator.
- If it is a varying length graphic string (VARGRAPHIC(n) with n≤127), each value has a 1-byte length indicator.

Pointers to the long-field values are stored in a special internal format that involves 6 bytes of control information in the stored row (2-byte length value and 4-byte tuple identifier (TID)).

Column Data Storage refers to the storage space occupied by the actual column values. The numbers shown are number of bytes.

- For DECIMAL data, the data is stored in a packed decimal format. Each digit (precision) occupies half a byte and the sign occupies half a byte. As the data is stored in whole bytes, you must round up to the next whole byte. If the number of digits is n, the field occupies `TRUNC [(n + 2) / 2]` bytes.
- For varying-length data columns, estimate the average length. If there is a wide variation in the individual lengths, estimate a higher number for the average. If

the rows are long, the DB2 database manager may move to new, empty, pages sooner in the loading process than otherwise is necessary.

Row Overhead is a fixed overhead for each row in the table. It consists of a 6-byte row header and a 2-byte offset into the page, for a total of 8 bytes.

Table 29. Formula for Calculating the Average Length of a Long-Field Column

LONG-FIELD VALUE OVERHEAD = (TRUNC [(average length + 3999) / 4000] x 20)
LONG-FIELD VALUE STORAGE = (TRUNC [(average length + 249) / 250] x 250)
AVERAGE LENGTH OF EACH STORED LONG FIELD (AVGCOLLEN) = LONG-FIELD VALUE OVERHEAD + LONG-FIELD VALUE STORAGE
Note: The above formula should be used to calculate the average length of a stored long field. The calculation needs to be done for each column of a table that is a long field. The following data types are long fields: VARCHAR(n) with n>254, VARGRAPHIC(n) with n>127, LONG VARCHAR, and LONG VARGRAPHIC. Graphic characters occupy 2 bytes of storage. When you determine the average length of a GRAPHIC or VARGRAPHIC column in characters, multiply that number by 2 to get the number of bytes.

LONG-FIELD Value Overhead

The value of a LONG FIELD is stored separately from the rest of the stored row. The value is stored as a chain of entries in an internal table. Each entry of the internal table is composed of 16 columns of 250 bytes each (some potentially null). Each record has 20 bytes of overhead (a 2-byte offset, a 6-byte row header, and a 12-byte unary link pointer chain). Thus, the overhead for a LONG-FIELD value depends on the actual length of the data and includes 20 bytes for each 4000-byte (16 columns of 250 bytes) entry required to store the value.

Long-Field Value Storage

Long-field values are stored in increments of 250 bytes. Each increment is one fixed-length 250-byte column value in the internal table. For example a 10-byte long-field value occupies 250 bytes of storage, plus the long-field-value overhead of 20 bytes. A 248-byte long-field value occupies 250 bytes, a 260-byte long-field value occupies 500 bytes storage, and so on.

Estimating the Number of Header Pages

The number of header pages for a dbspace can be established on the SQL ACQUIRE DBSPACE statement. In general, you should use the default value of 8 for this option.

A more precise estimate of the number of header pages follows. It is more complex than the general guidelines above, but will assist you in your calculations if you require a better estimation.

The header pages contain information of the objects defined in a dbspace. Each object defined in the dbspace, such as a table or an index, is recorded in the header pages via a control row. For more information on the types of objects that can be defined in a dbspace and the types of control rows that are inserted in the header pages, see the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual.

To estimate the number of header pages required:

1. Calculate the number of bytes required by the objects defined in the dbspace as follows:
   - Dbspace control information occupies 24 bytes.

- For each table created in the dbspace, add 32 + 2*c* bytes where *c* equals the number of columns in the table.
- For each index created on a table in the dbspace, add 20 + 2*d* bytes where *d* equals the number of indexed columns.
- For each table in the dbspace containing one or more long-field columns (LONG VARCHAR, LONG VARGRAPHIC, VARCHAR(*n*) where *n* > 254, or VARGRAPHIC(*n*) where *n* > 127), add 84 bytes.

2. Divide the total number of bytes required by 4080.
3. Round the result to the next highest integer.
4. The result is the number of header pages required for the dbspace.

For example, assume you are planning to acquire a dbspace that will contain 3 tables. Table A has 10 columns, 2 indexes each defined on a single column, and no long-field columns. Table B has 14 columns; 1 index containing 3 columns, and no long-field columns. Table C contains 3 columns, 1 index containing 1 column, and 2 long-field columns. The estimated number of header pages for this dbspace is as follows:

```
DBSPACE:                24

Table A:                32 + (2 x 10)    ← table
                        20 + (2 x 1)     ← index 1
                        20 + (2 x 1)     ← index 2

Table B:                32 + (2 x 14)    ← table
                        20 + (2 x 3)     ← index

Table C:                32 + (2 x 3)     ← table
                        20 + (2 x 1)     ← index
                        84               ← long-field columns
                        =============
Total:                  350 bytes

Divide by 4080          1 header page required.
```

Estimating the Number of Data Pages

The number of data pages required to store a table depends on whether the rows in the table are of fixed or variable length. The next section describes a method for calculating the pages required for storing tables with fixed length rows. For tables with variable length rows (rows with VARCHAR or VARGRAPHIC data), refer to "Estimating Data Pages for a Table with Variable Length Rows" on page 224.

Note: Long-field columns do not produce variable length rows because the long-field values are stored separately.

Pages Required for Storing Tables with Fixed Length Rows

The number of data pages required to hold the tables can be estimated after determining the average row lengths (AVGROWLEN) for each table and the effective page size (EPS) based on PCTFREE setting at the time the pages are to be loaded.

The number of data pages required is estimated as follows:

1. Determine the average row length (AVGROWLEN) as in Table 28 on page 216.
2. Determine the free space requirement (PCTFREE). Use a whole number for PCTFREE. That is, if the percent free is 10, use 10 for PCTFREE, not 0.10.
3. Calculate:

```
40 x PCTFREE + AVGROWLEN
```

4. Use the number calculated in step 3 to find the corresponding EPS in the following table:

Table 30. Effective Page Size Based on Free Space Requirement

40 x PCTFREE + AVGROWLEN	EPS (Effective Page Size)
8–17	4065 + AVGROWLEN
18–32	4050 + AVGROWLEN
33–52	4030 + AVGROWLEN
53–102	3980 + AVGROWLEN
103–252	3830 + AVGROWLEN
253–502	3580 + AVGROWLEN
503–1002	3080 + AVGROWLEN
1003–2002	2080 + AVGROWLEN
2003–4020	62 + AVGROWLEN
4021–4080	2 + AVGROWLEN
4081–	See note below.
Note: For the case where (40 x PCTFREE + AVGROWLEN) >= 4081: If AVGROWLEN <= 4080, number of rows per page = 1 and EPS = AVGROWLEN. If AVGROWLEN > 4080, row size exceeds DB2 limits. Reduce your row size and recalculate.	

5. Calculate the number of rows per page:

```
Rows per Page = MINIMUM (256, TRUNC [ EPS/AVGROWLEN ] )
```

6. Calculate the number of data pages required as follows:

   If the average long-field length is <= 4020, then:

```
REQUIRED     Number of Rows   Number of Rows x Number of Long Fields
DATA      =  -------------- + -------------------------------------
PAGES        Rows per Page     4020 / Average Long-Field Length
```

   Note: When evaluating the expression, truncate the denominator *4020/average long-field length* to the nearest integer and round up the results of both division expressions to the nearest integer before adding them.

   If the average long-field length is >4020:

```
REQUIRED     Number of Rows
DATA      =  -------------- + Number of Rows x Number of Long Fields
PAGES        Rows per Page

               Average Long-Field Length
             x -------------------------
                         4020
```

   Note: Round the results of both division terms up to the nearest integer before evaluating the expression.

If you are loading tables separately, calculate the number of pages required for each table separately. If you are loading the tables in an interleaved fashion, use the longest AVGROWLEN of all the tables in determining the Effective Page Size.

Notes:

1. Storage for long fields (LONG VARCHAR, LONG VARGRAPHIC, VARCHAR(n) with n>254, VARGRAPHIC(n) with n>127) columns must be

calculated apart from the rest of the row. AVGROWLEN will include six bytes for each long-field value. However, storage for the actual long field will be calculated separately.

2. If you have already established a database and are defining a new dbspace, you can get an estimate of the data pages required by modeling the dbspace. That is, create the tables in a test dbspace and load a sample of the data. Then you can issue queries against SYSTEM.SYSDBSPACES and SYSTEM.SYSCATALOG to find out how many pages were required for the data sample. The data for the real tables will be proportionately higher. When modeling data, avoid using nulls in the sample. Nulls tend to produce low results.

Examples of Estimating the Number of Data Pages

Example 1: The example work sheet shown in Table 31 is for a table that has just one CHAR(100) column supporting nulls.

Table 31. Example 1 — Calculating the Average Row Length

COLUMN OVERHEAD	
The number of columns supporting nulls	1
The number of VARCHAR(n) columns with n≤254	0
The number of VARGRAPHIC(n) columns with n≤127	0
The number (N) of Long Fields	
• N x 6	0
SUM OF COLUMN OVERHEAD FACTORS	1
COLUMN DATA STORAGE FACTORS	
• INTEGER: 4	0
• SMALLINT: 2	0
• DECIMAL: TRUNC [PRECISION/2 + 1]	0
• FLOAT: 8 (for double-precision)	0
• FLOAT: 4 (for single-precision)	0
• CHAR(n): n	100
• GRAPHIC(n): n x 2	0
• DATE: 4	0
• TIME: 3	0
• TIMESTAMP: 10	10
• VARCHAR(n): average length	0
• VARGRAPHIC(n): average length x 2	0
• Long Fields: calculated separately (See Table 29.)	0
SUM OF COLUMN DATA STORAGE FACTORS	100
ROW OVERHEAD FACTOR	8
AVERAGE LENGTH OF EACH STORED ROW	109

The number of DATA PAGES required to load 25000 rows into this table in a dbspace defined to have 10 % free space is:

1. Determine the Average Row Length.

   ```
   AVGROWLEN = 109
   ```

2. Determine the Free Space Requirement.

   ```
   PCTFREE = 10
   ```

3. Calculate 40 x PCTFREE + AVGROWLEN.

   ```
   40 x 10 + 109 = 509
   ```

4. From Table 30 on page 220, determine the Effective Page Size (EPS), using the number calculated in step 3 to find the corresponding EPS.

```
503 - 1002    .....    3080 + AVGROWLEN

EPS = 3080 + 109 = 3189
```

5. Calculate the number of rows per page.

```
Rows per Page = MINIMUM (256, TRUNC [ 3189/109 ] ) = 29
```

6. The number of data pages required is:

```
Number of Rows   Number of Rows x Number of Long Fields
-------------- + -------------------------------------  = 863
Rows per Page       4020 / Average Long-Field Length
```

Example 2: The example work sheet shown in Table 32 is for a table that has:

- 2 DECIMAL(6,0) columns supporting nulls (4 bytes each)
- 1 DECIMAL(9,0) column defined as NOT NULL (5 bytes)
- 1 INTEGER column defined as NOT NULL (4 bytes)
- 1 SMALLINT column supporting nulls (2 bytes)
- 1 CHAR(3) column supporting nulls (3 bytes)
- 1 CHAR(4) column supporting nulls (4 bytes)
- 1 GRAPHIC(10) column defined as NOT NULL (20 bytes)
- 1 DATE column supporting nulls (4 bytes)
- 1 TIME column defined as NOT NULL (3 bytes)
- 2 VARCHAR(10) columns supporting nulls (average 8 bytes each)
- 1 VARCHAR(15) column supporting nulls (average 12 bytes)
- 1 VARCHAR(250) column supporting nulls (average 32 bytes)
- 1 VARGRAPHIC(15) column supporting nulls (average 12 characters or 24 bytes)

Table 32. Example 2 — Calculating the Average Row Length of a Stored Row

COLUMN OVERHEAD	
The number of columns supporting nulls	11
The number of VARCHAR(n) columns with n≤254	4
The number of VARGRAPHIC(n) columns with n≤127	1
The number (N) of Long Fields	
• N x 6	0
SUM OF COLUMN OVERHEAD FACTORS	16
COLUMN DATA STORAGE FACTORS	
• INTEGER: 4	4
• SMALLINT: 2	2
• DECIMAL: TRUNC [PRECISION/2 + 1]	13
• FLOAT: 8 (for double-precision)	0
• FLOAT: 4 (for single-precision)	0
• CHAR(n): n	7
• GRAPHIC(n): n x 2	20
• DATE: 4	4
• TIME: 3	3
• TIMESTAMP: 10	0
• VARCHAR(n): average length	60
• VARGRAPHIC(n): average length x 2	24
• Long Fields: calculated separately	
SUM OF COLUMN DATA STORAGE FACTORS	137
ROW OVERHEAD FACTOR	8
AVERAGE LENGTH OF EACH STORED ROW	161

The number of DATA PAGES required to load 600 rows into this table in a dbspace defined to have 15 percent free space is:

1. Determine the Average Row Length.

   ```
   AVGROWLEN = 161
   ```

2. Determine the Free Space Requirement.

   ```
   PCTFREE = 15
   ```

3. Calculate 40 x PCTFREE + AVGROWLEN.

   ```
   40 x 15 + 161 = 761
   ```

4. From Table 30 on page 220, determine the Effective Page Size (EPS), using the number calculated in step 3 to find the corresponding EPS.

   ```
   503 - 1002    .....    3080 + AVGROWLEN
   EPS = 3080 + 161 = 3241
   ```

5. Calculate the number of rows per page.

   ```
   Rows per Page = MINIMUM (256, TRUNC [ 3241/161 ] ) = 20
   ```

6. The number of data pages required is:

   ```
   Number of Rows   Number of Rows x Number of Long Fields
   -------------- + ------------------------------------  = 30
   Rows per Page       4020 / Average Long-Field Length
   ```

Usually, you store a table this small in a dbspace with other tables. If a dbspace has more than one table, the total number of DATA PAGES required for the dbspace is the sum of the data page requirements of all the tables in the dbspace.

Example 3: The example work sheets shown in Table 33 and Table 34 on page 224 are for a table that has:

- 2 DECIMAL(6,0) columns supporting nulls (4 bytes each)
- 1 DECIMAL(9,0) column defined as NOT NULL (5 bytes)
- 1 INTEGER column defined as NOT NULL (4 bytes)
- 1 SMALLINT column supporting nulls (2 bytes)
- 1 CHAR(3) column supporting nulls (3 bytes)
- 1 CHAR(4) column supporting nulls (4 bytes)
- 1 GRAPHIC(10) column defined as NOT NULL (20 bytes)
- 2 DATE columns supporting nulls (4 bytes each)
- 1 TIMESTAMP column defined as NOT NULL (10 bytes)
- 2 VARCHAR(10) columns supporting nulls (average 8 bytes each)
- 1 VARCHAR(15) column supporting nulls (average 12 bytes)
- 1 VARGRAPHIC(15) column supporting nulls (average 12 characters or 24 bytes)
- 1 LONG VARCHAR column supporting nulls (average 32 bytes)

Table 33. Example 3 — Calculating the Average Row Length of a Stored Row

COLUMN OVERHEAD	
The number of columns supporting nulls	12
The number of VARCHAR(n) columns with n≤254	3
The number of VARGRAPHIC(n) columns with n≤127	1
The number (N) of Long Fields	
• N x 6	6
SUM OF COLUMN OVERHEAD FACTORS	22

Table 33. Example 3 — Calculating the Average Row Length of a Stored Row (continued)

COLUMN DATA STORAGE FACTORS	
• INTEGER: 4	4
• SMALLINT: 2	2
• DECIMAL: TRUNC [PRECISION/2 + 1]	13
• FLOAT: 8 (for double-precision)	0
• FLOAT: 4 (for single-precision)	0
• CHAR(n): n	7
• GRAPHIC(n): n x 2	20
• DATE: 4	8
• TIME: 3	0
• TIMESTAMP: 10	10
• VARCHAR(n): average length	28
• VARGRAPHIC(n): average length x 2	24
• Long Fields: calculated separately (See Table 34.)	
SUM OF COLUMN DATA STORAGE FACTORS	116
ROW OVERHEAD FACTOR	8
AVERAGE LENGTH OF EACH STORED ROW	146

Table 34. Example 3 — Calculating the Average LONG VARCHAR Stored Length

LONG VARCHAR VALUE OVERHEAD The number(N) of LONG VARCHAR columns • N x (TRUNC [(average length + 3999) / 4000] x 20)	20
LONG VARCHAR VALUE STORAGE TRUNC [(average length + 249) / 250] x 250	250
AVERAGE LENGTH OF EACH STORED LONG VARCHAR	270

The number of DATA PAGES required to load 25000 rows into this table in a dbspace defined to have 10 percent free space is:

1. Determine the Average Row Length.

   ```
   AVGROWLEN = 146
   ```

2. Determine the Free Space Requirement.

   ```
   PCTFREE = 10
   ```

3. Calculate 40 x PCTFREE + AVGROWLEN.

   ```
   40 x 10 + 146 = 546
   ```

4. From Table 30 on page 220, determine the Effective Page Size (EPS), using the number calculated in step 3 to find the corresponding EPS.

