IBM

International Systems Centers

IBM DATABASE 2
CONCEPTS AND FACILITIES GUIDE

GG24-1582

IBM DATABASE 2 Concepts and Facilities Guide

Document Number GG24-1582-00

Ueli Wahli, IBM Switzerland

International Systems Center - Santa Teresa
San Jose, California

This publication was produced using the
IBM Document Composition Facility
(Program number 5748-XX9) and
the master was printed on the IBM 3800 Printing Subsystem

First Edition (June 1983)

This edition applies to IBM DATABASE 2 (DB2) Release 1 (Program Number 5740-XYR), Query Management Facility (QMF) Release 1 (5668-972), and Data Extract (DXT) Release 1 (5668-973).

Requests for copies should be made to the IBM branch office that serves you.

Forms for reader's comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to:

IBM Corporation
International Systems Center - Santa Teresa
Department 471
P.O. Box 50020
San Jose, California 95150, U.S.A.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

ACKNOWLEDGEMENT

This guide is the result of a residency conducted at the International Systems Center - Santa Teresa.

We would like to acknowledge the excellent work done by the author:

- Ueli Wahli, IBM Switzerland

Peter Backlund
Wes Dayton
Colin White

International Systems Center - Santa Teresa
June 1983

This document:

- **GG24-1582 IBM DATABASE 2 Concepts and Facilities Guide**

  gives a functional overview of the IBM DATABASE 2 (DB2) relational data base management system. It is intended to be read by all DP Professionals who wish to obtain a good functional knowledge of the product.

It is one in a series produced by International Systems Center - Santa Teresa. Other documents in the series are:

- **GG24-1581 IBM DATABASE 2 Relational Concepts**

  which describes the relational approach to data base systems in general and to IBM DATABASE 2 in particular. The relationship between IMS/VS DL/I and DB2 applications is also covered. The intended audience is DP Professionals who wish to understand the relational model of data and how it is implemented in DB2.

- **GG24-1583 IBM DATABASE 2 SQL Usage Guide**

  which demonstrates the power of Structured Query Language (SQL), a data base management language which permits IBM DATABASE 2 users to access and manipulate data in relational data bases. The document is intended for DP Professionals who wish to obtain a good functional knowledge of SQL. It covers SQL by using a series of examples starting with the very basic and becoming increasingly complex.

Chapter 1. Introduction 1
Organization of the Book 1
Related IBM Program Products 3

Chapter 2. The DB2 Environment 5
System Environment 5

Chapter 3. Relational Data Model 7
Tables 7
  Columns 8
  Rows 8
    Data Types 8
    Nulls 9
  Primary Key 9
    Uniqueness 9
  Related Tables 9
    Foreign Key 10
Views 10
  Table Subsets 11
  Derived and Calculated Fields 12
  Table Combinations 12
Table Design 12
  Flexibility 13

Chapter 4. SQL Data Manipulation 15
SQL Introduction 15
  Automatic Navigation 15
    Automatic Access Path Selection 16
  Interactive SQL 16
  Embedded SQL 16
Data Manipulation Language (DML) 17
  Data Retrieval 17
    SELECT Statement 17
  Updating Data 19
    UPDATE Statement 19
  Deleting Data 20
    DELETE Statement 20
  Inserting Data 21
    INSERT Statement 21
  Search Conditions 22
    USER Keyword 24
  Builtin Functions 24
  Grouping - GROUP BY Clause 25
    HAVING Clause 26
  Ordering - ORDER BY Clause 26
  Use of Nulls 26
  Combining Data 27
    Simple Join 28
Using Views 29
  Updating through Views 29

With Check Option 30
Advanced DML 30
Advanced Join 30
Joining tables with identical column names 30
Joining a table to itself 31
Subselect 31
ALL and ANY with Subselects 32
Correlated Subselects 33
Subselects with EXISTS Keyword 34
Union 35

Chapter 5. Data Definition 37
DB2 Objects 37
Tables 37
Views 39
Table Spaces 39
Simple Table Spaces 40
Partitioned Table Spaces 40
Table Pages 41
Indexes 41
Clustering Index 42
Index Spaces and Pages 42
Data Bases 43
Storage Groups 43
Bufferpools 44
Synonyms 45
Data Definition Language 45
Naming Conventions 45
CREATE Statement 46
Defining a Storage Group 46
Defining a DB2 Data Base 46
Defining a Table Space 47
Defining a Partitioned Table Space 48
Defining a Table 49
Defining an Index 50
Defining a Partitioning Index 51
Defining a View 52
Defining a Synonym 53
ALTER Statement 54
Changing a Storage Group 54
Changing a Table Space 54
Changing a Table 54
Changing an Index 55
DROP Statement 55
COMMENT ON Statement 56
Modifying the Design 56
Modifying Tables 57
Table Re-Creation Procedures 57
Modifying a View 58
Modifying the Definition of Table Spaces 58
Modifying and Adding Indexes 58

Chapter 6. Data Management 59
Data Space Management 59

Letting DB2 manage the Data Sets 59
VSAM Datasets 60
Simple Table Spaces 60
Partitioned Table Spaces 60
User defined Datasets 61
Data Page Management 62
Row Format 63
Free Space Management 65
Index Management 65
Buffer Management 66
Concurrency Control - Locking 67
Table Space Locks 67
Lock Statement 69
Page Locks 69
Commit/Rollback 69
Deadlocks 70
Data Definition Effect on Concurrency 70
Loading Tables 71
Load Utility 71
Data Extract (DXT) 74

Chapter 7. Application Programming Environment 77
Language Support 77
Subsystem Support 77
Application Development Cycle 77
Program Structure 79
Embedded SQL 79
SQL Statement Format 79
INCLUDE Statement 80
SQL Communication Area (SQLCA) 80
Table Declaration 81
Generate Declarations (DCLGEN) 82
Retrieving Data into Host Variables 82
Updating, Inserting and Deleting Data 83
CURSOR Operations 83
DECLARE Cursor 83
OPEN Cursor 84
FETCH a Row 84
Updating or Deleting a Row 84
CLOSE Cursor 84
Error Handling 85
WHENEVER Statement 85
Concurrency Support - Locking 86
Commit/Rollback 86
Host Variables 87
Handling Null Values 89
Data Definition Statements 90
Dynamic SQL 90
SQL Descriptor Area (SQLDA) 91
PREPARE Statement 91
DESCRIBE Statement 92
EXECUTE Statement 92
Parameterized Execution 93
Immediate Execution 93

Using Cursors with Dynamic SQL 93
Program Preparation 94
Precompile 94
Compilation and Link-Edit 95
Creating an Application Plan (BIND) 95
BIND Command 96
REBIND Command 97
FREE Command 97
Program Execution 97
Automatic Bind 97
Dynamic Bind 98
Testing Facilities 98

Chapter 8. TSO Environment 99
The DB2 TSO Command Processor (DSN) 99
ISPF Support - DB2I 100
The DB2I Option Menu 100
Online Help 101
SPUFI 101
SPUFI Execution Flow 102
Generate Declarations (DCLGEN) 103
Binding 104
Program Preparation 104
Execute an Application Program 105
DB2 Commands 105
Utilities 105
TSO Batch Work 105
Query Management Facility (QMF) 106
QMF Objectives 106
QMF Languages 106
Use of SQL 107
Use of QBE style Language 107
QMF Command Language 107
Tailored Reports 109
Comparison of QMF with DB2I 110

Chapter 9. Security and Authorization 111
Views and Security 111
Column Subset 111
Row Subset by Field Value 111
Statistical Summary 112
Authorization 112
Resources 112
Users and Authorization-IDs 113
Capabilities 113
Single Capabilities 114
Group Capabilities or Administrative Authorities . . 115
Explicit Authorization 116
GRANT 116
REVOKE 117
Implicit Authorization 118
Operation - what is checked when ? 118
Data Set Protection 119
VSAM Password Protection 119

RACF 119

Chapter 10. The DB2 Catalog 121
Catalog Structure 121
Data Definition 121
Application Plans 122
Authorization Definition 124
Image Copy Data Sets 124
Using the DB2 Catalog 125

Chapter 11. Operation and Recovery 127
Operation of DB2 127
Command and Message Support 127
Startup and Shutdown 128
Controlling Data Bases 129
Starting a Data Base 129
Stopping a Data Base 129
Displaying the Status of a Data Base 129
The DB2 Log 130
Log Operation 130
Active Log Data Sets 130
Archive Log Data Sets 131
Boot Strap Data Set 132
Log Processing Options 132
Log Utilities 133
Print Log Map Utility 133
Change Log Inventory Utility 133
DB2 Data Base Utilities 134
Invocation and Control 134
Utility Job Status 135
Forced Utility Termination 136
Reorganization 136
Data Base Backup and Recovery 137
Image Copy 138
Incremental Image Copy 139
Merge Copy 139
Recovery 140
Index Recovery 141
Repair Utility 141
Catalog Usage 142
Catalog Maintenance 142
Catalog Backup and Recovery 142
System Restart and Recovery 143
MVS Failure 143
DB2 Subsystem Failure 143
Resource Recovery 143
Log Data Sets 143
Boot Strap Data Set 144
Catalog 144
Data Base 145
Utility Recovery 145

Chapter 12. Architecture 147
DB2 Structure 147

DB2 System Services 148
DB2 Data Base Services 149
IRLM 150
Interfacing to DB2 150
TSO 150
IMS and CICS 150
Attach Architecture 151
Thread 151
Thread Identification 152
Unit of Recovery 153
Commit 155
Two Phase Commit 155
TSO/Batch Attachment Facility 156

Chapter 13. Monitoring and Accounting 157
Optimization 157
System Parameters 157
Data Base Parameters 158
Monitoring 159
Display Command 159
STOSPACE Utility 159
RUNSTATS Utility 160
Statistics Facility 160
Accounting Facility 161
Auditing 162
Log Access Services 162

Chapter 14. IMS Environment 163
Overview 163
IMS Attachment of DB2 163
IMS Control Region 164
IMS Dependent Region 165
Authorization 166
Application Programming 166
Transaction Processing 167
Commit 167
In-doubt Threads 168
Application Testing 168
Operation 168
IMS Commands 169
Issuing DB2 Commands from IMS 169
Monitoring the IMS Attachment 170

Chapter 15. CICS Environment 171
Overview 171
CICS Attachment of DB2 171
CICS Resource Control Table 172
Authorization 175
Application Programming 175
Transaction Processing 176
Commit 176
In-doubt Threads 176
Application Testing 177
Execution Diagnostic Facility 177

Operation 177
CICS Commands 178
Issuing DB2 Commands from CICS 179
Monitoring the CICS Attachment 179

Chapter 16. Installation and Servicing 181
Prerequisites 181
Hardware 181
Software 181
Tape Content 182
SMP Considerations 183
Installation Tasks 183
Installation using ISPF 183
Post Installation Tasks 185
Enabling the TSO/Batch Attachment 185
IRLM Considerations 185
IMS Considerations 186
CICS Considerations 187
Updating Installation Parameters 187
Installation Verification 188
Sample Application 189
Serviceability 190
Problem Determination Tools 190
Formatted Dump 190
Trace 190
LOGREC Recording 191

Appendix A. Compatibility between DB2 and SQL/DS 193
SQL/DS Statements and Options not found in DB2 193
Statements that have different Effects in SQL/DS and DB2 194

Figure 1. The DB2 Environment 6
Figure 2. The EMPLOYEE Table 7
Figure 3. The DEPARTMENT Table 10
Figure 4. Join of EMPLOYEE and DEPARTMENT tables 12
Figure 5. A simple SELECT statement 18
Figure 6. Simple Join 28
Figure 7. DB2 Objects 38
Figure 8. Views of Tables 39
Figure 9. Number and Size of Partitions 61
Figure 10. Row Addressing within Page by Record-ID 63
Figure 11. Field Formats used by DB2 64
Figure 12. Index Tree Structure (Clustering Index) 66
Figure 13. Types of Table Space Locks 68
Figure 14. Table Space Locking 68
Figure 15. Data Extract (DXT) 75
Figure 16. Application Development Cycle 78
Figure 17. SQL Communication Area (PL/I Format) 81
Figure 18. Data Types of Host Variables 88
Figure 19. SQL Descriptor Area (PL/I Format) 91
Figure 20. DB2 ISPF Menus 100
Figure 21. DB2I Main Option Menu 101
Figure 22. SPUFI Execution Flow 102
Figure 23. DB2 Authorization-IDs 113
Figure 24. DB2 Log Hierarchy 131
Figure 25. DB2I Utilities Menu 135
Figure 26. Data Base Recovery Cycle 138
Figure 27. The MVS DB2 Environment 147
Figure 28. Subsystem and Batch Connection to DB2 151
Figure 29. Thread Identification 153
Figure 30. Communication Protocol with DB2 154
Figure 31. IMS Environment with DB2 164
Figure 32. CICS Environment with DB2 172
Figure 33. CICS Resource Control Table 173

CHAPTER 1. INTRODUCTION

This book introduces **IBM DATABASE 2 (DB2)**, a data base management system that provides a relational model of data. DB2 runs as a subsystem of MVS. The book's purpose is to enable managers, system programmers, data base support personnel, and other interested persons to evaluate DB2 for use in their organization. The book explains the **Concepts and Facilities** of the DB2 product.

ORGANIZATION OF THE BOOK

The remaining chapters of this book are as follows:

"Chapter 2. The DB2 Environment": This chapter introduces DB2 and the environment in which it is intended to run.

"Chapter 3. Relational Data Model": This chapter explains the fundamental ideas of the relational data model. Most of the content is independent of the actual implementation of the model by DB2.

"Chapter 4. SQL Data Manipulation": This chapter introduces the Structured Query Language (SQL) which is used for retrieval and update of relational data. A number of small examples is included to illustrate major aspects of the language.

"Chapter 5. Data Definition": This chapter explains the physical and logical objects used to define and store relational data, and the Data Definition Language (DDL) of SQL which is used to perform these tasks.

"Chapter 6. Data Management": This chapter describes how DB2 manages the physical data of relational tables. Physical data management, techniques for concurrent access, and initial loading of data is presented. An additional program product to extract data from existing DL/I, VSAM and SAM files, the Data Extract (DXT), is introduced.

"Chapter 7. Application Programming Environment": This chapter explains how application programs are coded and executed in a DB2 environment. The usage of SQL in application programs is introduced, and the process of binding (finding the best access paths) is described.

"Chapter 8. TSO Environment": This chapter explains how DB2 fits into the TSO environment under MVS. The interactive facility of DB2 (DB2I) under TSO/SPF, and an additional program product, the Query Management Facility (QMF), are introduced.

"Chapter 9. Security and Authorization": This chapter describes the security and authorization facilities of DB2.

"Chapter 10. The DB2 Catalog": This chapter introduces the DB2 catalog. The catalog is a set of relational tables used to describe and document all the objects (tables, etc) used by DB2.

"Chapter 11. Operation and Recovery": This chapter presents the operational aspects of DB2. The commands to control and operate DB2, the logging function and recovery of system and data, and all the DB2 utilities are described.

"Chapter 12. Architecture": This chapter explains the architectural structure of the DB2 system. The MVS address spaces and the functions they provide are presented.

"Chapter 13. Monitoring and Accounting": This chapter describes the parameters which allow the user to influence the performance of the system, as well as the facilities used to gather the required data for monitoring and accounting purposes.

"Chapter 14. IMS Environment": This chapter explains how application programs running under IMS may access DB2 relational data. It also shows how both systems (DB2 and IMS) are synchronized to commit or roll back data changes.

"Chapter 15. CICS Environment": This chapter explains how application programs running under CICS may access DB2 relational data. It also shows how both systems (DB2 and CICS) are synchronized to commit or roll back data changes.

"Chapter 16. Installation and Servicing": This chapter describes the installation process for DB2.

"Appendix A. Compatibility between DB2 and SQL/DS": In this appendix the differences in the implementation of the relational model and of the SQL language between DB2 and SQL/DS are explained.

RELATED IBM PROGRAM PRODUCTS

The following acronyms will be used throughout the document and always refer to the product mentioned below:

CICS Customer Information Control System / Operating System / Virtual Storage, Program Number 5740-XX1

DXT Data Extract, Program Number 5668-973

IMS Information Management System / Virtual Storage, Program Number 5740-XX2

QMF Query Management Facility, Program Number 5668-972

ISPF Interactive System Productivity Facility, Program Number 5668-960, and Interactive System Productivity Facility / Program Development Facility, Program Number 5665-268.

SQL/DS Structured Query Language / Data System, Program Number 5748-XXJ

IBM DATABASE 2 (DB2) is IBM's relational data base product for MVS establishments. DB2's primary direction is as an integral part of the data system in the MVS environment. As such, it can coexist as a Data Base Management System (DBMS) with IMS DB (DL/I) and in addition can be become the initial DBMS for new users.

The DB2 subsystem may be accessed concurrently by the IMS and CICS transaction managers, by TSO terminal users, or by TSO batch jobs. DB2 gives the IMS and CICS user the ability to access both DB2 and DL/I data from within the same application program. DB2 is used in the MVS environment as the base DBMS for the new Query Management Facility (QMF).

DB2 provides a relational data model. Data is defined and accessed in terms of tables. Tables consist of columns and rows. Since tables are simple and familiar, most users can easily understand them. Views may be defined on tables such that user/application "logical" tables need not conform to actual stored tables and may subset tables by rows and/or columns. This allows security to be controlled at the field content level.

Data definition, access, manipulation and authorization operations are supported by the Structured Query language (SQL). SQL is also used by SQL/DS, which is a relational system for the DOS/VSE and VM environments. SQL is a high level data language available to users through a TSO/ISPF interactive terminal interface (DB2I) and application programs written in Assembler, COBOL, PL/I, or FORTRAN.

DB2 promotes the continuous operation environment by providing dynamic creation and modification of descriptors, dynamic security definition, and on-line execution of utilities. The DB2 utilities (recovery, reorganization, etc) can operate on a complete table or a subset of a table (partition).

DB2 provides a recovery mechanism that ensures the integrity and recoverability of data regardless of the presence or absence of any other subsystem. Services such as logging and recovery are an integral part of the DB2 subsystem itself and operate in a coordinated manner with any transaction manager(s) which may be present.

One of the main objectives of DB2 is to reduce application design and development effort and to provide Ease of Use capabilities.

SYSTEM ENVIRONMENT

The DB2 subsystem may be accessed concurrently by IMS and/or CICS as transaction managers, by TSO on-line users, or by TSO batch

jobs. Figure 1 on page 6 shows the configurations that are possible using IMS DB (DL/I) and DB2 as data base managers (QMF is the new query product and DB2I is a DB2 supplied interactive front end). The detailed components of the diagram are discussed in later sections of this document.

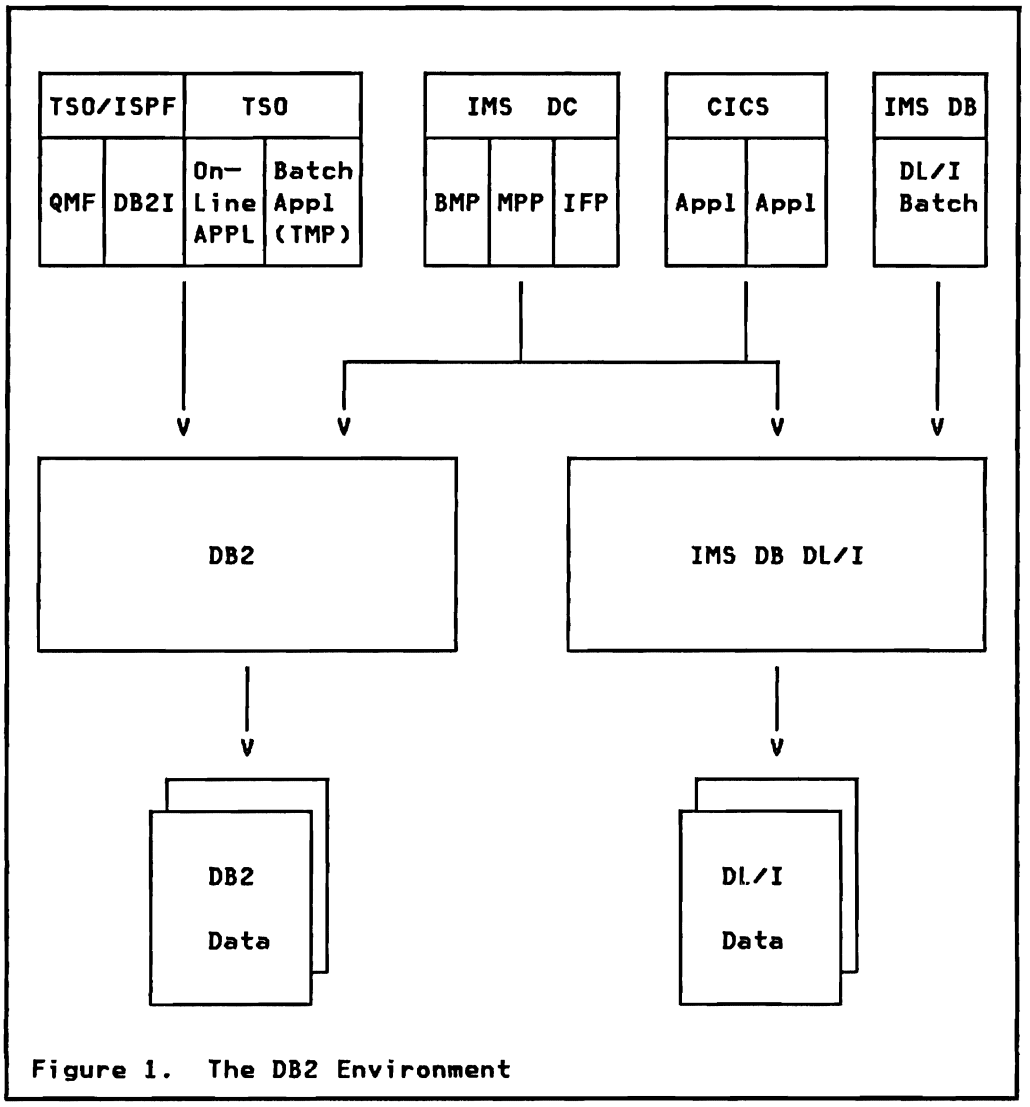


Figure 1. The DB2 Environment

This Chapter describes the fundamental ideas of the relational data model used by DB2. A **relation** in the relational data model can be thought of as a simple two-dimensional table having a specific number of columns and some number of unordered rows. Each row contains either a value or a null entry in each column.

Data is defined and accessed in terms of tables and operations on tables. The tabular format of data is easy to use. Simple data needs can be implemented very easily, and complex ones can be handled through a powerful set of operations on tables. Thus, the relational model supports a broad range of data requirements.

TABLES

In DB2 all data is presented as tables consisting of rows (horizontally) and columns (vertically), as shown in Figure 2.

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOBCODE	EDUCLVL	SALARY
000010	Haas	A00	3978	66	18	52750
000020	Thompson	B01	3476	61	18	41250
000030	Kwan	C01	4738	60	20	38250
000050	Geyer	E01	6789	58	16	40175
000060	Stern	D11	6423	55	16	32250
000070	Pulaski	D21	7831	56	16	36170
000100	Spenser	E21	0972	54	14	26150
000140	Nicholls	C01	1793	56	18	28420
000150	Adamson	D11	4510	55	16	25280
000220	Lutz	D11	0672	55	18	29840
000260	Johnson	D21	8953	52	16	17250
000270	Perez	D21	9001	55	15	27380
000300	Smith	E11	2095	48	14	17750
000310	Setright	E11	3332	46	12	15900

Figure 2. The EMPLOYEE Table

Tables in a relational data base are no different from any other tables. They are the same familiar and easy-to-use data structures you see and use every day in telephone books, airline schedules, newspapers, and many other places. Tables physically stored in DB2 relational data bases are called base tables.

Columns

Columns typically describe various aspects of some thing (such as a person). Each of the columns of Figure 2 on page 7 describes one aspect of an employee. The order of the columns is not significant since they are identified by their names and not by an implied column-order. Each column name may appear only once in a table. All values in a column have the same characteristics; for example all values in the JOBCODE column are integers.

Rows

A row in a table corresponds very much to a record of a file. A table row is the smallest unit of insertion and deletion. An insert operation adds one or more rows to a table, and a delete operation removes one or more rows from a table. Rows have no inherent order. Users have the flexibility to retrieve rows in the order they choose on individual requests. DB2 provides facilities to enforce uniqueness of rows.

Data Types

DB2 supports the following data types as field values:

INTEGER	Fullword 31-bit signed binary values.
SMALLINT	Halfword 15-bit signed binary values.
FLOAT	Double word long floating point values.
DECIMAL(p,q)	Packed decimal values of "p" (1 to 15) digits. A number of decimals "q" (0 to "p") to the right of an implied decimal point may be specified.
CHAR(n)	Fixed length character value of "n" (1 to 254) characters.
VARCHAR(n)	Variable length character value of up to "n" (1 to 32674) characters. The actual maximum length is dependent on physical storage characteristics and is about 4K (or optionally 32K).
LONG VARCHAR	This specification is equivalent to VARCHAR with a system calculated maximum and is supported for compatibility with the SQL/DS product.[1]

---

[1] Structured Query Language/Data System, Program Number 5748-XXJ

Nulls

Any field value in a table, regardless of its data type, may have the special value **null**. Null represents a value that is unknown or not applicable. A null value may be thought of as an empty space, or a space reserved for later insertion of data. Nulls can occur in a table by explicit user request through an insert or update operation, or implicitly when a new column is added to an existing table. Null values can be prohibited for specific columns when a table is created.

Primary Key

A primary key is a column, or a combination of multiple columns, that distinguishes a specific row from all other rows. A primary key provides uniqueness for that row. In most tables a single column will be enough to serve as the key, in some tables however it may be required to combine all the columns to form the key.

There may be more than one column that may serve as a possible primary key. Such columns (or combination of columns) are then called candidate keys, and one of them is normally chosen as primary key. The remaining candidate keys are called alternate keys.

Note: Primary key is not a DB2 term. The concept of uniqueness is supported in DB2 through indexes.

Uniqueness

As just seen above a candidate key provides uniqueness of rows within a table. In DB2 an **index** built on one or more columns is the vehicle to enforce this uniqueness. Indexes are discussed in more detail in "Chapter 5. Data Definition" under the heading "Indexes" on page 41.

Related Tables

Two tables may be related when they have a similar set of field values in a column. Since this is a crucial facility of the relational data base model it is easier to understand with an example as shown in Figure 3 on page 10.

DEPTNO	DEPTNAME	MGRNO
A00	Spiffy Computer Co	000010
B01	Planning	000020
C01	Info Center	000030
D01	Dev Center	
E01	Support Services	000050
D11	Manufacturing	000060
D21	Admin	000070
D31	Order Processing	
E11	Operations	000090
E21	Software Support	000100

Figure 3. The DEPARTMENT Table

The DEPTNO column of the department table consists of department numbers which are also used in the WORKDEPT column of the employee table in Figure 2 on page 7.

This relationship between the employee and the department table is through field values only. There are no physical links (like pointers) between the two tables.

- In a relational data model all relationships are represented through field values
- A table may have many such implied relationships to other tables

Note: There is another relationship between the employee and the department table through the columns EMPNO (in employee table) and MGRNO (managers employee number in department table).

Foreign Key

When the values of a primary key of one table is used in another table, the value of the key is referred to as a foreign key in the second table. In the department table of Figure 3 DEPTNO will most likely be the primary key for that table. Whenever the DEPTNO value is used in another table it is a foreign key (e.g. WORKDEPT in the employee table).

VIEWS

Although VIEWS are not part of the relational data model they are an important concept within the implementation of these concepts in DB2. For this reason the term VIEW is introduced here in the

conceptual overview chapter so that later chapters may refer to this definition.

A view is a logical (or "virtual") table that is derived from one or more tables. Views can also be derived from other views or combinations of views and tables. Views look like stored tables. They have rows and columns, and do not have an inherent order of rows. In general views can be used as if they were tables. Data in a view is not stored as a separate set of rows but is retrieved from the underlying base tables when the view is used for data manipulation.

Common uses of views are:

- To simplify data retrieval commands

  Data retrieval commands can be based on views such that data access requests made by terminal users or programs can be expressed in simpler terms. This capability can decrease both keystrokes and errors, especially in the use of complex data retrieval commands. Views reduce complexity by presenting a smaller and simpler version of the table upon which they are based.

- To limit user (or program) access to data

  A view can be used like a mask or filter to limit a user's perception of the stored data. For example, if a table holds a variety of data about employees, only specific persons should have access to salary data in the table. Views can be defined so that each user of the table is permitted to access and manipulate only specific parts of the table.

Table Subsets

Many views are subsets of base tables. These include:

- **Subset of the rows** of a base table. Such subsets are normally formed by selecting only certain values within one or more columns. An example would be all the employees of Figure 2 on page 7 with an education level (EDUCLVL) of 18.

- **Subset of the columns** of a base table. Such subsets are formed by selecting only certain columns of the base table. An example would be the employee table reduced to the columns EMPNO, LASTNAME, and WORKDEPT.

- **Row and column subset**. Any combination of row selection (by values) and column selection.

Derived and Calculated Fields

Views may include fields which are not part of the underlying base tables. Such fields include:

- Virtual fields where the field value is based on other fields within the same row (e.g. sum of quantity in stock plus quantity on order).
- Derived fields where the field value is calculated from a group of rows. An example would be the average salary by department, calculated from all the rows (employees) of each department.

Table Combinations

Views may be formed by a combination of multiple tables. Very often such a combination is formed using a common field. The operation involved here is called a **join** of tables.

An example of a view formed by joining the employee table (Figure 2 on page 7) with the department table (Figure 3 on page 10) based on the common field "department number" is shown in Figure 4. The resulting table includes selected columns of the employee table combined with the department name from the department table.

EMPNO	LASTNAME	DEPTNO	DEPTNAME
000010	Haas	A00	Spiffy Computer Co
000020	Thompson	B01	Planning
000030	Kwan	C01	Info Center
000050	Geyer	E01	Support Services
000060	Stern	D11	Manufacturing

Figure 4. Join of EMPLOYEE and DEPARTMENT tables

TABLE DESIGN

From a theoretical standpoint the design of a data base on a logical level should be the same for any data base management system. The logical design effort should resolve questions on which records are best suited to model the real business, which fields should go together in a record type, and what relationships exist between record types.

The responsibility of the physical design task is to structure these record types into physical data bases according to the requirements and facilities of the actual data base management system. An experienced designer, however, will take into account the available physical options already during the logical design phase.

With this reality applied to relational data base design a very important characteristic of the relational data base approach stands out. Here all relationships between data are modelled in the same way: as values in fields in the actual tables that are stored as physical data bases. This eliminates many complex considerations like which relationships should be implemented in the data structures (hierarchies or networks) and which should still be implemented as values. This leads to a simplification going from logical to physical design. The set of records derived in the logical design maps directly into relational tables.

The advantage of the relational approach with respect to data base design activity is that table structures will have a better chance to survive changing requirements and needs.

Flexibility

The relational approach gives full flexibility in regard to changing data requirements.

- Additional columns can be added to a table at any time without the requirement to unload and reload the table. Any existing method of working with the data (program or terminal user command) need not be changed unless they are to use the data in the new column. Values in new columns are null (absent) until some user or program fills values into each row that needs one.

- View definitions may insulate a design from subsequent changes. Users of an existing view are in general not impacted when the underlying base tables are changed.

- There will be less impact due to changes because all relationships are already carried through actual data values in the tables and not through links in an explicit data structure.

- If the data base design is not perfect from the very beginning, a relational data base system makes it easier to correct. A relational data base system is more forgiving to an incomplete data base design.

CHAPTER 4. SQL DATA MANIPULATION

SQL, the **Structured Query Language** is intended to provide access to data for users who have little training in computer programming. SQL is a high-level language for handling data stored in the form of tables as introduced in "Chapter 3. Relational Data Model." SQL is high-level in that data requests in SQL specify only the desired results, not data paths to effect the desired results. In other words, with SQL you specify **what you want** and **not how to get it.**

The full SQL as implemented in DB2 consists of three major components:

1. The **Data Manipulation Language** which is described in this chapter.

2. The **Data Definition Language** as presented in the next chapter.

3. The **Authorization Language** as presented in "Chapter 9. Security and Authorization."

SQL INTRODUCTION

SQL is the language used in DB2 for data manipulation. This language not only includes statements for retrieval operations as the acronym might suggest, but also modification operations for replace, insertion, and deletion of data.