   ```
   503 - 1002    .....    3080 + AVGROWLEN
   EPS = 3080 + 146 = 3226
   ```

5. Calculate the number of rows per page.

   ```
   Rows per Page = MINIMUM (256, TRUNC [ 3226/146 ] ) = 22
   ```

6. The number of data pages required is:

   ```
   Number of Rows   Number of Rows x Number of Long Fields
   -------------- + -------------------------------------  = 2923
   Rows per Page       4020 / Average Long-Field Length
   ```

Estimating Data Pages for a Table with Variable Length Rows

The following methods provide estimates for tables containing variable length data with data types VARCHAR and VARGRAPHIC. Tables with columns containing

variable length data types result in rows of differing lengths that can be distributed throughout a dbspace in different ways depending upon the order in which data is loaded. The distribution of the variable length rows in the dbspace can significantly affect the number of data pages occupied by a table.

There are three different methods you can use to more accurately estimate the data page requirements for tables with variable length rows:

Modeling An approach using a test dbspace containing a test table that contains a representative sample of the data. The accuracy of this estimates depends solely on the representativeness of the test table.

Worst case An approach that provides an estimate of the number of data pages that will accommodate the table regardless of the order of the rows. This approach will overestimate the number of pages in many cases, but will always ensure that you have allocated enough pages.

Splitting An approach requiring an approximation of the number of rows that fall within a range of row lengths. This may produce a more realistic number of pages than the worst case method but does not ensure that the table will fit.

Estimating Data Pages with Modeling: Establish a database before you begin modeling your data page requirements. Then, do the following:

1. Acquire a test dbspace.
2. Create the table in the dbspace and load a sampling of the data into it.
3. Ensure that the statistics for the table are up-to-date. If the statistics are not current, this can be done by performing a load (with statistics set on), or by performing an explicit UPDATE STATISTICS on the table.
4. Get the NACTIVE value for this dbspace from the SYSDBSPACES catalog table. Since there is only one table (and its companion table, if there is one or more long fields) in the dbspace, then the NACTIVE value indicates the number of data pages this table is currently using.
5. Multiply the NACTIVE value by a factor representing the relationship between the actual table size and this test table size. The result is an estimate of the number of data pages the actual table requires.

Consider the following when modeling your data page requirements in this way:

- Design a test table large enough to cover at least several data pages.
- Before creating the test table in the test dbspace, drop the dbspace and acquire it again. This ensures that previous use of the dbspace does not affect your modeling results.
- Try to arrange the various lengths of the rows in a sequence as close as possible to what you expect from your real table.

Estimating Using the Worst Case Method: This method is the safest method to use; it will ensure that you have enough pages regardless of the distribution of the rows in the table. However, it may overestimate your requirements.

To use this method you need to know the:

- Length of the longest row in the table
- Average length of a row in the table
- Number of rows in the table.

Then do the following:

1. Calculate the maximum row length (MAXROWLEN) by using the maximum length of the VARCHAR and VARGRAPHIC columns instead of the average length as shown in Table 28 on page 216.
2. Substituting MAXROWLEN for AVGROWLEN, perform steps 1 to 4 of the formula for estimating the number of pages as shown in "Estimating the Number of Data Pages" on page 219. This produces the Effective Page Size for the MAXROWLEN (denoted EPSmax).
3. Estimate the average lengths of columns in your table and calculate the average row length (AVGROWLEN) as described in Table 28 on page 216.
4. Calculate the worst case estimate using the following formula:

```
                                          AVGROWLEN x Number of Rows
Worst Case =  MINIMUM ( Number of rows, -------------------------- )
                                            EPSmax - MAXROWLEN + 1
```

Example using the Worst Case Method:

Consider a 500,000 row table being loaded into a dbspace with PCTFREE=10. Assume the overall AVGROWLEN value is 50 bytes. Assume the calculated MAXROWLEN value is 110 bytes for this table.

Calculate the EPSmax value as follows:

```
40 x PCTFREE + MAXROWLEN = 40 x 10 + 110 = 510
```

The corresponding EPSmax is 3190.

Substitute the values in the worst case formula:

```
                    50 x 500000
MINIMUM (500000, --------------) = MINIMUM (500000, 8114.2)
                  3190 - 110 + 1
```

To store this table you need at most 8115 data pages.

Estimating Using the Splitting Method: This method assumes that you can approximate the frequency of different ROWLENGTHs in the table to be stored. This method is as follows:

1. Split the set of all rows into several ROWLENGTH groups and calculate page requirements for each group as if it were a separate table using the formula described in "Estimating the Number of Data Pages" on page 219.
2. Add the page requirements for the groups to estimate the total table page requirements.

Try several different groupings of rows, making sure that each group is large enough to cover several data pages. If groups of rows do not cover several data pages, the estimate could be too high.

Different groupings will give different results. Select the highest overall page estimate to ensure that your estimate includes a contingency.

Example Using the Splitting Method:

Consider a 2000 row table to be loaded into a dbspace with PCTFREE=0. Assume the overall AVGROWLEN value to be 1000 bytes. Assume, also, that 25 percent of the rows in the table are longer than 800 bytes, with an AVGROWLEN value = 2500 bytes. The remaining 75 percent of the rows are less than 800 bytes long, with an AVGROWLEN value = 500 bytes.

We consider two groups of rows for this calculation:

Group 1
with 1500 rows and AVGROWLEN = 500

Group 2
with 500 rows and AVGROWLEN = 2500.

Perform the calculations described in "Estimating the Number of Data Pages" on page 219 treating each group as a table.

For Group 1 with AVGROWLEN = 500 and PCTFREE = 0, the EPS is 4080. Therefore we can fit 8 rows per page (4080/500 = 8.16) and we need 188 pages (1500/8 = 187.5) to store the 1500 rows.

For Group 2 with AVGROWLEN = 2500 and PCTFREE = 0, the EPS is 2562. Therefore we can fit 1 row per page (2562/2500 = 1.02) and we need 500 pages (500/1 = 500) to store the 500 rows.

Adding these two page requirements together gives an overall estimate of 688 data pages (188 + 500) to store the whole 2000 row table.

Compare this to the result if you used the formula for fixed length rows. Using only the formula described in "Pages Required for Storing Tables with Fixed Length Rows" on page 219 and the overall AVGROWLEN of 1000, the EPS is 4080. Therefore we can fit 4 rows per page (4080/1000 = 4.08) and we need 500 pages (2000/4 = 500) to store all 2000 rows. This is considerably less than the 688 pages estimated. The real number of data pages required is likely between 500 and 688 depending on the order in which the rows are being stored in the dbspace.

Estimating the Number of Index Pages

Generally speaking, you should take the default allocation for index pages in the dbspace (PCTINDEX=33). This is means that the number of index pages is approximately DATA PAGES / 2. This leaves you considerable freedom to vary the indexing you do on the tables in the dbspace. Another way of looking at this is that if the number of index pages is more than half the number of data pages, you may be trying to support too many indexes on the tables in the dbspace. As a result, you may experience performance problems on INSERT, UPDATE, and DELETE operations.

However, if the data in the dbspace is largely used for read only operations, you may want to create a lot of different indexes. If this is the case, you may need more index pages than is provided for by the default PCTINDEX value of 33 percent. For such read only (or read mostly) cases, you may want to do the detailed analysis of index page requirements to determine the appropriate PCTINDEX value based on the size of indexes you plan on supporting.

If the data in the dbspace is to have very few indexes with rather small key lengths, then you may want to do the detailed analysis of index page requirements to determine an appropriate PCTINDEX value that is smaller than the default.

The formula for calculating the appropriate PCTINDEX value is:

$$\text{PCTINDEX} = \frac{\text{TOTAL INDEX PAGES}}{\text{HEADER PAGES} + \text{DATA PAGES} + \text{TOTAL INDEX PAGES}}$$

In this formula, TOTAL INDEX PAGES is the sum of the number of index pages required for each planned index.

The next section provides the guidelines for estimating the number of index pages required for an index.

Estimating the Size of an Index

Index storage is allocated in pages. Each page contains data for only one index. The minimum storage required for any index is one page.

To estimate the amount of storage required for an index, you must consider the type of information in the index key and the amount of information in the table being indexed. The following table information must be considered for calculating the size of an index:

- The number of rows in the table
- The length of a key value
- Whether the key is variable or fixed in length
- The number of distinct keys in the table

  For indexes that are not unique, this result may be less than the total number of rows in the table. Each entry in a leaf page of the index consists of a key value and one or more row pointers, called Tuple Identifiers or TIDs, for the row having this key value.

  For unique indexes, each entry contains just one TID.

  These entries are called clusters.
- The amount of free space (PCTFREE) defined for the index. The PCTFREE value is the percentage of free space to be left on index pages during index creation.

For fixed length unique key indexes, the following calculations for index size will be accurate. For variable length or non-unique indexes, the calculations may either overestimate or underestimate the size of an index.

Generally, the size may be overestimated if:
- A large variable length column is the last column in the index or
- There are a large number of duplicates in the index.

The size may be underestimated if the varying length keys are not evenly distributed. For example, if the ordering of the keys in the index is from shortest to longest, then the lengths are not evenly distributed and space will be underestimated.

To calculate the size of the index perform the following steps:

1. Calculate the Effective Index Page Size

   The Effective Index Page Size (EIPS) is similar to the effective page size calculated for data pages. The formula for index pages differs for fixed length and variable length index keys.

   For fixed length index keys:

   ```
   EIPSmax = 4056 - (41 x PCTFREE)
   ```

   For variable length index keys:

   a. Calculate the maximum encoded length of each variable length column in the index (in bytes).

      For a short VARCHAR column, if it is the last column in the key,

      ```
      VARCOL(n) = maximum length of column
      ```

      Otherwise,

```
VARCOL(n) = (INTEGER((max length of column - 1) / 4) + 1) * 5
```

For a short VARGRAPHIC column, if it is the last column in the key,

```
VARCOL(n) = 2*(maximum length of column)
```

Otherwise,

```
VARCOL(n) = (INTEGER((2*max length of column-1)/4)+1) * 5
```

b. Calculate the maximum length of a key

```
KEYLENmax = the sum of the lengths of fixed columns (in bytes)
            + VARCOL(1) + ... + VARCOL(n)
            + 1 for the length of the key
            + 1 for each column that allows nulls
```

c. Use this KEYLENmax value to calculate the maximum length of a cluster with 1 TID.

```
CLUSTERmax = KEYLENmax
             + 1 for number of TIDs
             + 4 for the TID
```

d. Use this CLUSTERmax value to calculate the minimum effective index page size for leaf pages.

```
EIPSminleaf = 4056 - (41 x PCTFREE) - CLUSTERmax + 1
```

e. Use the KEYLENmax value again to calculate the maximum length of a nonleaf pair.

```
PAIRLENmax = KEYLENmax
             + 3 for the page number
             + 4 (if index is not unique)
```

f. Use this PAIRLENmax value to calculate the minimum effective index page size for nonleaf pages.

```
EIPSmin-nonleaf = 4056 - (41 x PCTFREE) - PAIRLENmax + 1
```

2. Calculate the average number of rows per key value.

   The average number of rows identified in one cluster is:

```
NUMBER_KEYS = Number of distinct keys

ROWSPERCLUSTER = Number of rows in table
                 ----------------------
                     NUMBER_KEYS
```

   If ROWSPERCLUSTER is greater than 255, then the key must be duplicated. In this case, the following calculations must be done:

```
NUMBER_KEYS = (TRUNC [1 + (ROWSPERCLUSTER/255) ]) x NUMBER_KEYS

                 Number of rows in table
ROWSPERCLUSTER = ----------------------
                       NUMBER_KEYS
```

3. Calculate the average length of a key value.

   a. Calculate the average encoded length of each variable length column in the index, if any, in bytes.

```
VARCOLavg(n) = (1.25 x average length of column) + 3
```

      These numbers must be rounded up to integer values.

      Once again, when determining the length of graphic data, allow 2 bytes for each character.

   b. Calculate the average length of a key in the index.

```
KEYLEN = the sum of the lengths of fixed columns (in bytes)
       + VARCOLavg(1) + ... + VARCOLavg(n)
       + 1 if there are any variable-length columns
       + 1 for each column that allows nulls
```

4. Calculate the cluster size for the index, using the value of ROWPERCLUSTER from step 2 on page 229.

   ```
   CLUSTERSIZE = 1 + KEYLEN + (4 x ROWSPERCLUSTER)
   ```

5. Calculate the number of keys that can be put on a leaf page, using the value of CLUSTERSIZE from step 4.

   ```
   #KEYSleaf = TRUNC [EIPS/CLUSTERSIZE]
   ```

   where EIPS is EIPSmax for an index with fixed length keys or EIPSminleaf for an index with variable length keys.

6. Calculate the number of leaf pages, using the the values of NUMBER_KEYS from step 2 on page 229 and #KEYSleaf from step 5:

   ```
   LEAF PAGES = TRUNC [1 + (NUMBER_KEYS/#KEYSleaf)]
   ```

7. Calculate the length of a nonleaf page entry with the value of KEYLEN from step 3b on page 229.

   ```
   PAIRLEN = KEYLEN + 3
                   + 4 (if index is not unique)
   ```

8. Use the value of PAIRLEN from step 7 to calculate the number of keys that can be put on a nonleaf page.

   ```
   #KEYSnonleaf = TRUNC [EIPS/PAIRLEN]
   ```

   where EIPS is EIPSmax for an index with fixed length keys or EIPSmin-nonleaf for an index with variable length keys.

9. Calculate the number of nonleaf pages required at each level, using the value of LEAF PAGES from step 6.

   ```
   level = 1
   NONLEAF PAGES(level)=
        TRUNC [1 + (LEAF PAGES/KEYSnonleaf)]
   ```

   While the number of nonleaf pages at the current level is greater than 1, do the following:

   ```
   level = level + 1
   NONLEAF PAGES(level) =
        TRUNC [1 + (NONLEAF PAGES(level-1)/KEYSnonleaf)]
   ```

10. Calculate the total number of index pages by adding the LEAF PAGES value from step 6 and the nonleaf pages for every level as calculated in step 9.

    ```
    INDEX PAGES = LEAF PAGES + NONLEAF PAGES(1) + ... + NONLEAF PAGES(n)
    ```

Estimating Internal Dbspace Size and DASD Needs for Sort Operations

Internal dbspaces are most commonly used as work areas for sorting data. It is helpful to predict the amount of space needed to perform a sort, in order to estimate how big your internal dbspaces should be.

This section will discuss how much space is required to perform a particular sort. Since multiple users can be performing a sort concurrently, it is more difficult to determine the maximum internal dbspace requirements for your database than for a given sort. This maximum depends both on the expected size of a sort, as well as how many sorts are expected to be occurring concurrently. You must also consider non-sort usage of internal dbspaces, such as to contain materialized views or intermediate query results. Refer to the *DB2 Server for VM System Administration* or

DB2 Server for VSE System Administration manual for more information about internal dbspace usage, including guidelines for determining the number and size of internal dbspaces for your database.

The size of internal dbspaces in a database is often dictated by the largest sort operation possible in that database, such as the sort needed to create an index on the largest table in the database.

When Do We Sort?

Sorting is performed whenever an operation requires that data be ordered or that duplicate values be eliminated, and no appropriate index exists that provides the necessary ordering. Even if an appropriate index exists, the Optimizer may decide not to use it.

In most cases, it is readily apparent where a sort can occur. The following is a list of all cases:

- Index creation, such as a result of the CREATE INDEX statement, the adding or activating of a PRIMARY KEY or UNIQUE CONSTRAINT, or the reorganizing of an invalid index. We sort on the index or key/constraint columns.
- UNION statement. We sort on the columns listed in the SELECT list of the queries being unioned. The sort eliminates duplicate values. (No duplicate elimination occurs for UNION ALL.)
- ORDER BY and GROUP BY clauses. Both these clauses request that data be ordered. We sort on the columns or expressions (ORDER BY can sort on the result of an expression in the SELECT list) listed in the clause.
- DISTINCT clause. This is another case of sorting to eliminate duplicate values. We sort on the columns listed in the clause.
- Merge/scan (type 2) join. This type of join requires that the columns on which we are joining be ordered. We sort on the join columns.

You can use the EXPLAIN command if you are unsure whether or not a particular query performs a sort. If you query performs more than one of the above, then it may perform multiple sorts. If you UNION or join more than two tables, another sort may be performed for each additional table, since we UNION and join tables two at a time.

For further information on sorting, refer to the *DB2 Server for VSE & VM Diagnosis Guide and Reference* manual.

Internal Dbspace Characteristics

The characteristics of an internal dbspace are different from those of a permanent dbspace:

- Each page is 4096 bytes.
- No free space is reserved on pages.
- There is no space at the end of a page reserved for pointers to each row on the page, and the limit of 256 rows per page is removed.
- There is always exactly one header page.
- Pages of internal dbspaces are never shadowed when they are modified.

There are no free space classes for internal dbspace pages, since data is always added at the end, and hence there is never a need to search for free space in which to store a row. This avoids the space wastage which can occur due to the

granularity of free space classes (a row will be stored on a page in an internal dbspace whenever the page has enough free bytes to hold the row).

You can see that internal dbspaces are much simpler than permanent dbspaces. Calculating the number of pages needed to hold a certain amount of data is also simpler.

Calculating Internal Dbspace Size Requirements

We will calculate the amount of space required to hold a copy of the working set of data during a sort. Specifically, we will calculate the size of the initial working set, since the working set can only get smaller due to the elimination of duplicate values. In building the initial working set, we retrieve a portion of the input data (enough to fill an internal sort buffer), sort it, and write the sorted portion to an internal dbspace. Duplicates may be eliminated during the sort. We will not consider the effects of duplicate elimination in these calculations, since these effects are dependent on the order in which data is encountered.

The following steps calculate the size of a sort row. The sort row is made up mostly of the columns by which we are ordering, that is the sort key, plus any other columns which must appear in the result.

1. Calculate the average length of a sort key.
   a. Calculate the average encoded length of each variable length ordering column (in bytes). The average length should not include trailing blanks (if any) since these blanks are not stored in the sort key.

      ```
      VARCOLavg(n) = (1.25 x average length of varying-length
                     ordering column n) + 3
      ```

      These numbers must be rounded up to integer values.

      The encoding of varying-length values incurs an overhead of approximately 25 percent.

   b. Calculate the average length of a sort key.

      ```
      SORTKEYLEN = the sum of the lengths of fixed-length ordering columns
                     (in bytes)
                   + VARCOLavg(1) + ... + VARCOLavg(n)
                   + 1 for each ordering column that allows nulls
      ```

      For index creation, the TID of the data row is part of the sort key/row. If the sort is for index creation:

      ```
      SORTKEYLEN = SORTKEYLEN + 4
      ```

2. Calculate the average length of a sort row. We add overhead for the sort row header, plus add any non-ordering columns which must appear in the result. There are no non-ordering columns for index creation. Non-ordering columns are not encoded.

   ```
   SORTROWLEN = SORTKEYLEN
                + 3 bytes (sort row header)
                + the sum of the lengths of fixed-length non-ordering columns
                  (in bytes)
                + the sum of the average lengths of varying-length non-ordering
                  columns (in bytes)
                + 1 for each non-ordering column which allows nulls
   ```

   For cases other than index creation, where the sort key contains at least one varying-length column, there will be the following additional overhead:

- A one-byte counter will indicate the number of varying-length key columns containing trailing blanks. This counter is used even if none of the columns contain trailing blanks. In this case it is set to zero.

  ```
  SORTROWLEN = SORTROWLEN + 1
  ```

- In addition to the column counter, the number of trailing blanks that each column originally had is recorded. If at least one column had trailing blanks, then a one-byte counter is allocated for each varying-length column.

  ```
  SORTROWLEN = SORTROWLEN + number of varying-length sort key columns
  ```

  If the data does not contain trailing blanks, then this overhead is not incurred.

3. Adjust for varying-length data.

   For varying length sort rows, the order in which rows are encountered and stored can affect the number of pages required. To account for this possibility, we can use a method similar to the Effective Index Page Size used in calculating the size of an index. Briefly, this method models the worst case where the last sort row we try to insert into a page is the largest possible sort row, and the page has one fewer bytes of free space available. This gives us our maximum space wastage per page, and will yield the upper bound on the number of pages we will use. To determine the Effective Internal Dbspace Page Size (EIDPS), do the following:

   a. For fixed-length data

      ```
      EIDPS = 4080
      ```

   b. For varying-length data, repeat the previous calculations to determine SORTROWLEN, substituting the maximum length of varying-length columns for the average length. This gives us MAX SORTROWLEN.

      ```
      EIDPS = 4080 - (MAX_SORTROWLEN + 1)
      ```

The following steps will calculate the number of pages required to hold a copy of all sort rows. This is the minimum size of internal dbspace that is required to perform the sort.

1. First calculate how many rows will fit on a page.

   ```
   ROWS_PER_PAGE = TRUNC [EIDPS/SORTROWLEN]
   ```

2. Determine the number of sort rows.

   For index creation, the number of sort rows is the same as the number of rows in the table. For cases where only a subset of the rows in a table participates in a sort, then the number of participating rows must be estimated based on your knowledge of the query and the contents of the table.

   ```
   NROWS = number of rows expected to participate in the sort
   ```

3. Compensate for effect of sort buffering

   A block of input rows is encoded and stored in a sort buffer. The contents of this buffer are then sorted and written out to pages of an internal dbspace. Since the buffer is large enough to fill several internal dbspace pages, and the space in the buffer is contiguous while each internal dbspace page has a header, then we must account for this in determining the number of pages required.

   a. Calculate how many rows are in the block of rows that would fill the sort buffer.

      ```
      ROWS_PER_BLOCK = TRUNC [40948 / SORTROWLEN]
      ```

   b. Calculate how many internal dbspace pages would be filled by a block of rows.

      ```
      PAGES_PER_BLOCK = ROWS_PER_BLOCK / ROWS_PER_PAGE
      ```

This number must be rounded up to an integer value.

c. Calculate how many full blocks the expected number of sort rows would generate.

```
FULL_BLOCKS = TRUNC [NROWS / ROWS_PER_BLOCK]
```

d. Calculate how many rows would be in the last (not full) block.

```
ROWS_LAST_BLOCK = NROWS - (ROWS_PER_BLOCK x FULL_BLOCKS)
```

4. Finally, using all the information we have derived so far, calculate the number of pages required. We add one more page to account for the header page of the internal dbspace.

```
NPAGES = (FULL_BLOCKS x PAGES_PER_BLOCK)
         + (ROWS_LAST_BLOCK / ROWS_PER_PAGE)    ← round up
         + 1
```

For a sort to be successful, the internal dbspace must be defined to have at least NPAGES pages.

Calculating Total Internal Dbspace and DASD Needs

So far we have calculated the size of the sort working set. After the initial working set has been created, we then merge all the sorted portions to yield a final sorted result. Multiple merge passes may occur before the final result is created. During this process, two copies of the working set exist, in two internal dbspaces. For successful completion of a sort, more than one internal dbspace must be available.

The final result may be smaller than intermediate results, due to such things as the elimination of the three byte sort row header, and the decoding of varying-length values in cases other than index creation. The amount of DASD required is reduced only in the case where a single merge pass is performed; that is, when one pass is made through the data from the initial working set to the final result. This only occurs on sorts which are sufficiently small, or where the data is already almost completely sorted.

When sorting for duplicate elimination, the merge process will remove duplicates. As with the duplicate elimination which occurred during sorting, it is difficult to predict the effect this will have. Note that, for calculating DASD requirements, we are only interested in the duplicate elimination which would occur during the first merge pass, since the second copy of the working set is created by this pass. The completion of the first merge pass is the point at which our peak DASD usage occurs.

We will not consider these cases, and calculate the amount of DASD required to perform the sort as:

```
number of DASD pages = NPAGES x 2
```

For the sort to complete successfully, the storage pool to which the internal dbspaces are assigned must have sufficient DASD pages available.

Appendix B. CMS EXECs

SQLINIT EXEC

The SQLINIT EXEC initializes a user machine for application server access. With this EXEC, users specify the application server they wish to access and any special options. Each user must run the SQLINIT EXEC.

Note: The user machine must be initialized regardless of whether you are operating in single user mode or multiple user mode.

Initializing a User Machine

Before a user can run any DB2 Server for VM application program, use the DBS Utility, run the preprocessors, or use ISQL:

1. The user machine must have read access to a database machine's production minidisk (Q-disk), read/write access to its own work minidisk (A-disk), and be able to communicate with the database machine (by IUCV or APPC/VM).

   For Information about providing minidisk access to user machines and allowing user machines to communicate with the database machine, see the *DB2 Server for VM System Administration* manual.
2. The user must log on and enter IPL CMS (if this was not done during the logon procedure).
3. The user must initialize the user machine for application server access using the SQLINIT EXEC. The syntax and options of the SQLINIT EXEC are discussed below.

   Note: Because the SQLINIT EXEC may issue the CMS NUCXLOAD and CMS NUCXDROP commands, it should not be run in the CMS/DOS environment.

Figure 59 on page 236 shows the format of the SQLINIT EXEC.

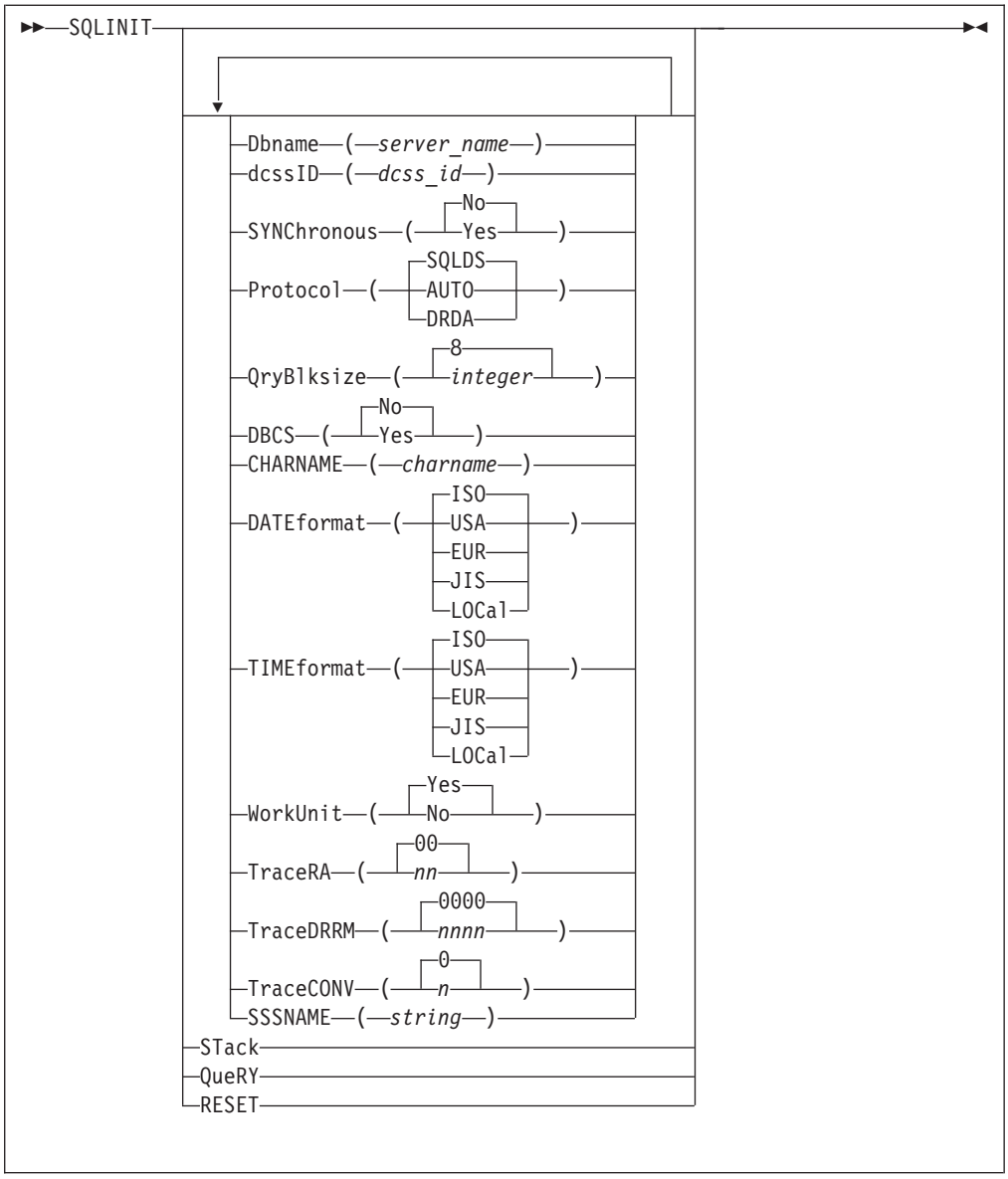


Figure 59. SQLINIT EXEC

The parameters of the SQLINIT EXEC are as follows:

Dbname
specifies the application server to be accessed. For the DBNAME keyword, you can use any initial substring (for example, D, DB, DBN, DBNA, or DBNAM). If DBNAME is omitted, the name of the last application server specified is used as a default. If SQLINIT cannot determine the last application server accessed, you are prompted to reissue the SQLINIT EXEC with the DBNAME parameter specified. The application server can be either:

- A DB2 Server for VM application server
- Any application server that supports IBM's implementation of the Distributed Relational Database Architecture (DRDA) protocol.

dcssID
specifies the name of the bootstrap package that contains the saved segment ID of the RA and ISQL components. This parameter should be specified only if

you want to use a specific saved segment for the database manager code; otherwise, it should be omitted. If you specify DCSSID, you **must** specify DBNAME.