SQL is a powerful language. **One SQL request may deal with all or selected rows of a table** (or multiple tables) **and access all or selected columns within these rows.**

The result of a SQL retrieve operation is again a table which is presented to the terminal user as a table, or to a user program one row at a time.

Automatic Navigation

DB2 uses the SQL statements provided by users to "automatically navigate" to the requested data. Users do not have to know how data is represented in storage in order to retrieve and use it. DB2 finds its own way to the data. When DB2 locates the requested data, it returns, updates, or deletes the entire collection of data that meets the conditions specified by the user.

As DB2 operates on collections of data and automatically selects a path to that data, DB2 users can be more productive. Automatic navigation allows them to concentrate on the essential logic and

data requirements of their application, instead of concerning themselves with the details of data representation and access paths.

Automatic Access Path Selection

In many data base management systems the application programmer must decide whether to process data sequentially or use one of possibly several indexes that the installation may have defined for that data. DB2 makes these decisions automatically. In fact, programmers never reference indexes in their programs.

DB2 selects an access path on request after the program has been written, or when changes to the underlying tables and indexes invalidate the current access path.

This ability to select an access path minimizes program maintenance for the installation. For example if the data base administrator decides to drop a seldom used index, DB2 will automatically select a new access path for those transactions that made use of that specific index. The logic of application programs themselves is unaffected and no program maintenance is required to cope with this change.

Interactive SQL

SQL statements may be issued interactively from a terminal, and the results may be browsed afterwards. This interactive SQL facility is available to authorized TSO SPF users. The technique is described in detail in "Chapter 8. TSO Environment" section "ISPF Support - DB2I" on page 100.

Embedded SQL

SQL statements may be embedded in application programs written in COBOL, PL/I, FORTRAN, and Assembler.

When SQL statements are embedded in programs they have a slightly different format. This format is described in "Chapter 7. Application Programming Environment" section "SQL Statement Format" on page 79.

DATA MANIPULATION LANGUAGE (DML)

The data manipulation part of SQL deals with retrieving, updating, inserting, and deleting of data stored in tables. In the following sections each major aspect of the language will be covered.

The SQL language presented here applies to interactive SQL as entered from a TSO terminal, as well as to embedded SQL used in application programs.

Data Retrieval

Data retrieval is by far the most basic task of DB2. Most programming and data languages process a record at a time. To use them, you code a sequence of instructions explaining how to get to data, what to look for, and what to do with it. With SQL you don't have to specify all this information. You select data with a single powerful **SELECT** statement and specify just what data you want.

SELECT Statement

To retrieve data SQL provides the **SELECT** statement. The basic form of the SELECT statement is:

```
SELECT what data
  FROM which tables
 WHERE search conditions
```

This format is similar to the kind of thinking one might use in retrieving specific information from some collection of data. A SELECT statement is just a formalized shorthand notation for asking questions about data. It is divided into three clauses, each of which helps to specify what you are looking for. The SELECT clause specifies the columns and the FROM clause the table or view from which you are going to retrieve data. The WHERE clause specifies all the conditions the data must meet in order to be retrieved.

Let us look at a simple example in Figure 5 on page 18. From the employee table we want to retrieve the last name, employee number, and work department of employees with an education level of 16.

This simple example shows selection of columns in the order wanted, and selection of rows through a condition on field EDUCLVL.

Notice the result of the SELECT statement. It is a set of rows, all of which satisfy the conditions specified in the **WHERE clause**. SQL works with sets of data, it retrieves or modifies all the rows in a table that meet the specifications of a single WHERE clause.

The SELECT statement:

```
SELECT LASTNAME, EMPNO, WORKDEPT
  FROM EMPLOYEE
 WHERE EDUCLVL=16
```

The resulting table:

LASTNAME	EMPNO	WORKDEPT
Geyer	000050	E01
Stern	000060	D11
Pulaski	000070	D21
Adamson	000150	D11
Johnson	000260	D21

Figure 5. A simple SELECT statement

All the rows of a table qualify if no WHERE clause has been specified. More details of the WHERE clause will be presented later in this chapter in section "Search Conditions" on page 22 because the WHERE clause applies to other SQL statements as well.

SELECT Clause: The SELECT clause specifies the fields (columns) to be selected or calculated. The format of the SELECT clause is:

```
SELECT [DISTINCT] expression, expression , ...
```

- If DISTINCT is specified it means that duplicates are to be eliminated from the query result as a whole. The resulting set of rows will therefore not include any duplicate rows. SELECT DISTINCT JOBCODE will get all the different JOBCODEs from the employee table.

- Each expression in the SELECT clause may be a simple column name or an arithmetic expression involving constants and column names from the selected table. Examples are:

  ```
  EDUCLVL
  PRICE * 0.85
  QONHAND + QONORDER
  ```

- Expressions in the SELECT clause may also involve builtin functions. These will be presented later in this chapter in section "Builtin Functions" on page 24.

- To select all columns from a table you can use the shorthand notation SELECT * FROM table.

FROM Clause: The FROM clause specifies one or more tables from which data is selected. Examples of selections from multiple tables are presented later in this chapter in sections "Simple Join" on page 28 and "Advanced DML" on page 30.

Updating Data

You update data in tables using the SQL **UPDATE** statement. The purpose of the UPDATE statement is to update one or more fields in one or more rows of a table. In general the UPDATE statement changes a set of rows and not just a single row (record).

UPDATE Statement

The SQL UPDATE statement is very similar to the SELECT statement. You must specify the same information about the data you wish to update, that is, the table that contains the data, the columns in which the data occurs, and a qualifying WHERE clause to select the proper rows. The basic form of the UPDATE statement is:

```
UPDATE which table
   SET new field values
 WHERE search conditions
```

The UPDATE clause specifies the table to be updated. The SET clause specifies one or multiple new field values, which may be constants or arithmetic expressions. The WHERE clause specifies all the conditions to be met by the rows to be updated. The details of the WHERE clause will be covered later in this chapter.

Let us look at some examples of UPDATE statements.

- To change the department number for all employees of a particular department in the employee table you would enter:

```
UPDATE EMPLOYEE
   SET WORKDEPT = 'E31'
 WHERE WORKDEPT = 'E01'
```

- To update multiple field values for a particular employee you would enter:

```
UPDATE EMPLOYEE
   SET PHONENO = '3623', JOBCODE = 64,
       EDUCLVL = 22
 WHERE EMPNO = '000030'
```

- To update all the rows of a table and give all the employees a 5% rise you would enter:

```
UPDATE EMPLOYEE
   SET SALARY = SALARY * 1.05
```

Deleting Data

You delete data in tables using the SQL **DELETE** statement. The purpose of the DELETE statement is to delete one or more rows of a table.

DELETE Statement

The SQL DELETE statement uses a syntax similar to the SELECT statement. You must specify which table contains the rows to be deleted and a qualifying WHERE clause to select the proper rows. The basic form of the DELETE statement is:

```
DELETE
  FROM which table
 WHERE search conditions
```

The FROM clause specifies the table from which one or multiple rows will be deleted. The WHERE clause specifies all the conditions to be met by the rows to be deleted. The details of the WHERE clause will be covered later in this chapter.

Let us look at some examples of DELETE statements.

- To delete all the rows of a particular department from the employee table you would enter:

```
DELETE
  FROM EMPLOYEE
 WHERE WORKDEPT = 'B01'
```

- To delete all the departments which have no manager assigned from the department table you would enter:

```
DELETE
  FROM DEPARTMENT
 WHERE MGRNO IS NULL
```

- To delete all the data (rows) from a table you would enter:

```
DELETE
  FROM table
```

  Note that although all data has been deleted from the table, it would still exist and new data could be entered immediately.

Inserting Data

To insert data into a table you use the SQL **INSERT** statement. The purpose of the INSERT statement is to add one new row to a table, or to add a whole set of rows selected from other tables.

INSERT Statement

The INSERT statement has two basic formats to add either one row, or multiple rows, to a table.

1. Simple insertion of one new row to a table:

```
INSERT INTO which table
            (list of columns)
     VALUES (list of values)
```

   In this format of the INSERT statement a single new row is added to a table. The new row is formed by placing various data-values into the named fields (columns), in the order named. All fields of the given row which are not named receive the null value. The list of columns may be omitted which is the same as naming all the columns in their normal order. Another way to force the insertion of null values is to use the keyword NULL in the value list.

2. Insertion of multiple rows selected or computed from other tables by a SELECT statement:

```
INSERT INTO which table
            (list of columns)
     SELECT columns
       FROM table(s)
      WHERE search conditions
```

   In this format of the INSERT statement many rows may be selected (and computed) from other tables and inserted into the target table. Rows which already exist in the target table are not affected by the insertion. The table(s) from which the inserted rows were selected are not affected at all. The number of fields selected must agree with the number of columns named in the list. The fields selected must be type-compatible with the target fields into which they are to be inserted. Data conversions between different numeric formats will be performed.

Note: DB2 does not impose any logical ordering on the rows of a table, and therefore no facility is provided to specify the "position" in the table of the newly inserted rows.

Now let us look at some examples of INSERT statements.

- Insertion of a new employee into the employee table

```
INSERT INTO EMPLOYEE
            (EMPNO, LASTNAME, WORKDEPT,
             PHONENO, EDUCLVL)
     VALUES ('000330', 'LEE', 'E21',
             '2103', 14)
```

  The field JOBCODE is set to null since it is not in the list of columns. The same result may be achieved through an INSERT statement without a column-list and marking the null field with the keyword NULL:

```
INSERT INTO EMPLOYEE
     VALUES ('000330', 'LEE', 'E21',
             '2103', NULL , 14)
```

- Insert selected employees with an education level of 18 into a work table for further evaluation:

```
INSERT INTO EMPWORK
            (EMPNO, NAME, DEPARTMENT)
     SELECT (EMPNO, LASTNAME, WORKDEPT)
       FROM EMPLOYEE
      WHERE EDUCLVL = 18
```

Search Conditions

One of the basic operation in SQL is to search through a table, choosing certain rows for processing. The criterion for choosing rows is called a "search condition." A search condition is specified in the WHERE clause of SELECT, UPDATE, and DELETE statements.

A search condition is a collection of one or more "predicates", connected by the logical connectors AND, OR, and NOT. Each predicate specifies a test to be applied to the rows of the table. For example the following search condition contains three predicates to find employees of department E01 with an education level greater than 20 or a job code of 54:

```
WHERE WORKDEPT = 'E01' AND
      (EDUCLVL > 20 OR JOBCODE = 54)
```

The basic format of a predicate is a comparison between two values or expressions. We represent this format as follows:

```
<expression> <comp-operator> <expression>
```

An <expression> may consist of a field name, a constant, or any combinations of these connected by one of the four arithmetic

operators (+ - * /). A <comp-operator> is one of the comparison operators (> >= = <= < ¬ ¬= ¬> ¬<).

Parentheses may be used in arithmetic expression and to group predicates to override precedence rules of operators. Single quote-marks must be used around character strings.

In constructing search conditions, the user should be careful to perform arithmetic only on numeric data types, and to make comparisons only between compatible data types.

SQL provides four special kinds of predicates which may be used in search conditions, in addition to the standard predicates which compare two expressions. These special predicates are:

BETWEEN The format of the BETWEEN predicate is as follows:

```
<expr> [NOT] BETWEEN <expr1> AND <expr2>
```

BETWEEN is therefore a shorthand notation for a greater-equal and a less-equal comparison. Example:

```
EDUCLVL BETWEEN 15 AND 20
```

IN The format of the IN predicate is as follows:

```
<expr> [NOT] IN <constant-list>
```

This type of predicate makes it possible to quickly compare the value of an expression with a list of constants. Example:

```
JOBCODE IN (54, 56, 60)
```

NULL The format of the NULL predicate is as follows:

```
<field-name> IS [NOT] NULL
```

A row of a table satisfies this predicate if the value of the designated field is (or is not) null.

LIKE The special LIKE predicate enables a user to search character string data which partially matches a given string. Its format is as follows:

```
<field-name> [NOT] LIKE 'string'
```

The quoted string on the right side of the LIKE is called a "pattern." The pattern may contain any character string, with special meanings for the characters "_" and "%". The "_" character represents any single character and the "%" character represents any string of zero or more characters. Examples:

```
LASTNAME LIKE '%MIT%'
   (would select SMITH, SCHMIT, MITTERAND, ...)

LASTNAME LIKE 'L__'
   (would select LEE, LEY, LOU, ...)
```

All possible predicates may be used in one single WHERE search condition as illustrated in the following example:

```
SELECT EMPNO, LASTNAME, JOBCODE,
       EDUCLVL, WORKDEPT
  FROM EMPLOYEE
 WHERE EMPNO BETWEEN '000010' AND '002000'
       AND ( LASTNAME LIKE 'K%' OR
             LASTNAME LIKE 'L%' )
       AND JOBCODE NOT NULL
       AND EDUCLVL IN (16, 18, 20)
       AND WORKDEPT > 'D01'
```

USER Keyword

The special keyword USER may be used within search conditions. It is set equal to the user-id of the user who is issuing the SELECT statement. USER behaves exactly like a fixed length character string constant of length 8, with trailing blanks if the user-id has less than eight characters.

Suppose the user-id is the employee number then the following SELECT will return the employee data of the issuing user:

```
SELECT *
  FROM EMPLOYEE
 WHERE EMPNO = USER
```

The user-id is identical to the TSO logon-id, the IMS SIGN-ON-id (or logical terminal name), and the CICS sign-on-id (or transaction code, terminal name, etc).

Builtin Functions

The following builtin functions can be used in the SELECT clause in addition to simple field-names and arithmetic expressions:

```
AVG  SUM  MIN  MAX  COUNT
```

The argument of a builtin function may be a field name (optionally preceded by DISTINCT), or an expression. DISTINCT means that duplicate values are to be eliminated before the function is applied. For example, COUNT(DISTINCT JOBCODE) computes the number of different job codes in the rows that satisfy the search condition.

AVG Computes the average of a numeric field

SUM Computes the sum of a numeric field

MIN Finds the minimum value of any field

MAX Finds the maximum value of any field

COUNT Counts either different field values when used as COUNT(DISTINCT field) or counts the number of qualifying rows when used as COUNT(*).

Since builtin functions compute a result over a set of rows they cannot be mixed with expressions which do not contain builtin functions. That is, every expression in the SELECT clause must use one of the builtin functions, or none can. Null values are ignored by builtin functions. Example:

```
SELECT COUNT(*), AVG(PRICE), MIN(PRICE),
       MAX(PRICE), SUM(PRICE * QUANTITY)
  FROM QUOTATIONS
```

Grouping - GROUP BY Clause

The grouping feature of DB2 permits a table to be conceptually divided into groups of rows with matching values in one or more fields, and then applies one or more builtin functions for each group. The fields which form the definition of the groups are listed in a special GROUP BY clause. The following example query finds the number of employees and their average salary for each department:

```
SELECT WORKDEPT, COUNT(*), AVG(SALARY)
  FROM EMPLOYEE
 GROUP BY WORKDEPT
```

A SELECT statement using the grouping feature returns only one row for each group. Therefore, the items selected by such a query must be properties of the group, not properties of individual rows. The SELECT clause may contain the fields contained in the GROUP BY clause, together with builtin functions on any other fields.

A grouping SELECT may have a standard WHERE clause which serves as a filter, keeping only those rows which satisfy the search condition. For example, to find the number of high skilled employees grouped by department and sex.

```
SELECT WORKDEPT, SEX, COUNT(*)
  FROM EMPLOYEE
 WHERE EDUCLVL > 17
 GROUP BY WORKDEPT, SEX
```

HAVING Clause

It is also possible to apply a qualifying condition to groups, causing the system to return a result only for those groups which satisfy the condition. This is done by a special HAVING clause which is written after the GROUP BY clause. The HAVING clause contains one or more group-qualifying predicates, connected by AND's and OR's. Each group-qualifying predicate compares some property of the group such as AVG(PRICE) with another group property or constant. The following example will only return results for departments with more than 5 employees:

```
SELECT WORKDEPT, COUNT(*), AVG(EDUCLVL)
  FROM EMPLOYEE
 WHERE JOBCODE > 15
 GROUP BY WORKDEPT
HAVING COUNT(*) > 5
```

Ordering - ORDER BY Clause

The results of a SELECT may be returned in a user defined order through an **ORDER BY** clause. Ascending and descending ordering is supported. Ordering may be requested by one or more items (fields or expressions) of the SELECT clause. These items are specified either by field name or by an integer number denoting an item number.

The ORDER BY clause is the last clause specified on a SELECT statement. If no ordering is requested rows are returned in system-determined order.

In the following example all employees are selected and returned in order of education level (descending) and salary (by default ascending).

```
SELECT EMPNO, LASTNAME, EDUCLVL, SALARY
  FROM EMPLOYEE
 ORDER BY 3 DESC, SALARY
```

Use of Nulls

Tables may be defined to allow null values in certain columns. If null values are allowed then users must be very clear about the distinction between null and the zero (for numeric fields) or blank (for character fields) value. A null value can signify that the value is not yet assigned, is unknown, or is not applicable.

Here are some considerations which must be taken into account when null values are allowed in a column:

- Null values can add complexity to the programming (or retrieving) task.

- Null values act in special ways when using search conditions.

  1. A null value never qualifies in a comparison predicate of a search condition. A null value is not greater, smaller, equal, or not equal than any other value.

  2. A null value is not even equal to another null value.

  3. A null value can only qualify when using the special null predicate:

     ```
     fieldname IS [NOT] NULL
     ```

- When inserting a row into a table values must be provided for all the columns where nulls are not allowed.

Combining Data

The relational operation that allows you to retrieve data from two or more tables by matching them on a common column is called a **join**.

Let us review the join operation we have performed in Figure 4 on page 12. We combined the employee table with the department table based on the common values of "department numbers" which are called WORKDEPT in the employee table and DEPTNO in the department table.

The SELECT statement which performs this operation reads as follows:

```
SELECT EMPNO, LASTNAME,
       DEPTNO, DEPTNAME
  FROM EMPLOYEE,
       DEPARTMENT
 WHERE WORKDEPT = DEPTNO
```

In the SELECT clause we specify the names of the columns we want to retrieve. In the FROM clause we specify the two table names where these columns exist. In the search condition of the WHERE clause we specify the compare operation to be performed. For all the matching department numbers the SELECT statement will retrieve EMPNO and LASTNAME from the employee table, DEPTNO and DEPTNAME from the department table, and merge them into the resulting table of Figure 4.

Rules for row qualification:

- A row from one of the participating tables which does not qualify the join-condition will not appear in the result of the join.

- A join condition is never satisfied by a null value.

- If no join-condition is given in the WHERE clause, all possible combinations of rows from tables in the FROM clause will be returned. This type is called "Cartesian Product" and does not usually produce a meaningful result.

Simple Join

Simple joins are combinations of two tables based on one common column as the example shown above.

The same employee and department tables may also be joined by another common set of values. These are "employee numbers" called EMPNO in the employee table and MGRNO in the department table.

Such a join may be used to get a table of the departments with the name of the manager replacing the managers personnel number. See Figure 6 for the required SELECT statement and the resulting table.

The SELECT statement:

```
SELECT DEPTNO, DEPTNAME, LASTNAME
  FROM DEPARTMENT, EMPLOYEE
 WHERE MGRNO = EMPNO
```

The resulting table:

DEPTNO	DEPTNAME	LASTNAME
A00	Spiffy Computer Co	Haas
B01	Planning	Thompson
C01	Info Center	Kwan
D11	Manufacturing	Stern
D21	Admin	Pulaski
E01	Support Services	Geyer
E21	Software Support	Spencer

Figure 6. Simple Join

Note: This table includes only departments which have a manager assigned (null values cannot qualify) and for which the managers number is actually included in the employee table.

More about joins is presented later in this chapter in section "Advanced Join" on page 30.

USING VIEWS

Views as described in "Chapter 3. Relational Data Model" in section "Views" on page 10 may be used in SQL statements the same way as tables are used.

Assuming that a view has been defined for the employee table including only columns EMPNO, LASTNAME, and PHONENO, then the following SELECT statements are identical:

```
SELECT EMPNO, LASTNAME
  FROM EMPLOYEE
 WHERE EMPNO > 100

SELECT EMPNO, LASTNAME
  FROM view-name
 WHERE EMPNO > 100
```

The same rules apply to views and tables when retrieving data through SELECT statements.

Updating through Views

Views may also be used to update tables. However this facility is limited to views that are simple subsets of a single base table. The view must consist of a **subset of the columns and/or the rows of a single table.** A view which contains data from more than one table (e.g. a join) cannot be updated.

DELETE: Simple views (single table subsets) may be used unrestricted in DELETE statements. Any row which qualifies for the view may be deleted.

UPDATE: Simple views may be used in UPDATE statements to change field values of selected rows. If a view is a subset of rows it is possible that the changed row does not qualify for the view any more.

Suppose that the view includes only employees of department E01. An update statement is used to change the department number for one employee. This changed row no more qualifies for the view and cannot be retrieved through the view following the update. Such updates which invalidate rows according to the view definition may be prevented through the specification of a "check option". This is described in the next section.

INSERT: Simple views may be used in INSERT statements to add new rows to the underlying base table. The view must include all the columns which do not allow null values, otherwise an INSERT through the view is impossible. Field values must be specified for all columns which do not allow nulls.

You may insert a row which does not qualify to the view definition (row subset), unless the "check option" is applied to prevent such inserts.

With Check Option

When defining a view as a subset of rows a "check option" may be activated. This specification will prevent any UPDATE's or INSERT's of rows which do not qualify according to the view definition.

Suppose the view includes only rows of the employee table with a JOBCODE between 50 and 60. In this case the "check option" prevents from changing an employees job code outside the specified limits, and from inserting new rows with job codes outside these limits.

Note: The default is no checking, and therefore allows inserts of rows outside the view definition, and changes to rows so that the resulting row disappears from the view.

ADVANCED DML

The data manipulation language presented so far is just a subset of the enormous potential of SQL. SQL provides several features which enable complex data base queries, possibly involving several tables, to be expressed in a single SQL statement. These features, which are described in this section, may be used in combination with each other and with simpler language features described previously.

Advanced Join

The join operation which we have seen earlier in this chapter in section "Simple Join" has many more possibilities. More than two tables may be joined in a single operation, and the search condition may involve compare operators other than equal (e.g >, >=, etc).

Some of the advanced possibilities are presented here in form of examples, for the full details of the language consult the STSC SQL Usage Guide and/or the DB2 reference manuals.

Joining tables with identical column names

When joining two tables by a common column which has the same name in both tables we have to identify the individual columns used in the SELECT and WHERE clause.

Suppose the department column of the employee table had the name DEPTNO (instead of WORKDEPT). To perform the join illustrated in "Combining Data" on page 27 we would use the following SELECT statement:

```
SELECT EMPNO, LASTNAME,
       EMPLOYEE.DEPTNO, DEPTNAME
  FROM EMPLOYEE,
       DEPARTMENT
 WHERE EMPLOYEE.DEPTNO = DEPARTMENT.DEPTNO
```

By qualifying the column names with the table name we properly identify the identical column names in the SELECT clause (EMPLOYEE.DEPTNO) and in the WHERE clause (EMPLOYEE.DEPTNO = DEPARTMENT.DEPTNO).

Joining a table to itself

A similar technique as above must be used when joining a table to itself. The table name is repeated two or more times in the FROM clause, indicating that the join consists of combinations of two (or more) rows from the same table. Because the table name is no longer unique, each table name in the FROM clause must be given a unique table label of 1 to 18 characters.

Suppose we have to report pairs of employees with the same JOBCODE, but whose salaries differ by more than 1000. The following SELECT statement would perform this task:

```
SELECT E1.JOBCODE,
       E1.LASTNAME, E1.SEX, E1.SALARY,
       E2.LASTNAME, E2.SEX, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.JOBCODE = E2.JOBCODE
   AND E1.SALARY > E2.SALARY + 1000
```

This SELECT would return rows containing the job code and names, sex, and salaries of two employees where the first earns much more than the second.

Note: The EMPLOYEE table is labeled E1 and E2, and this label is used as column qualifier when comparing field values for employee "E1" with employee "E2".

Subselect

SQL allows a value or set of values in the search condition of a SELECT, UPDATE, or DELETE statement to be computed by another SELECT statement called a **SUBSELECT**. The result of a subselect is effectively substituted directly into the search condition (or HAVING clause) in which it appears.

If the subselect returns a single value, it can be used on the right-hand side of any comparison. If it may return more than one value the special predicate IN can be used, or one of the special functions ANY or ALL.

Subselect with single value: The following SELECT statement finds employees of job code 58 with a less than average salary for that job code:

```
SELECT EMPNO, LASTNAME, SALARY
  FROM EMPLOYEE
 WHERE JOBCODE = 58
   AND SALARY < ( SELECT AVG(SALARY)
                    FROM EMPLOYEE
                   WHERE JOBCODE = 58 )
```

The subselect returns the average salary which is then used to find employees whose salary is less than this average.

Subselect with IN predicate: The following SELECT statement reports project activities for employees of department E11:

```
SELECT *
  FROM PROJECT
 WHERE EMPNO IN ( SELECT EMPNO
                    FROM EMPLOYEE
                   WHERE WORKDEPT = 'E11' )
```

The subselect returns a list of employee numbers which is substituted into the IN predicate of the search condition.

ALL and ANY with Subselects

In the search condition the comparison operators =, ¬=, >, etc. can be modified by the keyword ALL or ANY when used with a subselect. These keywords permit the subselect to return a set of values, and they determine how the set is to be treated in the search condition. Let us look at the comparison operator '<' as an example, but the remarks below apply to the other operators as well:

expression < subselect	Subselect must return exactly one value to be compared against the expression (mostly a field name).
expression < ALL subselect	Subselect may return multiple values. The expression must be smaller than **each** of the values returned to qualify.
expression < ANY subselect	The expression must be smaller than **at least one** of the values returned to qualify.

The following example finds an employee of job code below 54, whose salary is greater than any other employees of job code below 58.

```
SELECT LASTNAME, SALARY
  FROM EMPLOYEE
 WHERE JOBCODE < 54
   AND SALARY > ALL ( SELECT SALARY
                        FROM EMPLOYEE
                       WHERE JOBCODE < 58 )
```

Correlated Subselects

In all the examples of subselects above, the subselect is evaluated once and the resulting value (or set of values) is substituted into the outer-level predicate. The **correlation** feature of DB2 permits a subselect to be executed repeatedly, once for each row of the outer-level select. This type of correlated subselect is used to compute some property of the outer-level row which is needed to evaluate a predicate.

The technique used is the same as for advanced joins. A correlation name is added to the table name in the FROM clause (same as table label) and used to qualify a column name with the outer level table name in the subselect.

The following SELECT returns departments from the department table which have more than 5 employees:

```
SELECT DEPTNO, DEPTNAME
  FROM DEPARTMENT D
 WHERE 5 < ( SELECT COUNT(*)
               FROM EMPLOYEE
              WHERE WORKDEPT = D.DEPTNO )
```

Subqueries may be nested to multiple levels. The next example returns employees who earn more than their managers:

```
SELECT EMPNO, LASTNAME
  FROM EMPLOYEE E
 WHERE E.SALARY >
         ( SELECT SALARY
             FROM EMPLOYEE M
            WHERE M.EMPNO =
                    ( SELECT MGRNO
                        FROM DEPARTMENT
                       WHERE DEPTNO = E.WORKDEPT ) )
```

The same question may be answered by using a join in the subselect:

```
SELECT EMPNO, LASTNAME
  FROM EMPLOYEE E
 WHERE E.SALARY >
       ( SELECT M.SALARY
           FROM DEPARTMENT D,
                EMPLOYEE M
          WHERE D.MGRNO = M.EMPNO
            AND D.DEPTNO = E.WORKDEPT )
```

Subselects with EXISTS Keyword

A subselect can be used to test for the existence of a row satisfying some condition. The subselect is connected to the outer level SELECT through the special search condition **EXISTS**. The subselect does not return a value. It only indicates if the subselect search was successful or not.

In the following example we look for departments in the department table which have an invalid manager number assigned, one that does not exist in the employee table.

```
SELECT DEPTNO, DEPTNAME
  FROM DEPARTMENT D
 WHERE NOT EXISTS
             ( SELECT *
                 FROM EMPLOYEE
                WHERE EMPNO = D.MGRNO )
```

The next statement deletes departments which have no manager assigned and have no employees either. This example illustrates the use of a correlated subselect with the EXISTS keyword in a DELETE statement:

```
DELETE
  FROM DEPARTMENT D
 WHERE MGRNO IS NULL
   AND NOT EXISTS
             ( SELECT *
                 FROM EMPLOYEE
                WHERE WORKDEPT = D.DEPTNO )
```

Union

The UNION operator of SQL provides a way of combining two or more SELECT statements into a single SELECT statement. Each of the SELECT's is executed to produce an answer set, then these sets are combined and duplicate rows are eliminated.

The UNION operator is often useful to merge lists of values derived from two or more different tables. The following example merges employees who work in 'D' departments with employees who work on 'AD' projects:

```
  SELECT EMPNO
    FROM EMPLOYEE
   WHERE DEPTNO LIKE 'D%'
UNION
  SELECT DISTINCT EMPNO
    FROM PROJEMPS
   WHERE PROJNO LIKE 'AD%'
ORDER BY 1
```

Rules for connecting SELECT statements through the UNION operator:

- The data types of corresponding items selected by all the SELECT statements must be identical (not just compatible), e.g. SMALLINT is not identical to INTEGER, character data must be of same type and length, and matching columns must obey the same rules regarding nulls.
- The ORDER BY clause must follow all the SELECTS if an order is requested for the union, and the ordering must be expressed in terms of column numbers (not names).
- UNIONs are not allowed in subselects.
- UNIONs are not allowed in the definition of views.

In "Chapter 4. SQL Data Manipulation" we have seen the language to retrieve and update data in relational tables. The purpose of this chapter is to present the language used to define the logical and physical characteristics of tables, associated indexes, and views based upon these tables.

The chapter is structured into two parts. The first part describes the **DB2 Objects** used to store data of relational tables, and in the second part the **Data Definition Language (DDL)** of SQL is presented.

DB2 OBJECTS

The term "DB2 Objects" applies to data bases, storage groups, tables spaces, tables, indexes, and views. This term is just a convenient way of referring to anything you can define or manipulate with SQL. All these objects serve to define important characteristics of a DB2 data base.

This section gives a brief description of each object. This description is merely to give you a conceptual view of how each object fits into the overall DB2 picture. Figure 7 on page 38 shows you how they relate to each other.

Note: Only data base designers and administrators need to know all the DB2 objects. Users of SQL data manipulation only deal with tables and views, and are not concerned with the physical characteristics of these objects.

Tables

All data in DB2 is stored in tables. A table is a collection of rows that have certain columns (fields). When you define a table in DB2, you're defining an ordered set of columns. In the employee table of Figure 3 on page 10 the ordered set of columns includes DEPTNO, DEPTNAME, and MGRNO representing the department number, department name, and the employee number of the manager.

The terms **"fields"** and **"columns"** are used interchangeably. At the intersection of a column and a row we have a **"field value"**. The storage representation of a row is called a **record**.

SQL is used to retrieve or change data in a table. A value is the smallest unit of data that can be retrieved and updated. A row is the smallest unit of data that can be inserted or deleted. SQL can also be used to add a new column to a table.