You can specify ID instead of DCSSID for the keyword. No other abbreviation is valid. For more information about using saved segments for the database manager code, refer to the *DB2 Server for VM System Administration* manual.

DCSSID identifies a bootstrap package that invokes RA or ISQL code that resides in a discontiguous saved segment. If DCSSID is not specified, the *dcss_id* value from the *resid* SQLDBN file on the production disk is used. If the *resid* SQLDBN file is not available, and you are in a VM/ESA environment, the *dcss_id* value from the SQLDCSID DEFAULT file (if available) is used. If neither value is available, SQLDBA SQLRMBT and SQLDBA SQLISBT are used. See "SQLINIT, SQLSTART, Bootstrap Modules and SQLDBN files" on page 242 for more information on this topic.

Note: In a VM/ESA environment, *resid* may or may not be the same as *server_name*.

SYNChronous
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 restriction: you cannot use SQLHX or CANCEL to cancel SQL statements. The only way to end an LUW is to use the DB2 Server for VM FORCE operator command, or to re-IPL CMS.

Use synchronous communication primarily when running a well tested production application against local application servers. The default value for SYNCHRONOUS is NO.

Protocol
indicates the application requester access protocol to be used for communicating with the application server.

If you specify the SQLDS option, the SQLDS protocol is used for communication between a DB2 Server for VM application requester and a DB2 Server for VM application server. If this option is specified, the application requester cannot connect to a non-DB2 Server for VM application server. Use this option if both the application server and the application requester are part of DB2 Server for VM system and both use the same default CCSIDs. SQLDS is the default value.

Note: If PROTOCOL(SQLDS) is specified, the CCSID defaults set for the application requester with the SQLINIT EXEC are not used; the CCSIDs set for the application server are used.

If you specify the AUTO option, the application requester uses the SQLDS protocol when communicating with a DB2 Server for VM application server and the DRDA protocol when communicating with other application servers. If both the application requester and the application server use AUTO protocol but have different default CCSIDs, CCSID conversion is done correctly for requests and replies. The AUTO option lets you access both like and unlike systems without changing the PROTOCOL option and reissuing SQLINIT. You should use this option in the following cases:

- The user needs access to both like and unlike systems.

- The CCSID defaults are not the same on the application server and the application requester. For correct CCSID conversion the application server must also use AUTO protocol.
- You need an LUWID associated with each task so that you can easily trace a task back to its originating site.

If you specify the DRDA option, the application requester will use the DRDA protocol when communicating with a like or unlike application server. If the database machine is running code prior to Version 3 Release 3, the SQLDS protocol is forced for that connection. Of the three options, DRDA has the greatest performance overhead and storage requirements.

Notes:

1. The PROTOCOL value is ignored in single user mode and the SQLDS protocol is used for the connection.
2. The DRDA and AUTO options can only be specified if:
   - The DRDA facility is installed on the DB2 Server for VM application requester
   - The other application server to which you want to connect supports IBM's implementation of the DRDA protocol.

QryBlksize
specifies the block size of the returned rows of data when blocking performs FETCHes. The number is specified in denominations of 1K and can range anywhere from 1K to 32K. The default value is 8K.

Note: The QRYBLKSIZE parameter is ignored when the SQLDS protocol is used for the connection.

DBCS
specifies whether DBCS character handling of SO/SI pairs is done or not. This value is used by ISQL, the DBS Utility, and the preprocessors instead of the value currently found in the SYSTEM.SYSOPTIONS table. If NO is specified, keywords are converted from lowercase to uppercase by ISQL and the DBS Utility. If YES is specified, error checking is done on DBCS data by ISQL, the DBS Utility, and the preprocessors. The default value for DBCS is NO.

CHARNAME
specifies the CCSID values (CCSIDSBCS, CCSIDMIXED, and CCSIDGRAPHIC) used by the application requester, and is used to determine how to fold characters from lowercase to uppercase. Its value must be a valid character name, such as those found in the CHARNAME column of the SYSTEM.SYSCCSIDS table. CHARNAME is supported for the DRDA and AUTO PROTOCOL options. The DB2 Server for VM product is shipped with CHARNAME of the user machine initially set to INTERNATIONAL.

The SQLPREP EXEC, the DBS Utility, and the Resource Adapter use the CHARNAME value for folding support in both single user mode and multiple user mode. See the *DB2 Server for VSE & VM Application Programming* manual for information on the SQLPREP EXEC.

DATEformat
specifies the date format. The default is ISO. This parameter is for information only. It represents the date format in which the user wants to see date values returned.

TIMEformat
specifies the time format. The default is ISO. This parameter is for information only. It represents the time format in which the user wants to see time values returned.

WorkUnit
specifies whether CMS Work Unit support is to be used for an application. The default is Yes.

TraceRA
specifies the parts of the Resource Adapter (RA) that are to be traced and the level of the trace. The positional digits correspond to the following Resource Adapter subcomponents and functions:
- RA control flow
- Communications.

When 0 is specified, tracing is turned off. When 1 is specified, tracing is done in limited detail. When 2 is specified, tracing is done in greater detail. The default value for TRACERA is 00.

Note: A data stream trace is obtained by tracing the communications subcomponent.

The following CMS FILEDEF command was entered to define the default trace output file:

```
FILEDEF ARITRAC TAP2 SL (BLKSIZE 4096 NOCHANGE PERM
```

You can enter a different CMS FILEDEF command to override the default. The options you specify on your CMS FILEDEF command will not be overridden unless you reenter a CMS FILEDEF command to change them.

Note: The trace output, requested using the TRACERA, TRACEDRRM, and TRACECONV parameters, is stored in a single file.

TraceDRRM
specifies the parts of the DRRM component that are to be traced and the level of the trace. The positional digits correspond to the following DRRM subcomponents and functions:
- Parser
- Generator
- Dictionary
- RDIIN Manager.

When 0 is specified, tracing is turned off. When 1 is specified, tracing is done in limited detail. When 2 is specified, tracing is done in greater detail. The default value for TRACEDRRM is 0000.

The following CMS FILEDEF command was entered to define the default trace output file:

```
FILEDEF ARITRAC TAP2 SL (BLKSIZE 4096 NOCHANGE PERM
```

You can enter a different CMS FILEDEF command to override the default. The options you specify on your CMS FILEDEF command will not be overridden unless you reenter a CMS FILEDEF command to change them.

Notes:

1. The TRACEDRRM parameter is ignored when the SQLDS protocol is used for the connection.
2. The trace output, requested using the TRACERA, TRACEDRRM, and TRACECONV parameters is stored in a single file.

TraceCONV
specifies that the data conversion component is to be traced and the level of the trace.

When 0 is specified, tracing is turned off. When 1 is specified, tracing is done in limited detail. When 2 is specified, tracing is done in greater detail. The default value for TRACECONV is 0.

The following CMS FILEDEF command was entered to define the default trace output file:

```
FILEDEF ARITRAC TAP2 SL (BLKSIZE 4096 NOCHANGE PERM
```

You can enter a different CMS FILEDEF command to override the default. The options you specify on your CMS FILEDEF command will not be overridden unless you reenter a CMS FILEDEF command to change them.

Note: The trace output, requested using the TRACERA, TRACEDRRM, and TRACECONV parameters, is stored in a single file.

SSSNAME
specifies the name of the status shared segment. This parameter is optional. For more information on defining the status shared segment for the DB2 Server for VM system, refer to the *DB2 Server for VM System Administration* manual.

STack
places all values currently set for the parameters of the SQLINIT EXEC, except DCSSID, onto the CMS stack in the same sequence as shown for QUERY.

QueRY
displays all values currently set for the parameters of the SQLINIT EXEC, except DCSSID. It also displays the resource adapter code release level, CCSID values, LDATELEN value, and LTIMELEN value. (See "SQLGLOB EXEC" on page 244 for information about LDATELEN and LTIMELEN.)

Note: The value returned for CHARNAME is valid only if the value specified for the PROTOCOL parameter is **not** SQLDS. If the PROTOCOL parameter value is SQLDS, the CHARNAME value returned for the application requester is the same as the CHARNAME value of the application server to which it is connected (even if they are not the same).

The following is sample output from an SQLINIT QUERY.

```
ARI0717I Start SQLINIT EXEC: 05/29/92 14:52:40 EDT.
SELECTED TABLE IS: SQL/DS
 DBNAME=SQLDBA
 DBCS=NO
 SYNCHRONOUS=NO
 DATEFORMAT=ISO
 TIMEFORMAT=ISO
 TRACERA=00
 LDATELEN=0
 LTIMELEN=0
 RELEASE=3.3.0
 WORKUNIT=NO
 QRYBLKSIZE=8
 PROTOCOL=SQLDS
 CHARNAME=INTERNATIONAL
 CCSIDSBCS=500
 CCSIDMIXED=0
 CCSIDGRAPHIC=0
 TRACEDRRM=0000
 TRACECONV=0
 SSSNAME=
ARI0796I End SQLINIT EXEC: 05/29/92 14:52:40 EDT
```

RESET
resets all values currently set for the parameters of the SQLINIT EXEC, except DCSSID. The next time the SQLINIT EXEC is invoked, the defaults are used.

The SQLINIT EXEC parameter values are stored in the CMS LASTING GLOBALV file. Each time the SQLINIT EXEC is run, the parameter values are appended to the LASTING GLOBALV file. To maintain the LASTING GLOBALV file size, duplicate entries can be removed. Subsequently, when the user reenters the SQLINIT EXEC, the parameter value is established as follows:

1. If the user specifies a parameter on the SQLINIT EXEC, that parameter value is used.
2. If a parameter is not specified on the SQLINIT EXEC, the value stored in the LASTING GLOBALV file is used. That is, the default is the value used on the most recent SQLINIT EXEC.
3. If there are no values in the LASTING GLOBALV file (no values were specified on a previous SQLINIT EXEC, or SQLINIT RESET has reset the entries to blanks in the LASTING GLOBALV file), the application server-wide defaults established by the SQLGLOB EXEC are used.
4. If nothing is available, that is, if no application server-wide defaults established by the SQLGLOB EXEC exist, the SQLINIT EXEC will supply hardcoded defaults, except for the DBNAME parameter.

The parameter values remain in the LASTING GLOBALV file until explicitly changed through a subsequent SQLINIT EXEC with new parameters. SQLINIT RESET resets the entries to blanks, for any SQLINIT EXEC parameter values currently stored in the LASTING GLOBALV file.

The LASTING GLOBALV file is left on the user's A-disk. This means that users do not have to run the SQLINIT EXEC every time they log on. (This also means that users do not need to run the SQLINIT EXEC from their PROFILE EXECs.) The only times the user needs to rerun the SQLINIT EXEC are:

- When the user wants to change the default application server
- When the user wants to change any of the SQLINIT EXEC parameter values.

SQLINIT, SQLSTART, Bootstrap Modules and SQLDBN files

The SQLINIT EXEC provides for a user's program to communicate with the database machine by copying to the user's A-disk the following bootstraps:

```
dcss-id SQLRMBT Q  --->  ARISRMBT MODULE A
dcss-id SQLISBT Q  --->  ARISISBT MODULE A
```

The bootstrap modules reside on the production minidisk (Q-disk).

Prior to Version 3 Release 1, the SQLINIT EXEC used the ARISRMBT module to obtain default SQLINIT EXEC values. The SQLINIT EXEC now uses the LASTING GLOBALV file instead of this bootstrap module to obtain default values. The bootstrap module is still produced to maintain compatibility with load modules generated prior to Version 3 Release 1.

The ARISRMBT module is for the resource adapter, but it is incomplete. The resource adapter needs to know the name of the database machine with which it is to communicate.

Note: In a VM/ESA environment, the resource adapter only needs to know the database (resource) name.

The resource adapter also needs to know the default DCSSID. At this time ARISRMBT does not contain this information. The SQLINIT EXEC uses a CMS file called a *SQLDBN file* to locate information about a database. The SQLDBN file is created by the SQLSTART EXEC. When the SQLSTART EXEC is invoked, it starts the database manager code in a particular machine to access a particular application server. The SQLSTART EXEC creates a CMS file on the production minidisk to record this information (if the CMS file does not already exist). The name of the file is taken from the DBNAME parameter if the *resid* is the same as the *server-name*; otherwise, the *resid* will be resolved using the RESID NAMES file. The *filetype* is SQLDBN. So, suppose you log on the SQLDBA database machine and enter:

```
SQLSTART DBNAME(DB01) DCSSID(MYBOOT)
```

The SQLSTART EXEC accesses or creates the DB01 SQLDBN Q file. DB01 contains the following information:

1. The *server-name* of the application server being accessed (DB01)
2. The name of the database machine that is accessing the application server (SQLDBA)

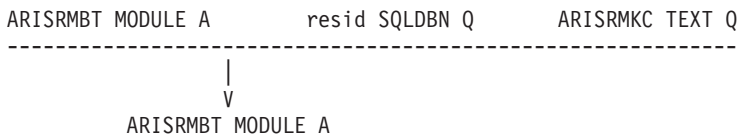
   Note: The name of the database machine is not needed to complete the bootstrap in a VM environment.
3. The name of the bootstrap or DCSSID being used (MYBOOT).

When the database machine is shut down, the *resid* SQLDBN file remains on the production minidisk. (The name of the file is taken from the DBNAME parameter if the *resid* is the same as the *server-name*; otherwise, the *resid* will be resolved using the RESID NAMES file.) It is updated whenever a database machine is started to access the application server and one of the following is true:

- The DCSSID specified on the SQLSTART EXEC is different from the one stored in the SQLDBN file
- The AMODE specified on the SQLSTART EXEC is different from the one stored in the SQLDBN file

- The DBNAME specified on the SQLSTART EXEC is different from the one stored in the SQLDBN file
- The database machine trying to access the application server is different from the one that last created the SQLDBN file.

The SQLINIT EXEC uses these SQLDBN files to complete the resource adapter bootstrap module. That is, the SQLINIT EXEC looks for the SQLDBN file having the *resid* that corresponds to the DBNAME parameter. If the DBNAME is greater than 8 bytes, it uses the SQLDCSID DEFAULTS file. Otherwise, the SQLINIT EXEC reads the information in the SQLDBN file and then generates the complete resource adapter bootstrap on the work minidisk:



The new module is called ARISRMBT. It will replace any existing ARISRMBT MODULE on the user's A-disk. ARISRMBT serves two purposes:
- It identifies where the resource adapter code is to be loaded
- It tells the resource adapter where to direct its communications.

Note the *resid* SQLDBN files will not be available to user machines that:
- Access an application server that resides on a different processor
- Access an application server that does not own the Production (Q) minidisk to which the user has a link.

If the SQLDBN file is not available, the SQLINIT EXEC looks for the SQLDCSID DEFAULT file for a default saved segment (DCSSID). If the SQLDBN file or SQLDCSID DEFAULT file is not found on the Production (Q) disk, the default SQLDBA SQLRMBT and SQLDBA SQLISBT bootstrap modules are used. If the *dcss-id* from the default SQLDBN file is not the one desired, specify the *dcss-id* on the SQLINIT EXEC to override it.

Note: The SQLDCSID DEFAULT file is only used in a VM environment. The SQLDCSID DEFAULT file is created by the SQLGENLD EXEC. See the *DB2 Server for VM System Administration* manual for more information on saved segments.

The ISQL bootstrap module, on the other hand, only identifies where the ISQL code is to be loaded. Because ISQL uses the resource adapter also, there is no need to identify the database machine in the ISQL bootstrap.

When an application initially calls the database manager, the bootstraps are executed to load the resource adapter and to help establish communication with the database machine. The database name is used for APPC/VM communication in VM environments.

Note that the bootstrap modules **are left on the user's A-disk**. This means that a user does not have to run the SQLINIT EXEC every time he or she logs on. (This also means that users do not need to run the SQLINIT EXEC from their PROFILE EXECs.) The only times the user needs to rerun SQLINIT are:
- When the user wants to change the default application server. If for any reason the bootstraps are not valid, it is detected and a message is issued to the user.

- When the user wishes to use bootstraps that have been defined after running the SQLINIT EXEC.

  For example, if bootstrap modules are defined, the user runs the SQLINIT EXEC, and then new bootstrap modules are defined, the user will have to run the SQLINIT EXEC again to take advantage of the new bootstrap modules.

When the DBNAME parameter is not specified on the SQLINIT EXEC, the name of the application server last accessed will be obtained from the ARISRMBT module residing on the user's A-disk. New versions of the ARISRMBT and ARISISBT modules will then be generated to reflect the information stored in the SQLDBN file for that application server.

You can run the SQLINIT EXEC any number of times from within another EXEC.

SQLGLOB EXEC

Use the SQLGLOB EXEC to set the default parameter values for the SQLINIT EXEC, except DCSSID, for your local DB2 Server for VM application server. The default values will only be used for application requests that have linked to the production disk of the local application server. The syntax of the SQLGLOB EXEC is similar to that of the SQLINIT EXEC.

The SQLGLOB EXEC creates a file on the production disk, called SQLGLOB DEFAULTS, containing all the default values for the SQLINIT EXEC, except DCSSID. If a user runs the SQLINIT EXEC without specifying some of the parameter values and these values are not stored in the LASTING GLOBALV file, then the missing parameter values are taken from the SQLGLOB DEFAULTS file that was created with the SQLGLOB EXEC. The syntax of the SQLGLOB EXEC is shown in Figure 60 on page 245.

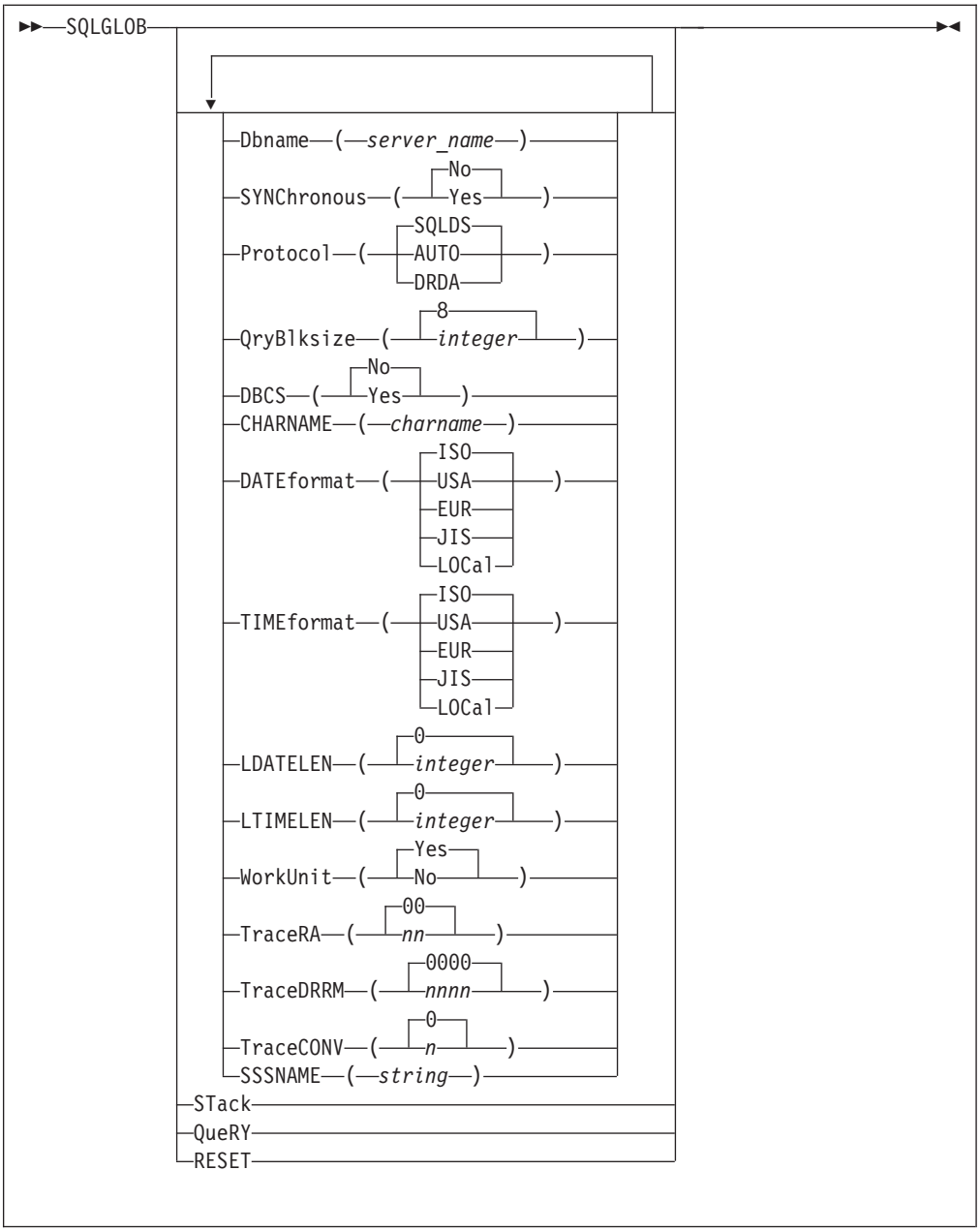


Figure 60. SQLGLOB EXEC

The parameters of the SQLGLOB EXEC are as follows:

Dbname
specifies the application server to be accessed. For the DBNAME keyword, you can use any initial substring (for example, D, DB, DBN, DBNA, or DBNAM). If DBNAME is omitted, the name of the last application server specified is used as a default. The application server can be either:

- A DB2 Server for VM application server
- Any application server that supports IBM's implementation of the Distributed Relational Database Architecture (DRDA) protocol.

Note: Access to non-DB2 Server for VM application servers is only possible if the DRDA facility is installed on the DB2 Server for VM application requester.

SYNChronous
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 restriction: you cannot use SQLHX or CANCEL to cancel SQL statements. The only ways to end an unwanted LUW is to use the FORCE command, or to re-IPL CMS.

Use synchronous communication primarily when running a well tested production application against local application servers. The default value for SYNCHRONOUS is NO.

Protocol
indicates the application requester access protocol to be used for communicating with the application server.

If you specify the SQLDS option, the SQLDS protocol is used for communication between a DB2 Server for VM application requester and a DB2 Server for VM application server. If this option is specified, the application requester cannot connect to a non-DB2 Server for VM application server. Use this option if both the application server and the application requester are part of DB2 Server for VM system and both use the same default CCSIDs. SQLDS is the default value.

Note: If PROTOCOL(SQLDS) is specified, the CCSID defaults set for the application requester with the SQLINIT EXEC are not used; the CCSIDs set for the application server are used.

If you specify the AUTO option, the application requester uses the SQLDS protocol when communicating with DB2 Server for VM application servers and the DRDA protocol when communicating with other application servers. If both the application requester and the application server use AUTO protocol but have different default CCSIDs, CCSID conversion is done for requests and replies. The AUTO option lets you access both like and unlike systems without changing the PROTOCOL option and reissuing SQLINIT. You should use this option in the following cases:

- The user needs access to both like and unlike systems.
- The CCSID defaults are not the same on the application server and the application requester. For correct CCSID conversion the application server must also use AUTO protocol.
- You need an LUWID associated with each task so that you can easily trace a task back to its originating site.

If you specify the DRDA option, the application requester will use the DRDA protocol when communicating with a like or unlike application server. If the database machine is running code prior to Version 3 Release 3, the SQLDS protocol is forced for that connection. Of the three options, DRDA has the greatest performance overhead and storage requirements.

Notes:

1. The PROTOCOL value is ignored in single user mode and the SQLDS protocol is used for the connection.
2. The DRDA and AUTO options can only be specified if:
   - The DRDA facility is installed on the DB2 Server for VM application requester

- The other application server to which you want to connect supports IBM's implementation of the DRDA protocol.

QryBlksize
specifies the block size of the returned rows of data when blocking performs FETCHes. The number is specified in denominations of 1K and can range anywhere from 1K to 32K. The default value is 8K.

Note: The QRYBLKSIZE parameter is ignored when the SQLDS protocol is used for the connection.

DBCS
specifies whether DBCS character handling of SO/SI pairs is done or not. This value is used by ISQL, the DBS Utility, and the preprocessors instead of the value currently found in the SYSTEM.SYSOPTIONS table. If NO is specified, keywords are converted from lowercase to uppercase by ISQL and the DBS Utility. If YES is specified, error checking is done on DBCS data by ISQL, the DBS Utility, and the preprocessors. The default value for DBCS is NO.

CHARNAME
specifies the CCSID values (CCSIDSBCS, CCSIDMIXED, and CCSIDGRAPHIC) used by the application requester, and determines how to fold characters from lowercase to uppercase. Its value must be a valid character name, such as those found in the CHARNAME column of the SYSTEM.SYSCCSIDS table. CHARNAME is supported for the DRDA and AUTO PROTOCOL options. The DB2 Server for VM product is shipped with CHARNAME of the user machine initially set to INTERNATIONAL.

The SQLPREP EXEC, the DBS Utility, and the Resource Adapter use the CHARNAME value for folding support in both single user mode and multiple user mode. See the *DB2 Server for VSE & VM Application Programming* manual for information on the SQLPREP EXEC.

DATEformat
specifies the date format. The default is ISO. This parameter is for information only. It represents the date format in which the user wants to see date values returned.

TIMEformat
specifies the time format. The default is ISO. This parameter is for information only. It represents the time format in which the user wants to see time values returned.

LDATELEN
gives the length of the local date format. The values may range from 10 to 254, and 0. The default is 0, which indicates that LOCAL format is not available.

LTIMELEN
gives the length of the local time format. The values may range from 8 to 254, and 0. The default is 0, which indicates that LOCAL format is not available.

WorkUnit
specifies whether CMS Work Unit support is to be used for an application. The default is Yes. CMS Work Units are supported in VM environments only.

TraceRA
specifies the parts of the Resource Adapter (RA) that are to be traced and the level of the trace. The positional digits correspond to the following Resource Adapter subcomponents and functions:
- RA control flow

- Communications.

When 0 is specified, tracing is turned off. When 1 is specified, tracing is done in limited detail. When 2 is specified, tracing is done in greater detail. The default value for TRACERA is 00.

The following CMS FILEDEF command was entered to define the default trace output file:

```
FILEDEF ARITRAC TAP2 SL (BLKSIZE 4096 NOCHANGE PERM
```

You can enter a different CMS FILEDEF command to override the default. The options you specify on your CMS FILEDEF command will not be overridden unless you reenter a CMS FILEDEF command to change them.

Note: The trace output, requested using the TRACERA, TRACEDRRM, and TRACECONV parameters is stored in a single file.

TraceDRRM
specifies the parts of the DRRM component that are to be traced and the level of the trace. The positional digits correspond to the following DRRM subcomponents and functions:
- Parser
- Generator
- Dictionary
- RDIIN Manager.

When 0 is specified, tracing is turned off. When 1 is specified, tracing is done in limited detail. When 2 is specified, tracing is done in greater detail. The default value for TRACEDRRM is 0000.

The following CMS FILEDEF command was entered to define the default trace output file:

```
FILEDEF ARITRAC TAP2 SL (BLKSIZE 4096 NOCHANGE PERM
```

You can enter a different CMS FILEDEF command to override the default. The options you specify on your CMS FILEDEF command will not be overridden unless you reenter a CMS FILEDEF command to change them.

Notes:

1. The TRACEDRRM parameter is ignored when the SQLDS protocol is used for the connection.
2. The trace output, requested using the TRACERA, TRACEDRRM, and TRACECONV parameters is stored in a single file.

TraceCONV
specifies that the data conversion component is to be traced and the level of the trace.

When 0 is specified, tracing is turned off. When 1 is specified, tracing is done in limited detail. When 2 is specified, tracing is done in greater detail. The default value for TRACECONV is 0.

The following CMS FILEDEF command was entered to define the default trace output file:

```
FILEDEF ARITRAC TAP2 SL (BLKSIZE 4096 NOCHANGE PERM
```

You can enter a different CMS FILEDEF command to override the default. The options you specify on your CMS FILEDEF command will not be overridden unless you reenter a CMS FILEDEF command to change them.

Note: The trace output, requested using the TRACERA, TRACEDRRM, and TRACECONV parameters is stored in a single file.

SSSNAME
specifies the name of the status shared segment. This parameter is optional, and is intended for use with products such as the IBM SystemView Information Warehouse DataHub Support/VM software. For more information on defining the status shared segment for the DB2 Server for VM system, refer to the *DB2 Server for VM System Administration* manual.

STack
places all values currently set for the parameters of the SQLGLOB EXEC onto the CMS stack in the same sequence as shown for QUERY.

QueRY
displays all values currently set for the parameters of the SQLGLOB EXEC. It also displays the resource adapter code release level and CCSID values. The following is sample output from an SQLGLOB QUERY.

```
ARI0717I Start SQLGLOB EXEC: 05/29/92 10:27:36 EDT.
DBNAME=SQLDBA
DBCS=NO
SYNCHRONOUS=NO
DATEFORMAT=ISO
TIMEFORMAT=ISO
TRACERA=00
LDATELEN=0
LTIMELEN=0
RELEASE=3.3.0
WORKUNIT=NO
QRYBLKSIZE=8
PROTOCOL=SQLDS
CHARNAME=INTERNATIONAL
CCSIDSBCS=500
CCSIDMIXED=0
CCSIDGRAPHIC=0
TRACEDRRM=0000
TRACECONV=0
SSSNAME=
ARI0796I End SQLGLOB EXEC: 05/29/92 10:27:36 EDT
```

RESET
resets all values currently set for the parameters of the SQLGLOB EXEC. The next time the SQLGLOB EXEC is run, the defaults are used.

SQLCIREO EXEC

This EXEC reorganizes the indexes on the catalog tables. The following diagram shows the format of the SQLCIREO EXEC.

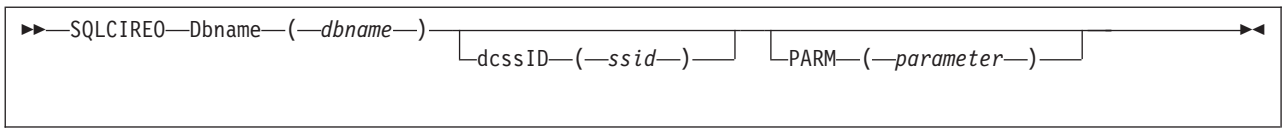


Figure 61. SQLCIREO EXEC

Dbname
specifies the name of the database for which you want to reorganize the catalog indexes. Any initial substring for DBNAME can be used as the keyword (for example DB or D).

dcssID
specifies the name of a bootstrap package that identifies a saved segment. You can use DCSSID or ID for the keyword. If not specified, DCSSID defaults to SQLDBA. The SQLDBA bootstrap package causes the database manager code to be loaded into the DMSFREE area.

PARM
specifies additional DB2 Server for VM initialization parameters. If you specify the PARM parameter, it must follow the other SQLCIREO parameters.