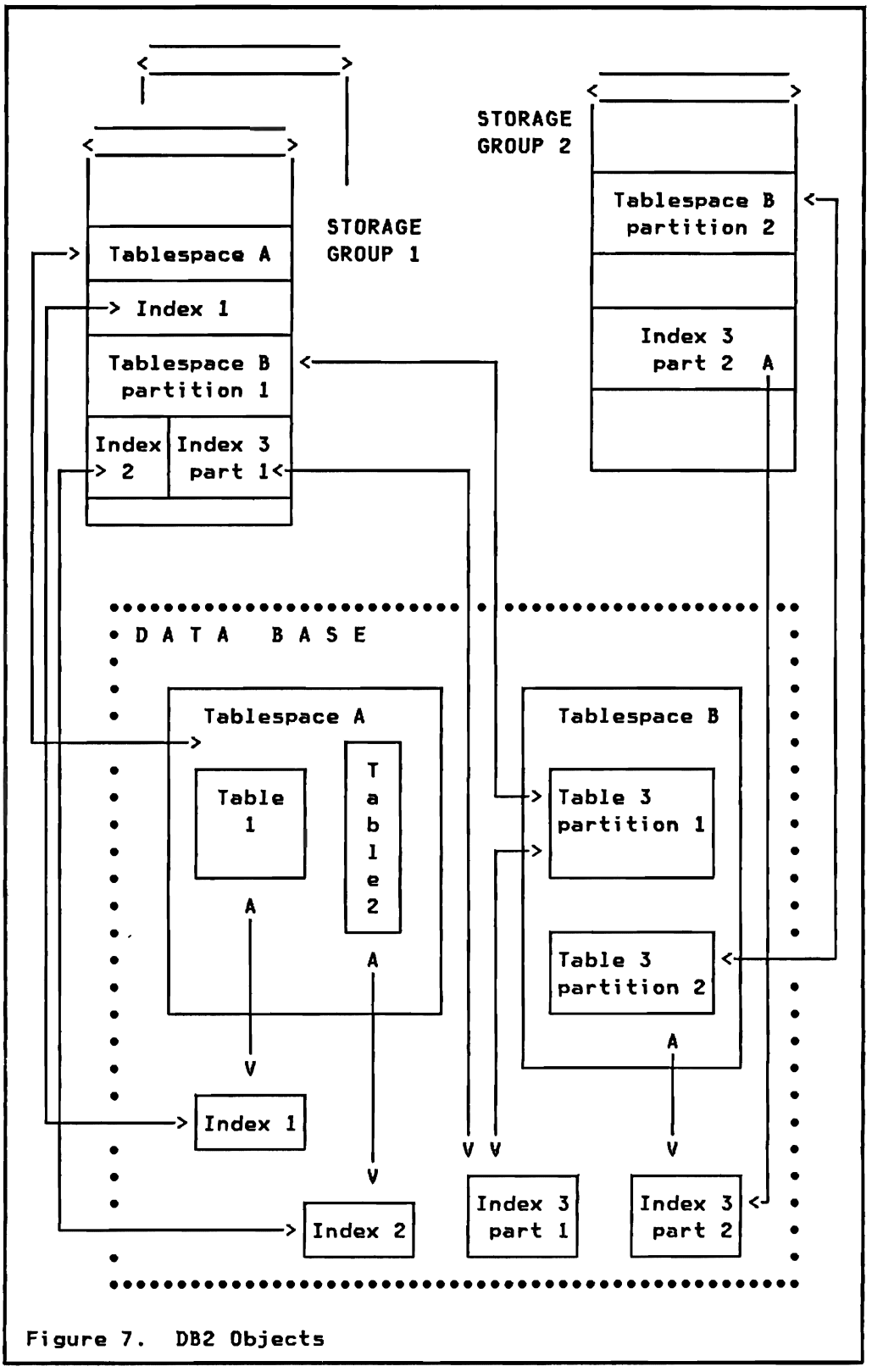


Figure 7. DB2 Objects

Views

A view is an alternate way of representing table data. Views are derived from one or more tables. Views may also be derived from other views, or a combination of views and tables. The tables from which views are derived are referred to as base tables. See Figure 8 for the relationship between views and tables. Views are used in SQL data manipulation statements just like tables.

The definition of a view is stored by DB2. A view, on the other hand, has no separate storage representation. When creating a view you define an alternate representation of the data that is stored in tables.

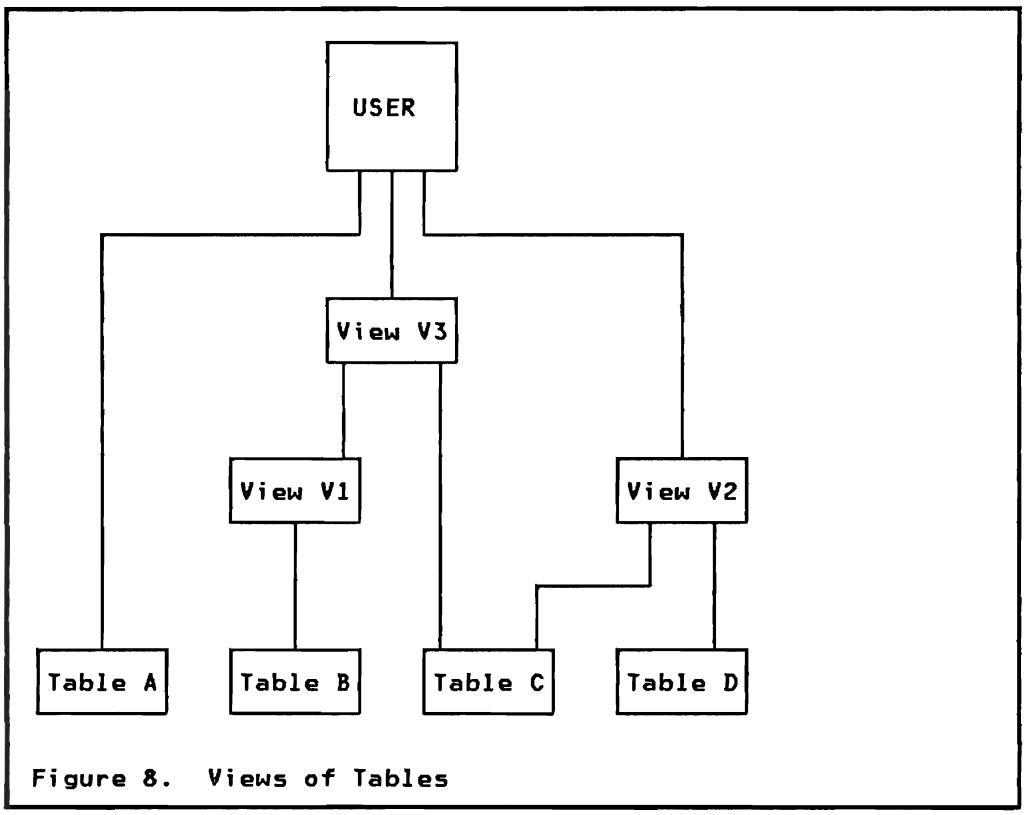


Figure 8. Views of Tables

Table Spaces

You define a table space to hold tables. Each table space is divided into equal-sized units, called **pages**, which contain table data. A table space consists of 1 to 64 VSAM entry-sequenced data sets (ESDS) and can contain up to 64 gigabytes of information. A gigabyte is roughly 1,000,000,000 (or 10**9) bytes.

Table spaces are important operational structures. In DB2 they are the recoverable units, that is you recover table spaces, not data bases. Very large table spaces (up to 64 gigabytes) would

however be impractical for recovery. To make large table spaces practical, DB2 optionally provides **Partitioned Table Spaces.** Tables spaces are divided into partitions on the basis of ranges of data values.

Simple Table Spaces

Table spaces which are not partitioned are referred to as "simple" table spaces. A simple table space can hold one or more tables. Following are some considerations about when to have just one table in a simple table space and when to have multiple tables:

- The time to scan a table will probably be longer when tables are intermixed in the table space.
- If there are many small tables you may gain a performance advantage and save storage space if they are together in one table space.
- If two tables are related through the same or similar primary key, and are mainly used together, it might be an advantage to have them in one table space.
- You might want to have one table space per user department.
- When certain DB2 utilities are executing, the entire table space is unavailable.
- If an application requires exclusive use of a table by issuing a LOCK TABLE statement, the entire table space will be locked, preventing other tables from being accessed. This is due to the fact that data pages might contain data from multiple tables. The LOCK statement is presented in "Chapter 6. Data Management" section "Table Space Locks" on page 67.

Partitioned Table Spaces

A partitioned table space holds exactly one table. The table space is divided into partitions, and each partition is stored in one VSAM ESDS. The partitions are defined as ranges of an index based on one or multiple columns.

The principal benefits of partitioned table spaces are:

- Very large tables can be split into manageable units.
- Partitions are independent from each other. They can be reorganized and recovered individually (but the entire table space is locked during that time).
- They can be assigned to different storage groups each of which may be assigned to a different device type. This allows

installations to store active data on faster devices than history data.

Table Pages

Tables spaces are physically divided into storage units called pages. Each page holds one or more rows of a table (or multiple tables for simple table spaces). Two page sizes are available, 4K and 32K. The larger page size must be chosen if rows with a length of more than 4K bytes exist in the table.

The size of the pages is selected through the assignment of a **Bufferpool** to the table space. Bufferpools are presented later in this chapter. The VSAM control interval size of data sets holding DB2 data pages is equal to 4K bytes.

Indexes

You use indexes to get to data in different ways. Suppose you have a phone number and need to find the corresponding name. Since the regular phone book is ordered by name, you would have to scan the phone numbers in sequence to find the given number. If you could reorder the entries according to phone numbers your task would be easy.

Indexes in DB2 logically reorder data so that DB2 can get to data without a lengthy sequential search. Indexes are based on one or more columns of a table. **Programs or users accessing DB2 data never explicitly specify indexes.** Indexes are only used by DB2.

There are four reasons for having indexes:

1. To ensure unique values in a column, or in a combination of multiple columns. Unique indexes can be used to force uniqueness of the rows of a table. To enforce that no duplicate employee numbers are put into the employee table, you define a unique index based on employee number.

2. To improve performance. In most cases when there is an index on a column, access to data is faster than without an index. You should define an index on columns which are often used in search conditions or joins.

3. To cluster data of a table in a certain order. See "Clustering Index" below.

4. A partitioned table must have an index which defines the partition ranges.

You can have multiple indexes for a table. Each index is based on the values of data in one or more columns of the table, with a

total length not exceeding 255 bytes. An index may be defined ascending (default) or descending on each column involved.

An index is an object completely separate from the data in your table. It is a structure that DB2 builds for you in response to your request to create an index. DB2 automatically maintains the index after creation.

Even though you may have many indexes to a table, you cannot assume that good performance is just dependent on the number of indexes. If a table is constantly having rows inserted, deleted, or updated, its indexes must always be updated. Thus too many indexes may degrade performance. Remember also that each index requires external storage.

An index on a table can be created any time after the table is created. Except for changes in performance users of the table will be unaware of the existence of the index. It is only DB2s decision whether or not to use an index to access the table.

Clustering Index

DB2 indexes can also cluster data. A clustering index is an index which causes the rows of the table to be stored in a physical order that approximates the order of the indexed column(s). You use a clustering index to store data the way it will be most frequently accessed, that is, by employee number, account number, or alphabetically by name.

Since a clustering index defines the way data is stored, you can define **only one clustering index per table**. If no index has the clustering attribute, then DB2 uses the first index defined as a clustering index. In a table space with only one table, the data is in exact clustering sequence after a reorganization.

It is mandatory to have a clustering index for a partitioned table (space). The table is divided into partitions by ranges of this clustering index. The index itself is physically partitioned too.

Index Spaces and Pages

An index space is automatically allocated when an index is created. Index data is stored in index pages within the index space. The physical page size on external storage is always 4K. A physical page can consist of between 1 and 16 subpages. A subpage is the unit of index locking.

Data Bases

A DB2 data base names a collection of tables and indexes, and the spaces that contain them. The DB2 data base is not the same as a DL/I data base. It serves as an operational object for start/stop (availability of data) and for authorization (accessibility of data).

A DB2 data base is therefore a convenient way of referring to a set of tables and its associated indexes. It is recommended that a data base contains related tables - related by application, department, etc.

DB2 provides defaults for both data base and table space membership. These defaults permit authorized users to define tables without explicitly specifying a data base or table space. The default data base (named DSNDB04) is setup at installation time and may be tailored to the installation needs.

Applications programmers need not be concerned with DB2s data base structure. They only deal with tables and views in their programs.

Storage Groups

A storage group is a set of DASD volumes to which you assign a name. Storage groups allow you to delegate to DB2 some of the responsibility for managing the external storage used for DB2 data bases. When external DASD storage space is required for a table space, DB2 allocates the space from the volumes previously specified for the associated storage group.

Rules for storage groups:

- All the volumes in a storage group must be of the same device type.
- Storage groups may overlap. The same volume may belong to more than one storage group.
- Volumes can have standard OS or VSAM data sets allocated on them.

Storage groups simplify the allocation of data sets. By letting DB2 handle your storage, you are relieved of the burden of defining and deleting data sets yourself.

You have the option to manage external storage yourself. This is necessary when there is a need to control data set placement within a volume.

Rules for user controlled space allocation:

- Define required VSAM data sets
- Follow DB2 data set naming convention
- Add secondary data sets when table expands

DB2 provides a default storage group (named SYSDEFLT) at installation time. Table Spaces will be allocated within this group if the user does not explicitly request otherwise.

Bufferpools

Bufferpools are areas of virtual storage which DB2 uses during the execution of an application program or an interactive SQL request to temporarily store pages of table spaces. When an application program needs access to a row of a table, the page containing that row is brought into a buffer. If data is changed, that buffer must be written back to the table space. If the data the application requires is already in the buffer, it can be used immediately.

DB2 has four different bufferpools to choose from. They are named BP0, BP1, BP2, and BP32K. Your choice of bufferpool when defining a table space implicitly determines the page size of the table space. If you choose bufferpool BP0, BP1, or BP2, the page size will be 4K bytes. Selecting bufferpool BP32K gives a page size of 32K bytes. The only time you should consider using bufferpool BP32K is when the total length of the rows in your table exceeds 4K bytes.

Bufferpools of 4K buffers are assigned to table spaces depending on application requirements. Critical application tables might be assigned to a bufferpool of their own.

The actual number of buffers in a bufferpool is dynamically assigned by DB2, between a minimum and a maximum number defined in the DB2 installation module DSNZPARM (see "Chapter 16. Installation and Servicing" on page 181). The installation default bufferpool size for bufferpool BP0 is a minimum of 56 pages, and a maximum of 112 pages. The other bufferpools are by default inactive.

The lower limit of buffers is 14 for a 4K buffer pool, and 3 for BP32K. The upper limits are 1029 for a 4K buffer pool, and 133 for BP32K.

Synonyms

Synonyms are alternate names for tables or views. Each user may individually assign a synonym to any table or view. By doing this a user can refer to a table through a short identification.

A synonym is only effective for the user who created it. If many users want to use the same synonym, they must each define that synonym.

DATA DEFINITION LANGUAGE

The Data Definition statements of SQL provide facilities for creating, changing, and deleting all the DB2 objects. One of the unique advantages of SQL is that these Data Definition statements may be used in a normal user session and mixed together with other types of statements, such as data manipulation statements. It is not necessary to stop DB2 or to invoke special utility programs to create new tables in one of the data bases. Thus a user may create a table for storing and manipulating some temporary result, and drop the table when it is no longer needed.

Data Definition statements automatically update the **DB2 Catalog**, where the definitions of all DB2 objects are maintained. See "Chapter 10. The DB2 Catalog" for further information.

Naming Conventions

DB2 uses naming conventions for its objects. Within the explanation given below we use the following terms:

- A "short-id" is up to 8 characters, 1st alphabetic (A-Z,@,#,$), rest alphabetic or numeric.
- A "long-id" is up to 18 characters, 1st alphabetic, rest alphabetic, "_", or numeric.

Following are the DB2 naming conventions:

Storage Group short-id

Data Base short-id

Table Space short-id

The full name for table spaces however is formed by qualification with the DB2 data base name [db-name.space-name], and allows for table spaces with the same name in different DB2 data bases.

Bufferpool BP0, BP1, BP2, BP32K

Table	long-id The full name for tables is formed by qualification with the authorization-id [auth-id.tablename], and allows for multiple tables with the same name for different users. The authorization-id (user-id) of the creator of a table is automatically added by DB2.
View	long-id, same rule as for table
Index	long-id, same rule as for table
Synonym	long-id
Columns	long-id, unique within a table

CREATE Statement

SQL CREATE statements are used to define DB2 objects. The general format of a CREATE statement is:

```
CREATE object name
       parameters...
```

For each of the different objects of DB2 which can be defined a different format of the CREATE statements applies. The following subsections present the important aspects of the different CREATE statements but not all possibilities and options will be explained.

DB2 provides defaults for most of the parameters of the CREATE statement. In the given examples the default is always the first option listed for a parameter.

Defining a Storage Group

The CREATE STOGROUP statement defines a set of volumes, on which DB2 may allocate space for table spaces and indexes.

```
CREATE STOGROUP stogroup-name
       VOLUMES  (volser,volser,...)
       VCAT     vsam-catalog-name
       PASSWORD vsam-cat-control-password
```

Defining a DB2 Data Base

You use the CREATE DATABASE statement to define a data base.

```
CREATE DATABASE   db-name
       STOGROUP   stogroup-name
       BUFFERPOOL BPx
```

The optional STOGROUP and BUFFERPOOL parameters are used to establish default values to be used in subsequent create table space statements. If a create table space does not explicitly name a storage group or bufferpool then the values from its data base will be used.

Defining a Table Space

Table Spaces are the physical spaces that hold tables. The CREATE TABLESPACE statement is used to predefine space on DASD for subsequent table definition.

```
CREATE TABLESPACE space-name
       IN         db-name
       USING      STOGROUP stogroup-name
                  PRIQTY k-bytes SECQTY k-bytes
                  ERASE NO | YES
       BUFFERPOOL BPx
       CLOSE      YES | NO
       LOCKSIZE   ANY | PAGE | TABLESPACE
```

IN: Names the DB2 data base the table space belongs to. The data base name defaults to the DB2 default data base DSNDB04 if not given.

USING: Mainly used to define the primary and secondary space allocation (in kilo-bytes) on volumes of a named storage group. The default and minimum sizes are 3 pages (12 or 96 KB) each if omitted, the maximum sizes are 131068. The maximum data set size is 2 gigabytes (GB), and you may have up to 32 data sets per table space. The maximum table space size therefore is 64 GB.

ERASE: Allows the user to have the data sets of the table space physically erased when the table space is dropped (through the DROP statement).

BUFFERPOOL: Assigning a specific bufferpool to override the bufferpool specification which applies to the data base.

CLOSE: This options allows the user to specify if VSAM data sets should remain open or be closed at the end of an application usage. Closing the data sets allows the table space to be taken off-line and also frees resources within DB2.

LOCKSIZE: Specifies the level of locking for the table space. If your design involves sharing of tables among several application programs, you need to be aware of the options available to you to control the degree of concurrency.

DB2 uses **locks** to minimize the interference between concurrent users. Locks are a mechanism which DB2 uses to prevent concurrent users from accessing inconsistent data. Locks apply to table spaces or pages. As long as a page or table space is locked, other application programs that need to use the data in that page or table space must wait until the lock is released. Notice that DB2 does not lock a specific table. If a table space contains more than one table, then all tables are locked when the table space is locked.

If an application merely needs to access (not change) the data, the lock is known as a **shared** lock. A shared lock allows other applications to access the table space or page, but not to change it. If an application needs to change the data, the lock is known as an **exclusive** lock. An exclusive lock prohibits all access to the table space or page until the lock is released.

You can choose one of three different locking options when you define a table space:

TABLESPACE The entire table space is locked when an application program is using it. Good for the performance of one application, but others will be locked out.

PAGE Only pages that contain referenced data are locked. This allows more concurrent usage than table space locking, but at a higher resource utilization.

ANY Let DB2 choose the most appropriate mode of locking. This is the default and normally the most effective option. DB2 decides which locksize to apply when analyzing the SQL statements of an application program.

Defining a Partitioned Table Space

Partitioned Table Spaces contain multiple VSAM ESDS's. The partitions are defined through additional parameters of the CREATE table space statement.

```
CREATE TABLESPACE space-name
       IN          db-name
       USING       STOGROUP stogroup-name
                   PRIQTY kb SECQTY kb
                   ERASE NO|YES
       NUMPARTS    number-of-partitions
         ( PART n USING STOGROUP stogroup-name
                        PRIQTY kb   SECQTY kb,
           PART m USING STOGROUP stogroup-name
                        PRIQTY kb   SECQTY kb )
```

NUMPARTS: Defines the number of partitions (2 to 64). The actual dividing into partitions by keyrange is done later by defining the clustering index belonging to the table stored in this table space.

The number of partitions implicitly defines the maximum size of each partition. This maximum size is 4 gigabytes (per partition) for 1 to 16 partitions, 2 GB for 17 to 32 partitions, and 1 GB for 33 to 64 partitions. The total maximum number of bytes stored in a partitioned table space is 64 GB, the same as for a simple table space.

PART: This subclause defines these partitions which use a different storage group, or have a different size (PRIQTY, SECQTY) than the defaults defined in the USING clause of the CREATE TABLESPACE statement. The list may be omitted if all partitions are stored in the same storage group and use the same primary and secondary allocations.

Defining a Table

The CREATE TABLE statement is used to define a new table to DB2.

```
CREATE TABLE     table-name
                 ( column-name1  data-type  [NOT NULL],
                   column-name2  data-type  [NOT NULL],
                   column-...    )
       IN        db-name.table-space-name
       EDITPROC  edit-routine
       VALIDPROC validation-routine
```

Tablename: Is either a user name of up to 18 characters, or a qualified name in the format auth-id.tablename. An unqualified user name will be prefixed by DB2 with the "auth-id" of the issuing user.

Columns: After the tablename we define all the columns of the table, with their data types, and optionally followed by the specification NOT NULL. Data types were presented in "Chapter 3. Relational Data Model" in section "Data Types" on page 8. The specification of NOT NULL prohibits fields without a value.

IN: Defines the DB2 data base and the table space the table is assigned to. If no table space name is given (clause omitted or entered as IN DATABASE db-name), then DB2 will define and allocate a default table space for the table.

EDITPROC: Defines an editing routine to be invoked just after a record (corresponding to a table row) is retrieved, and just before a record is stored. Editing routines allow for data compression/decompression and encrypting.

VALIDPROC: Defines a validation routine to be invoked just before a record is stored in the table space. Validation routines allow user checking of data entered into tables.

Some rules for edit- and validation routines:

- DB2 passes the row and a description of the row with all the column names and data types to edit- and validation routines.
- Routines set a return code to signal DB2 if the row may be inserted (or changed) into the table space.
- Fields used for indexes are extracted by DB2 before passing the row to the edit routine for encoding. Index keys are therefore always stored in external data format.
- Routines may not invoke SVC or DB2 services.

The CREATE statement below is used to define the sample employee table used in the previous chapters. See Figure 2 on page 7 for reference.

```
CREATE TABLE EMPLOYEE
             ( EMPNO     CHAR(6)      NOT NULL,
               LASTNAME  VARCHAR(15)  NOT NULL,
               WORKDEPT  CHAR(3)      NOT NULL,
               PHONENO   CHAR(4),
               JOBCODE   DECIMAL(3),
               EDUCLVL   SMALLINT,
               SALARY    DECIMAL(8,2)  )
       IN    DSN8DAPP.DSN8SEMP
```

Defining an Index

Indexes are defined to force uniqueness within a table and to improve performance. Multiple indexes may be defined per table. An index may be defined on a single column or multiple columns.

The CREATE INDEX statement has the following format:

```
CREATE [UNIQUE]
       INDEX      index-name
       ON         table-name
                  ( column-name1 [DESC],
                    column-name2  ...    )
       USING      STOGROUP stogroup-name
                  PRIQTY kb  SECQTY kb
                  ERASE NO | YES
       BUFFERPOOL BPx
       CLOSE      YES | NO
       SUBPAGES   1 | 2 | 4 | 8 | 16
       [CLUSTER]
```

UNIQUE: If you want your table to contain only unique rows with respect to the values of a column or multiple columns, you have to specify UNIQUE. For example you want the employee numbers of the employee table to be unique, whereas the last names tend not to be unique.

Index name and Table name: Both follow the explanation given for table names in section CREATE TABLE.

USING, BUFFERPOOL, CLOSE, and ERASE: Same explanation as for CREATE TABLESPACE.

ON: Defines the column or columns on which the index is based. By default the ordering is ascending (ASC), but individual columns may be ordered in descending order (DESC).

SUBPAGES: Each physical page can be defined as consisting of between 1 and 16 subpages. A subpage is the unit of index locking. Physically, however, index pages are always stored as 4K blocks. Physical blocks are the units for data transfer.

CLUSTER: This optional specification defines the index as a clustering index. DB2 will try to store the data rows of the base table in sequence of the clustering index for faster sequential access. A table may have only one clustering index. The first index defined will be used as a clustering index if CLUSTER is not specified.

A CREATE INDEX statement automatically allocates an index space, scans the table, and builds the actual index. Indexes should normally be defined before a table is loaded with data.

The following statement is used to define a unique index on employee numbers for the employee table:

```
CREATE UNIQUE INDEX XEMPNO
       ON EMPLOYEE (EMPNO)
```

Defining a Partitioning Index

A partitioned table (space) must have a clustering index which defines the keyranges to be used to divide the table into partitions.

The CLUSTER clause is extended for the specification of the keyrange of each partition:

```
CREATE [UNIQUE]
       INDEX       index-name
       ON          table-name
                   ( column-specification )
       CLUSTER
         ( PART 1  VALUES (limit-key1-col1, .. colm ),
           PART 2  VALUES (limit-key2-col1, .. colm ),
           ..
           PART n  VALUES (limit-keyn-col1, .. colm ) )
```

The number of partitions must match the NUMPARTS specification of the associated table space. For each partition the limit-keys of all the columns involved are specified. These are the highest key values (for descending columns the lowest key values) to be stored in that partition of the table (space).

Defining a View

A view is nothing more than a "virtual" table which is derived from one or more base tables. The definition of a view is stored in the form of a SQL SELECT statement. The view appears to be a table which can be queried and in certain cases updated in the same way as a real table.

The CREATE VIEW statement is used to store the definition of the view:

```
CREATE VIEW view-name
            (column-name, ...)
       AS   SELECT ...
       [WITH CHECK OPTION]
```

Viewname: Is handled exactly like a table name. The viewname is prefixed with an authorization-id either explicitly by the user, or automatically by DB2.

Columns: Optional, used to substitute one or more names for columns in the view. If no column names are given, the columns of the view will have the same names as the columns of the underlying base table(s). If the view is a join of two tables which include identical column names then new column names must be assigned.

AS SELECT ...: Substitute a SELECT statement that defines the view. This SELECT statement may include a WHERE clause with a search condition, a GROUP BY clause, a HAVING clause, but no ORDER BY clause. The SELECT statement may be based on multiple tables and/or other views.

WITH CHECK OPTION: Indicates that all updates and inserts using the view are to be checked against the view definition. The checking involved was presented in "Chapter 4. SQL Data Manipulation" section "Using Views" on page 29.

Note: Remember that only simple views which are row and/or column subsets of one base table may be updated.

Examples of Views

- Column subset

```
CREATE VIEW EMPNAME
       AS   SELECT EMPNO, LASTNAME
              FROM EMPLOYEE
```

  A SELECT without a WHERE clause retrieves all the rows of a table.

- Row subset

```
CREATE VIEW DEPTD11
       AS   SELECT *
              FROM EMPLOYEE
             WHERE WORKDEPT=D11
```

  If the view definition has a SELECT * clause, the view will have as many columns as the underlying table. New columns added later to the table will not appear in this view.

- Statistical data

```
CREATE VIEW AVGSAL (DEPT, AVGSAL)
       AS   SELECT WORKDEPT, AVG(SALARY)
              FROM EMPLOYEE
             GROUP BY WORKDEPT
```

  This view includes one row per department, with two columns containing the department number and the average salary.

Defining a Synonym

Synonyms are alternate names for tables or views. The format of the CREATE SYNONYM statement is:

```
CREATE SYNONYM synonym
       FOR     authorization-id.table-name
```

A common use of synonyms is when a group of users all wish to share a table. One user, JONES, creates a table called TEMPL. Each other user can refer to that table using the fully qualified name JONES.TEMPL, or each user defines a synonym for the table as shown below, and refers to it as EMPLOYEE.

```
CREATE SYNONYM EMPLOYEE
       FOR     JONES.TEMPL
```

ALTER Statement

SQL ALTER statements are used to modify attributes of the physical design. Not all changes of the physical design may be implemented through an ALTER statement. Some changes may only be executed by deleting the current definitions (SQL DROP Statement) and redefining the objects.

Changing a Storage Group

DASD volumes may be added to or removed from a storage group by an ALTER STOGROUP statement:

```
ALTER STOGROUP stogroup-name
      ADD      VOLUMES (volser,...)
      REMOVE   VOLUMES (volser,...)
```

Data (table spaces) which already exists on volumes being removed from a storage group is not affected by the ALTER statement. However, a new table space will no longer be allocated to a "removed" volume.

Changing a Table Space

The ALTER TABLESPACE statement is used to change the processing options associated with a table space:

```
ALTER TABLESPACE db-name.space-name
      BUFFERPOOL BPx
      LOCKSIZE   ANY | PAGE | TABLESPACE
      CLOSE      YES | NO
```

Changing a Table

The ALTER TABLE statement is used to add one new column to an existing table, or to change the name of the validation routine associated with the table.

```
ALTER TABLE     table-name
      ADD       column-name  data-type
      VALIDPROC validation-routine | NULL
```

When a column is added to a table, all values of this column are implicitly set to the null value. A new column cannot be defined with the NOT NULL option. Non null values are stored in existing rows of the table through SQL UPDATE statements.

Other modifications to a table like deleting a column, changing the data type of a column, or changing the edit-routine, are not allowed through an ALTER statement. For these changes the table must be deleted (DROP statement) and recreated. This is presented

later in this chapter in section "Modifying the Design" on page 56.

Changing an Index

Minor changes to the definition of an index can be applied with the ALTER INDEX statement:

```
ALTER INDEX      index-name
      BUFFERPOOL BPx
      CLOSE      YES | NO
```

DROP Statement

The SQL DROP statement is used to remove permanently from DB2 the specified object, and all other objects which are directly or indirectly dependent on the one being dropped.

DROP STOGROUP: DB2 does not allow a DROP STOGROUP as long as there is a table space associated with it. You must first drop all the table spaces (and probably save the table data before that), before you are allowed to drop the storage group.

DROP DATABASE: Removes all the data (table spaces, tables, indexes) associated with the data base.

DROP TABLESPACE: Removes all the tables of the table space, and all the indexes and views based on these tables.

DROP TABLE: The specified table is deleted from the table space, and all indexes and views based on the table are removed too. The contents of the table are lost. If the associated table space was created implicitly together with the table, it is dropped too. The physical storage of the table is reclaimed when the table space is reorganized.

DROP INDEX: Removes the definition and the data of an index. For applications which were using the index DB2 will automatically re-evaluate the data needs at their next usage and find alternate paths to access the required data.

DROP VIEW: Removes the definition of the specified view, and of all other views based on this view, from the system. The underlying table(s) on which the view is defined are not affected. Applications using the view are invalidated and may not be executed any more.

To enable these applications again you have to:

- change the SQL statements in the application to use base tables or other views, or
- redefine views with the same name, or
- assign synonyms with the deleted views names to other views, and
- have the application data access paths re-evaluated (this process is called BIND and is explained in "Chapter 7. Application Programming Environment" section "Creating an Application Plan (BIND)" on page 95).

DROP SYNONYM: Removes an alternate table or view name from the system. Applications using this name are affected in the same way as when views are dropped.

COMMENT ON Statement

To document the design of tables SQL provides the COMMENT ON statement. This allows the user to associate one line of text with a table, and one line of text with each column of a table.

```
COMMENT ON TABLE  table-name
           IS     'text-string'

COMMENT ON COLUMN table-name.column-name
           IS     'text-string'
```

The text may be up to 254 characters. It is stored in the DB2 Catalog and may be queried by SQL SELECT statements on catalog tables. The DB2 Catalog is presented in "Chapter 10. The DB2 Catalog" on page 121.

MODIFYING THE DESIGN

Modifying a data base is an extension of data base design and implementation. The primary means of changing a DB2 data base is either by using the ALTER statement or by dropping an object and re-creating it with different specifications.

As we have seen in a previous section, the possibilities of the ALTER statement are limited to the changing of processing options (locking, bufferpools, etc) and adding new columns to a table. Other changes can only be implemented by deleting (dropping) existing definitions, and creating new ones.

Modifying Tables

Modifying a table does not mean changing the data within the table. It means to change the original specifications used when creating the table. The scope of the ALTER statement is limited to adding new columns, and to change validation routines. See "Changing a Table" on page 54.

The following definitional changes to a table must be performed by dropping the table and recreating it:

- Changing the data type of a column
- Changing the precision/length of a column
- Changing if nulls are allowed or not allowed
- Changing the edit-routine specification
- Removing a column

Table Re-Creation Procedures

As presented in "DROP TABLE" on page 55, dropping a table deletes the table data and removes all the indexes and views based on the table. Therefore the following steps are necessary when changing the definition of a table:

1. Define a new table with all the changes using a different name. Column names may be the same or different, but data types must be compatible.
2. Define all the necessary indexes on the new table.
3. Copy the old data into the new table:

```
INSERT INTO   new-table
       SELECT *     or     SELECT columns
         FROM old-table
```

4. Drop the old table (this deletes all the views too).
5. Restore the original table name as a view of the new table:

```
CREATE VIEW old-table
       AS   ( SELECT *
                FROM new-table )
```

6. Define views corresponding to those of the old table. They can be defined based on the new table, or based on the view with the old table name.
7. Recreate the authorization for the table and views.

Modifying a View

In many cases, changing user requirements could be satisfied by modifying an existing view. For example, the user wants to change the names of columns.

You cannot change a view directly, there is no ALTER VIEW statement. You have to drop the view and redefine it. This will invalidate applications using the view and DB2 will dynamically re-evaluate the access paths of these applications when they are invoked the next time.

Modifying the Definition of Table Spaces

Some of the processing options for table spaces may be changed by the ALTER TABLESPACE statement presented earlier in this chapter.

Any other change to the table space must be done by dropping and redefining it. The steps involved are similar to those involved in dropping and redefining a table.

Modifying and Adding Indexes

Only the CLOSE and BUFFERPOOL option of an index may be changed by the ALTER INDEX statement. Any other change to the definition of an index must be done by dropping the index and recreating it.

All applications using a table should have their access paths re-evaluated when an index is dropped or added. This is done automatically at the next invocation of the application when an index has been dropped. However, existing applications are not re-evaluated when new indexes are added. It is the users or data base administrators responsibility to have these applications re-evaluated (by using the REBIND command).

In this chapter we look how DB2 manages the physical data of relational tables. The physical data management is presented in the first section, then we look at the management of concurrent access (locking), and finally how data is loaded into DB2 data bases.

DATA SPACE MANAGEMENT

In "Chapter 5. Data Definition" we have seen the logical objects DB2 uses to manage table data. Now we look more into the physical organization.

All the data in a DB2 data base is stored in VSAM entry-sequenced data sets (ESDS). You can define and maintain these VSAM data sets yourself or let DB2 do it. If you wish to do it yourself, you will need to use the VSAM Access Method Services to define and delete the data sets. If you let DB2 take care of your external storage, you will need to create one or more **storage groups**. A storage group is a DB2 object that is a set of DASD volumes of a single device type, to which a name is assigned.

Letting DB2 manage the Data Sets

Having defined one or more storage groups as presented in the previous chapter in section "Defining a Storage Group" on page 46, DB2 will handle your external storage satisfactorily for most cases. Here is what DB2 does for you when you use storage groups:

- Defines the necessary VSAM data sets using VSAM Access Method Services
- The data sets are automatically deleted when a table space is dropped
- A new data set is automatically allocated when the limit number of bytes of a data set is reached for a simple table space

You should make use of storage groups whenever you can. Letting DB2 manage your external storage will make data management easier.

VSAM Datasets

Data of relational tables is stored in VSAM ESDSs. A data space used by DB2 to store data can be:

- A table space with one or more tables
- An index space containing one index

DB2 uses VSAM for DASD space management and data set cataloging. However, after the data sets are created, they are formatted and used by DB2 and are not processable by VSAM services.

Simple Table Spaces

Simple table spaces contain one or more tables. The size of a simple table space is limited to 32 VSAM ESDSs of up to 2 GB (gigabytes) each, thus allowing for a total of 64 GB.

How DB2 manages a simple table space:

- Initially DB2 allocates one data set of the size given in the PRIQTY parameter of the CREATE TABLESPACE statement.
- Secondary extents are allocated as necessary up to the VSAM limit of 123 extents, or until the limit size of 2 GB is reached. Each secondary extent is of the size given in the SECQTY parameter (CREATE statement).
- If, and only if, the 2 GB limit size of one data set is reached, DB2 allocates another data set up to the maximum of 32 data sets.

Partitioned Table Spaces

Partitioned table spaces contain a single partitioned table. They are divided into up to 64 partitions with a total size of up to 64 GB. Each partition is one VSAM ESDS, and all the partitions have the same maximum size. The relationship between number of partitions and the maximum partition size is illustrated in Figure 9 on page 61.

Number of partitions	Maximum partition size
1 - 16	4 GB (gigabytes)
17 - 32	2 GB
33 - 64	1 GB

Figure 9. Number and Size of Partitions

How DB2 manages a partitioned table space:

- Initially DB2 allocates a data set of the size given in the PRIQTY parameter of the CREATE TABLESPACE statement.
- Secondary extents are allocated as necessary up to the VSAM limit of 123 extents, or until the limit size of the partition is reached. Each secondary extent is of the size given in the SECQTY parameter (CREATE statement).

User defined Datasets

You may elect to control the external storage of table spaces yourself by defining and maintaining data sets using VSAM Access Method Services. Some reasons why you want to do that are:

- Depending on your requirements you may wish to allocate data sets on specific cylinders of a volume.
- The default staging parameters DB2 uses for MSS data sets (STAGE, NODESTAGEWAIT) may not satisfy your requirements.
- DB2 does not use TO and FOR parameters when defining a data set. This means that clusters may be deleted any time. If you wish to prevent deletion of a data set before a specific date or before a certain number of days have elapsed, then you must create and manage your own VSAM data sets.

Managing DB2 external storage yourself involves a number of steps:

1. Define the required data sets before you issue a CREATE TABLESPACE or CREATE INDEX statement. You need a data set for each simple table space, each non-partitioned index, each partition of a partitioned table space, and each partition of the index of a partitioned table space.

- Data set naming conventions must be strictly followed:

  ```
  vsamcat.DSNDBx.dbname.spacename.I0001.Annn

  vsamcat    = VSAM catalog name
  DSNDBC     = for cluster
  DSNDBD     = for data component
  dbname     = name of DB2 data base
  spacename  = name of tablespace or indexspace
  I0001      = constant
  Annn       = A001 for first data set or partition
             = A002 for second ...
  ```

- The data set must be defined as NONINDEXED, with a RECORDSIZE of 4089, a CONTROLINTERVALSIZE of 4096, and SHAREOPTIONS(3,3).

- Some options may be specified but will be ignored by DB2 during processing: SPANNED, REUSE, SPEED, BUFFERSPACE, WRITECHECK, EXCEPTIONEXIT.

- Example:

  ```
  DEFINE CLUSTER -
   (NAME(CAT1.DSNDBC.PERSONNEL.EMPLOYEE.I0001.A001)  -
    CYLINDERS(100 10)  VOLUMES(WORK01)  NONINDEXED   -
    CONTROLINTERVALSIZE(4096)  RECORDSIZE(4089)      -
    UNIQUE  MASTERPW(ZUERICH)  ERASE                 -
    SHAREOPTIONS(3 3) )                              -
  DATA                                               -
   (NAME(CAT1.DNSDBD.PERSONNEL.EMPLOYEE.I0001.A001)) -
  CATALOG(CAT1/BARBARA)
  ```

2. Define additional data sets for expanding tables. Increment the data set number at the end of the data set name. DB2 issues a warning message when the size of the data set approaches 2 GB.

Data Page Management

The rows of the relational tables are stored in data pages. A data page is either 4K or 32K bytes, and is stored as 1 or 8 VSAM control intervals of 4K bytes.

The rows stored in a page are also called records. A row is always fully contained within a page, no row (or record) will span across two pages.

Records within a page are addressed by a 4-byte **Record-ID (RID)** as illustrated in Figure 10 on page 63.

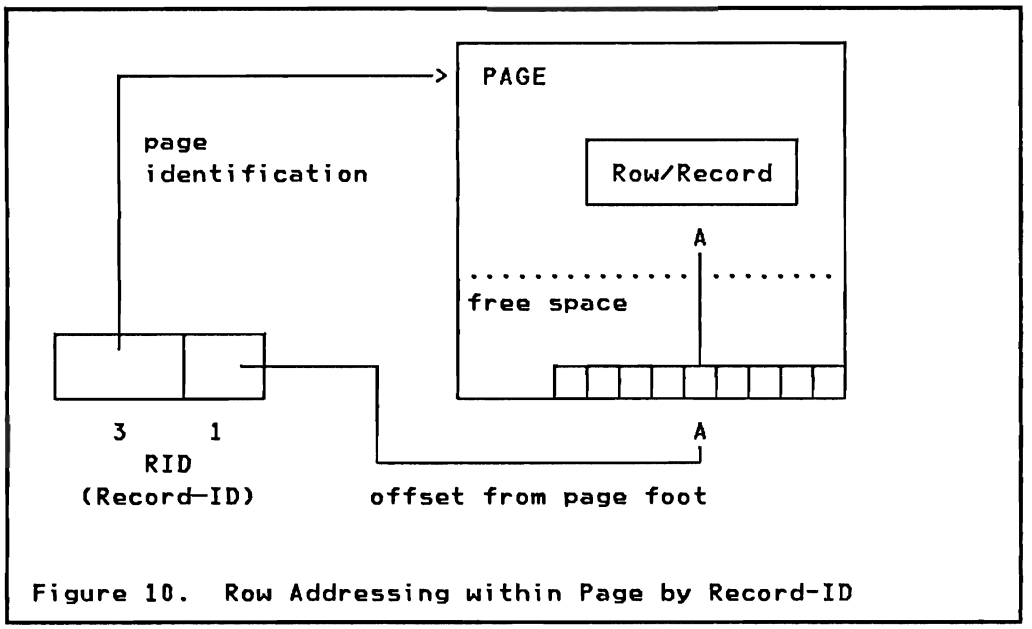


Figure 10. Row Addressing within Page by Record-ID

This record addressing schema combines the speed of direct addressing with the flexibility of indirection. Records may be moved within a page without any restriction.

Records are added to pages in load sequence. For a table space with one single table the rows may be physically stored in the sequence of a clustering index through the reorganization utility.

Row Format

Rows or records are stored in a data page as a contiguous string of characters. Rows may be encoded through a user written edit routine defined with the CREATE TABLE statement (see "Defining a Table" on page 49).

Non-encoded rows are stored in one of two basic formats:

1. Fixed length rows (of tables without varying length character fields) are just byte strings. All fields in the row are fixed length and byte aligned.

   Note: When a new column is added to a table with fixed length rows, the rows will be treated as variable length rows, even if the column is of fixed length.

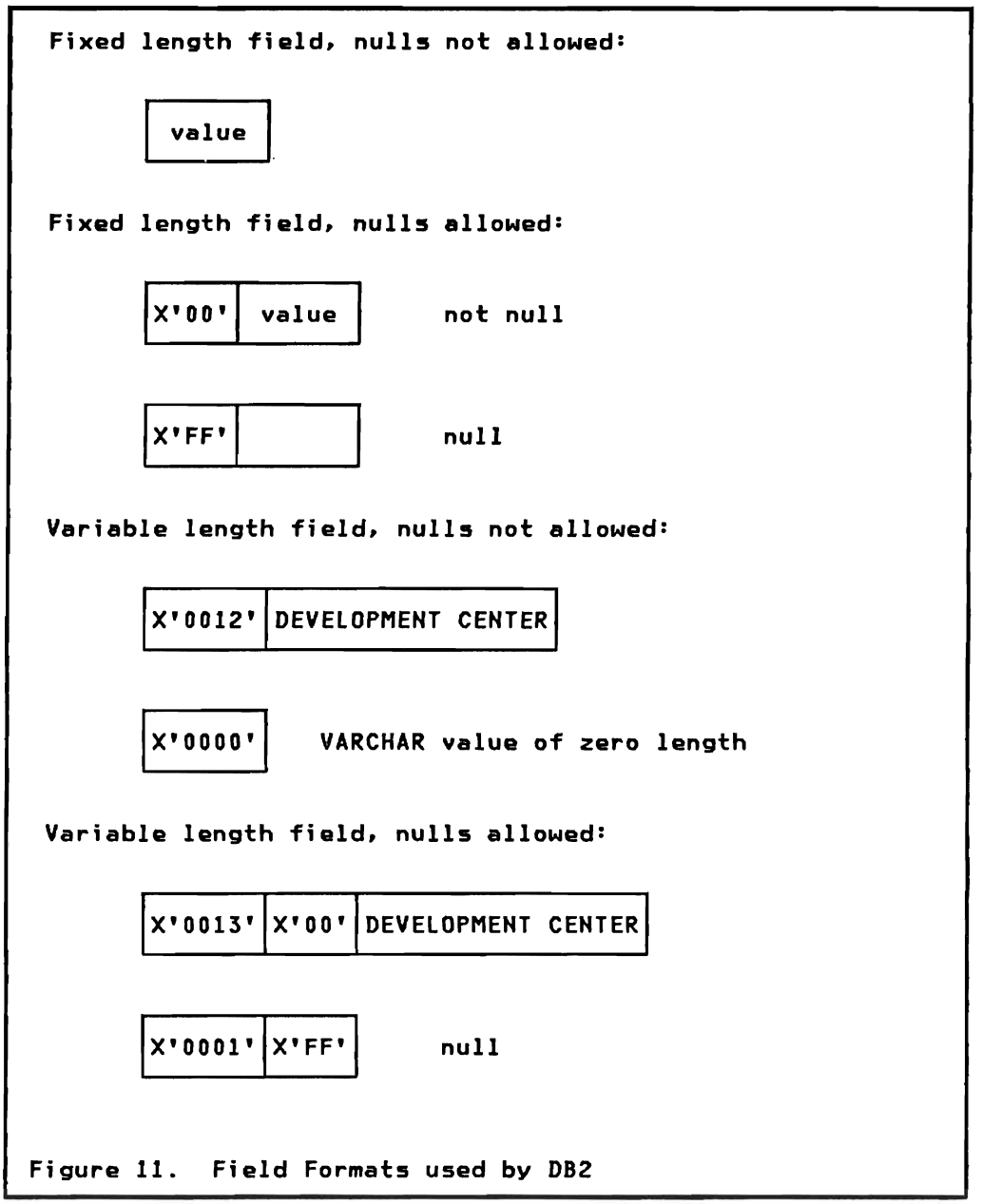


Figure 11. Field Formats used by DB2

2. Variable length rows contain one or more varying length character fields. Each varying length field has a two-byte length prefix. The value of the length prefix does not include this prefix. There are no gaps after varying length fields. All fields are stored contiguously, each one is byte aligned.

If null values are allowed for a field, an extra indicator byte is stored in front of the field value. This byte is X'00' if the field value is not null, and X'FF' if the field value is null. For a varying length field which allows null, the indicator is stored after the length prefix in front of the data value. The length does include the indicator byte.

See Figure 11 on page 64 which summarizes the different field formats used.

All numeric fields are encoded by DB2 so that the logical compare (CLC) instruction can be used to compare field values for all data types. INTEGER fields for example are encoded by inverting the high order sign bit.

Free Space Management

Free space within a page is handled by DB2 and not by VSAM. Rows deleted from the table are noted as free space within the page. When space is required for new rows or lengthened rows, DB2 will use the page's free space, or gather free space of deleted rows by packing active rows together. If not enough space is available within the page, the row is stored according to an insert strategy.

Index Management

DB2 indexes are stored in **index spaces**. An index space is similar to a simple table space, except that each index space handles exactly one index. The index space is created with the CREATE INDEX statement, there are no DDL statements for index spaces as such. Indexes and table data are stored in separate data sets. They logically belong to the same DB2 data base.

The physical page size in an index space is always 4K bytes. Each physical page consists of between 1 and 16 subpages.

Indexes are implemented as tree structures resembling the VSAM index implementation. See Figure 12 on page 66 illustrating an index tree structure for a clustering index. For a non-clustering index the arrows (pointers) from the index leaf pages would point to data pages in random order.

Sequential access of the table data in index sequence is using leaf pages only, whereas direct access uses all levels of the index tree.

Three levels of an index structure can handle approximately 8 million records (rows). The index tree remains balanced even after insertions and deletions.

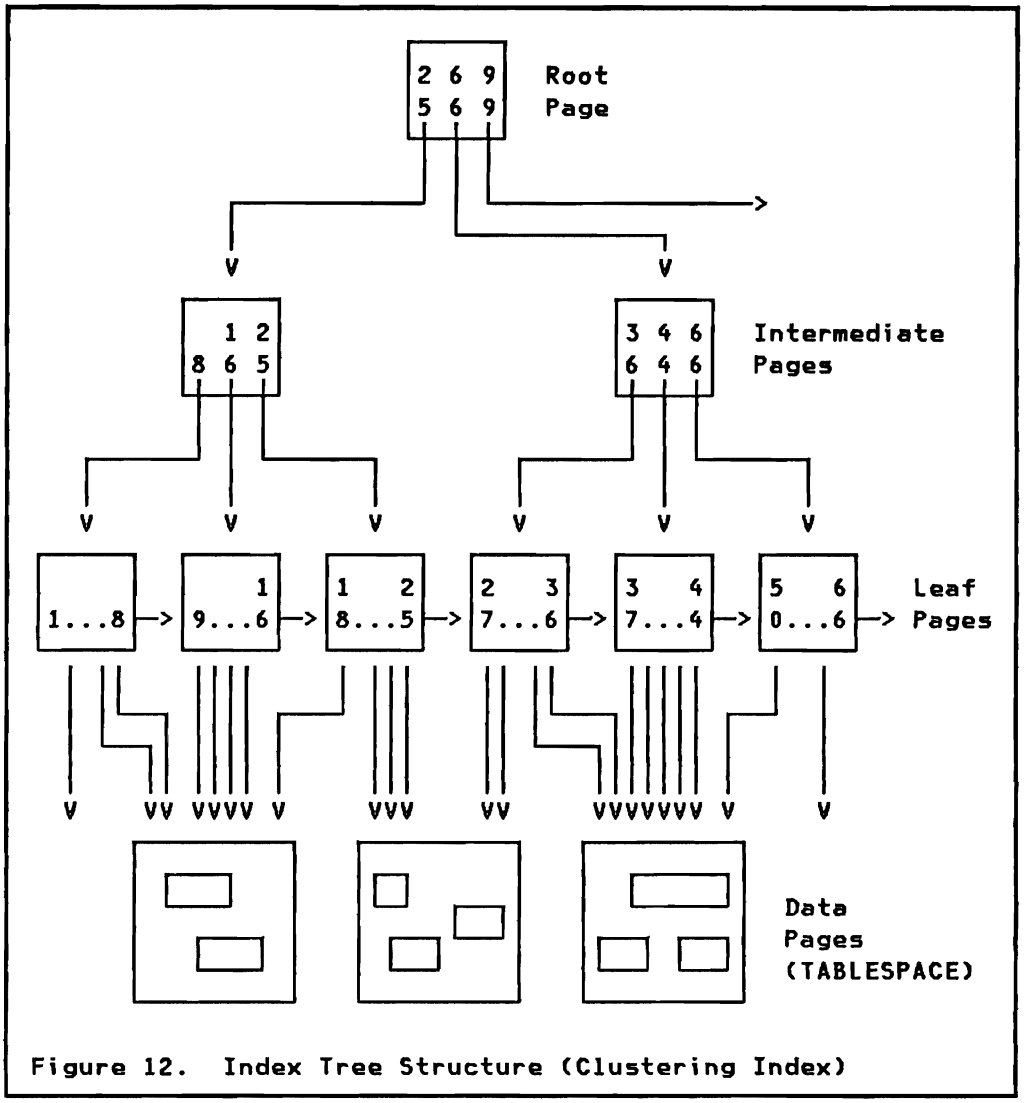


Figure 12. Index Tree Structure (Clustering Index)

Buffer Management

DB2 can handle up to four bufferpools (BP0,BP1, BP2, and BP32K), each of which may contain a large number of buffers.

Within a bufferpool, all the buffers are of the same size. Buffers are chained together on a "use chain," that is, they are chained together in the order in which they have been last used. More recently used buffers are at the top of the use chain; less recently used buffers are at the bottom. When a buffer is needed, DB2 selects the buffer at the bottom of the chain.

CONCURRENCY CONTROL - LOCKING

DB2 allows **any number of readers and updaters simultaneously** against tables. The necessary locking/unlocking is performed automatically by the system (at table space or page level); the user does not have to request and/or release locks explicitly.

The general rules for data consistency are:

- All modifications are subject to commitment.
- An application can never access data which has been updated by another application and is not yet committed.
- All pages updated are locked for exclusive use until commitment, at which time all page locks will be released.

DB2 employs locks to support data integrity, that is, to ensure that there is no undesirable interference between concurrent usage of data. While this mechanism guarantees the integrity of data it also affects the availability of the data to applications.

The data units of locks are table spaces and pages. Prior to accessing a table, DB2 will acquire a lock on the table space. The level of locking is dependent on the type of SQL request against the table. The level determines whether the table space is to be shared concurrently with other users, and what operations (read-only or update) other users may perform.

In addition to the locks at the table space level, individual pages may be locked when they are referenced.

Table Space Locks

The types of table space locks are summarized in Figure 13 on page 68. Table space locks are acquired during allocation of DB2 resources, e.g. normally when the application program issues the first SQL call.

The actual level of table space locking is dependent on the intent of the application (reference or change) against the table, and the LOCKSIZE attribute of the table space. The LOCKSIZE is specified with either the CREATE or ALTER TABLESPACE statement. The matrix of Figure 14 on page 68 shows the choice of lock acquired at the table space level.

S **Share** - Application references the table space (read only), and shares it with other read-only applications. No page locks are necessary.

X **Exclusive** - Application requests exclusive use of the table space. No page locks are necessary.

IS **Intent Share** - Application only references the table space, and tolerates other updating applications. Individual page locks at share level are necessary.

IX **Intent Exclusive** - Application makes changes to the table space, and tolerates other updating applications. Individual page locks at share or exclusive level are necessary.

SIX **Share with Intent Exclusive** - Application makes changes to the table space, and tolerates other read-only applications. Individual page locks at exclusive level are necessary for pages being updated.

Figure 13. Types of Table Space Locks

When the LOCKSIZE has been set to ANY, DB2 will select the appropriate locking level. This level is determined by the request, whether a few or many pages of the table space will be involved in the satisfaction of the request. If only a few pages are affected, page locking protocols will be employed; if many pages are affected, table space locking is selected.

Application Intent ↓	LOCKSIZE of CREATE TABLESPACE		
	TABLESPACE	PAGE	ANY
Reference (Read-only)	S	IS	S or IS
Change (Update)	X	IX or SIX	X or IX or SIX

Figure 14. Table Space Locking

Note: In certain circumstances DB2 may decide to override LOCKSIZE=PAGE and use table space locking (only S or X locks). The user is always informed on this decision.

Lock Statement

The user is allowed to request an explicit lock on the table space through the SQL LOCK statement.

```
LOCK TABLE table-name IN SHARE|EXCLUSIVE MODE
```

Although a table is named in the LOCK statement the table space is actually locked at share (S) or exclusive (X) level. SHARE mode allows concurrent readers (but not updaters), whereas EXCLUSIVE will not permit any other application to access the table space.

By acquiring a single lock on a table space performance may be enhanced considerably compared to the locking of individual pages. On the other hand, the LOCK statement in general delays other applications until the program has terminated.

Page Locks

Page locks are essentially concerned with concurrent usage of a table space where at least one user is updating. A page lock is for **shared (S)** use (other applications may concurrently read), or **exclusive (X)** use (no other application may access the page in any way). The choice of shared or exclusive is made depending on whether the operation performed is read (share lock) or update (exclusive lock).

Page locks are also applied to index pages. Index pages may contain data (keys) which are uncommitted, and are therefore locked individually. Index locks are on a subpage rather than a physical page (4K bytes).

Commit/Rollback

Changes in DB2 tables must either be committed or discarded. Committed changes are final and the changed data is available to other applications. Discarded changes are removed from the tables and the original data is available again to other applications.

COMMIT: Is automatically executed at the "normal" end of an application. It can also be invoked through the SQL COMMIT statement:

```
COMMIT
```

COMMIT releases all locks on pages acquired since the last commit point. All changes to table data are available to other users. Changes can no longer be backed out.

ROLLBACK: Is invoked through the SQL ROLLBACK statement:

```
ROLLBACK
```

ROLLBACK removes all changes done by the application from the tables. All locks on pages are freed as for COMMIT. The original data in the tables is available to other users.

Note: The COMMIT and ROLLBACK statements are only available in the TSO environment. Similar functions with an identical effect are available in IMS and CICS environments (see "Commit/Rollback" on page 86).

Note: Tablespace locks are only freed when the application terminates. The exception to this is in CICS, where in certain situations tablespace locks may be freed at commit time.

Deadlocks

In acquiring locks dynamically with DB2 it is possible that two (or more) requesters will become involved in a deadlock situation. Such a deadlock is resolved by DB2 based on the number of log records written on behalf of each requester.

In the TSO environment the requester with the least number of log records written since the last commit point will receive an **error return code** to the SQL request which caused the deadlock.

In IMS and CICS environments the selected requester is abended (with possible ROLLBACK) as in current DL/I application programs.

Data Definition Effect on Concurrency

Data definition operations (CREATE, ALTER, etc) performed on a data base can lock out the following activities until the work is committed or rolled back:

- Other definitional operations on that data base.
- Utility operations on that data base. Those already in progress are NOT suspended.
- Any dynamic SQL statements being prepared/executed.
- Any BIND operations against that data base. (BINDing is the process of finding the best access paths to table data and is presented in "Chapter 7. Application Programming Environment").

LOADING TABLES

Data can be loaded into DB2 tables by two different methods:

1. Using the LOAD utility. The LOAD utility loads data from a sequential data set into DB2 tables.

   - Input from the sequential data set may be reformatted.
   - Input data set may be created by SQL/DS.
   - Input may be created by the DXT program product.[2]

2. Using SQL INSERT statements in a user program or interactively.

Load Utility

The LOAD utility loads data from a BSAM sequential data set or SQL/DS unload tapes into DB2 tables. Like all DB2 utilities it runs as a JES- or TSO-initiated batch job under MVS.

You can use the utility to load an empty table, or to resume loading a table that has already been partially loaded. LOAD both loads records into one or more tables and builds any indexes you have defined for the table(s). The utility executes in three phases:

1. The load phase is a sequential pass through the input data set loading the table and writing work records for indexes.
2. The sort phase sorts the work records.
3. The build phase creates all the indexes from the sorted work file.

Before updating a table built using the LOAD utility, you should make an image copy of it using the COPY utility. This requirements ensures that you can recover that data to its loaded state. The recovery utilities (including COPY) are presented in "Chapter 11. Operation and Recovery."

---

[2] Data Extract (DXT), Program Number 5668-973

Formatting capabilities of the LOAD utility:

- Load multiple tables from one sequential input data set.
- Load one field of the input records into multiple tables.
- Truncate input data to fit into table column width.
- Select which fields in the input data set contain data for tables, loading from some and ignoring others.
- Convert from external print format into internal table format.
- Load fields of one data type into columns of another compatible data type (e.g. INTEGER into FLOAT). The LOAD utility automatically performs conversion.
- Load only from selected input records through the specification of a search condition.

The LOAD utility is controlled through a LOAD statement:

```
LOAD DATA
     INDDN        ddname
     RESUME       NO  | YES
     LOG          YES | NO
     WORKDDN      SYSUT1 | ddname
     SORTDEVT     device-type
     SORTNUM      number-of-sort-ds
     CONTINUE IF (startpos:endpos) = [X]'string'
INTO TABLE        table-name
                  ( fieldname,
                    fieldname datatype,
                    fieldname POSITION(start:end) datatype,
                    fieldname...NULLIF(start:end) = [X]'string',
                    ... )
     PART         partition-number
     FORMAT       UNLOAD | SQL/DS
     WHEN         (startpos:endpos) = [X]'string'  |
                  fieldname = [X]'string'
```

INDDN: Names the DDNAME of the input data set.

RESUME: Loading into an empty table, or resume loading into an existing table.

LOG: Logging may be suppressed to speed up the load process, but the utility must be restarted from the beginning if loading fails.

WORKDDN: Ddname of the work file when loading an indexed table.

SORTDEVT and SORTNUM: Specification of number and device type of SORT work data sets.

CONTINUE IF: Specifies which input records of the input data set are followed by continuation records.

INTO TABLE: Names a table to be loaded. Multiple INTO TABLE statements may follow a LOAD statement, each identifying a table to be loaded from the input data set. The list of field names specifies the table columns to be loaded, and the position and data type of these fields in the input data set. If the POSITION is not specified it is assumed to be just after the previous field, if the data type is not specified it is assumed to be identical to the column in the table. In addition to the data types supported for table columns, the data type of numeric fields may be qualified by the keyword EXTERNAL. Fields in EXTERNAL format are printed numbers with optional sign and decimal point (e.g. 234, -3.5). If the data types of input and table column are different the load utility will convert the input into the table format.

PART: Specifies that only one partition of a tablespace is loaded.

FORMAT: If FORMAT is specified then no field list is allowed since the format is uniquely predefined to be either the DB2 UNLOAD format, or the SQL/DS unload format. No data conversions may be performed with FORMAT specified.

WHEN: Optionally specifies which records of the input data set are to be loaded into the table. The condition may be expressed as either a field of the table or a position in the input record to be compared to a constant.

Example: Load of the employee table from records of an 'EMPDS' input data set which have 'ZCH' in position 7 to 9. The sequence of the fields in the input records is: EMPNO, constant, WORKDEPT, LASTNAME, PHONENO, JOBCODE, EDUCLVL, and SALARY. All the numeric fields are in printable format.

```
LOAD DATA
     INDDN      EMPDS
     STATS      DETAIL
INTO TABLE      EMPLOYEE                        -
                ( EMPNO,
                  LASTNAME  CHAR(15) POSITION(13:27),
                  WORKDEPT  POSITION(10:12)
                  PHONENO   POSITION(28:31),
                  JOBCODE   DECIMAL  EXTERNAL(3),
                  EDUCLEVL  SMALLINT EXTERNAL(2),
                  SALARY    DECIMAL  EXTERNAL(10) )
     WHEN       (7:9) = 'ZCH'
```

Data Extract (DXT)

The Data Extract (DXT), program number 5668-973, extracts data from files residing in IMS DL/I data bases, VSAM and SAM data sets. The extracted data is collected into files meeting the input format for the DB2 load utility.

DXT operates in an MVS environment. For processing VSAM and SAM data sets, DXT runs as a standard batch job. When processing DL/I data, DXT runs as either an IMS batch or BMP job. DXT can communicate with a user or product executing on another processor be means of RSCS, JES2, or JES3 networking. See Figure 15 on page 75 for the operating environment.

DXT dialogs enable users of DXT to interactively construct extract requests through ISPF menu driven screen facilities. The dialog feature includes:

- model extract requests that the user may tailor to his needs
- panels that help the user with execution JCL
- facilities to automatically submit DXT jobs for execution

Data descriptions stored in the DB/DC Data Dictionary are optionally available to the DXT user via a DXT interface program.

DXT is not designed for use by end users. Normally a DP professional would define and set up extract requests.