The valid initialization parameters are:
ARCHPCT=*n*
CHARNAME=*name*
CHKINTVL=*n*
DUMPTYPE=F | N | P
LOGMODE=A | L | N | Y
NCSCANS=*n*
NDIRBUF=*n*
NPAGBUF=*n*
PARMID=*name*
SLOGCUSH=*n*
SOSLEVEL=*n*
TRACCONV=*n*
TRACDBSS=*nnnnnnnnnn*
TRACDRRM=*nnnn*
TRACDSC=*nn*
TRACEBUF=*n*
TRACRDS=*nnnnnn*
TRACWUM=*n*
TRACSTG=*n*

Initialization parameters are described in the *DB2 Server for VM System Administration* manual and the *DB2 Server for VSE & VM Operation* manual.

SQLCIREO automatically supplies the initialization parameters DBNAME (based on what you supply in the EXEC parameter DBNAME), SYSMODE=S, and STARTUP=I.

To avoid the processing involved in switching log modes, use the same LOGMODE that you normally use.

Because the catalog index reorganization utility runs in single user mode, the only way to trace it is with the TRACDBSS, TRACDSC, and TRACWUM initialization parameters. (The TRACE operator command cannot be used in single user mode.)

If you are using tracing, you may want to enter your own CMS FILEDEF and LABELDEF commands before invoking SQLCIREO. These optional CMS FILEDEF and LABELDEF commands are described in the *VM/ESA: CMS Application Development Reference* for your VM system.

For example, to reorganize the indexes for the PRODX database, you might enter:

```
SQLCIREO D(PRODX) PARM(LOGMODE=A)
```

The catalog index reorganization utility directs messages to SYSPRINT. If you do not supply a FILEDEF command for SYSPRINT, SQLCIREO assigns SYSPRINT to the terminal. This FILEDEF command directs the utility's messages to the CMS file REORG PRT A:

```
FILEDEF SYSPRINT DISK REORG PRT A (RECFM FBA LRECL 121 BLOCK 1210
```

The utility prints informational messages that describe what actions it has taken. Packages are not invalidated if they use an index reorganized by the SQLCIREO utility.

If the catalog index reorganization utility abnormally ends, all changes it has made to the database are rolled back. You should rerun the utility after correcting the problem.

SQLDBID EXEC

The SQLDBID EXEC displays the name of the default application server that has been established by the SQLINIT EXEC.

Figure 62 shows the format of the SQLDBID EXEC.

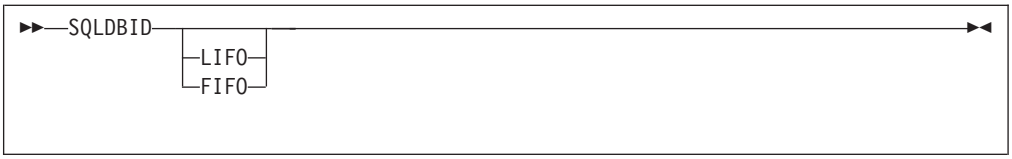


Figure 62. SQLDBID EXEC

The SQLDBID EXEC resides on the production minidisk. The SQLDBID EXEC displays the name of the default application server that will be accessed if the resource adapter bootstrap on the A-disk is used.

The SQLDBID EXEC does no more than display information. If the name of the application server displayed is not the one you want to access, you must rerun the SQLINIT EXEC.

If you specify the LIFO or FIFO parameters, the information provided by the SQLDBID EXEC is stacked onto the most recently created buffer of the program stack (system provided data queue). If you specify the FIFO parameter, the information is stacked on a first in, first out basis. If you specify the LIFO parameter, the information is stacked on a last in, first out basis. If neither parameter is specified, message ARI0320I, specifying the application server name, is issued.

SQLRMEND EXEC

The SQLRMEND EXEC manages the communication links between an application program and an application server. It does this in two ways:

- If you have more than one program called from an EXEC, SQLRMEND can ensure the integrity of each program by dropping the link used by the current program, or dropping the resource adapter code which drops all links between all programs and all application servers they access

- If you have one program, SQLRMEND can drop the resource adapter code to free storage from your virtual machine.
- If your program runs the CMS subset, a ROLLBACK or COMMIT of all LUWs is prevented. This is to ensure that any LUWs in the native CMS MODE are not affected.

Usually the resource adapter code and control blocks are not dropped and outstanding work is not committed until the end of the VM command. When programs are invoked from within EXECs, the "end of command" is at the end of the EXEC, not the end of the program. If you want to drop the resource adapter code or perform COMMIT/ROLLBACK processing at the end of a program (not at the end of the EXEC), you must run the SQLRMEND EXEC. Figure 63 shows the format of the SQLRMEND EXEC.

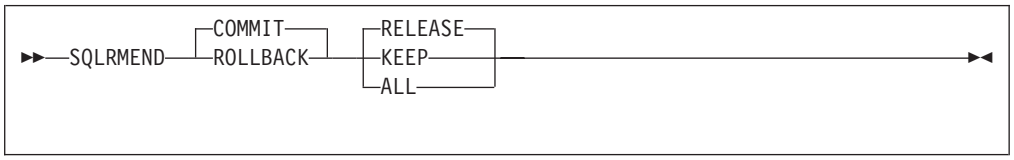


Figure 63. SQLRMEND EXEC

The SQLRMEND EXEC resides on the production minidisk.

The SQLRMEND EXEC can be used to drop the DB2 Server for VM resource adapter code and its control blocks from within the DMSFREE (user free storage) area of a user's virtual machine. It allows users who invoke more than one program from within an EXEC to free the storage used by the resource adapter. The storage can then be used by other programs.

The SQLRMEND EXEC can also be used to perform COMMIT/ROLLBACK processing on all outstanding work. It allows users who invoke more than one program from within an EXEC to COMMIT or ROLLBACK all outstanding work before the next program is executed.

You may have separate logical units of work active at one time. You could do this by having one or more programs running in separate CMS Work Units. When you use SQLRMEND in this situation, the second option (RELEASE, KEEP, or ALL) determines which CMS Work Units are affected.

Note: You cannot use COMMIT ALL or ROLLBACK ALL in CMS subset mode to prevent the user from affecting any work done previously in a normal CMS mode. The SQLRMEND EXEC will not allow these parameters in CMS subset mode.

COMMIT RELEASE
is the default. Specifies the currently active LUW within the active work unit is to be committed, the communication link to be released. Resource adapter code is dropped only if there is just one CMS Work Unit.

If you have more than one CMS Work Unit, only the current LUW is committed, and the link between the program and the database it is accessing is dropped.

COMMIT KEEP
specifies the active LUW within the active work unit is to be committed and the communication link and the resource adapter code are to be kept.

COMMIT ALL
specifies all active LUWs for all suspended and active work units are to be committed, the communication link(s) to be released, and the resource adapter code is to be dropped.

ROLLBACK RELEASE
specifies the active LUW in the active work unit is to be rolled back and the communication link is to be released. Resource adapter code is dropped only if there is just one CMS Work Unit.

If you have more than one CMS Work Unit, the current link between the program and the application server it is accessing is dropped, and the current logical unit of work is rolled back.

ROLLBACK KEEP
specifies the currently active LUW in the active work unit is to be rolled back and that the communication link and the resource adapter code are to be kept.

ROLLBACK ALL
specifies all active LUWs for all suspended and active work units are to be rolled back, the communication link(s) to be released, and the resource adapter code is to be dropped.

COMMIT
is equivalent to COMMIT RELEASE.

RELEASE
is equivalent to COMMIT RELEASE.

ROLLBACK
is equivalent to ROLLBACK RELEASE.

KEEP
is equivalent to COMMIT KEEP.

ALL
is equivalent to COMMIT ALL.

The KEEP option will keep the communication link. Therefore if you want to access an application server again from within the EXEC, your next program will be able to use the same SQL connection (user ID and application server) without issuing an explicit CONNECT.

Unless you are maintaining more than one CMS Work Unit, the RELEASE option will drop the communication link and the resource adapter code. Only the storage used for the resource adapter control blocks will be freed. If you want to access the DB2 application server again from within the EXEC, you do not need to enter anything to get the resource adapter code back into storage. Just invoke the DB2 Server for VM application. The resource adapter code will automatically be reloaded.

If you can have more than one CMS Work Unit, the RELEASE option drops the current link. Logical units of work in other CMS Work Units are still active.

Example

The following example is a portion of an EXEC that runs in a VM system, and invokes two programs. The second program runs in a separate CMS Work Unit. After it has completed, the SQLRMEND EXEC is invoked with the COMMIT ALL

option, which commits the logical units of work active in each program, drops both links, and frees the storage used by the resource adapter.

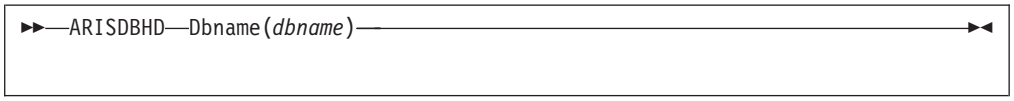
```
   •
   •
   •
EXEC SQLINIT DB(DB01)     ←Set up access to DB01 in first work unit
SQLPROG1                  ←Run program that accesses DB01
EXEC SQLINIT DB(DB02)     ←Set up access to DB02 in second work unit
SQLPROG2                  ←Run program that accesses DB02
EXEC SQLRMEND COMMIT ALL  ←commits both logical units of work, drops
                                 both links, and drops resource adapter
```

The SQLINIT EXEC is invoked to switch application servers for the second program.

ARISDBHD EXEC

The ARISDBHD EXEC deletes SQL/DS HELP text, including administrator supplied topics, for one or more languages. It does not delete the message repository.

To run the ARISDBHD EXEC, you must have:
- Read access to the SQL/DS service minidisk or SFS directory
- Read access to the SQL/DS production minidisk or SFS directory
- The connect password for SQLDBA.



The parameters of the ARISDBHD EXEC are as follows:

Dbname(*dbname***)**
Replace *dbname* with the name of the database in which the HELP text is to be deleted.

When you run the ARISDBHD EXEC:
1. Specify which HELP text languages to delete.
2. Confirm that you want to delete the HELP text for the specified languages.
3. Execute the delete procedure.

Step 1 When you invoke the ARISDBHD EXEC, you will be prompted for the connect password for SQLDBA. After entering the password, the contents of the SYSLANGUAGE table are reformatted and displayed. Specify the language key for each language whose HELP text you wish to delete (separated by commas or blanks) or `ALL` to specify all HELP text for all languages in the database.

If you decide to stop the procedure at this point, enter `QUIT`.

Each time you select a language, it is flagged on the screen; a null entry will process your selections.

Step 2 The languages that you indicated for deletion are displayed on the next screen. Confirm that you want to delete all these languages by entering `YES`. To exit from the procedure without deleting any HELP text languages, enter `NO`.

Step 3 When prompted, enter the owner ID and virtual address of the CMS HELP text for the language specified.

If you know that there is no CMS HELP text for the language specified, enter `SKIP` to bypass the language.

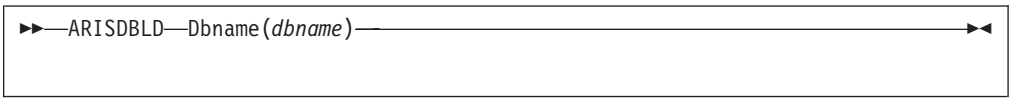
If you accidentally delete the CMS HELP text for a different language by entering the wrong virtual address, you can restore the environment by executing the ARISDBMA EXEC for the deleted language and rerunning this EXEC with the correct address.

If you wish to delete an entire language (both the messages and the HELP text), use the ARISDBLD EXEC instead. The ARISDBHD EXEC deletes both ISQL help and CMS help. This EXEC displays a list of currently installed languages, which may contain languages that have already had their help deleted but are still active (that is, are listed in the SYSLANGUAGE table and have the appropriate message repository available). You will be prompted to select the HELP text languages for deletion. You should not specify languages whose HELP text has already been deleted. Since this EXEC cannot be used to delete a message repository, the SYSLANGUAGE table and the ARISNLSC MACRO will not be updated in any way. This EXEC will not affect the default language setting.

ARISDBLD EXEC

The ARISDBLD EXEC deletes the SQL/DS messages and HELP text, including administrator supplied topics, for one or more languages.

To run the ARISDBLD EXEC, you must have:
- Read access to the SQL/DS service minidisk or SFS directory
- Read access to the SQL/DS production minidisk or SFS directory
- The connect password for SQLDBA.



The parameters of the ARISDBLD EXEC are as follows:

Dbname(*dbname***)**
Replace *dbname* with the name of the database in which the messages and HELP text are to be deleted.

When you run the ARISDBLD EXEC it prompts you to:

1. Specify which languages to delete
2. Determine which is to become the new default language, if the current default language is to be deleted, and two or more languages will remain on the system
3. Confirm that you want to delete the messages and HELP text for the specified languages.

Step 1 When you invoke the ARISDBLD EXEC, you will be prompted for the connect password for SQLDBA. After entering the password, the contents of the SYSLANGUAGE table are reformatted and displayed. Specify the language key for each language whose HELP text you wish to delete (separated by commas or blanks).

If you decide to stop the procedure at this point, enter `QUIT`.

Each time you select a language, it is flagged on the screen; a null entry will process your selections.

Step 2 To delete the current default language with two or more languages remaining on the system, you must specify which of the remaining languages will be the new default language. When the current default language is flagged for deletion, a list of the remaining languages and keys is displayed. Specify the key for the new default language.

If only one language remains on the system, it will automatically become the new default language.

Step 3 The languages that you flagged for deletion are displayed on the next screen. Confirm that you want to delete all these languages by entering `YES`. To exit from the procedure without deleting any HELP text languages, enter `NO`.

Step 4 When prompted, enter the owner ID and virtual address of the CMS HELP text for the language specified.

If you know that there is no CMS HELP text for the language specified, enter `SKIP` to bypass the language.

If you accidentally delete the CMS HELP text for a different language by entering the wrong virtual address, you can restore the environment by executing the ARISDBMA EXEC for the deleted language and rerunning this EXEC with the correct address.

A minimum of one language **must** be left on the ARISNLSC macro and the SQLDBA.SYSLANGUAGE table; it is not possible to delete **all** the languages.

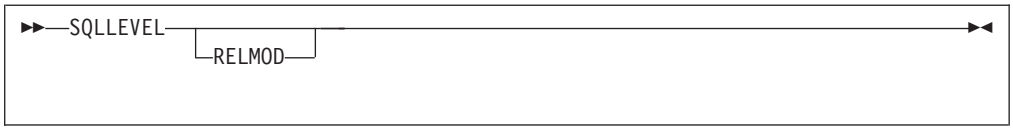
For each language being deactivated, the following occurs:
- The ARISNLSC MACRO is updated. If the current default language has been marked for deletion, you must specify a new default language.
- The message repository for this language is deleted from the production minidisk or directory.
- The ISQL HELP text is deleted by updating the SYSTEXT2 table.
- The ARISDBMC EXEC is invoked to delete CMS help.

SQLLEVEL EXEC

The SQLLEVEL EXEC displays the SQL/DS release level that is installed. For example:
To run the SQLLEVEL EXEC, you must have:

```
*** SQL/DS VERSION 7 RELEASE 1 MODIFICATION 0 ***
```

- Read access to the SQL/DS service minidisk or SFS directory
- Read access to the SQL/DS production minidisk or SFS directory.



The parameters for the SQLLEVEL EXEC are as follows:

RELMOD

If you include this parameter, the EXEC places the version, release, and modification levels in the CMS stack. (These values are all integers.)