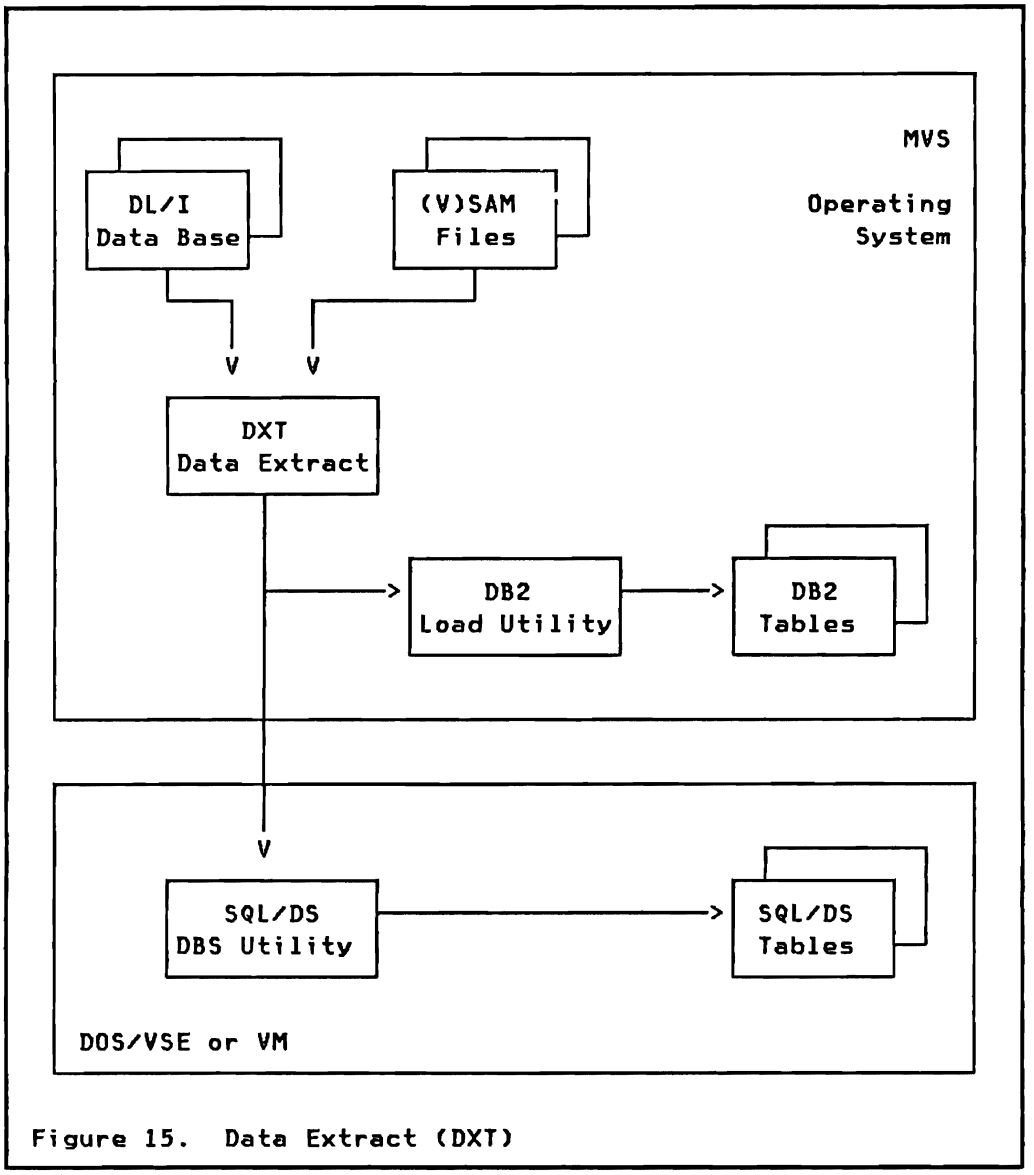


Figure 15. Data Extract (DXT)

In this chapter we look at application programming in a DB2 environment. The most important difference between application programming with DB2 and application programming with other data base systems is the data manipulation language. The same language introduced in "Chapter 4. SQL Data Manipulation"and "Chapter 5. Data Definition" is used in DB2 application programs. Some further SQL statements may only be used in application programs.

LANGUAGE SUPPORT

DB2 application programs may be written in **PL/I, COBOL, FORTRAN,** and **ASSEMBLER LANGUAGE.** The same SQL statements may be embedded in any of the supported languages.

SUBSYSTEM SUPPORT

An application program using DB2 tables can be written as IMS/VS BMP (Batch Message Processing Program), MPP (Message Processing Program), or MFP (Message driven Fast Path Program), a CICS/VS transaction, or a TSO batch job. The DB2 data base requests are the same in all these environments. Special considerations regarding the different environments will be presented in later chapters.

APPLICATION DEVELOPMENT CYCLE

Figure 16 on page 78 shows the flow of how an application program is prepared, compiled, and executed in a DB2 environment. The steps, each of which will be presented in more detail in this chapter, are:

DCLGEN: Generation of declarations of tables as source language data structures. From table definitions stored in the DB2 Catalog, declarations of rows as source language structures are generated for PL/I or COBOL programs, and stored as members of a partitioned data set.

PRECOMPILE: The DB2 precompiler analyzes the application source program, and includes table declarations from a partitioned data set. SQL statements are saved as a DBRM (Data Base Request Module) in a partitioned data set, and replaced in the source program by calls to a DB2 interface module.

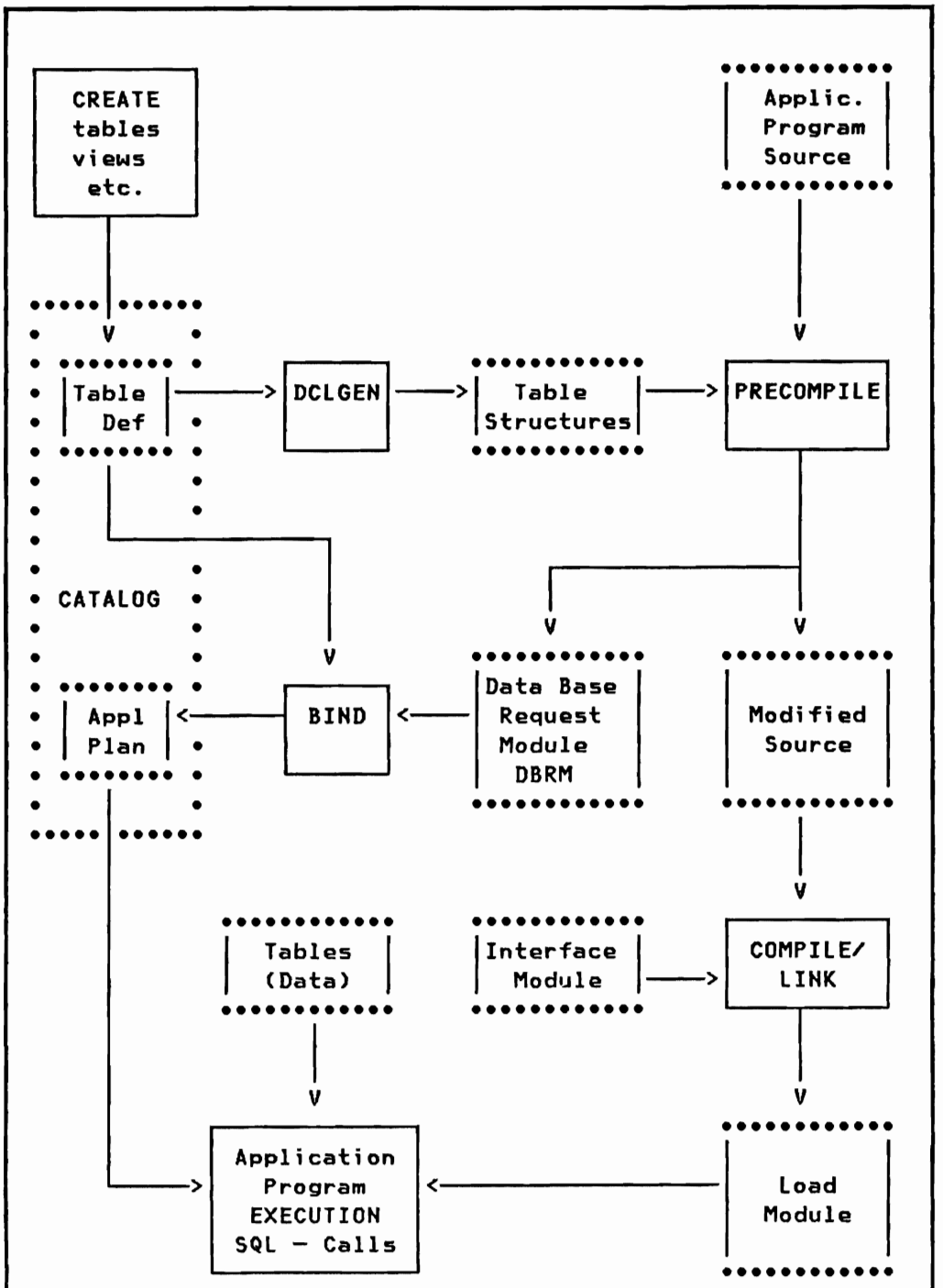


Figure 16. Application Development Cycle

BIND: The BIND process finds the access paths for the SQL requests of the DBRM, based upon table definitions, available indexes, etc. The results of BIND are stored in the catalog as an application PLAN.

COMPILE and LINK: Standard compilation by a language processor and subsequent linkage editing. A DB2 language interface module is link edited to the application load module.

EXECUTION: The load module is executed. With the first SQL call the application PLAN is loaded and tables are accessed using the search strategy established during BIND.

PROGRAM STRUCTURE

The structure of a SQL application program does not differ from the structuring of any other program. In the following sections the actual techniques and facilities of SQL specific coding are presented.

EMBEDDED SQL

SQL statements are embedded in application source code anywhere in the program. They will be analyzed by the PRECOMPILER and replaced by valid statements in the host language.

In addition to the SQL data manipulation and data definition statements presented in earlier chapters, SQL statements unique to application program coding are available. These include special support for declarations of tables, SQL communications area, and usage of host variables within SQL statements.

SQL Statement Format

When SQL statements are embedded in programs they require some delimiters to set them apart from programming language statements. Following are the conventions used in programming language source code:

PL/I: SQL statements are preceeded by EXEC SQL and terminated with a semicolon (;). They must be within columns 2 to 72.

```
EXEC SQL  SELECT *
            FROM EMPLOYEE
           WHERE EMPNO = '00010' ;
```

COBOL: SQL statements are preceeded by EXEC SQL and terminated by END-EXEC. They must be within columns 12 to 72.

```
EXEC SQL  SELECT *
            FROM EMPLOYEE
           WHERE EMPNO = '00010'
END-EXEC
```

ASSEMBLER: SQL statements are preceded by EXEC SQL and continued on subsequent cards by standard practice with a non-blank character in column 72.

```
      EXEC SQL  SELECT *                                  *
                FROM   EMPLOYEE                           *
                WHERE  EMPNO = '00010'
```

FORTRAN: SQL statements follow FORTRAN coding conventions with a continuation indicator in column 6. They must be within columns 7 to 72.

```
       EXEC SQL  SELECT *
      C            FROM EMPLOYEE
      C           WHERE EMPNO = '00010'
```

Note: All the examples presented later in this chapter are coded as EXEC SQL statements without the language specific terminator or continuation character.

INCLUDE Statement

DB2 provides a facility to include source code from a library into the program. This function is very similar to %INCLUDE in PL/I programs, and COPY in COBOL and ASSEMBLER.

```
EXEC SQL INCLUDE member-name
```

Any kind of source code may be included through the SQL INCLUDE statement. The included code may contain language source and/or SQL statements. One restriction must be noted however:

- Nested INCLUDE's are not supported, that is, the included code must not contain EXEC SQL INCLUDE statements.

SQL Communication Area (SQLCA)

Each program using DB2 must include the SQL Communication Area (SQLCA). The SQLCA is the means by which DB2 communicates with the application program. The structure of the SQLCA is shown in Figure 17 on page 81. Of all the fields within the SQLCA note that SQLCODE will contain a return code after execution of any SQL statement.

```
DCL 1 SQLCA,
      2 SQLCAID     CHAR(8),           <-- 'SQLCA'
      2 SQLCABC     BIN FIXED(31),     <-- length (136)
      2 SQLCODE     BIN FIXED(31),     <-- return code
      2 SQLERRM     CHAR(70) VAR,      <-- error message
      2 SQLERRP     CHAR(8),           <-- module (error)
      2 SQLERRD(6)  BIN FIXED(31),     <-- special codes
      2 SQLWARN,                       <-- warning codes
        3 SQLWARN0  CHAR(1),           <-- 'W' any warning
        3 SQLWARN1  CHAR(1),           <-- truncated string
        3 SQLWARN2  CHAR(1),           <-- null val elimin.
        3 SQLWARN3  CHAR(1),           <-- # hostvar wrong
        3 SQLWARN4  CHAR(1),           <-- upd/del all rows
        3 SQLWARN5  CHAR(1),           <-- used SQL/DS
        3 SQLWARN6  CHAR(1),
        3 SQLWARN7  CHAR(1),
      2 SQLEXT      CHAR(8);           <-- reserved
```

Figure 17. SQL Communication Area (PL/I Format)

The SQLCA may be explicitly coded by the programmer, included by means of source COPY or %INCLUDE, or generated by the special SQL INCLUDE statement:

```
EXEC SQL INCLUDE SQLCA
```

Table Declaration

Tables and views used in SELECT statements may optionally be declared in the source program in the format of a SQL DECLARE statement.

```
EXEC SQL DECLARE table-name TABLE
          ( EMPNO     CHAR(6)     NOT NULL,
            LASTNAME  VARCHAR(15) NOT NULL,
            WORKDEPT  CHAR(3)     NOT NULL,
            PHONENNO  CHAR(4),
            JOBCODE   DECIMAL(3),
            EDUCLVL   SMALLINT,
            SALARY    DECIMAL(8,2)  )
```

This declaration allows the precompiler to check SQL data manipulation statements against the table declaration. Incorrect usage of SQL may thus be detected before the BIND process.

Note: The field definition part of the table is identical to the format used in the CREATE TABLE statement shown in "Defining a Table" on page 49.

Normally the SQL table declaration is followed by a structure declaration using the host language. The host language

declaration is used to actually store and access data retrieved by SQL SELECT statements. The PL/I declaration of the employee table would look like:

```
DECLARE
  1 EMPLOYEE,
     3 EMPNO       CHAR(6),
     3 LASTNAME    CHAR(15) VARYING,
     3 WORKDEPT    CHAR(3),
     3 PHONENNO    CHAR(4),
     3 JOBCODE     DEC FIXED (3),
     3 EDUCLVL     BIN FIXED (15)
     3 SALARY      DEC FIXED (8,2);
```

Generate Declarations (DCLGEN)

DB2 provides the DCLGEN facility to create the SQL declaration of a table or view and the host language declaration of the same table for PL/I and COBOL source programs. The TSO subcommand DCLGEN may be invoked after a table has been defined to DB2 through a CREATE TABLE statement.

```
DCLGEN TABLE     (table-name)
       LIBRARY   (library-name(member-name))
       LANGUAGE  (PLI | COBOL)
       ACTION    (ADD | REPLACE)
       NAMES     (prefix)
       STRUCTURE (structure-name)
```

The output of DCLGEN is stored in a partitioned (or sequential) data set with a data set name built as auth-id.library-name.PLI (or COBOL). If no STRUCTURE name is given, the host language structure is named 'DCLtable-name'. The fields will be named as the columns of the table or view. By specifying NAMES the fields may be numbered sequentially as 'prefix1' etc.

The output of DCLGEN thus comprises the EXEC SQL DECLARE statement for the table and the PL/I or COBOL structure declaration representing a table row.

Retrieving Data into Host Variables

An expansion of the SELECT statement allows you to retrieve **one row of data** directly into host variables.

```
EXEC SQL SELECT *
           INTO :EMPLOYEE
           FROM EMPLOYEE
          WHERE EMPNO = '000260'
```

The **INTO clause** of SELECT directs DB2 to retrieve the requested row of data into the host structure 'EMPLOYEE'. Host variables

are identified by the preceding colon (:). This prefix is optional when no naming conflicts exist, but as a rule host variables should be prefixed. DB2 supports two level structures only.

The INTO clause may also be coded to put field values into a set of non-contiguous host variables:

```
EXEC SQL SELECT LASTNAME, SALARY
           INTO :NAME, :MONEY
           FROM EMPLOYEE
          WHERE EMPNO = '000260'
```

Normally a SELECT statement retrieves a set of rows. SELECT with the INTO clause will return an error code if more than one row is retrieved. A WHERE clause referencing a column having a unique index can insure that exactly one row qualifies. SQL provides the **CURSOR facility** to deal with multiple rows.

Updating, Inserting and Deleting Data

SQL UPDATE, DELETE, and INSERT statements as presented in "Chapter 4. SQL Data Manipulation" may be used unrestricted in application programs. In addition variations of UPDATE and DELETE statements may be used to change or delete one single row of data retrieved. These are described in the next section.

CURSOR Operations

The SQL cursor operations allow application programs to manipulate tables or views one row at a time. The program can step through the set of rows retrieved by a SELECT statement, and UPDATE or DELETE that specific row.

The result table of a cursor operation is conceptually a sequential data set containing the rows retrieved as a result of a SELECT statement. The application program then "reads" each row until "end-of-file" is reached. The set of rows can contain from zero to many rows.

DECLARE Cursor

The first step in a cursor operation is to declare the cursor and associate a SELECT statement with it.

```
EXEC SQL DECLARE cursor-name CURSOR FOR
                        SELECT columns ...
                               FROM/WHERE/GROUP BY/HAVING
                 FOR UPDATE OF column, column, ...
                      ORDER BY column, ...
```

Declaring a cursor does not retrieve any data. Data manipulation is performed through OPEN, FETCH, UPDATE, DELETE, and CLOSE statements. Updates to a row are only allowed if the 'FOR UPDATE OF' clause has been included in the cursor declaration. Rows may be presented to the application program in a specified order by using the 'ORDER BY', but this prevents updating.

OPEN Cursor

The OPEN statement is used to prepare for data retrieval by DB2. No data is returned to the program yet.

```
EXEC SQL OPEN cursor-name
```

FETCH a Row

The FETCH statement returns the fields of one row into the host variables. Each execution of the FETCH statement returns the data of one row qualifying the WHERE search condition of the SELECT statement declared with the cursor. Normally the FETCH statement is imbedded into a loop running as long as data is available.

```
EXEC SQL FETCH cursor-name
          INTO :variable, ...
```

Updating or Deleting a Row

The retrieved row may optionally be updated or deleted. Updating is only allowed for fields specified in a FOR UPDATE OF clause when the cursor is defined.

```
EXEC SQL UPDATE table-name
            SET field = value, ...
          WHERE CURRENT OF cursor-name

EXEC SQL DELETE table-name
          WHERE CURRENT OF cursor-name
```

CLOSE Cursor

After having finished the handling of data the cursors operation is terminated by a CLOSE statement. The same cursor may then be opened again.

```
EXEC SQL CLOSE cursor-name
```

Error Handling

When DB2 processes a SQL statement of the program, it places a return code into the SQLCODE variable of the SQLCA (see Figure 17 on page 81). The return codes are:

<0 Error in execution of SQL statement.

>0 Exceptional but valid condition, e.g. no row qualifies for a SELECT statement.

0 All OK, no errors or exceptional conditions. However, some conditions (like truncation of character data when the host variable is too short), give a return code of zero but set flags in SQLWARNx of the SQLCA.

The return code may be tested through normal programming language statements, or actions may be taken automatically by setting up a WHENEVER statement prior to data manipulation.

WHENEVER Statement

The WHENEVER statement allows for automatic testing of SQL return codes on all subsequent SQL statements (appearing later in the source program). Using WHENEVER you do not have to test the return code yourself after each SQL statement execution.

You may have up to three WHENEVER statements active, each of them testing for one of the possible conditions.

```
EXEC SQL WHENEVER NOT FOUND    CONTINUE | GOTO :label

EXEC SQL WHENEVER SQLWARNING   CONTINUE | GOTO :label

EXEC SQL WHENEVER SQLERROR     CONTINUE | GOTO :label
```

The NOT FOUND condition tests for a retrieval which could not find any qualifying rows, the SQLWARNING condition tests for all other positive return codes (or zero with SQLWARNx flags set), and the SQLERROR condition tests for all negative return codes.

Concurrency Support - Locking

In general the application programmer is not concerned about locks. DB2 handles all the locking automatically. Every row that has been updated is locked (actually the whole page containing the row is locked) until the next commit (synchronization) point.

The application program may lock a whole table by using the SQL LOCK statement.

```
EXEC SQL LOCK TABLE table-name IN SHARE|EXCLUSIVE MODE
```

SHARE mode allows other read-only programs, whereas EXCLUSIVE does not allow any concurrent access to the table space containing the named table.

The scope of locking may also be influenced by parameters defined during the BIND process for the application program. The **BIND ISOLATION** parameter identifies the degree to which the program wants to be isolated from the effects of other programs. It specifies how long any row retrieved by the program can be guaranteed not to change.

Cursor Stability (CS): This specification indicates that any row retrieved can not be changed by other programs until this program moves off the row (or closes the cursor). This choice has the least impact on performance of other programs.

Repeatable Read (RR): Indicates that any row retrieved is locked until this program reaches a synchronization point (commit point). No other program may change any row which has been retrieved already by this program. Repeatable Read (RR) should be specified when your program needs to process several rows before making a decision to update, or when the same set of rows must be retrieved multiple times with no changes guaranteed in the mean time.

Note: Since Cursor Stability and Repeatable Read are specified at BIND time for the whole program, they affect all the tables accessed by the application program.

Commit/Rollback

Changes made by an application program to the various tables it uses may be committed (made permanent) at a certain time through the SQL COMMIT statement. This occurs automatically at the end of the program, or by request one or multiple times during execution of the application program. If committing can take place early, then the resources locked because of pending changes will be freed earlier, resulting in greater system throughput.

Similarly, if the application decides that changes made to tables are not valid, it must "rollback" those changes, resulting in the same state as they were prior to the updates.

The SQL COMMIT and ROLLBACK statements as introduced in "Chapter 6. Data Management" section "Commit/Rollback" on page 69 can be explicitly invoked from a TSO application program through the appropriate SQL statements:

```
EXEC SQL COMMIT

EXEC SQL ROLLBACK
```

Changes to DB2 tables are either committed or removed. All locks are released and changed (or original) data is available to other application programs. Any open cursors are closed.

In the IMS environment COMMIT is requested by an IMS synchronization point (e.g. GU/CHKP call), and ROLLBACK by a DL/I ROLL call.

In the CICS environment COMMIT is requested by a CICS synchronization point, and ROLLBACK by a transaction abend or CICS ROLLBACK.

HOST VARIABLES

Host variables are variables of the programming language used for the application. They provide selection values for search conditions and storage areas for results of retrieve operations. They are declared like any other variable used in the program. Here are some of the rules regarding the usage of host variables:

- Host variables are prefixed by a colon (:) when used in SQL statements. Although the colon is optional in most cases, it is recommended to have the colon coded for clarity and differentiation from table column names.
- The host variable must be of a data type compatible to the data type of the table column. Conversions between compatible data types are supported by DB2.
- Host variables may be two level structures. This is very convenient for the retrieval of one row of data into a structure. An example was shown in "Retrieving Data into Host Variables" on page 82.

PL/I

SMALLINT	BIN FIXED(15)
INTEGER	BIN FIXED(31)
DECIMAL(p,q)	DEC FIXED(p,q)
FLOAT	BIN FLOAT(53)
CHAR(n)	CHAR(n)
VARCHAR(m)	CHAR(m) VAR
LONG VARCHAR	CHAR(x) VAR

COBOL

SMALLINT	PIC S9(4) COMP
INTEGER	PIC S9(9) COMP
DECIMAL(p,q)	PIC S9(r)V9(q) COMP-3. (r=p-q)
FLOAT	COMP-2.
CHAR(n)	PIC X(n)
VARCHAR(m)	xx struct 49 PIC S9(4) COMP. <- length 49 PIC X(m) <- data
LONG VARCHAR	as VARCHAR

ASSEMBLER

SMALLINT	DS H
INTEGER	DS F
DECIMAL(p,q)	DS PLp'0000.000' (p=7 and q=3)
FLOAT	DS D
CHAR(n)	DS CLn
VARCHAR(m)	DS H,CLm <- length and data
LONG VARCHAR	as VARCHAR

FORTRAN

SMALLINT	INTEGER*2
INTEGER	INTEGER
DECIMAL(p,q)	.. no equivalent
FLOAT	REAL*8
CHAR(n)	CHAR*n
VARCHAR(m)	.. no equivalent, use CHAR*n
LONG VARCHAR	as VARCHAR

Figure 18. Data Types of Host Variables

- A host variable may be qualified by a higher level structure identifier which is exactly one level higher.

```
DCL 1 STR,
      2 ...
      2 EMPDATA,
        3 MONEY    BIN FIXED,
        3 ...
SELECT SALARY FROM EMPLOYEE
  INTO :EMPDATA.MONEY;
```

  See Figure 18 on page 88 for host variable data types in the supported languages.

- You need an indicator variable for each host variable where the value retrieved might be the null value. This technique is presented below.

Handling Null Values

Null values are NOT returned into host variables. You must set up an additional indicator variable (binary halfword) for each field which may have a null value. If a null value must be returned to the program, DB2 will mark the indicator variable with a negative value and leave the data variable untouched. An error results when no indicator variable is available to DB2.

The indicator variable is specified in the INTO clause of the SELECT statement immediately following the host variable:

```
DCL  MGRNO      CHAR(6),
     XMGRNO     BIN FIXED;     /* NULL INDICATOR */

EXEC SQL SELECT MGRNO
           INTO :MGRNO:XMGRNO
           FROM DEPARTMENT
          WHERE DEPTNO = 'D01'

IF XMGRNO < 0 THEN .....
```

If the host variable is a structure you need an indicator variable for each field of the structure. The indicator variable is coded as an array of halfwords, where each element of the array corresponds to one of the fields of the structure. The dimension of the indicator array is equal to the number of fields, even if nulls are not allowed for some of the fields.

```
DCL  1 EMPLOYEE,
       3 EMPNO        CHAR(6),
       3 LASTNAME     CHAR(15) VARYING,
       3 WORKDEPT     CHAR(3),
       3 PHONENNO     CHAR(4),
       3 JOBCODE      DEC FIXED (3),
       3 EDUCLVL      BIN FIXED (15)
       3 SALARY       DEC FIXED (8,2),
     EMPIND(7)        BIN FIXED (15),
     EMPLNUMBER       CHAR(6);

EXEC SQL SELECT *
           INTO :EMPLOYEE:EMPIND
           FROM EMPLOYEE
          WHERE EMPNO = :EMPLNUMBER;
```

Data Definition Statements

All SQL data definition statements introduced in "Chapter 5. Data Definition"may also be executed from a program. This includes creation and definition of tables and other DB2 objects.

For example, an application program might store the result of a SELECT statement into a "temporary" work table for further analysis and selection.

```
EXEC SQL CREATE TABLE  TEMP
                       (EMPNO   CHAR(6) NOT NULL,
                        JOBCODE DECIMAL (3))
EXEC SQL INSERT INTO   TEMP (EMPNO, JOBCODE)
                SELECT EMPNO, JOBCODE
                  FROM EMPLOYEE
                 WHERE EDUCLVL > 16
```

DYNAMIC SQL

For some kind of applications it is desirable to execute SQL statements which are not known until the program is actually running. An example is a program to support an interactive user who wishes to enter requests and receive results at a terminal. In this case, the SQL statements can not be imbedded in the program and recognized by the precompiler, since the statements are read from a terminal when the program is running. To support applications such as this, SQL provides facilities for translating and executing SQL statements at run-time. These facilities are provided by the **dynamic SQL statements** PREPARE, DESCRIBE, and EXECUTE, in conjunction with a structure called the SQL Descriptor Area (SQLDA).

SQL Descriptor Area (SQLDA)

The SQL Descriptor Area is a control block that DB2 uses to pass information concerning the dynamic SQL statement to the application program. A declaration of a SQLDA may be included in a program by means of a SQL INCLUDE statement.

For PL/I, the SQLDA structure shown in Figure 19 is generated through:

```
EXEC SQL INCLUDE SQLDA;
```

The SQLDA basically describes all the columns returned by a SELECT statement to allow an application program to deal easily with individual values.

```
DCL 1 SQLDA          BASED(SQLDAPTR),
      2 SQLDAID      CHAR(8),              <-- 'SQLDA'
      2 SQLDABC      BIN FIXED (31),       <-- length
      2 SQLN         BIN FIXED (15),       <-- max nr of fields
      2 SQLD         BIN FIXED (15),       <-- actual nr
      2 SQLVAR     (SQLSIZE REFER(SQLN)),<-- fields
        3 SQLTYPE    BIN FIXED(15),        <-- data type
        3 SQLLEN     BIN FIXED(15),        <-- field length
        3 SQLDATA    PTR,                  <-- ptr to data
        3 SQLIND     PTR,                  <-- ptr to indicator
        3 SQLNAME    CHAR(30) VARYING,     <-- column name
DCL   SQLSIZE        BIN FIXED(15);        <-- nr of fields
DCL   SQLDAPTR       PTR;                  <-- ptr to sqlda
```

Figure 19. SQL Descriptor Area (PL/I Format)

PREPARE Statement

The SQL PREPARE statement is used to "prepare" a SQL statement for execution, and associate a statement name with it. A prepared statement may then be referenced by name in DESCRIBE and EXECUTE statements, and in cursor declarations.

```
EXEC SQL PREPARE statement-name
            INTO :sqlda-structure
            FROM :string-expression
```

PREPARE causes the string expression to be parsed and checked for errors. If the optional INTO clause is provided then a description of the sizes and types of variables needed to receive the results of the SELECT statement is generated in the named structure (which should be an SQLDA).

For documentation purposes it is suggested to "declare" the statement name:

```
EXEC SQL DECLARE statement-name STATEMENT
```

Example of a SQL PREPARE statement:

```
EXEC SQL DECLARE SRCSTMT STATEMENT;
DCL  SOURCE      CHAR(255) VAR;
     /* assume source read from input */
     SOURCE = 'SELECT EMPNO, LASTNAME
                 FROM EMPLOYEE
                WHERE EMPNO=''000010''  ';
EXEC SQL PREPARE SRCSTMT
            INTO :SQLDA
            FROM :SOURCE;
```

DESCRIBE Statement

If the dynamically supplied statement might be a SELECT, then the program may require information about the sizes and types of variables needed to receive the results of the SELECT. The SQL DESCRIBE statement may be used to obtain this information in a SQLDA, so that the program can dynamically allocate buffers of the correct size and type to receive the results.

```
EXEC SQL DESCRIBE statement-name
             INTO :sqlda-structure
```

The statement must have been PREPAREd as above. The SQLDA structure contains an indicator telling if the statement was a SELECT, and if yes, the sizes and types of all values to be returned if the SELECT is executed.

The INTO clause of the PREPARE provides the same function as a subsequent DESCRIBE statement.

EXECUTE Statement

The SQL EXECUTE statement allows you to execute a dynamic SQL statement that has previously been PREPAREd.

The SQL statement must not be a SELECT statement. These must be handled by extensions of cursor operations which are described later in this section.

```
EXEC SQL EXECUTE statement-name
           USING :hostvar, ...
           USING DESCRIPTOR :sqlda-structure
```

The optional USING clause is used to pass parameter values to be substituted into the SQL statement. Parameters are passed either as a list of host variables, or via an SQLDA host variable list.

Parameterized Execution

Some values of the dynamic SQL statement may be left open (unknown) when the statement is prepared. Such "parameters" are coded as question marks (?) in the SQL statement, and actual values are supplied with the EXECUTE statement. This technique is demonstrated in the example below:

```
EXEC SQL DECLARE PARMSTMT STATEMENT;
DCL  SOURCE      CHAR(255) VAR,
     DEPART      CHAR(3);      /* set by program */

SOURCE = 'DELETE FROM  EMPLOYEE  WHERE WORKDEPT = ? ';
EXEC SQL PREPARE PARMSTMT                          |
            FROM :SOURCE  ;                        |
EXEC SQL EXECUTE PARMSTMT                          |
           USING :DEPART  ;   --------------------┘
```

Immediate Execution

Dynamic SQL statements without any reference to host variables may be executed immediately without previous preparation. The statement is analyzed, translated, and executed.

```
SOURCE = 'DELETE FROM EMPLOYEE WHERE EMPNO=''000010''';
EXEC SQL EXECUTE IMMEDIATE :SOURCE ;
```

Using Cursors with Dynamic SQL

The cursor operations introduced earlier in this chapter must be used to execute a dynamic SQL SELECT statement. Since the coding is just a minor variation of the cursor technique described above, only an overview of the required statements is shown below.

```
EXEC SQL DECLARE stmt    STATEMENT
EXEC SQL DECLARE cursor  CURSOR FOR stmt

EXEC SQL PREPARE stmt  INTO :sqlda  FROM :source

/* examine and fill SQLDA */

EXEC SQL OPEN    cursor  USING :hostvar-parameter, ...
EXEC SQL FETCH   cursor  USING :sqlda
EXEC SQL CLOSE   cursor
```

The cursor is declared FOR a dynamic statement. The statement must be PREPAREd before the cursor can be opened. Parameter

values can be passed through the OPEN statement to replace question marks in the SQL statement. The FETCH statement retrieves a row of data into host variables, which have been defined in an SQLDA structure.

PROGRAM PREPARATION

When your application program design and coding are complete, you are ready to prepare the source statements for execution. The steps necessary for program preparation were introduced briefly at the beginning of this chapter in section "Application Development Cycle" on page 77.

Precompile

Before a program can be executed, it must be compiled by a compiler of the appropriate host language. Before compilation can occur, however, the SQL statements imbedded in the host language program must first be prepared by the **DB2 Precompiler** for compilation as host language statements.

The DB2 Precompiler scans every statement of the program and produces a modified program in which every SQL statement has been replaced by one or more statements of the host language.

Below is a list of some of the actions the Precompiler performs:

- Matching host variables with SQL statements. The variable names and definitions are used to check the validity of the SQL statements.

- Validation and syntax checking of SQL statements. Some validation of the SQL statement is possible if declarations of the tables used are included in the source program.

- Replace the SQL statements by host language statements. Some statements (like WHENEVER and DECLARE) are replaced by a comment only, other statements are replaced by a call to the SQL language interface module.

```
CALL DSNHLI (SQLPLISTn)
```

- Store extracted SQL statements in the format of a **Data Base Request Module (DBRM)**. The DBRM contains information about each precompiled SQL statement. It is put into a partitioned data set, where it is available for a later process, called the BIND process.

The DB2 Precompiler can be invoked at any time to compile embedded SQL statements. DB2 does not have to be active, because the precompiler does not refer to any DB2 data. This means that the names of tables and fields used in SQL statements are not

validated against the current DB2 data bases. These checks are performed at BIND time.

Compilation and Link-Edit

After precompilation the application program is compiled using the standard compilation procedure suitable for the host language.

Link-editing is performed after compilation. The DB2 language interface module must be included during this phase.

Creating an Application Plan (BIND)

After compilation and link-editing, the application program is almost ready for execution. Before you can execute the program, however, you must establish a relationship between the program and its DB2 data. This process is called **binding**. Binding a program must occur on the same system that invokes DB2, and DB2 must be active. Binding does four things:

1. **SQL statement validation:**

   All SQL statements of the DBRM are validated against the table definitions. The program cannot be executed until all SQL statements are syntactically correct.

2. **Authorization checking:**

   The BIND authorization process checks if the person invoking the BIND is authorized to access the tables the program requests. More about the security and authorization mechanism is presented in "Chapter 9. Security and Authorization."

3. **Access path selection:**

   The BIND process selects the path to the data. It considers all indexes available to access the data, and decides which ones (if any) to use when building a path to the data.

4. **Application plan creation:**

   If all SQL statements are correct, and authorization to access the data is available, the BIND process builds an application plan.

   An application plan contains information about the program and about the data the program intends to use. It also contains the access modules the program will call to access data in a table. The application plan is stored in the DB2 Catalog.

Because application plans are stored by DB2, an application program that uses a plan can be executed many times without repeating the BIND process. However, rebinding occurs automatically if attributes of the data change and cause the plan to become invalid. For example, deletion of an index which is used in an access path will invalidate the application plan.

BIND Command

The BIND process is invoked through the TSO BIND subcommand, or an SPF full screen menu provided by DB2:

```
BIND PLAN      (plan-name)
     MEMBER    (dbrm-member-name,...)
     LIBRARY   (dbrm-lib-name,...)
     ACTION    (ADD | REPLACE)
     RETAIN    (NO | YES)
     VALIDATE  (RUN | BIND)
     ISOLATION (RR | CS)
     FLAG      (I | W | E | C)
```

PLAN: Gives the name of the plan to be stored by DB2.

MEMBER: Names the member (or members) of the DBRM library to be included in the plan. Multiple members are used if application modules have been independently (pre-)compiled and then link edited together.

LIBRARY: Names the libraries containing the DBRM members selected. A library defined through a DBRMLIB DD statement is searched if no LIBRARY parameter is given.

ACTION: Indicates if a new plan is added, or an existing plan replaced.

RETAIN: YES must be specified to keep (retain) existing EXECUTE authority for the plan in the case of a replace action. If not specified only the person invoking BIND is authorized to execute the program (use the plan).

VALIDATE: Indicates if full validity checking can be deferred to run time, or must be applied already during BIND. The checks which can be deferred to run time are "table existence" and "access authority."

ISOLATION: Indicates the degree to which the program wants to be isolated from the effects of other programs. The two options, which were presented in section "Concurrency Support - Locking" on page 86, are **Repeatable Read (RR)** and **Cursor Stability (CS)**.

FLAG: Indicates the level of messages (Informational, Warning, Error, Completion) to be produced by BIND.

REBIND Command

Under some circumstances you might want to invoke the rebind process directly. REBIND evaluates all currently available indexes to select an access path to the data. Changes that might prompt you to rebind an application plan include: new or deleted indexes, new columns added to a table, and a dropped table or view.

```
REBIND PLAN      (plan-name,...) | PLAN(*)
       FLAG      (..)
       VALIDATE  (..)
       ISOLATION (..)
```

Rebinding may occur for selected plans ("plan-name,.."), or for all the plans the user has BIND authority for ("PLAN(*)"). All other options are equal to the BIND command.

FREE Command

Application plans may be explicitly deleted when the corresponding application programs are not used any more. The DB2 FREE command is used to delete one or more application plans from the catalog:

```
FREE PLAN (plan-name)
```

PROGRAM EXECUTION

Once an application plan has been created with the BIND process, the program can be executed. Each time that the program is executed, DB2 verifies that the information in the application plan is consistent with the corresponding information in the DB2 catalog. Some of the validation checking may occur at run time, e.g. the testing for the existence of a table may be deferred to run time.

In the TSO environment the program may be executed through the RUN command:

```
RUN PROGRAM (program-name)
    PLAN    (plan-name)
```

Automatic Bind

If changes have occurred to the access path a program uses with its application plan, the BIND process is automatically initiated at run time. This automatic binding is triggered by an invalidated application plan. For example, an application plan is invalidated if an index is dropped which was used by the plan.

Dynamic Bind

Dynamic BIND occurs during execution when dynamic SQL statements are executed. Since these statements are not known in advance (at precompilation time), no data base request module (DBRM) was available for BIND. Dynamic SQL statements are bound just when they are executed from the application program. No application plan is generated by dynamic bind.

Testing Facilities

Programs using DB2 data bases are tested like any other programs. You have to set up a **Test Environment** for DB2 application testing. This includes:

- Test JCL procedures for precompile, compile, link-edit, binding, and execution.
- Test data as subsets of production data. This is very easy using DB2 CREATE TABLE and INSERT statements.
- Test input data to test each path and error routine in the program.
- TSO interactive SQL execution. The SQL statements used in an application program can be tested from TSO terminals using a tailored DB2 facility. This is presented in "Chapter 8. TSO Environment."
- TSO Test command.
- Interactive compilers like TESTCOB for COBOL programs.

This chapter describes how DB2 fits into the TSO environment of an MVS system. The following items were the main considerations in the implementation of the interface between TSO and DB2:

- **Usage of ISPF**

  ISPF is used as base of interaction between a terminal user and DB2. This dialog is supported through a number of ISPF panels attached to an extension of the ISPF main option menu. ISPF is not required for DB2, but access to DB2 is much easier with ISPF than without it.

- **Ease of use**

  The TSO/ISPF support was designed as a user friendly interface for a terminal user. All tasks necessary to design, implement, and run an application using DB2 data sets are fully supported through this interface.

- **Designed for DBAs and programmers**

  The interface supports the tasks of a DBA, which are: data base design, implementation, backup, recovery, etc. It also supports the whole application programming cycle (precompile, compile, link-edit, bind, execute) presented in "Chapter 7. Application Programming Environment."

- **Interactive execution of SQL**

  SQL statements may be submitted to DB2 in an interactive manner and the resulting output is displayed at the terminal. This facility allows the testing of a sequence of SQL statements as they will appear in an application program, before any actual coding has been done.

The TSO support of DB2 consists of a DB2 command processor (DSN), a number of TSO CLISTS, and ISPF panels to invoke the DB2 functions interactively.

THE DB2 TSO COMMAND PROCESSOR (DSN)

DB2 provides a TSO command processor as an interface between TSO terminal users and DB2. Under this command processor the user may invoke the DB2 subcommands DCLGEN, BIND, REBIND, FREE, and RUN.

The DSN command processor may also be invoked in batch as described in section "TSO Batch Work" on page 105.

ISPF SUPPORT - DB2I

The ISPF support of DB2 consists of a set of ISPF panels, from which the different DB2 functions are invoked. The functions (panels) are selectable from one main DB2I option menu, which itself is attached to the ISPF primary option menu through a user extension. The hierarchy of the most frequently used menus is shown in Figure 20.

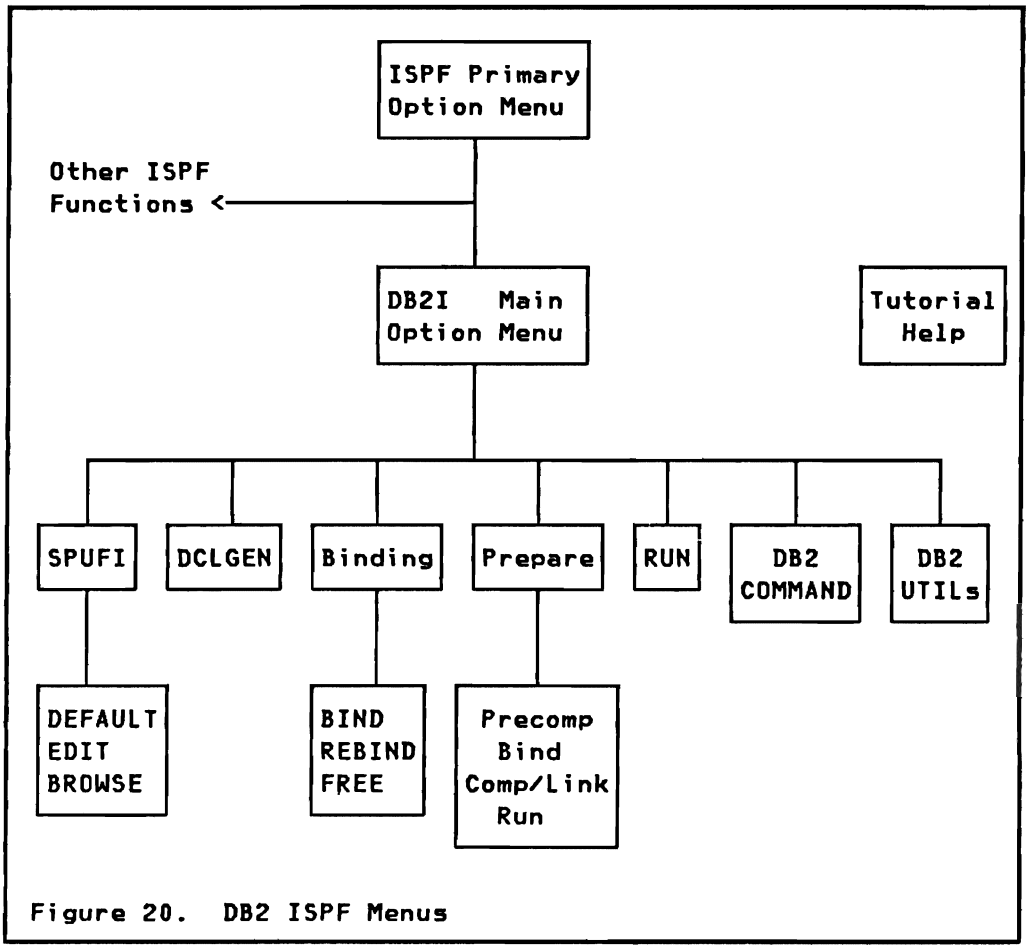


Figure 20. DB2 ISPF Menus

The DB2I Option Menu

All the functions of DB2 supported through ISPF are available from the main DB2I option menu. The layout of this menu is shown in Figure 21 on page 101.

```
DSNEPRI                          DB2I MENU

===> _

SELECT ONE OF THE FOLLOWING DB2 FUNCTIONS:

   1  SPUFI                      Process SQL statements.

   2  DCLGEN                     Generate SQL and source language declarations.

   3  BIND/REBIND/FREE           Issue BIND, REBIND, FREE for application plans.

   4  PROGRAM PREPARATION        PRECOMPILE, BIND, COMPILE, LINK, and RUN.

   5  RUN                        Run a SQL program.

   6  DB2 COMMANDS               Issue DB2 commands.

   7  UTILITIES                  Invoke DB2 utilities.

   X  EXIT                       Leave DB2I.

PRESS:  END to exit              HELP for more information
```

Figure 21. DB2I Main Option Menu

Online Help

To give DB2I users quick access to information about the system, DB2 provides ISPF tutorial information that supports application programming and data base administration tasks. Specifically, ISPF tutorial information is provided for all the tasks described in the following sections.

TSO HELP panels are also provided with the system to aid the application programmer or data base administrator who is entering commands, precompiling or binding, or using utilities without using DB2I.

SPUFI

SPUFI (SQL Processor Using File Input) is an interactive facility available under ISPF through which DB2 application programmers can submit and test SQL statements. Using SPUFI, application programmers can create and edit SQL statements they plan to include in their programs and then execute those statements and

receive the results at the terminal. SPUFI makes it easy to test the SQL parts of the programs before the programs are compiled and run (maybe even before the programs are coded).

SPUFI is also a very important tool for data base administration and maintenance. The following tasks can be performed using SPUFI:

- Execution of SQL data definition statements (DDL), such as creating a table, an index, etc.
- Retrieving information from the DB2 catalog. The catalog contains all the information about DB2 objects (tables, data bases, indexes, etc), and SQL may be used to access the catalog. SPUFI provides a convenient vehicle for entering and executing those queries. Queries can be stored and executed whenever you want.
- To control authorization to DB2 data.
- To copy and load test data from operational tables to use for testing application programs that modify data.

SPUFI Execution Flow

Execution of SQL statements from the SPUFI ISPF panel invokes the steps shown in Figure 22. All the steps are optional and are selected on the SPUFI menu.

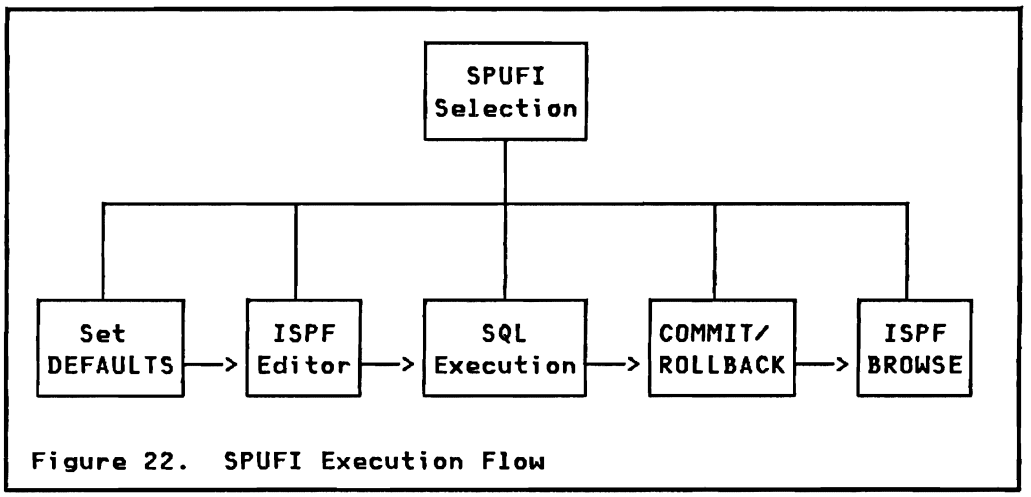


Figure 22. SPUFI Execution Flow

DEFAULTS: Setting of defaults for SQL execution. These defaults include parameters like isolation level (repeatable read or cursor stability), maximum rows returned by a SELECT, output data set characteristics (record format and length), number of lines per page, and maximum sizes for fields on the output listing. The defaults step can be skipped in most instances.

EDIT: Editing of an input data set with the ISPF editor. The input data set may be a sequential data set or a member of a partitioned data set. The name of the input data set is entered on the SPUFI menu. The user enters or modifies SQL statements to be executed. A library of input streams may be maintained and selected for execution.

SQL EXECUTION: The SQL statements of the input data set are submitted to DB2 for execution. Execution of SQL statements continues until either all input has been processed, or a SQL statement terminates with an error return code.

COMMIT/ROLLBACK: Commit or Rollback of the data base changes is either automatic or user controlled. By choosing **automatic commit** on the SPUFI menu, the user requests that all changes are committed after successful processing, or all changes are rolled back after an error has been encountered in a SQL statement.

With the user controlled commit option SPUFI will ask the user after processing of the SQL statements about his intention. The user may then select commit, rollback, or wait (decide later). Waiting with commit/rollback leaves all the changed data locked and inaccessible by other users.

BROWSE: The ISPF BROWSE program is invoked to display the output data set. The output data set contains echoed SQL input statements, results from SQL SELECT statements, and a message line indicating success or failure for each SQL statement. The SQLCA (communication area) is displayed if the statement was not successful.

If the user has chosen auto-commit, the first output screen from BROWSE is displayed after signalling END to the ISPF editor. ISPF provides an exit via the TSO attention key (PA1), which causes the end of the dialog, and in SPUFI's case a rollback of all data base changes.

Generate Declarations (DCLGEN)

An ISPF menu is provided to submit a DCLGEN command to DB2. DCLGEN will generate the declaration of a table (EXEC SQL DECLARE TABLE statement) and its corresponding host structure for COBOL or PL/I, and store these source statements in a sequential or partitioned data set. These declarations may then be included into a source program with the EXEC SQL INCLUDE statement.

DCLGEN was presented in "Chapter 7. Application Programming Environment" section "Table Declaration" on page 81.

Binding

All the functions of binding are supported through four ISPF panels. Refer to "Chapter 7. Application Programming Environment" section "Creating an Application Plan (BIND)" on page 95 for a description of the BIND process. The first panel provides selection for BIND, REBIND, or FREE.

BIND: The BIND panel allows binding of one application plan from a set of DBRMs (data base request modules). All the options for the BIND command submitted to DB2 may be specified on this menu.

REBIND: The REBIND panel allows to rebind one or more application plans. It supports all the options of the REBIND command.

FREE: The FREE panel allows to submit a FREE command to DB2. The FREE command deletes one or more application plans from the DB2 catalog.

Program Preparation

Program preparation as presented in "Chapter 7. Application Programming Environment" is supported by two ISPF panels.

The first panel invokes the DB2 Precompiler and contains all the necessary specifications like host language, input data set, DBRM output data set, include library, and output print options.

The second panel is used to invoke, in sequence, the following processes:

1. BIND, to create an application plan from the precompiler output DBRM.
2. Compiler (or Assembler), to compile the modified source program.
3. Linkage editing, to create a load module.
4. Run, to execute the application program.

All processes may be run in TSO foreground, as one batch job, or the panel may be used to build the batch JCL without submitting the job.

Execute an Application Program

The RUN panel is used to execute a user program (in TSO foreground) that contains SQL calls. It can also be used to invoke a TSO command that in turn invokes a program containing SQL calls.

DB2 Commands

The COMMAND panel is used to execute a DB2 command from a TSO terminal. DB2 commands are used to start or stop DB2 data bases, display current status, or recover a table space. Most DB2 commands may be submitted from the COMMAND panel. More details about DB2 commands will be presented in "Chapter 11. Operation and Recovery."

Utilities

The UTILITY panel allows you to submit and control DB2 utilities. Utilities may be started by preparing and submitting a job, restarted after a failure, displayed (current status), or terminated prematurely.

More about individual utilities will be presented in "Chapter 11. Operation and Recovery"(see the UTILITY panel in Figure 25 on page 135) and "Chapter 13. Monitoring and Accounting."

TSO BATCH WORK

Batch work can be run in TSO background under the TSO Terminal Monitor Program (TMP). The input stream invokes TSO command processors, in particular the DB2 command processor DSN. DSN subcommands like RUN, DCLGEN, and BIND, are then used to execute an application program, or to generate table declarations or application plans. An example of a TMP job is shown below:

```
//jobname  JOB  ...,USER=userid,...
//GO       EXEC PGM=IKJEFT01,DYNAMBR=20
//ddname    DD  DSN=....,...  user OS files
//SYSTSPRT  DD  SYSOUT=A
//SYSTSIN   DD  *
DSN
DCLGEN TABLE (EMPLOYEE) LANGUAGE(PLI)
BIND   PLAN(EMPPLAN) MEMBER(EMPPROG)
RUN    PROGRAM(EMPPROG) PLAN(EMPPLAN)
END
/*
```

The program invoked on the EXEC card is the TMP and DYNAMBR is the maximum number of dynamically allocated data sets.

Batch programs invoked through the RUN subcommand are subject to the following restrictions:

- No parameters may be passed
- No checkpoint/restart facilities are provided (no restart from a commit point)
- No capability to use SPUFI.

QUERY MANAGEMENT FACILITY (QMF)

The Query Management Facility (QMF), program number 5668-972, is a strategic end user query and report writer facility for relational data base management systems. It is designed to assist end users in the composition and execution of "queries" written in either the SQL or QBE-style languages. These queries permit the user to create, retrieve, and modify data in relational tables managed by DB2. QMF also contains the ability to produce tailored reports.

QMF provides a "full screen" interface and makes extensive use of menus. It uses the Interactive System Productivity Facility (ISPF) and GDDM. Hence, QMF can be used from an ISPF terminal but not from a CICS or IMS terminal.

QMF Objectives

QMF has been developed primarily for non-DP professionals. The main objective of QMF is to provide the terminal user with the maximum amount of function in manipulating a DB2 database, while reducing to a minimum the knowledge that the user must have of any other program. The intention is that the end user should not require any understanding of programming or computers. The user should only need to learn the languages required to create, manipulate and execute QMF queries. Additionally, QMF users are expected to have some understanding of the meaning of the data to be manipulated.

QMF Languages

The QMF user needs to know the following languages:

- the QMF command language, plus
- the Structured Query Language (SQL), and/or
- the Query By Example (QBE) style language

Use of SQL

The objective of QMF is to provide an easy to use "front end" interface to DB2 for the composition and execution of SQL queries. This interface has the following characteristics:

- allows full screen entry and editing of all DB2 interactive SQL statements:
  - Data Manipulation Statements (DML) - SELECT, UPDATE, INSERT and DELETE
  - Data Definition Statements (DDL) - CREATE, DROP, ALTER
  - Control Statements - GRANT, REVOKE, LOCK, etc.
- provides model queries which can be modified and executed
- allows the user to create and save SQL queries that contain parameters that can be varied (substituted) at execution time
- allows the user to execute the query and format the results at the terminal or on hardcopy (see "tailored reports" below)

A sample SQL query follows:

```
SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DEPARTMENT
 WHERE ADMRDEPT = 'A00'
```

Use of QBE style Language

QMF will provide a QBE style syntax to perform a subset of the SQL functions. An example of a QBE query equivalent to the SQL one above is shown below:

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
P.	P.	P.	=A00

The QBE style language will give the user the capability to retrieve, insert, update and delete data in relational tables. All of the QMF facilities provided to create, edit, execute and format SQL queries and results are also available to users of QBE style queries. QBE is a graphic language supplied by QMF to give users an alternative language to SQL.

QMF Command Language

The QMF command language is used to manipulate and manage QMF objects. QMF objects are the user's representation of SQL/QBE queries, relational tables, query results, session

characteristics, etc. QMF objects are normally stored in DB2 relational tables. During QMF operation these objects are read into a virtual storage area known conceptually as the QMF work area.

The terminal is a window through which QMF objects can be examined, altered or executed. By entering QMF commands at the terminal, the user can cause data to move from user or QMF data bases to corresponding objects in the work area and vice versa. Objects in the work area represent the latest object being used by the user or the QMF system. The objects that can be manipulated are:

QUERY A QUERY is a request in SQL or QBE describing the action to be performed on user data by DB2.

DATA The DATA object is a table that is produced by the execution of the most recent retrieval query. This table is shown to the terminal user in the form of a REPORT. The REPORT object is not separately stored but is produced by formatting the DATA object based on definitions contained in the FORM object.

FORM A FORM describes the formatting to be given to a query result before it is displayed or printed.

PROC A PROCedure is a sequence of QMF commands that are stored and executed as a group. QMF commands are discussed in more detail below.

PROFILE The user PROFILE specifies certain default parameters to be used by QMF when the user issues commands. The user can modify the profile during the session.

The user "moves the window" from one object to another by executing the QMF DISPLAY command. If the user wishes to update an existing object, the DISPLAY command is entered with the name of the object. The data base object is copied from the data base into the corresponding work area object and then the work area copy is displayed. The user can alter the object simply by typing over data to be changed or by typing new data into blank areas.

The user can request help at any time by pressing a PF key. The help panel displayed is appropriate to the work area object that the user is viewing or the command that is being used.

QMF will display a prompt panel if a QMF command is entered incorrectly. This panel contains an appropriate error message and the parameters that the user has supplied. The panel also contains reminders of what the parameters mean. The user can then correct the data in error and/or add any missing data and then enter the command for execution.

Standard ISPF scrolling facilities are available for the user to browse through multiple screen data.

The main QMF commands are:

DISPLAY This command is used to display QMF objects. It is also used to list the contents of a relational table.

RUN This command is used to initiate the execution of a SQL or QBE QUERY or QMF command PROCedure.

SAVE This command is used to save a work area object in the data base.

ERASE This command is used to delete an object from the data base.

PRINT This command is used to prepare a hardcopy listing of an object on a system printer.

IMPORT This command is used to read a TSO data set into a QUERY, FORM or PROC work area object.

EXPORT This command is used to write a QUERY, DATA, FORM or PROC work area object into a TSO data set.

Tailored Reports

As has already been discussed above, the output of a query is stored in the DATA object. The DATA object can either be displayed on the screen or sent to a printer. The format of displayed or printed data is controlled by an associated FORM object. QMF will always produce a default FORM for the user. Thus, a user only has to be concerned about FORM objects if he wishes to change the default format of the displayed or printed data.

Three types of format or report can be produced by QMF:

LISTING This report is equivalent to listing all of the fields specified in a retrieval query. For every row passed to QMF from DB2, a row of output will be produced. Variations on this report include establishing control breaks and computing summaries.

SUMMARY This report summarizes the rows passed to QMF based on specifications in the FORM object. With this type of report all the data retrieved from the data base is summarized, i.e. no detail lines are produced. Like the listing report, variations can include control breaks and summaries, i.e. summary of a summary.

ACROSS The across report is very similar to the summary report in that table data is being summarized, but the data is summarized in two dimensions. Another way of looking at this type of report is to say the data is summarized based on two groups. Like the other report types, control breaks and subtotals can be produced.

COMPARISON OF QMF WITH DB2I

QMF is provided to support end users having little or no knowledge of data processing. QMF contains an easy to use interface to formulate queries in the graphic QBE type language, or the SQL language. A data base of queries, results, etc, can be maintained, and any of these objects may subsequently be formatted and printed.

The user has full control over the formatting of results. He can maintain a library of predefined requests and formatting rules. A fixed sequence of requests may be saved as a procedure which may be executed any time.

The **ISPF support** of DB2, called **DB2I**, is designed for use by application programmers and data base administrators. It supports execution of application development tasks (precompile, bind, etc), testing SQL statements, and administrative work in regard to the definition and maintenance of DB2 data bases and the installations authorization mechanism.

Predefined files of SQL statements may be submitted for execution using the SPUFI technique. Input and output data sets are user specified. ISPF edit is used to compose the input, whereas ISPF browse is used to view the results. The user has to be familiar with the TSO/ISPF file system. The results are formatted by DB2, the user has no formatting or reporting capabilities.

Conclusion: There is very little overlap between the purpose of DB2I and QMF.

- DB2I (SPUFI) is the right tool for the data base administrator and the application programmer
- QMF is the right product for general users

The main objectives of DB2 security and authorization are

- to provide effective control over the data resources at the required level (e.g. control at field value level) and
- to allow centralization or decentralization of control at the required level, e.g. DB2 users may wish to control the use of their resources without the permission or assistance of administrative personnel.

The fulfillment of these objectives makes it possible to implement a secured system without putting a burden on the end user of DB2 data.

VIEWS AND SECURITY

The authorization mechanism of DB2 provides access control at the level of a table or view. To protect individual values within a table, a view must be defined to exclude all the columns and/or rows containing these protected values. Views, therefore, provide security at the **field value level**.

Column Subset

A view defined as a column subset is used to protect sensitive data stored in columns of a table. For example, the following view protects JOBCODE and SALARY data in the employee table:

```
CREATE VIEW EMPL1
         AS SELECT EMPNO, LASTNAME, WORKDEPT,
                   PHONENO, EDUCLVL
              FROM EMPLOYEE
```

Row Subset by Field Value

A view defined as a row subset is used to restrict access to a table based on one or multiple **field values**. For example, the following view allows access to employees of department 'D11' only:

```
CREATE VIEW EMPLD11
         AS SELECT *
              FROM EMPLOYEE
             WHERE WORKDEPT = 'D11'
```

Statistical Summary

A view may also be used to present summary data only to the user, thus protecting individual values. For example, the following view provides average salaries by department:

```
CREATE VIEW EMPLAVGSAL
        AS SELECT WORKDEPT, AVG(SALARY)
             FROM EMPLOYEE
            GROUP BY WORKDEPT
```

AUTHORIZATION

The authorization mechanism of DB2 is based on **resources** to be protected, and **capabilities** assigned to an **authorization-ID** (user). The comprehensive and flexible authorization schema does not rely on the concept of a central, all powerful "data base administrator". Each "owner" of a resource may delegate capabilities (privileges) to other users, and revoke them again at a later time.

Resources

The following list contains all the DB2 resources that can be protected against unauthorized access:

Tables, Views Access to the data in each table or view can be protected. Users may be allowed to use SELECT, DELETE, INSERT, and UPDATE statements against the table or view. With UPDATE one may even control which columns can be updated. Changing a table definition (ALTER) may also be controlled.

Data Base Creation and deletion of tables and table spaces within a data base, starting/stopping of the data base, etc., can be controlled.

Utilities The invocation of each utility can be protected by data base.

Appl. Plan Creation (BIND), replacement (REBIND), deletion (FREE), and invocation (execution) may be controlled.

DB2 Catalog The catalog is automatically protected against update through SQL statements. By default users are allowed read-only access to the catalog. An installation may further protect catalog data by providing selective views only.

Table Spaces Use of a table space may be controlled.

Storage groups Use of a storage group may be controlled.

Buffer Pools Use of a buffer pool may be controlled.

DB2 Commands Issuing of each command can be protected.

Users and Authorization-IDs

All DB2 users are identified by an **authorization-ID**. An authorization-ID can be up to 8 characters in length. A DB2 user may have more than one authorization-ID, and several users may share an authorization-ID at the same time.

Figure 23 shows the origin of authorization-IDs for the different kinds of DB2 users. DB2 assumes that the attaching subsystem has validated the authorization-ID, so there is no additional checking in DB2. DB2 does not check from which subsystem the authorization-ID is coming. The same authorization-ID may be used in different subsystems for the same or for different persons.

User Type	Authorization-ID
TSO terminal user	TSO logon-ID or sign-on exit value
Batch Job	USER parameter of job card, sign-on exit value, or installation default.
IMS	Sign-on user-id, or logical terminal name
IMS BMP (non MSG)	USER parameter of job card, or PSB name
CICS	AUTH parameter in RCT (Resource Control Table): user-ID, terminal name, transaction-ID, etc.

Figure 23. DB2 Authorization-IDs

Capabilities

The authorization of a user to access a certain resource can be done for a **single capability** or for a **group capability**.

Single Capabilities

There are different capabilities for which users may be authorized with respect to the resources mentioned earlier in this chapter.

General Capabilities: Give the user the authority for an action that is not related to a specific resource, but applies to all the resources of a certain category. A selection of general capabilities follows:

ALTERBP The attributes of any buffer pool may be altered.

BINDADD Creation of a new application plan.

CREATEDBA New data bases may be created. The user automatically gets DBADM authority (described later in section "Group Capabilities or Administrative Authorities" on page 115) for all the data bases created.

CREATEDBC New data bases may be created. The user automatically gets DBCTRL authority (described later in section "Group Capabilities or Administrative Authorities" on page 115) for all the data bases created.

Resource-specific Capabilities: Give the user the authority for an action that is related to the specified resource(s). A selection of resource-specific capabilities follows:

ALTER The definition of the specified tables may be altered.

DELETE Rows may be deleted from the specified tables or views.

INDEX Indexes may be created for the specified tables.

INSERT Rows may be inserted into the specified tables or views.

SELECT Rows may be selected from the specified tables or views.

UPDATE The values for the specified list of columns within the specified tables or views may be updated.