Appendix C. Querying the Status of an Application (VM Only)

SQLQRY is implemented as a CMS immediate command and enables you to query the status of the application that you are currently running on the user machine. It is initialized by the DB2 Server for VM resource adapter when the first database statement is processed, and can be issued while your application is running. Since it is implemented as a CMS immediate command, it can be used even when your application is not accepting other input. See the *VM/ESA: CMS User's Guide* manual for more information about CMS immediate commands.

The SQLQRY command is particularly useful if problems arise while you are switching between application servers. In these cases, use SQLQRY to determine the application server to which you are currently connected.

Notes:

1. You can only enter the SQLQRY command from the user machine after the resource manager has been loaded and while an application is running.
2. You cannot use the SQLQRY command if you are using the SYNChronous(YES) option with the SQLINIT EXEC, if you are not receiving CP messages (for example, if you specified CP SET MSG OFF), or if your application has locked the keyboard. See "SQLINIT EXEC" on page 235 for information on the SYNChronous(YES) option.

The following information is displayed at the terminal when you enter the SQLQRY command:

EXTNAME
displays the user ID of the application requester to which you are currently connected. It also displays the CMS Work Unit number, if CMS Work Units are in use.

RDBMS
displays the name, class, and release level (version, release, and modification level) of the application server being accessed. If the Protocol(DRDA) or Protocol(AUTO) option is specified with the SQLINIT EXEC and the SQLQRY command is issued before handshaking (the process of establishing a connection) is completed, "n/a" will be displayed for both the class and release level of the application server. If the Protocol(SQLDS) option is specified with the SQLINIT EXEC and the SQLQRY command is issued, "SQLDS/VM" will be displayed for the application server class and "n/a" will be displayed for the application server release level, because handshaking does not take place. See "SQLINIT EXEC" on page 235 for information on the Protocol parameter.

STATUS
displays the communication state. COMM indicates that the Work Unit sent an SQL statement to the database machine and has been waiting for a reply since the time shown. APPL indicates that the Work Unit returned control to the application at the time shown. VRA indicates that the VM Resource Adapter is processing your request.

LUWID
displays the logical unit of work identifier, which uniquely identifies an LU6.2 conversation. Its value is *netid.luname.instance_number.sequence_number*, where *netid* and *luname* are up

to 8 characters long, *instance_number* is 12 characters long, and *sequence_number* is 4 characters long. LUWID is only used for conversations that use the AUTO and DRDA Protocol options. If the middle portion of the LUWID contains *IDENT, then the application server is a local one or is in a TSAF collection; in these cases, no LU name and TPN are displayed. If TCP/IP is being used, the LUWID has the format IPADDRESS.PORT.INSTANCE_NUMBER, where IPADDRESS is 8 characters long, PORT is 4 characters long, and INSTANCE_NUMBER is 12 characters long.

LU displays the logical unit name, if the access is through VTAM.

TPN displays the transaction program name. Its character and hexadecimal versions are both displayed. For a DB2 application server, this is the same as the resource ID.

TCPIP displays the IP address of the target host system. It is only displayed when TCP/IP is being used.

PORT displays the port number of the target application server. It is only displayed when TCP/IP is being used.

Example

Figure 64 displays sample output from an SQLQRY command issued in a VM/ESA environment with Protocol(AUTO) and two active CMS Work Units.

```
11:09:51  * MSG FROM SQLUSR6 : Status of Server Conversations on 2000-09-20.
11:09:51  * MSG FROM SQLUSR6 : EXTNAME = SQLUSR6.1
11:09:51  * MSG FROM SQLUSR6 : RDBMS  = SQLRDB1            SQLDS/VM V6.1.0
11:09:51  * MSG FROM SQLUSR6 : STATUS = COMM   TIME = 2000-09-20.11:09:43
11:09:51  * MSG FROM SQLUSR6 : LUWID = IBMNET01.*IDENT.45F2ABCD236D42.0001
11:09:51  * MSG FROM SQLUSR6 :
11:09:51  * MSG FROM SQLUSR6 : EXTNAME = SQLUSR6.2
11:09:51  * MSG FROM SQLUSR6 : RDBMS  = IBMSTLDB2          DB2      V2.3.0
11:09:51  * MSG FROM SQLUSR6 : STATUS = APPL   TIME = 2000-09-20.11:07:32
11:09:51  * MSG FROM SQLUSR6 : LU = STLMVS04     TPN = "6DB     (X'07F6C4C2')
11:09:51  * MSG FROM SQLUSR6 : LUWID = IBMNET01.TORLU001.45F2ABCD236DFE.0001
11:09:51  * MSG FROM SQLUSR6 :
11:09:51  * MSG FROM SQLUSR6 : EXTNAME = SQLUSR6.3
11:09:51  * MSG FROM SQLUSR6 : RDBMS  = SQLMACGM           SQLDS/VM  V7.1.0
11:09:51  * MSG FROM SQLUSR6 : STATUS = COMM   TIME = 2000-09-20.11:07:32
11:09:51  * MSG FROM SQLUSR6 : TCPIP = 9.21.23.32       PORT = 8030
11:09:51  * MSG FROM SQLUSR6 : LUWID = G9151720.L372.B1622DADEF8A
```

Figure 64. Sample Output from SQLQRY in a VM Environment with Protocol(AUTO) and CMS Work Units

You can have multiple active CMS Work Units in a user machine, each accessing an application server. In this example, information is displayed for two CMS Work Units.

EXTNAME contains the user ID of the application requester, concatenated with the CMS Work Unit number.

Because Protocol(AUTO) is used, a unique LUWID is assigned to each conversation.

- Work Unit #1:

  - Accesses a SQL/DS V3.3.0 application server called SQLRDB1. Since no LU name or TPN are displayed, the application server must be a local one or is in a TSAF collection. For the same reason, the middle portion of the LUWID is *IDENT.
  - Has a STATUS of COMM, indicating that it must have sent an SQL statement to the application server and has been waiting for a reply (since 11:09:43, as indicated by TIME).
- Work Unit #2:
  - Accesses a DB2* V2.3.0 application server called IBMSTLDB2. The LU name and TPN are displayed because the application server is in a VTAM network. The DB2 application server uses the default DRDA TPN of X'07F6C4C2'.
  - Has a STATUS of APPL, indicating that it has already returned control to the application (at 11:07:32, as indicated by TIME).
- Work Unit #3:
  - Accesses a DB2 Server for VM Version 7 Release 1 application server called SQLMACGM. The HOST name and SERVICE port connection are displayed because the application server is in a TCP/IP network.

Appendix D. Maximums

The following tables describe logical data maximums and ISQL maximums. Information about database maximums and system maximums can be found in the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual.

Table 35. Logical Data Maximums

Restricted Parameter	Maximum
Number of Tables per Database	8,000,000
Number of Indexes per Database	8,000,000
Number of Views per Database[1]	2,549,490
Number of Programs per Database[1]	2,549,490
Number of Tables per DBSPACE[4]	255
Number of Indexes per Table	255
Number of Columns per Table	255
Number of unique CCSID combinations per Table	80
Number of Columns per View[3]	≈ 140
Number of Columns in a SELECT-list	255
Length of a Row in a Table (Bytes)[2]	4,080
Number of Columns in an Index	16
Length of an Index Key (Bytes)	255
Number of Foreign Keys per Table	32,767

Notes:

1. The number of views plus the number of programs cannot exceed 2,549,490. This limit assumes that you create the maximum number of dbspaces possible (9998) for packages. Each dbspace can contain 255 packages. If you only create 10 dbspaces for packages, you only have room for 2,550 packages.
2. Not including long field columns.
3. There is no specific limit on the number of columns in a view, because it depends on many factors which affect this limit. A view of up to 140 columns should work in most situations.
4. This maximum includes tables implicitly created as well as user-defined tables. Each table with one or more long fields requires a table created implicitly to hold the long fields. Long fields are LONG VARCHAR, LONG VARGRAPHIC, VARCHAR(n) where n > 254, and VARGRAPHIC(n) where n > 127.

ISQL Maximums

Table 36. ISQL Maximums

Restricted Parameter	Maximum
Maximum number of columns in a query	45
Maximum length of a command (bytes)	2,048

The maximum number of columns in a query may be further reduced by the width of the columns selected.

Appendix E. SQLGLOB Parameters (VSE Only)

DB2 Server for VSE stores certain environmental parameters in a VSAM file called "SQLGLOB." ISQL, DBSU and the preprocessors retrieve the CHARNAME and DBCS values from this SQLGLOB file. The online and batch Resource Adapters also access this file to determine certain environmental parameters as they communicate with a remote application server.

The SQLGLOB VSAM file will hold both *GLOBAL* and *USER* parameters. There is one set of global SQLGLOB parameters, which is the system-wide default values. These global parameters are initially set with the IBM-supplied default values during installation using the ARISGDEF procedure and subsequently updated using the DSQG transaction. A CICS user can choose to override the global SQLGLOB parameters by setting up his or her own user SQLGLOB parameters. This is done by executing the DSQU transaction. There is one set of user SQLGLOB parameters for every CICS user who executed the DSQU transaction. A batch user can choose to override the global SQLGLOB parameters by setting up their own user SQLGLOB parameters. This is done by executing the program ARIRBGUD (JCL: ARISBGUD.Z) and specifying the Update (U) command and a user ID. The ARIRBGUD program can also be used to update the global SQLGLOB parameters, to query one of the user's parameters or all of the user's parameters, or to delete a user's set of parameters.

The SQLGLOB VSAM file is defined to the system and initially updated with the IBM-supplied default global SQLGLOB parameter values during product installation.

The SQLGLOB parameters and their initial IBM-supplied global default values are described below:

QryBlksize
Specifies the block size used to return rows of data when DRDA blocking is used to perform FETCHes. The number is specified in denominations of 1K and can range anywhere between 1K and 32K.

This option is only used when the application requester is communicating with a remote application server.

The IBM-supplied global default QryBlksize is 8K.

CHARNAME
Specifies the character set name, which determines the CCSID values for CCSIDSBCS, CCSIDMIXED and CCSIDGRAPHIC used by the application requester, and determines how ISQL and the preprocessors fold characters from lowercase to uppercase. Its value must be a valid character set name, such as those found in the CHARNAME column of the SYSTEM.SYSCCSIDS table.

This value is used by ISQL, DBSU, and the preprocessors instead of the value currently found in the SYSTEM.SYSOPTIONS table.

The IBM-supplied global default CHARNAME is INTERNATIONAL.

DBCS
Specifies whether DBCS character handling of SO/SI pairs is done or not. This value is used by ISQL, DBSU, and the preprocessors instead of the value currently found in the SYSTEM.SYSOPTIONS table.

YES
Specifies that error checking is done on DBCS data by the preprocessors, DBSU and ISQL. If double byte characters are to be used, DBCS must be set to YES.

NO
Specifies that error checking is not done on DBCS data by the preprocessors, DBSU and ISQL.

The IBM-supplied global default DBCS is NO.

SYNCPOINT
Specifies how commits or rollbacks are to be coordinated by the CICS/VSE syncpoint manager.

1 Specifies a **one-phase** commit is to be done. In this case, the CICS/VSE sync point manager is not involved and unprotected APPC conversations are used.

2 Specifies a **two-phase** commit is to be done. In this case, protected APPC conversations will be used to connect to the DRDA server and the CICS/VSE sync point manager will be used to coordinate two-phase commits. If a user is updating a remote server and other CICS resources which participate in two-phase commit within a logical unit of work, SYNCPOINT must be set to 2.

This option is only used when the online CICS application requester is communicating with a remote application server through SNA.

The IBM-supplied global default SYNCPOINT is 1.

TRACERA
Specifies a two digit number (nn) which specifies the parts of the Batch and Online Resource Adapter that are to be traced and the level of the trace. Trace data is collected only when the application is connected to a remote server. The positional digits correspond to the following Resource Adapter subcomponents and functions:

- Resource Adapter control flow
- Communications.

0 Tracing is turned off.

1 Tracing is done in limited detail.

2 Tracing is done in greater detail.

TRACERA is ignored on local connections.

The IBM-supplied global default TRACERA is 00.

TRACEDRRM
Specifies a four digit number (nnnn) which specifies the parts of the DRRM component that are to be traced and the level of the trace. Trace data is

collected only when the application is connected to a remote server. The positional digits correspond to the following DRRM subcomponents and functions:

- Parser
- Generator
- Dictionary
- RDIIN Manager.

0 Tracing is turned off.

1 Tracing is done in limited detail.

2 Tracing is done in greater detail.

TRACEDRRM is ignored on local connections.

The IBM-supplied global default TRACEDRRM is 0000.

TRACECONV
Specifies a 1 digit number (n) which specifies that the data conversion component is to be traced and the level of the trace. Trace data is collected only when the application is connected to a remote server.

0 Tracing is turned off.

1 Tracing is done in limited detail.

2 Tracing is done in greater detail.

TRACECONV is ignored on local connections.

The IBM-supplied global default TRACECONV is 0.

Communications Protocol
Specifies which network access method (SNA or TCP/IP) to use for remote connections.

S Specifies that SNA is used for the connection to the remote application server. In this case, the remote application server entry in the DBNAME Directory should contain SNA routing information (SYSID and REMTPN). S is the default for this field.

T Specifies that TCP/IP is used for the connection to the remote application server. In this case, the remote application server entry in the DBNAME Directory should contain TCP/IP routing information (TCPPORT and TCPHOST, or TCPPORT and IPADDR).

This option is only used when the online CICS application requester is communicating with a remote application server. Note that Batch Applications must always use TCP/IP to communicate with a remote server.

The IBM-supplied global default COMMUNICATIONS PROTOCOL is SNA.

Transactions for Updating SQLGLOB Parameters

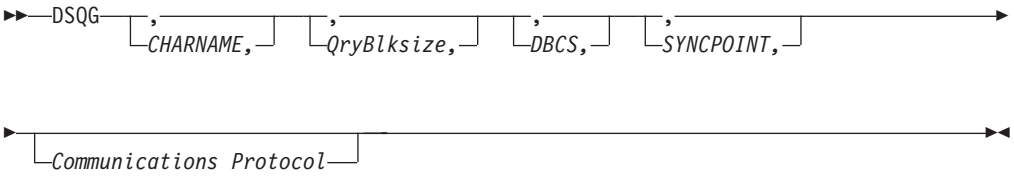
This section describes the various CICS transactions that a user can use to manage parameters stored in the SQLGLOB VSAM file.

DSQG - Update global SQLGLOB Parm Transaction

The **DSQG** transaction is a CICS transaction which updates a subset of the global SQLGLOB parameters.

This must be defined as a **secured** transaction. That is, this transaction must be defined with a TRANSEC value greater than 1, so that it *cannot* be initiated by any user on the CICS system. Only authorized CICS users should be allowed to invoke this transaction.

This transaction has five parameters as shown in the following syntax diagram. See "Appendix E. SQLGLOB Parameters (VSE Only)" on page 265 for more information on these DSQG parameters.



Each time the DSQG transaction is executed, the global SQLGLOB parameters are replaced as follows:

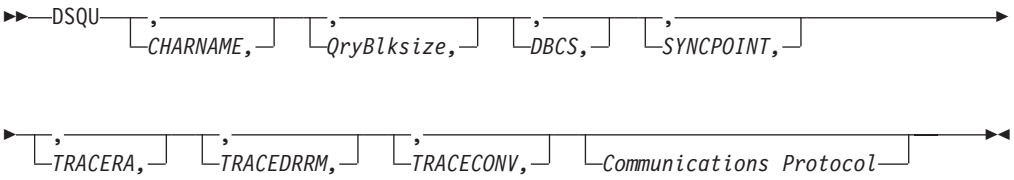
1. If the user specifies a parameter on the DSQG transaction, that parameter value is used.
2. If a parameter is not specified on the DSQG transaction, the current global SQLGLOB parameter is used.

The global SQLGLOB parameters remain in effect until they are explicitly changed. The global SQLGLOB parameters can be changed by invoking the DSQG transaction and specifying the new parameters, or by invoking program ARIRBGUD (the SQLGLOB File Batch Update/Query Program) and specifying the Update (U) command with the user ID *SYSDEF*.

DSQU - Update user SQLGLOB Parm Transaction

The **DSQU** transaction is a CICS transaction which initializes the SQLGLOB parameters for the signed-on user ID. The user SQLGLOB parameters, like the global SQLGLOB parameters, are stored in the SQLGLOB VSAM file.

This transaction has eight parameters, as shown in the following syntax diagram. See "Appendix E. SQLGLOB Parameters (VSE Only)" on page 265 for more information on these DSQU parameters.



Each time the DSQU transaction is executed, the user SQLGLOB parameters are replaced. When a user reissues the DSQU transaction, the parameter value is established as follows:

1. If the user specifies a parameter on the DSQU transaction, that parameter value is used.
2. If a parameter is not specified on the DSQU transaction, the current user SQLGLOB parameter is used. That is, the default is the value used on the most recent DSQU transaction. However, if the user SQLGLOB parameter does not exist, the global SQLGLOB parameter is used.

The user SQLGLOB parameters remain in effect until they are explicitly changed or until they are explicitly deleted through a subsequent DSQD transaction. The user SQLGLOB parameters can be changed by invoking the DSQU transaction and specifying the new parameters or by invoking the program ARIRBGUD (SQLGLOB File Batch Update/Query Program) and specifying the Update (U) command with the appropriate user ID and new parameters. The user SQLGLOB parameters can be deleted by invoking the DSQD transaction or by invoking the program ARIRBGUD and specifying the Delete (D) command with the appropriate user ID.

All DRDA connections initiated by online CICS transactions, except those initiated by ISQL, CBND, or any task that was started by the EXEC CICS START command, will use the signed-on user's SQLGLOB parameters, if they exist. If they do not exist, these DRDA connections will use the global SQLGLOB parameters.

ISQL, CBND, and any task that was started by the EXEC CICS START command will use the global SQLGLOB parameters for DRDA connections regardless of who is signed on, with the following exceptions:

1. ISQL uses the *user* DBCS parameter to determine whether DBCS character handling is required or not.
2. ISQL uses the *user* CHARNAME parameter to get the folding table to fold input from the terminal from lowercase to uppercase, but it uses the *global* CHARNAME for CCSID data conversion.

All DRDA connections initiated by VSE batch application programs will use the SQLGLOB parameters (if they exist) of the user ID specified on the SQL CONNECT statement. If they do not exist, these DRDA connections will use the global SQLGLOB parameters.

DSQQ - Query SQLGLOB Parm Transaction

The **DSQQ** transaction is a CICS transaction which displays all the SQLGLOB parameters. Which version of the SQLGLOB parameters (user or global) is displayed depends on whether or not the userid parameter is specified on the DSQQ command.

This transaction has one parameter, as shown in the following syntax diagram:

userid
Specifies the user ID whose user SQLGLOB parameters are to be displayed.

If the userid parameter is specified and the user SQLGLOB parameters of the specified *userid* exist, DSQQ will display the user SQLGLOB parameters.

If the userid parameter is omitted, DSQQ will display the global SQLGLOB parameters.

DSQD - Delete user SQLGLOB Parm Transaction

The **DSQD** transaction is a CICS transaction which deletes a signed-on user ID's user SQLGLOB parameters.

This transaction has no parameters, as shown in the following syntax diagram:

After the signed-on user's user SQLGLOB parameters are deleted, any subsequent DRDA connections done on behalf of this signed-on user ID will use the global SQLGLOB parameters.

Batch Program to Update/Query the SQLGLOB File

If a "CONNECT user ID" is needed for a remote server, but the user ID does not exist as a CICS user ID, then a batch program and JCL are supplied to allow the SQLGLOB file to be updated for any user ID. This stand-alone program, ARIRBGUD, allows a new user ID to be inserted or an existing user ID to be updated, deleted or queried based on the input given to the program. Input for this program is provided from SYSIPT "cards" (80 byte records).

Table 37 describes the layout of the input for ARIRBGUD:

Table 37. ARIRBGUD Input Layout. Description

Start — End Column #s	**Field Name**	**Usage**
1 - 1	COMMAND	'Q' = query, 'D' = delete, or 'U' = update. This field is mandatory.
3 - 10	USERID	This is the user ID in the SQLGLOB file that is to be operated upon. This can be '*' if COMMAND is 'Query'. This field is mandatory with an 8 byte maximum length.
12 - 29	CHARNAME	Character Set Name; it determines CCSID values and folding (18 bytes maximum).
31 - 32	QRYBLKSIZE	Block size used for Blocked Fetches, in integral 'K' bytes, from 1 to 32.
34 - 34	DBCS	'Y' if DBCS is allowed. 'N' if DBCS is not allowed.
36 - 36	SYNCPOINT	'1' or '2', for one- or two-phase COMMIT (DRDA 1 or DRDA 2). This field is only used by the CICS Application Requester for remote DRDA connections.
38 - 39	TRACERA	Resource Adapter Trace flags, two digits, each either '0', '1' or '2'.
41 - 44	TRACEDRRM	DRRM Trace flags, four digits, each either '0', '1' or '2'.
46 - 46	TRACECONV	CONV Trace Flags, 1 digit, either '0', '1' or '2'.
48 - 48	COMMPROTO	Communications Protocol to be used for Remote database access: 'S'=SNA or 'T'=TCP/IP. Only used by CICS , not Batch.

Notes:

1. The COMMAND and USERID are always required.

2. For the Query COMMAND only, the user ID can be specified as '*', which means display all user IDs in the SQLGLOB file.
3. For a Query or Delete COMMAND, all other fields are ignored.
4. For an Update COMMAND, fields that are left blank remain unchanged in an existing record for the user ID. If the user ID does not already have an existing record in SQLGLOB, then the System Default record values are used.
5. The System Default record (user ID = X'FF's) CANNOT be deleted by this program. It CAN be updated or queried. Considering this user ID cannot be displayed, it is displayed as the string "*SYSDEF*", and this string must be used for the Update or Query COMMAND as the "CONNECT User ID".

The following is an example of the SYSLST output from the program ARIRBGUD, with a default of 120 print positions per print line. Input records are shown in bold italics. Note that the first byte of the listing is a printer carriage control character.

```
1ARIRBGUD - Batch Query/Update of DB2 for VSE SQLGLOB File    1999/12/31 23:59:59

0                                  CCSID CCSID CCSID QRYBLK      SYNC  ----TRACE---  COMM
 COMMAND USER     CHARNAME         SBCS  MIXED DBCS    SIZE DBCS POINT RA DRRM CONV  PROTO

0----> Q UUUUUUUU
 QUERY   uuuuuuuu ccccccccccccccccc sssss sssss sssss    12K   Y    1   00 0000   0   TCPIP

0----> U UUUUUUUU                  32 N 2 11 2222   S
 UPDATE  UUUUUUUU ccccccccccccccccc sssss sssss sssss    32K   N    2   11 2222   0     SNA

0----> D UUUUUUUU
 DELETE  UUUUUUUU

0----> Q UUUUUUUU
 ARI0485I The user SQLGLOB parameters for user UUUUUUUU do not exist.

0----> U BADPARM                   99
 ARI0494E Invalid input parameter entered. Parameter = QRYBLKSIZE.

0----> X JUNK STUFF
 ARI4599E Invalid COMMAND given, must be 'Q'(Query), 'U'(Update) or 'D'(Delete).

0----> D *SYSDEF*
 ARI4598E You can not DELETE the system default record from the SQLGLOB file.

0----> Q *
 ----> List of all Users in the SQLGLOB file:
 QUERY   uuuuuuuu ccccccccccccccccc sssss sssss sssss  bbbbb   d    s   tt tttt   t   ppppp
 QUERY   uuuuuuuu ccccccccccccccccc sssss sssss sssss  bbbbb   d    s   tt tttt   t   ppppp
      ...
      ...
      ...
 QUERY   *SYSDEF* ccccccccccccccccc sssss sssss sssss  bbbbb   d    s   tt tttt   t   ppppp
```

The following is an example of the SYSLST output from ARIRBGUD, because of a JCL error or because the file has never been created:

```
1ARIRBGUD - Batch Query/Update of DB2 for VSE SQLGLOB File    1999/12/31 23:59:59

0ARI0487E The SQLGLOB file does not exist.
```

The JCL to execute the program ARIRBGUD (SQLGLOB File Batch Update/Query Program) can be found in the IBM-supplied job ARISBGUD.Z.

Using Online and Batch Resource Adapter Tracing

The online (CICS) and batch Resource Adapter tracing is used for problem analysis. You would normally not turn tracing on unless requested to do so by IBM Support. The Online Resource Adapter trace output can be directed to a tape or a disk file, which must be defined in the CICS start up JCL and predefined in the CICS Destination Control Table. See the *DB2 Server for VSE Program Directory* for details about setting up the Destination Control Table. The batch Resource Adapter trace can only be directed to a tape file, which must be defined in the batch job JCL. Tracing is only performed for application statements that access remote DRDA servers.

Tracing is activated by updating the SQLGLOB file record for the SQL CONNECT statement user IDs (or the default Online user IDs). A '1' or '2' character placed in the TRACERA, TRACEDRRM or TRACECONV component field activates tracing for the respective subcomponents. A '0' character in this field deactivates tracing. Care should be taken when updating the trace fields in the Global Default user ID because this would cause tracing to be activated for ALL users who do not have a record in the SQLGLOB file; also, under CICS, all trace data is intermingled in the single trace output file. You can use the DSQU CICS transaction or the ARIRBGUD program (described in "Batch Program to Update/Query the SQLGLOB File" on page 270) to update the SQLGLOB file records.

Online Trace File JCL

The following is JCL that must be placed in the CICS start up JCL to define the trace output file:

- For a tape file:

```
// ASSGN SYS018,181
// TLBL ARITRAC,'name of trace file'
```

- For a disk file:

```
// ASSGN SYS018,DISK,VOL=volxxx,SHR
// DLBL ARITRAC,'name of trace file',0,SD
// EXTENT SYS018,volxxx,n,n,nnn,nn
```

Batch Trace File JCL

The following is JCL that must be placed in the batch job JCL to define the trace output file:

```
// ASSGN SYS005,181
// TLBL ARITRAC,'name of trace file'
```

Formatting the Online or Batch Trace File

After the trace is finished, you can invoke the trace formatting utility that comes with the DB2 Server for VSE system. The above JCL must also be used when the trace file is used as input to the trace formatter utility, except 'SYS004' must be used instead of 'SYS005' or 'SYS018' when your trace file is on tape. 'SYS018' must be used when your Online trace file is on disk. Note that the Batch trace file **cannot** be on disk. See "Formatting DB2 Server for VSE Trace Output" in *DB2 Server for VSE & VM Operation*.

Before the Online Resource Adapter trace file can be formatted, it must be closed using the following CICS command:

```
CEMT SET QUEUE(ARIT) DISABLED CLOSED
```

Before the batch Resource Adapter trace file can be formatted, it must be closed using the following JCL statement ″after″ the batch job step being traced:

```
// MTC WTM,SYS005,2
```

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10594-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
APL2
AS/400
C/370
CICS
CICS/VSE
DATABASE 2
DataHub
DataPropagator
DB2
DFSMS/VM
DRDA
Distributed Relational Database Architecture
Extended Services for OS/2
IBM
Information Warehouse
IBMLink
MVS
OS/2
OS/400
QMF
RACF
System/370
SystemView
VM/ESA
VSE/ESA
VTAM

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

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

Bibliography

This bibliography lists publications that are referenced in this manual or that may be helpful.

DB2 Server for VM Publications

- *DB2 Server for VSE & VM Application Programming*, SC09-2889
- *DB2 Server for VSE & VM Database Administration*, SC09-2888
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2983
- *DB2 Server for VSE & VM Diagnosis Guide and Reference*, LC09-2907
- *DB2 Server for VSE & VM Overivew*, GC09-2995
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2990
- *DB2 Server for VSE & VM Master Index and Glossary*, SC09-2890
- *DB2 Server for VM Messages and Codes*, GC09-2984
- *DB2 Server for VSE & VM Operation*, SC09-2986
- *DB2 Server for VSE & VM Quick Reference*, SC09-2988
- *DB2 Server for VM System Administration*, SC09-2980
- *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2987
- *DB2 Server for VSE & VM SQL Reference*, SC09-2989

DB2 Server for VSE Publications

- *DB2 Server for VSE & VM Application Programming*, SC09-2889
- *DB2 Server for VSE & VM Database Administration*, SC09-2888
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2983
- *DB2 Server for VSE & VM Diagnosis Guide and Reference*, LC09-2907
- *DB2 Server for VSE & VM Overivew*, GC09-2995
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2990
- *DB2 Server for VSE & VM Master Index and Glossary*, SC09-2890
- *DB2 Server for VSE Messages and Codes*, GC09-2985
- *DB2 Server for VSE & VM Operation*, SC09-2986
- *DB2 Server for VSE System Administration*, SC09-2981
- *DB2 Server for VSE & VM Performance Tuning Handbook*, GC09-2987
- *DB2 Server for VSE & VM SQL Reference*, SC09-2989

Related Publications

- *DB2 Server for VSE & VM Data Restore*, SC09-2991
- *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: 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
- CP and CMS control block information is not provided in book form. This information is available on the IBM VM/ESA operating system home page (http://www.ibm.com/s390/vm).
- *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*, GC33-1645
- *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*, GC33-0803

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 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 4*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Implementation Programmer's Guide*, SC21-9529
- *VM/Directory Maintenance Licensed Program Specification*, GC20-1836
- *IBM Distributed Relational Database Architecture Reference*, SC26-4651
- *IBM Systems Network Architecture, Format and Protocol Reference*, SC30-3112
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269
- *IBM Systems Network Architecture, Logical Unit 6.2 Reference: Peer Protocols*, SC31-6808
- *Distributed Data Management (DDM) General Information*, GC21-9527

CCSID Publications

- *Character Data Representation Architecture, Executive Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

DB2 Server RXSQL Publications

- *DB2 REXX SQL for VM/ESA Installation and Reference*, SC09-2891

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

- *Introducing QMF*, GC27-0714
- *Installing and Managing QMF for VSE*, GC27-0721
- *QMF Reference*, SC27-0715
- *Installing and Managing QMF for VM*, GC27-0720
- *Developing QMF Applications*, SC27-0718
- *QMF Messages and Codes*, GC27-0717
- *Using QMF*, SC27-0716

Query Management Facility (QMF) for Windows Publications

- *Getting Started with QMF for Windows*, SC27-0723
- *Installing and Managing QMF for Windows*, GC27-0722

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 RMS User's Guide and Reference*, SC35-0141

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

- *DB2 for VM Control Center Operations Guide*, GC09-2993
- *DB2 for VSE Control Center Operations Guide*, GC09-2992
- *DB2 Replication Guide and Reference*, SC26-9920

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Readers' Comments — We'd Like to Hear from You

DB2 Server for VSE & VM
Database Administration
Version 7 Release 1

Publication No. SC09-2888-00

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 _____

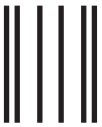
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC09-2888-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CANADA LTD.
DB2 Server for VSE & VM
2S/240/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada M3C 1H7

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/4300-50
Program Number: 5697-F42

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-2888-00

Spine information:

DB2 Server for VSE & VM Database Administration Version 7 Release 1