BIND Binding, rebinding, and freeing of an existing application plan.

EXECUTE Execution of an application plan (program) is allowed.

CREATETAB New tables may be created within the specified data bases.

CREATETS New table spaces may be created within the specified data bases.

DROP The specified data bases may be dropped. This also drops all the contained tables. An object (e.g. table) within a data base can only be dropped by the creator, or an administrative authority.

IMAGCOPY The COPY utility may be run against table spaces and indexes within the specified data sets. Similar resource specific capabilities exist for all the other utilities.

USE OF The specified objects (buffer pool, storage group, table space) may be used.

Group Capabilities or Administrative Authorities

Some of the single capabilities have been grouped together and may be assigned to special DB2 users for administrative purposes. These administrative authorities are:

SYSADM **System Administration:**

A user with this authority has **total control over any DB2 resource**. Such a user may grant to or revoke from any user the authority to access any resource. When DB2 is installed one user is the initial SYSADM. He may grant this authority to other users.

DBADM **Data Base Administration.**

This kind of data base administrator has total control over those data bases he is responsible for. For these data bases he can create tables, alter table definitions, see and modify data in the tables, and run utilities.

DBCTRL **Data Base Control.**

This kind of data base administrator has the same capabilites as DBADM, except that he is not allowed to see and modify table definitions and data in the tables. He may run read-only and updating utilities.

DBMAINT **Data Base Maintenance.**

This kind of data base administrator has the same level of control as DBCTRL, except that he is only

authorized to run read-only utilities (e.g. image copy and statistics).

SYSOPR **System Operator.**

The system operator is only authorized to issue selected DB2 operational commands. This user has no access to data bases (DB2 data).

Explicit Authorization

Authorization is explicitly **GRANT**ed to or **REVOKE**ed from a user through two SQL statements.

GRANT

The SQL GRANT statement is used to give a certain capability to a DB2 user. The format of GRANT is:

```
GRANT capability resource-list
   TO authorization-ID-list
      [WITH GRANT OPTION]
```

The **capability** is either a single or a grouped capability, as listed earlier in this chapter. Depending on the capability none, one or more **resources** can be specified. Resources may be data bases, tables, views, columns, table spaces, storage groups, buffer pools, or application plans. The authorization is granted to one or more users identified by their **authorization-IDs**. Authorization may also be granted to all the users:

```
GRANT capability resource-list
   TO PUBLIC
```

An authorization may be granted to a user **WITH GRANT OPTION**. It allows the user to pass this authority further to other users. Following are the main rules for granting authorization:

- A capability can be given to (or revoked from) a user at any time.
- Authorization for an object cannot be done before the creation of that object.
- Authorization for an object is automatically revoked (deleted from the catalog) when the object is dropped.
- To issue the GRANT statement requires an explicit or implicit authorization. Explicit authorization to issue the GRANT statement for the same capabilities and resources is obtained through: WITH GRANT OPTION. Implicit authorization is explained later in this chapter.

Examples of GRANT

- All the users are allowed to look into the employee table:

```
GRANT SELECT ON TABLE EMPLOYEE TO PUBLIC
```

- Allow an application programmer to insert rows into an existing table:

```
GRANT INSERT ON TABLE EMPLOYEE TO PROG01
```

- Two users are allowed to change definitions of two tables:

```
GRANT ALTER ON TABLE EMPLOYEE, DEPARTMENT
   TO PROG02, PROG03
```

- A user is allowed to execute a program (application plan):

```
GRANT EXECUTE ON PLAN EMPMUT TO TERM33
```

- A second system administrator is enabled:

```
GRANT SYSADM TO SECADM
```

- A data base administrator is given control over the personnel data base, and he is allowed to grant authority to other users:

```
GRANT DBADM ON DATABASE PERSONNEL
   TO ADMIN2 WITH GRANT OPTION
```

REVOKE

The SQL REVOKE statement is used to take away a certain capability from a DB2 user. The format of REVOKE is:

```
REVOKE capability resource-list
  FROM authorization-ID-list
```

Following are the main rules for the REVOKE statement:

- A REVOKE may only be issued by the grantor (who GRANTed the capability earlier), or by a SYSADM user.

- There is a **cascading effect** for the REVOKE statement. If a capability is granted WITH GRANT OPTION from one user to the next in a sequence, then a REVOKE of the same capability from the first user results in REVOKE's for all dependent users.

```
User0:  GRANT SELECT ON  TABLE EMPLOYEE
           TO USER1 WITH GRANT OPTION
User1:  GRANT SELECT ON  TABLE EMPLOYEE
           TO USER2 WITH GRANT OPTION
User2:  GRANT SELECT ON  TABLE EMPLOYEE
           TO USER3 WITH GRANT OPTION

User0: REVOKE SELECT ON  TABLE EMPLOYEE FROM USER1
       (revokes all the above authorizations)
```

- Only a capability that has been granted can be revoked. For example, if DBCTRL has been granted, you cannot revoke the IMAGCOPY capability individually.

Implicit Authorization

There are three user categories which are implicitly authorized, i.e. authorized without GRANT/REVOKE statements.

The **creator of an object** is automatically given full authority WITH GRANT OPTION on that object. This access to the object cannot be revoked, unless the object itself is deleted.

Every **SYSADM** administrative authority has by default full access to any DB2 resource. He may also DROP (delete) the resource, but he cannot revoke the authority to access the resource from the creator.

A **DBADM** user has implicit access to all the tables created by other users in the data bases he is controlling. He cannot grant, however, capabilities on these tables to other users.

Operation - what is checked when ?

There are different events when authority is checked. The following section describes the events that are of importance for authority checking.

Pre-compile: Since DB2 is not required at precompile time, no authority checking is done at that time.

DCLGEN: SELECT authority is required for all tables whose declarations are to be generated.

ALTER or DROP of definition: When a table, view, or index definition is changed or dropped, all application plans dependent on it will be marked invalid. Automatic bind is called the next time the application is used.

BIND (TSO subcommand): To issue a BIND command the user must be authorized with the BINDADD (new plan) or BIND (existing plan) capability. The binder must have the proper authority for all the

tables and views used in this application plan. Dependent on the VALIDATE option of the BIND command (section "BIND Command" on page 96 in "Chapter 7. Application Programming Environment") some authorization checking may be delayed to execution time.

Automatic bind: The latest binder (user who issued BIND the last time) is checked again for authorization on used tables and views.

Dynamic bind: Is called for dynamic SQL statements. The user executing these SQL statements must be properly authorized to access the tables and views referenced.

Execution time: The user who executes the program (plan) must have EXECUTE authority on that application plan. In addition it is possible that authorization must be checked for the latest binder of the plan if that authority was missing at BIND time.

DATA SET PROTECTION

The DB2 security and authorization mechanism is of value only if the active data base and the DB2 system data sets are properly protected. This may be accomplished by VSAM password protection and/or RACF.

VSAM Password Protection

Active DB2 data base data sets may be protected by a VSAM password.

User defined VSAM data sets are password protected through the VSAM DEFINE statement. The MASTER password must then be specified in the CREATE TABLESPACE or CREATE INDEX statement.

```
CREATE TABLESPACE space-name
       etc.
       DSETPASS   dataset-password
```

Data sets defined using DB2 **storage groups** are password protected by specifying a password (DSETPASS) in the CREATE statement as shown above.

RACF

In addition to or in place of VSAM password protection, RACF may be used to limit access to DB2 data sets. All the DB2 system data sets and libraries (e.g. INCLUDE libraries) may be RACF protected.

Individual users (authorization-IDs) must be given RACF ALTER access to **user defined VSAM data sets** prior to creating the table space or index.

Data sets in **storage groups** may be automatically RACF protected by assigning the RACF ADSP (Automatic Data Set Protection) attribute to DB2 authorization-IDs.

The DB2 system catalog contains data about all the data DB2 manages. This includes:

- Descriptors for all the DB2 objects defined and used by the Data Definition Language (DDL)
- Application plans created by the BIND process
- Authorization privileges granted to DB2 users
- Data set names and volume-ID's of image copy data sets

Another portion of system data managed by DB2 is stored in the **Directory**. This part of DB2s system data is not accessible by the user, and is therefore not presented here.

CATALOG STRUCTURE

The system catalog is stored as a set of DB2 tables. These tables of the catalog are like any other tables in DB2. You can use SQL SELECT statements to look at data in the catalog tables the same way you use them to retrieve data from any other table in the system. Examples of catalog usage through SQL statements are presented later in this chapter.

The catalog cannot be updated using the SQL data manipulation language. The catalog is maintained through the SQL data definition language (DDL), authorization language (GRANT and REVOKE), and the BIND process.

The next sections describe the different types of data stored in DB2 catalog tables. Some tables will be presented conceptually, but not all the catalog tables will be mentioned, and none of them will be explained in detail.

Data Definition

Descriptors of all the DB2 objects defined through the SQL data definition language are stored in catalog tables. To illustrate this facility a small selection of the catalog tables used to describe DB2 objects is shown below:

SYSTABLES The SYSTABLES table contains one row for every table or view defined in the system. Some of the fields of SYSTABLES are the NAME of the table (or view), the CREATOR (user-ID), the TYPE (table or view), the DBNAME (data base name), the TSNAME (table space

name), and the COLCOUNT (number of columns in the defined table).

Additional **statistical information** like the number of data rows, the number of pages holding rows of this table, and the percentage of the total number of pages of the table space holding rows of this table, can be stored in SYSTABLES (using the **RUNSTATS** utility) after data has actually been loaded.

An additional field called REMARKS holds text information associated with the table through the "COMMENT ON Statement" on page 56.

SYSCOLUMNS The SYSCOLUMNS table holds one row for every column of each table that has been created or defined. Some of the fields of SYSCOLUMNS are the NAME of the defined column, the TBNAME (name of the table which contains this column), the COLTYPE (data type of defined column), and the LENGTH of the column.

Other catalog tables hold information about buffer pools, DB2 data bases, storage groups, table spaces, indexes and synonyms.

SQL SELECT statements may be used to retrieve information from catalog tables. The following examples retrieves the definition of all the columns of the employee table (SYSIBM is the creator of catalog tables):

```
SELECT NAME, COLTYPE, LENGTH
  FROM SYSIBM.SYSCOLUMNS
 WHERE TBNAME = 'EMPLOYEE'
```

Application Plans

The BIND process presented in "Chapter 7. Application Programming Environment" section "Creating an Application Plan (BIND)" on page 95 creates entries in several catalog tables:

SYSDBRM The SYSDBRM table contains information about the programs which constitute an application plan; there is one entry for each program. It includes the name of the DBRM, the time and date of the precompile run, the host language, and the name of the application plan of which this DBRM is a part. The SQL text of the DBRM is stored in a separate catalog table SYSSTMT.

SYSPLAN The SYSPLAN table contains one row for every application plan. It contains the NAME of the plan, the CREATOR (user-ID), the BINDDATE (date of last bind operation of this plan), the VALIDATE and ISOLATION options of the BIND command, and a flag

indicating if the plan is valid or must be rebound dynamically.

SYSUSAGE The SYSUSAGE table contains information about dependencies that application plans have on storage objects and views. For example an application is dependent on an index of a certain table.

The catalog tables holding information about application plans may be used, for example, to find which plans need a REBIND after an index is deleted, or for which plans a REBIND might be worthwile after a table space has been reorganized.

- Find all application plans (DNAME is the dependent object name), dependent on the index (BNAME is the base object) before deleting the index:

```
SELECT DNAME
  FROM SYSIBM.SYSUSAGE
 WHERE BNAME    = 'index-name'
   AND BCREATOR = 'auth-id'
```

- Find invalid application plans (VALID equal N) after deleting the index:

```
SELECT NAME
  FROM SYSIBM.SYSPLAN
 WHERE VALID = 'N'
```

Another example of catalog usage is to find all the application plans which use a certain table. This is useful when the table has been reorganized, or when an index has been added, and a REBIND might find better access paths to the data:

```
SELECT DNAME
  FROM SYSIBM.SYSUSAGE
 WHERE BNAME    = 'table-name'
   AND BCREATOR = 'auth-id'
   AND DTYPE    = 'P'
```

Authorization Definition

A number of catalog tables hold information about authorization granted from one user to another.

SYSTABAUTH The SYSTABAUTH table records information about privileges held by users over tables and views. Each row contains the GRANTOR and the GRANTEE, the name of the table or view, the timestamp when the capability was granted, and the functions which may be performed (SELECT, INSERT, etc.).

SYSPLANAUTH The SYSPLANAUTH table records the DB2 users holding authority to BIND or EXECUTE an application plan.

Other tables record the authorization on DB2 data bases, individual table column update, general capabilities, and usage of buffer pool, storage groups, and table spaces.

SQL SELECT statements may be used to find all the users having access to a certain resource. The following SELECT retrieves all the users who may update the employee table:

```
SELECT DISTINCT GRANTEE
  FROM SYSIBM.SYSTABAUTH
 WHERE (DELETEAUTH = 'Y' OR DELETEAUTH = 'G' OR
        INSERTAUTH = 'Y' OR INSERTAUTH = 'G' OR
        UPDATEAUTH = 'Y' OR UPDATEAUTH = 'G')
   AND TTNAME   = 'TEMPL'
   AND TCREATOR = 'DSN8'
```

Image Copy Data Sets

The **SYSCOPY** table contains records showing the latest image copy taken for each table space in every data base. Information maintained in SYSCOPY includes the table space name, the run date, and all the information where the image copy data set is stored.

This information is used by the recovery utility to automatically allocate the latest image copy when recovery of a DB2 data base data set is necessary.

The information may be used by a data base administrator to retrieve the inventory of image copies of a table space:

```
SELECT ICDATE, DEVTYPE, DSNAME, DSVOLSER
  FROM SYSIBM.SYSCOPY
 WHERE TSNAME = 'table-space-name'
   AND DBNAME = 'data-base-name'
 ORDER BY DSNAME, ICDATE
```

USING THE DB2 CATALOG

In the previous sections some small examples were presented to illustrate the use of SQL SELECT statements on catalog tables. However, since some of the catalog tables contain sensitive information, the authority to issue SELECT statements should not be granted to every user on all the catalog tables.

To allow a more selective access of catalog data, an installation may define a number of views restricting users to data they are allowed to see. For example:

- A view TABAUTH defined on the SYSTABAUTH table selects rows which describe authorizations the user has been granted by others, or has granted to others. He may not see authorization records where both the grantor and the grantee are other users.

- A view DBAUTH defined on the SYSDBAUTH table (authorization on data bases) selects rows on authorization granted to the user by others, or granted by the user to others.

The DB2 catalog may therefore be used as a tool for the Data Base Administrator (DBA) to manage all the DB2 objects, the application plans, and the user authorizations. An installation should carefully control the access to the DB2 catalog using the DB2 Authorization mechanism.

In this chapter we look at the operational environment of DB2. This includes:

- Commands to control and operate a DB2 system and its data bases
- The logging function which provides a base for system and data base recovery
- DB2 operational utilities for reorganization, backup and recovery

OPERATION OF DB2

DB2 must be defined as a formal MVS subsystem in order that it may use the services provided by the MVS Sub-systems Interface (SSI). This is done at DB2 installation time by creating an entry in the SYS1.PARMLIB member IEFSSNxx. This entry is used by MVS to

- activate a DB2 "early processing module" when the master scheduler is initialized, and to
- define the subsystem recognition character (SRC) which identifies DB2 commands issued at an MVS console

The DB2 "early processing module" is activated at master scheduler initialization time. It waits for a DB2 START command to startup the whole DB2 system.

DB2 commands are identified by a '-' preceding the command verb. This is the subsystem recognition character (SRC) assigned to DB2. When running multiple DB2 systems in one MVS system, each DB2 system must have its own SRC.

Command and Message Support

DB2 commands may be issued from an MVS console, a TSO terminal, or an authorized IMS or CICS terminal. The IMS and CICS facilities will be presented in special chapters devoted entirely to the IMS and CICS environments.

Users of TSO terminals may enter DB2 commands either through the DB2 command processor (DSN), or through the ISPF menu presented in "Chapter 8. TSO Environment." RACF may be used to ensure that a TSO user is authorized to connect to DB2.

Responses to DB2 commands are sent back to the terminal (or console) issuing the command. DB2 unsolicited system messages

are sent to the MVS console only (or secondary consoles defined at DB2 installation time). DB2 messages are prefixed by DSNcxxxt, where "c" is the DB2 component, "xxx" the message number, and "t" the message type (A, E, or I).

Startup and Shutdown

DB2 is started from an MVS console through the START command:

```
-START DB2
```

The start command is processed by the "early processing module" of DB2, which will interface with JES to start the DB2 address spaces. More details on the DB2 address spaces is presented in "Chapter 12. Architecture."

There are no parameters on the start command to indicate normal or emergency (re-)start. The necessary restart processing is determined by DB2. Transactions (application programs) which were active at the last DB2 shutdown are automatically recovered.

DB2 is stopped from an MVS console, or any authorized terminal attached to DB2, through the STOP command.

```
-STOP DB2                    or

-STOP DB2 MODE (QUIESCE)
```

In this normal mode of stopping the subsystem all current work is allowed to finish. No new transactions (application programs) are allowed to start processing.

DB2 may be forced to shutdown fast through the FORCE mode of the stop command:

```
-STOP DB2 MODE (FORCE)
```

In this mode all current work is aborted at the first opportunity. Active work is rolled back, and no new transactions are allowed to start.

Immediate termination of DB2 is available through the MVS cancel command. Active work is interrupted immediately and rolled back at the next DB2 startup.

Controlling Data Bases

DB2 data bases, table spaces and index spaces are controlled through DB2 start, stop, and display commands. The unit of these commands is normally a DB2 data base, but it is also possible to act on individual table space or index spaces.

Starting a Data Base

A DB2 data base is made available for processing through the start command:

```
-START DB    (db-name-list)
       ACC   (RW | RO | UT)
       SPACE (table-space-list, index-space-list)
```

DB or DATABASE: Specifies one or more DB2 data bases to be started. All data bases may be started as DB (*).

ACCess: Specifies the level of access allowed. The default access is read-write (RW). The access may be restricted to read-only (RO), or utilities only (UT).

SPACE: This optional parameter allows to restrict the start command to individual table spaces and/or index spaces of one data base named with the DB keyword.

Stopping a Data Base

A DB2 data base is made unavailable for processing through the stop command.

```
-STOP  DB    (db-name-list)
       SPACE (table-space-list, index-space-list)
```

One or more DB2 data bases, or individual table spaces and/or index spaces, may be stopped.

Displaying the Status of a Data Base

The status of a DB2 data base, or the status of individual table/index spaces, may be displayed through the DISPLAY command:

```
-DISPLAY DB    (db-name-list)
         SPACE (table-space-list, index-space-list)
         ACTIVE
         RESTRICT
```

The optional keywords ACTIVE and RESTRICT allow the user to display information about data bases allocated to applications (ACTIVE), or data bases which are in a status that restricts their

use (RESTRICT). An example of a restricted data base is a data base started for read-only processing. The default is display information on ACTIVE and RESTRICTed data bases.

THE DB2 LOG

The DB2 Log contains all the necessary information for data base recovery, application recovery, and system recovery. The log does not contain records for accounting, traces, and performance evaluation.

Log records fall into different categories. Some of these are:

- Data set control, recording open and close of DB2 data sets.
- Transaction related records, marking the beginning and end of processing, as well as "before" and "after" update images of table records.
- DB2 system checkpoint records.

Log Operation

The log is implemented as a three level hierarchy as shown in Figure 24 on page 131.

Log records are placed sequentially into **log buffers**. Log buffers are formatted as VSAM CI's of 4K bytes. Log records longer than 256 bytes may span into the next CI. Each log record is assigned a continuously increasing relative byte address (RBA) in a linear address range from 1 to 2**48 (log RBA's are 6 bytes long). Log RBA's are therefore not RBA's of a physical VSAM data set.

Active Log Data Sets

Log buffers (CI's) are written to a set of predefined DASD **active log data sets** which are used on a cyclic basis. Active log data sets are VSAM ESDS's, and 2 to 53 may be defined. Optionally a second set of VSAM ESDS may be defined to support a dual copy of the active log.

Log buffers are written to the active log when

- all log buffers are full
- a user specified number of log buffers is full (1-256)
- the log buffer is forced by the **log write ahead** facility

The log write ahead facility enforces that recoverable information is written to the log data set before the associated

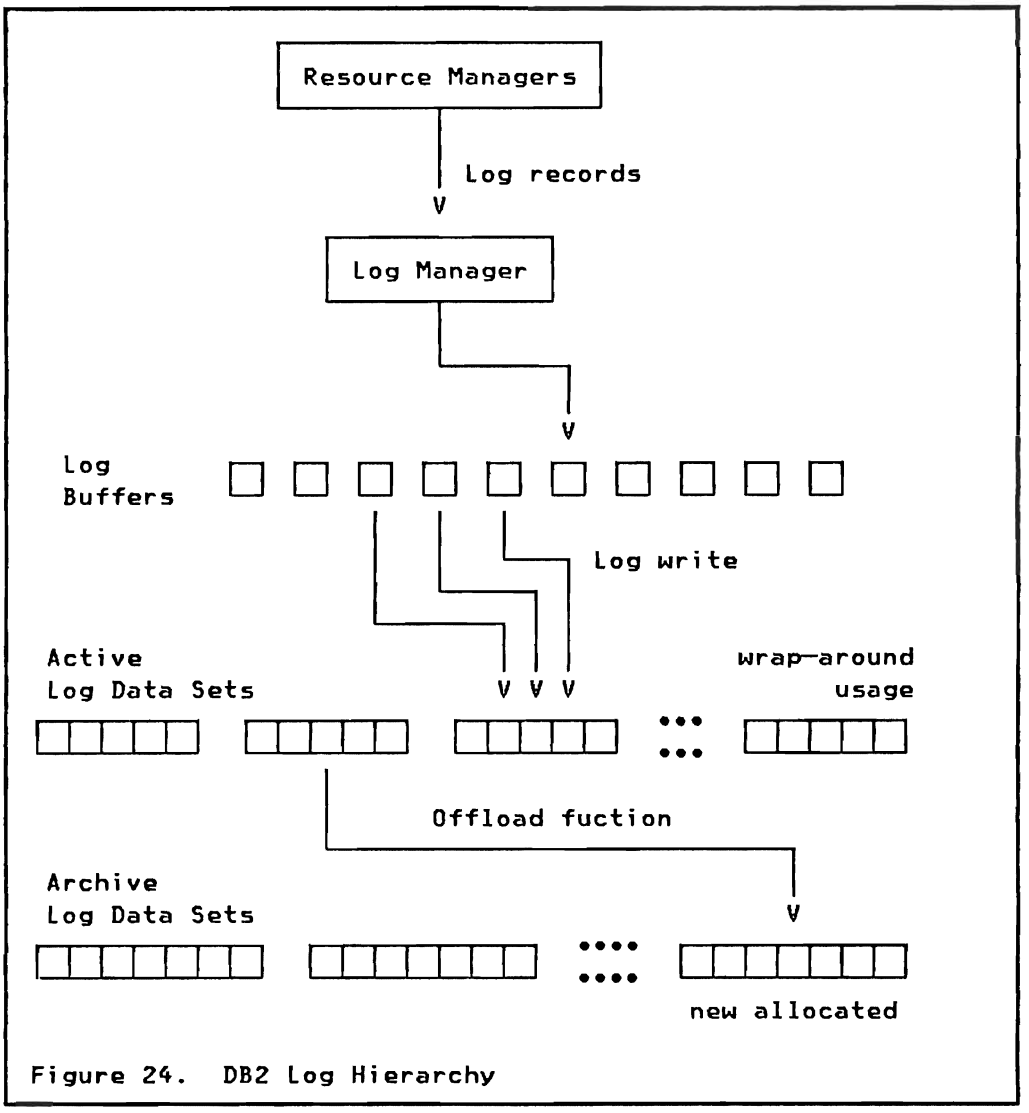


Figure 24. DB2 Log Hierarchy

action is performed (e.g. a table page change is logged before the page is written back to the table space).

Archive Log Data Sets

Whenever an active log data set is full, its CI's are **offloaded** to a new **archive log data set**. Archive log data sets are sequential data sets (BSAM/QSAM), which are dynamically allocated and optionally cataloged by DB2 during the offload operation. The blocksize of archive data sets is a multiple of 4K in the range of 8K to 28K. Each record in the archive data set corresponds to a VSAM CI of an active log data set.

Up to 1000 archive log data sets may be controlled by DB2. Dual mode is also available for archive data sets and DB2 will maintain two sets of archive logs.

Boot Strap Data Set

The DB2 Boot Strap Data Set (BSDS) is a VSAM key sequenced data set that contains an inventory of all log data sets.

- All the available active log data sets are recorded in the BSDS at installation time. Data sets may be added and removed when DB2 is not running.

- Each newly allocated archive log data set is reflected in the BSDS.

The BSDS therefore maps each log RBA into an active log data set and/or into an archive data set. Using the BSDS the log manager may dynamically allocate a required log data set to retrieve any log record with a given log RBA.

A dual option is supported for the BSDS, and all changes are made to both data sets. An error on one BSDS disables dual mode and the erroneous data set is unallocated. The installation may then redefine the VSAM data set using access method services (AMS), and tell DB2 to reestablish dual mode through the -RECOVER BSDS command. The new data set is dynamically allocated and all the information copied from the good BSDS.

Whenever an archive log is created by the log offload function, a copy of the BSDS is created as well. This may be used in case of an error with single BSDS or when both BSDS are damaged.

Log Processing Options

An installation has the following control over the DB2 logging function:

- **Number of active logs**

  The number of active log data sets may be defined by the installation in the range between 2 and 53 inclusive.

- **Dual active log**

  Dual copies of active log data sets may be handled by DB2. In case of an error on one set of active logs DB2 will continue to run with a single active log. The two copies are not synchronized. A data set full condition on either copy will trigger the offload function, and a range of log RBA's will be offloaded to an archive data set. Log records from both copies may be used to offload the RBA range.

- **Dual BSDS**

  The BSDS may be maintained in dual mode. DB2 provides a command to reinstate dual BSDS's after an error on one copy.

- **Log write control**

  The interval between writes to the active log is controlled through a threshold number of buffers which must be full.

- **Number of log buffers**

  The installation may specify the amount of output log buffers (32K - 4000K), and the amount of input log buffers (28K - 60K). Input log buffers are used for the offload process and for data base recovery.

- **Dual archive log**

  Dual copies of archive logs may be written by the offload function. This dual mode is independent from dual active logs. The records contained on a pair of archive data sets are identical. Archive logs may be automatically cataloged.

- **Archive log attributes**

  The unit type, blocksize, and space allocation parameters for archive logs, as well as the maximum number of archive logs to be recorded in the BSDS before wrap-around occurs, are all user specifiable.

Log Utilities

Two DB2 utilities are available to manage the recording of log data set in the BSDS.

Print Log Map Utility

The Print Log Map Utility lists the log inventory information stored in the BSDS. For both copies of every active and archive log data set the log RBA range which is stored on that data set is printed.

The utility may be run when DB2 is up or when DB2 is stopped. The BSDS is accessed through JCL specification.

Change Log Inventory Utility

The Change Log Inventory Utility is an offline utility which may only be run when DB2 is stopped. Utility statements are provided to:

- Define a new active log data set for DB2 usage

- Delete an active log data set which had an error

- Replace an active log data set of a given RBA range
- Define an archive log data set of a given RBA range
- Delete an archive log data set

The utility is used to reflect normal changes to the BSDS like adding a new active log data set, or as a BSDS recovery tool in case the only BSDS, or both dual BSDS, are inaccessible.

DB2 DATA BASE UTILITIES

A set of utilities is provided to support the DB2 data base environment. The following design consideration were taken into account:

- **Improved availability of data**

  All utilities are executed when DB2 is running. Depending on the utility function, data remains available to applications for update or for read-only access. Some utilities allow the invoker to tradeoff availability with utility performance. If utility execution time is to be minimized, the utility will make the entire data object unavailable, where as if availability is critical the utility will make only small amounts of data unavailable at any one time.

- **Improved usability**

  Logical functions may be invoked via one utility, instead of having JCL for a number of steps (e.g. reorganization). The utility itself will determine the steps and sequence necessary to perform the desired function. DB2 incorporates a mechanism in the catalog to keep track of backup copies.

  All utilities have a builtin checkpoint/restart function. Checkpoints are taken frequently during execution and restart may be performed in case of a utility or system failure. The checkpoint feature is implemented in DB2, no OS checkpoint is involved.

DB2 utilities use a special low-level interface to get DB2 data. This allows for fast sequential processing of table data.

Invocation and Control

DB2 utilities are invoked via batch jobs. The simplest way to run a utility job is by using the appropriate TSO command procedure (CLIST) provided by DB2. You invoke the CLIST using the DB2I interactive menu facility, or directly under TSO. The DB2 provided menu is shown in Figure 25 on page 135.

```
                              DB2 UTILITIES
===>

SPECIFY THE FOLLOWING TO EXECUTE A DB2 UTILITY:
 1  SPECIFY UTILITY   ===>          Enter code letter of desired utility
                                   a REORG      d MERGECOPY   g RUNSTATS
                                   b LOAD       e RECOVER     h STOSPACE
                                   c COPY       f REPAIR
 2  CONTROL CARDS DS  ===>
 3  EDIT JCL FIRST .. ===>          Enter YES to EDIT JCL before submitting
 4  SUBMIT JCL ..... ===>          Enter YES to submit JCL for execution
 5  UTILITY-ID ..... ===>          Enter a unique identifier string
 6  DB2 NAME ........ ===>          Enter DB2 subsystem name

ENTER NAME OF DATASET FOR COPY, MERGECOPY, LOAD, OR REORG:
 7  RECDSN OR COPYDSN ===>

SPECIFY OPTION FOR RESTARTING UTILITY:
 8  AFTER LAST COMMIT ===>          Enter YES to start after last commit point
 9  AT START OF PHASE ===>          Enter YES for beginning of next phase
ENTER QUALIFIER TO DISPLAY OR TERMINATE A UTILITY:
10  DISPLAY ......... ===>          Enter UTILITY ID string or *.
11  TERMINATE ....... ===>          Unter UTILITY ID string or *.

PRESS:  ENTER to process  END to exit  HELP for more information
```

Figure 25. DB2I Utilities Menu

To submit a job for a DB2 utility via the CLIST you:

- Define the utility control statements in a standard ISPF data set.
- Invoke the CLIST, supplying the name of the input data set as one of the parameters.

The job will be built, optionally edited by the invoker, and submitted for execution.

Utility Job Status

You can query the status of utility jobs by issuing the DB2 DISPLAY UTILITY command, or using the ISPF utility control menu:

```
-DISPLAY UTILITY (utility-ID)
```

The utility-ID is the identification which was supplied in the utilities menu when the utility job was started.

Forced Utility Termination

A long running utility may be forced to terminate through the TERM UTILITY command, or using the ISPF utility control menu:

```
-TERM UTILITY (utility-ID)
```

This command may be used by the user who submitted the utility job, and by a SYSADM or SYSOPR user.

Reorganization

The reorganization utility is used to reorganize a table space, an index, or a partition of a partitioned table or index space. Reorganization operates in up to four phases:

UNLOAD The table space (or index, or partition) is read and written to a sequential data set. Unloaded data is in clustering index sequence for a simple table space with one table only, or in physical sequence otherwise.

RELOAD The unloaded data is read from the sequential data set and loaded into the table space. Work records are generated for indexes defined on tables within the table space.

SORT If there are indexes this phase will sort the index work records before updating the indexes.

BUILD After the sort phase the indexes are updated to reflect the new location of records.

An image copy must be taken after reorganization before any updates are allowed on the table space. Physical space of dropped tables is reclaimed by reorganization.

The reorganization control statement to reorganize a table space is:

```
REORG TABLESPACE table-space-name
      PART       partition-number
      UNLDDN     SYSREC | ddname
      WORKDDN    SYSUT1 | ddname
      SORTDEVT   device-type
      SORTNUM    numer-of-sort-ds
      UNLOAD     CONTINUE | PAUSE | ONLY
```

The reorganization control statement to reorganize an index is:

```
REORG INDEX      index-name
      ..etc.
```

TABLESPACE or INDEX, PART: Identifies the space or partition to be reorganized.

UNLDDN and WORKDDN: Define the "ddnames" for the sequential output files for table data and index data.

SORTDEVT and SORTNUM: Define the number and device type of sort work data sets.

UNLOAD: Specifies actions to take place after the data has been unloaded. The default CONTINUE will immediately reload the table data. PAUSE will cause the utility to end after unload. User defined VSAM data sets may now be redefined (e.g. to increase space, change the volume). The utility job may then be resubmitted with the RESTART option to reload the data. ONLY will cause the utility to end after unload and remove the status of the utility completely. The table definition may now be changed with the ALTER TABLE statement to change the name of the validation procedure. The data may then be reloaded using the LOAD utility (See "Load Utility" on page 71 in "Chapter 6. Data Management") with the 'FORMAT UNLOAD' option. This is a way to enforce validation of existing data.

Data Base Backup and Recovery

The DB2 Backup and Recovery concept allows to recover a table space, partition, or a group of pages. It is based on:

- **Logging:** All the changes to data pages are logged on the DB2 system log.
- **Image copies:** Two utilities are provided to maintain image copies of table spaces.
- The **Recovery utility** may be used to recreate a table space from an image copy and log records.

The cycle and data flow of the data base recovery system is shown in Figure 26 on page 138.

Index spaces are not recovered using image copies and log records. They are re-created using the CREATE INDEX function, either explicitly by the user, or by the index recovery function of DB2.

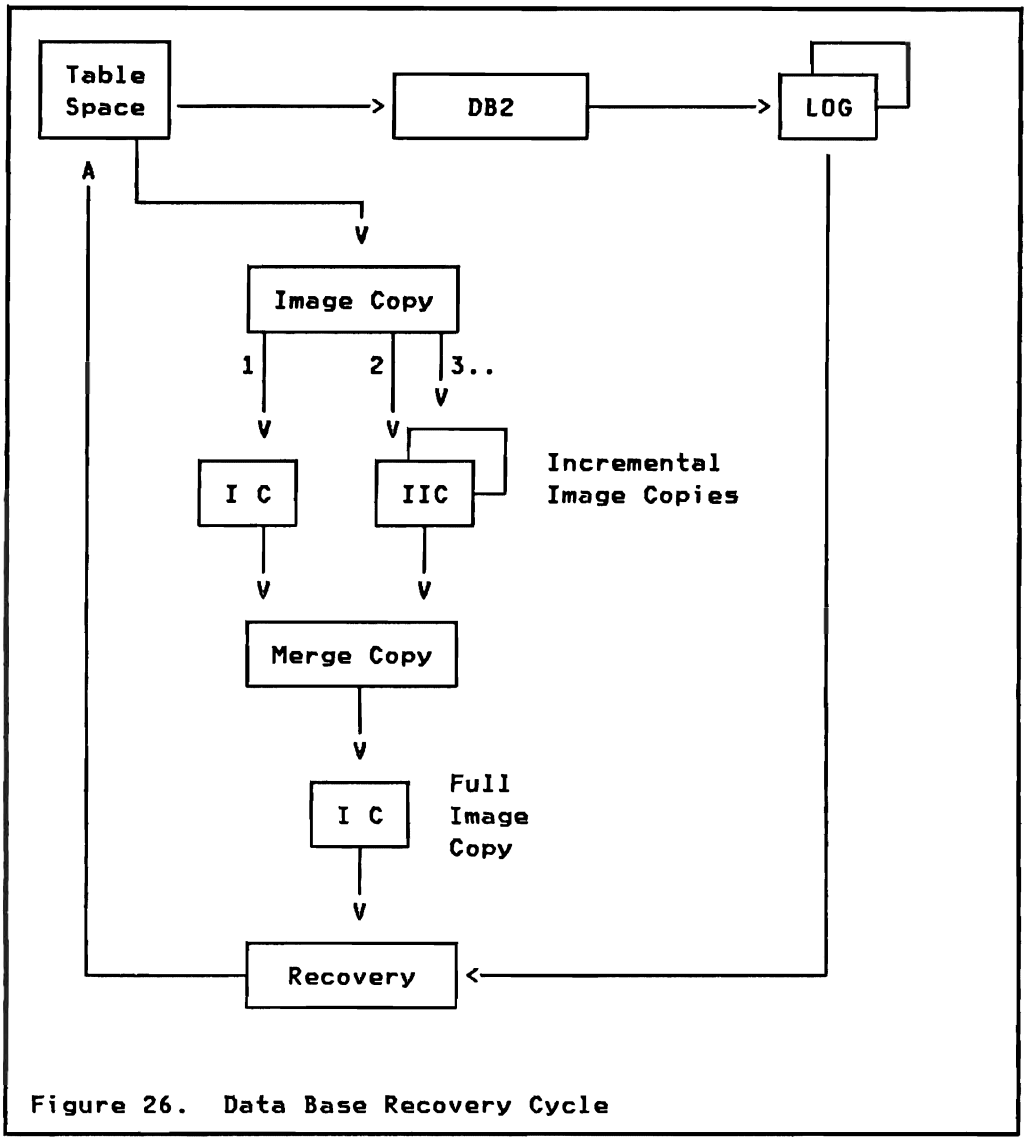


Figure 26. Data Base Recovery Cycle

Image Copy

The image copy utility makes a page image copy of a DB2 table space (or data set of a table space) to a sequential data set. The table space remains available (read-only or update) to applications during that time. Allowing read-only access during the copy will provide best performance. The sequential output data set (image copy) is recorded in the catalog table SYSCOPY.

The image copy utility statement is:

```
COPY TABLESPACE table-space-name
     DEVT       image-copy-device-type
     DSNUM      ALL | data-set-number
     COPYDDN    SYSCOPY | ddname
     SHRLEVEL   REFERENCE | CHANGE
```

TABLESPACE: Identifies the table space to be image copied.

DSNUM: Used to copy one data set only. This is supported for partitioned table spaces (fixed number of data sets), as well as for simple table spaces (number of data sets may increase).

COPYDDN and DEVT: Specify the "ddname" and device type of the image copy data set.

SHRLEVEL: Specifies if read-only (REFERENCE) or update (CHANGE) applications are allowed during the image copy time.

Incremental Image Copy

A significant option of the image copy utility is the **incremental image copy**. Incremental image copy allows the user to copy only those data pages which have been changed since the last image copy (full or incremental). In most cases, only a limited number of pages will have been changed since the last incremental image copy, and therefore only a small percentage of the entire table space will have to be copied. Incremental image copies are also recorded in the catalog table SYSCOPY.

The utility statement for incremental image copy has the FULL NO option added:

```
COPY TABLESPACE table-space-name
     FULL       NO
     ...etc.
```

Merge Copy

The merge copy utility allows the user to merge multiple incremental image copies together into one incremental image copy, or to merge all incremental image copies with the latest image copy into a new full image copy. The usage of merge copy is optional, however, it may provide the basis for faster recovery.

The utility statement for merge copy is:

```
MERGECOPY TABLESPACE table-space-name
          DEVT       device-type
          DSNUM      ALL | data-set-number
          COPYDDN    SYSCOPY | ddname
          WORKDD     ddname
          NEWCOPY    NO  | YES
          DELETE     YES | NO
```

NEWCOPY: Specifies if a full new image copy is to be created (YES), or if only incremental image copies are merged into a new incremental image copy.

DELETE: Specifies if the old full image copy record in the catalog may be deleted after a new full image copy has been created (applies to NEWCOPY YES only). If NO is specified then image copy information prior to the old full image copy is deleted only. Merged incremental image copy records in the catalog are always replaced by the new information.

WORKDD: A work data set is needed if many incremental image copies must be merged. If omitted only some incremental image copies may have been merged and another execution of the merge copy utility is required.

Recovery

The recovery utility is used to recover a damaged table space, a partition or data set of a table space, or a range of pages (tracks) of a table space.

Recovery consists of up to three distinct phases. They are:

ICREST The latest full image copy is restored. If the latest full image copy is unreadable recovery will use the previous image copy.

ICMERG If incremental image copies are outstanding they are applied next. The user does not have to run the merge copy utility before starting recovery.

LOGAPPLY The last phase applies all outstanding log changes. The log manager will retrieve the required change log records from current log buffers, active log data sets, and if required even from archive log data sets.

Selective data recovery is used to replace erroneous tracks of a table space data set. DB2 keeps track of the range of damage in case of I/O errors on data pages. DB2 then recovers these tracks by recovering the pages into alternate tracks. This option provides the fastest recovery possible.

The utility statement for recovery is:

```
RECOVER TABLESPACE table-space-name
        DSNUM      ALL | data-set-number
        [ERROR RANGE]
```

TABLESPACE and DSNUM: Specify the data set(s) to be recovered.

ERROR RANGE: Optionally specifies that erroneous tracks are to be recovered only using alternate track assignment.

Index Recovery

There is no image copy - recovery mechanism for indexes. Indexes may be recovered in two ways:

1. DROP the damaged index, and use the CREATE INDEX function to recreate the index. Note that all the application plans which are dependent on that index will be automatically rebound at their next usage.

2. Use the RECOVER INDEX command which will internally invoke the INDEX build function:

```
RECOVER INDEX index-name
```

Repair Utility

The REPAIR utility allows the user to repair data in pages, whether it is user data (table data) or DB2 control data. Data may be examined by clustering index key, RID (record identifier), or page number/offset, and discrepancies may be repaired.

The whole utility execution is one "unit of work". At the end of execution either all the work is committed (if no errors occurred), or all the work is rolled back.

The utility statement for the repair utility is:

```
REPAIR
```

Additional control statements are used to LOCATE data, VERIFY existing values, DELETE a record (row), REPLACE data with new values, and DUMP data in printable format.

These statements are shown in the format of an example and the detailed options are not explained.

```
LOCATE  TABLESPACE table-space-name
        PAGE   X'xxxxxx'       to access whole page
        RID    X'xxxxxxxx'     to access a record (row)
        KEY    data            to access a record (row)

VERIFY  OFFSET offset
        DATA   X'xxxx..'

DELETE

REPLACE OFFSET offset
        DATA   X'xxxx..'

DUMP    OFFSET offset
        LENGTH length
```

Catalog Usage

The backup and recovery utilities use the DB2 catalog to record information about image copies, incremental image copies, and ranges of inaccessible pages.

SQL SELECT statements may be used by authorized users to retrieve data from these catalog tables.

CATALOG MAINTENANCE

As presented in "Chapter 10. The DB2 Catalog," the catalog consists of a number of tables. These tables may be accessed by SQL SELECT statements, but they cannot be modified by other SQL data manipulation statements.

It is therefore very crucial to provide a backup and recovery mechanism for the catalog itself.

Catalog Backup and Recovery

The catalog tables must be backed up by the image copy utility as any other tables. This is the DB2 administrators responsibility. Image copies and incremental image copies of catalog tables will be recorded in the SYSCOPY table (same as for any table). The recovery utility may be used to recover a catalog table in the same way as to recover a user table.

Image copy descriptors of the SYSCOPY table itself are recorded on the DB2 log. This permits recovery of the SYSCOPY table, without which no other recovery operation is possible.

SYSTEM RESTART AND RECOVERY

In the following section some information is presented on how to recover in case of failures of components of the DB2 system or its MVS environment.

MVS Failure

In case of an MVS failure due to software, hardware, or power supply, the following steps are required to restart the DB2 environment:

1. IPL MVS
2. START DB2. This will backout uncommitted work and make data available again.
3. Start other subsystems (IMS, CICS) connecting to DB2. The synchronization required between DB2 and IMS and CICS will be presented in later chapters.

DB2 Subsystem Failure

If the DB2 subsytem fails it is restarted through the DB2 START command. Uncommitted work will be backed out and synchronization with IMS or CICS is performed as explained under MVS failure.

Resource Recovery

On a failure of one of the system resources, in most cases, DB2 will be able to continue with processing, and the error may be corrected. Some of the situations are explained in the following section, but not all the possible cases are listed here.

Log Data Sets

There are a number of possible problems with log data sets. Some of the actions performed are:

- Running out of space on active log:

  The system issues warning messages when the last available active log data set is 75% full. DB2 continues until no more space is available, then it will stop and wait until the log offload function terminates and frees one active log data set. Probably the user should define more active log space when DB2 is shutdown the next time.

- Read I/O error on active log:

  If during offload then DB2 will offload from the dual copy active log. If no dual log exists, then the archive log has a discontinuity. Probably the user should image copy all data bases to insure recoverability.

  If an I/O error occurs during recovery, DB2 will try to locate the required log records on dual copy log, or on archive logs. If unsuccessful, recovery fails. Same situation if read I/O error on archive log.

- Write I/O error on archive log:

  Offload dynamically allocates a new archive log and restarts offload from beginning again. User might change the offload unit (disk, tape).

Boot Strap Data Set

The BSDS is crucial to log operation. Dual mode is suggested for secure operation.

- In case of I/O errors on a BSDS, DB2 falls back into single mode. The administrator must redefine the damaged BSDS with VSAM AMS, and then reinstate dual mode using the DB2 RECOVER BSDS command.

- The BSDS should never become full. Dummy records are inserted at DB2 installation time for the number of archive log data set defined.

If both BSDS are lost, they may be recovered from an archive copy. As explained in the logging section, an archive copy of the BSDS is automatically created whenever an active log is offloaded into an archive log. The change log inventory utility may be used to update information about active and archive log data sets.

Catalog

Catalog recovery is performed using the DB2 recovery utility. Since there exist dependencies between some of the catalog tables (e.g. any catalog table is dependent on a correct SYSCOPY table), the "higher" level catalog table must always be recovered first. Details about dependencies in the catalog are not presented here, but the reader should be aware that catalog recovery might require some more understanding and knowledge of the structure of DB2. For a more complete discussion of this subject, please see IBM DATABASE 2 Operation and Recovery Guide.

Data Base

Data base recovery is performed using the DB2 recovery utility. Some of the reasons for data base recovery are:

- Write I/O error:

  DB2 keeps track of an error range of bad pages. If this range is small in regard to the table space "error range" recovery may be performed. Applications are not stopped and no return code is passed back to an SQL statement in case of a write error. Other applications may not access the bad pages until recovery has been completed.

- Read I/O error:

  A return code is passed back to the SQL statement. The DISPLAY DB command may be used to find if an error range is recorded. Recovery may be delayed unless many read errors occur. The table space must be recovered fully, or with the "error range" option.

- Logical error in a data page:

  The page is flagged as broken, and the application program is abended. The data set may be recovered, or the REPAIR utility must be used to "fix" the broken page.

Utility Recovery

When a utility fails, its status is recorded in the system. The utility may be restarted using the same CLIST with the RESTART option. Utilities use a DB2 checkpoint/restart facility and the restart will automatically be from the last checkpoint. Optionally restart may be forced to the beginning of the current (failed) phase.

In this chapter we look at the overall architecture of DB2, and at how other subsystems (TSO, IMS, CICS) attach to DB2. The product structure and attach architecture is presented at an externalized level without going into implementation internals.

DB2 STRUCTURE

The address space structure of a DB2 system under the MVS operating system is shown in Figure 27.

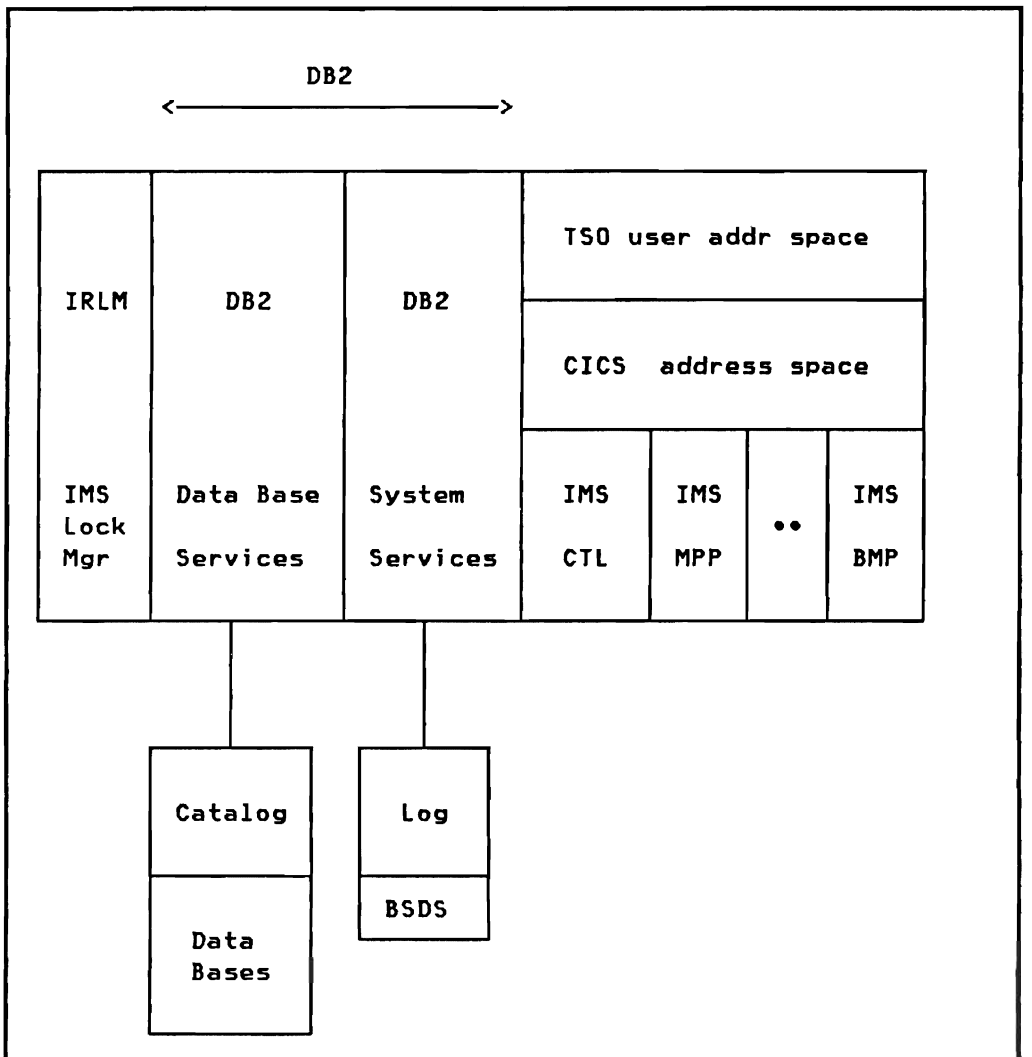


Figure 27. The MVS DB2 Environment

The address spaces and their major functions are:

IRLM The IMS Resource Lock Manager provides locking services to the DB2 address spaces. Locks are used to isolate different DB2 users from each other. IMS may use the IRLM as well for its own locking.

DB2 DB Services: Provides the access to the DB2 catalog and all user tables.

DB2 System Services: Provides logging, restart, and operational control services.

TSO User Any number of TSO users may concurrently access DB2 data.

CICS Transactions running in a CICS address space may access DB2 data.

IMS Transactions running in message processing programs (MPP), batch message programs (BMP), and Fast Path programs (IFP), may access DB2 data.

DB2 System Services

This address space provides the overall DB2 execution environment, and the set of services and functions which are required by the DB2 DB Services. A selection of functions provided is:

- **Attachment oriented functions** are those required to create and maintain a connection between DB2 and the other subsystems in which application programs are executing. These include connection protocols for TSO (through the DB2 command processor and SPUFI), JES batch (through the DB2 command processor under the TSO TMP), IMS, and CICS.

- **Service oriented functions**, such as:

  - Logging
  - Serialization of functions
  - System messages
  - Virtual storage management
  - DB2 command support
  - Trace, statistics, and accounting

- **Coordination oriented functions** are functions which require coordination among several DB2 components, such as

  - Allocation/deallocation
  - Commit and abort
  - Recovery
  - Checkpoint
  - Startup and shutdown

DB2 Data Base Services

This address space, in combination with the DB2 System Services, provides a relational data base management capability. Application programs requiring access to data which is under the control of DB2 may execute in any of the environments supported by DB2. This access is established via DB2 System Services, which uses the MVS Subsystem Interface (SSI).

DB2 DB Services provides facilities for definition of data bases (table spaces, tables, etc) and their access control, and for manipulation of the data. Some of the subfunctions are:

- **Data space management** controls storage groups, volumes, and all the VSAM data sets used to hold the data of relational tables.

- The stand-alone **precompiler** scans the application source code and converts SQL statement into host language calls. The SQL statements are extracted into a data base request module (DBRM), which is input to the binding process.

- The **relational data system** includes the bind function to convert a DBRM into an application plan. During execution it receives the application's manipulative or definitional SQL requests from the plan, or as dynamic SQL statements.

- The **data manager** processes the SQL requests, and performs the necessary locking (using the IRLM), logging (using DB2 System Services), and path maintenance.

- The **buffer manager** controls the movement of data pages between auxiliary storage and virtual storage buffers.

- The **utility component** provides all the data base utilities (reorganization, backup, recovery, statistics). For improved performance the utilities use internal protocols rather than the application program interface. These internal protocols provide fast sequential access.

IRLM

The **IMS Resource Lock Manager** provides the locking services on DB2 resources. One IRLM may service both DB2 (for locks on DB2 resources) and IMS (for locks on DL/I resources), or separate IRLM's may each service either DB2 or IMS.

The IRLM, in regard to DB2 resources, provides locking within one MVS system. The IRLM, and hence DB2, does **not provide data sharing** between multiple DB2 systems running in one or multiple MVS systems.

The IRLM may be started automatically when DB2 is started and the IRLM is not present.

INTERFACING TO DB2

Transactions entered by IMS, CICS, or TSO users may issue SQL requests for DB2 data. DB2 provides modules called 'attach facilities' which, running under an IMS/CICS/TSO subtask, use MVS sub-system interface (SSI) protocols to interface requests from IMS/CICS/TSO to DB2, and to coordinate resource commitment.

TSO

In the TSO environment the Terminal Monitor Program (TMP) attaches the DB2 supplied DSN command processor which links to the user application.

A range of DB2 functions, including the running of TSO applications, may be invoked from the DB2 Interactive Interface (DB2I) as presented in "ISPF Support - DB2I" on page 100. One DB2 function which may be invoked is called SPUFI. This provides the capability to execute a file of SQL statements.

IMS and CICS

In the IMS environment the transactions using SQL calls run in MPP, BMP or Fast Path regions. In the CICS environment the transactions run as normal CICS applications.

In addition to the SQL requests, the IMS and CICS transactions (but not TSO) may also issue DL/I calls. An application program may therefore access DL/I and DB2 data in parallel. DL/I batch jobs and CICS shared data base regions may not issue SQL calls.

ATTACH ARCHITECTURE

DB2 provides attachment facilities that allow other subsystems or batch address spaces to access DB2 resources. A conceptual view of these facilities is shown in Figure 28.

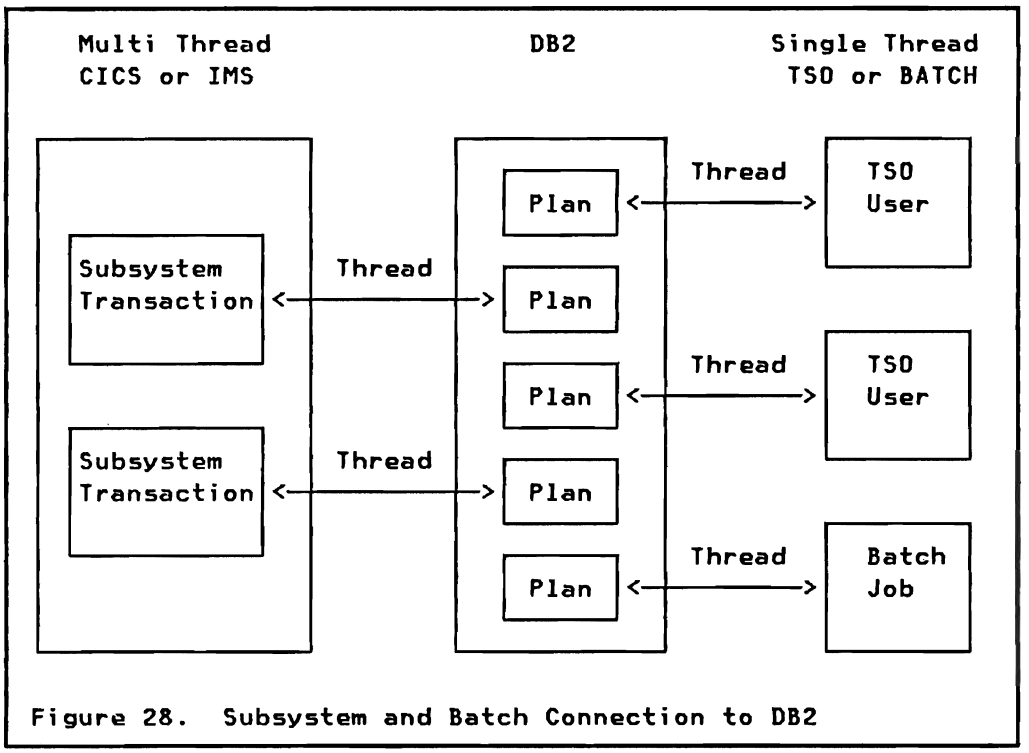


Figure 28. Subsystem and Batch Connection to DB2

Attachment facilities are provided for TSO, Batch jobs (running under the TSO TMP), IMS, and CICS.

Thread

A thread can be considered as a path from the subsystem or batch address space to a specific application plan. When an application makes its first SQL call DB2 is aware that an external subsystem is starting a piece of work for DB2. It will require certain DB2 resources, and will have to be tracked. Just as an operating system uses the term 'job' to describe work coming in from the external environment, so DB2 uses the term **thread** to describe a unit of work which originates in an external subsystem.

A thread is begun with the first SQL call of an application. DB2 sets up control blocks and allocates resources to the thread.

- The data base resources are described in the application plan which is created by the bind process. The plan contains details of DB2 data to be accessed, and the calls to the data manager used to execute the SQL calls.

During execution of the application program DB2 tracks the progress of the thread through its interactions with DB2 data and points of resource commitment.

DB2 supports two types of connections:

Single thread connections Only one thread is allowed per connection. This is the case for TSO and batch address spaces.

Multi-thread connections Multiple threads can be established between a connected subsystem and DB2. Each thread services the requests of a specific subsystem transaction. This connection type is used for IMS and CICS.

A thread is ended when the application program terminates and all changes made by it have either been committed or rolled back.

Thread Identification

When a thread is created through the first SQL call the user (or application program) must identify itself to DB2. The identifiers associated with a thread are defined below and shown in Figure 29 on page 153:

Connection-ID Identifies the connecting subsystem (e.g. the IMSID of the stage-1 generation of the IMS system).

Authorization-ID Identifies the user who is making the request. This ID is used for authorization checking as described in "Chapter 9. Security and Authorization." A user exit (sign-on exit) is available and allows the user in certain situations to change the authorization-ID.

Plan-ID Identifies the name of the application plan that is executed. Predefined plan names are used for SPUFI and DCLGEN from a TSO terminal.

Correlation-ID Uniquely identifies the thread and correlates it with the requester. In IMS and CICS the same application program may exist concurrently. The plan-ID is therefore not specific enough to identify the thread.

	BATCH	TSO	IMS	CICS
Connection-ID	'BATCH'	'TSO'	imsid	cicsid
Authorization-ID	USER-ID on jobcard or Install. Default	Logon-ID	Message Region: - LTERM or - SIGNON-ID Non-message driven Region: - Jobcard USER - or PSBNAME	USER-ID or Terminal-ID or Transact-ID or other Specified in Resource Control Table (RCT)
Plan-ID	Plan name	Plan name Special for SPUFI	Program name (modifiable by user module)	RCT specification or Transaction-ID
Correlation-ID	Jobname	Logon-ID	PST#.PSBNAME	thread#.tran-ID

Figure 29. Thread Identification

Unit of Recovery

An application issues SQL calls, and in the case of IMS and CICS may also issue DL/I calls. An application may regard all calls between two synchronization points as being part of the same **unit of recovery.**

The first unit of recovery is started with the first SQL call (thread creation). The unit of recovery ends at the next synchronization point, and a new unit of recovery is started at the next SQL call.

TSO: A unit of recovery is ended explicitly through a COMMIT or ROLLBACK SQL statement. When an application program terminates normally, a COMMIT is invoked automatically. In case of an abnormal termination of the application program an automatic ROLLBACK is invoked. The thread is terminated when the application program ends. Several units of recovery of one user may be processed serially in one thread.

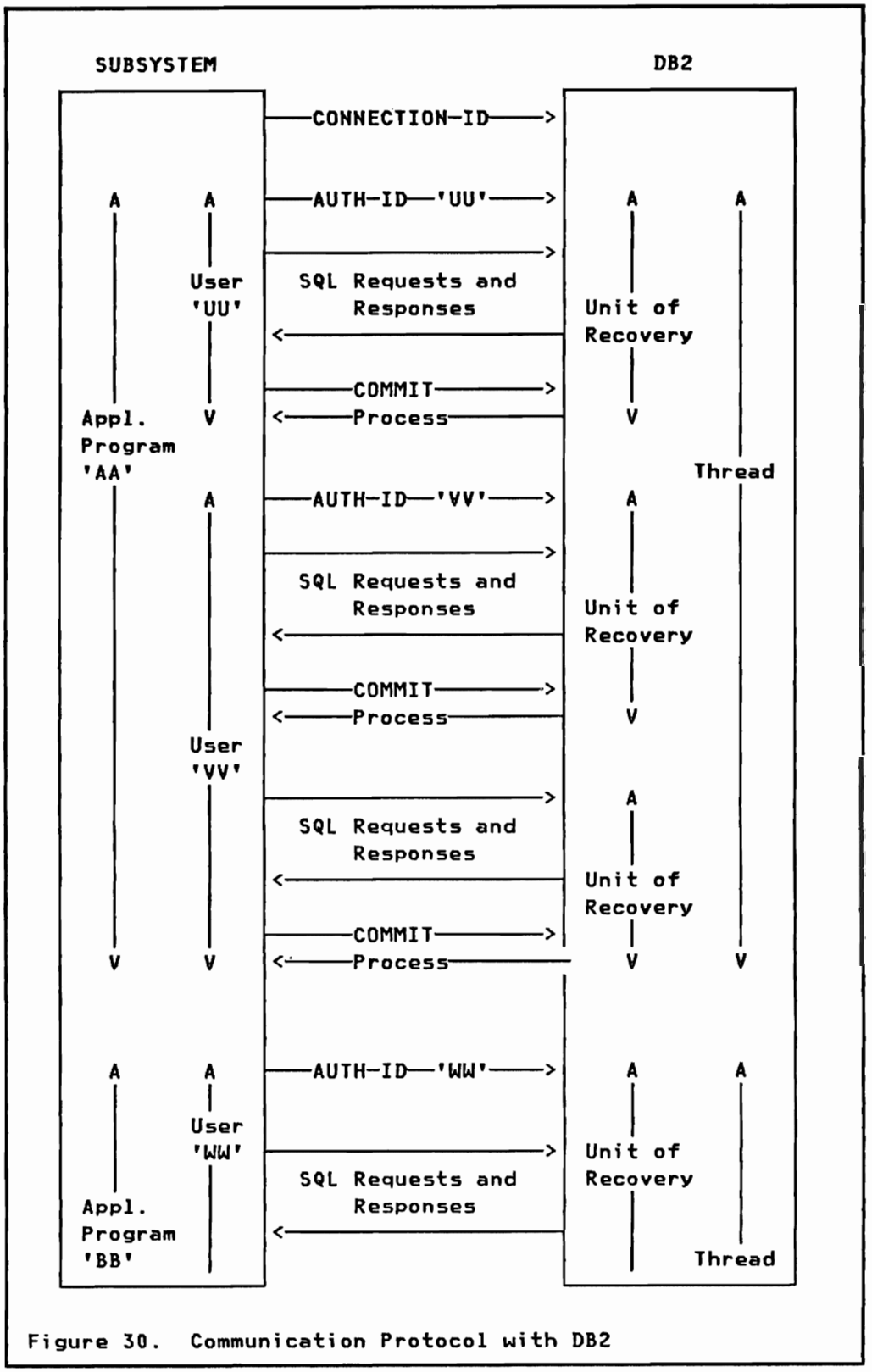


Figure 30. Communication Protocol with DB2

IMS and CICS: A unit of recovery is ended when a synchronization point is reached in the IMS or CICS subsystem. Application programs may not issue SQL COMMIT or ROLLBACK statements directly. More details about these environments are presented in later chapters.

Protocol: The communication protocol used between a subsystem (e.g. IMS) and DB2 is illustrated in Figure 30 on page 154.

Multiple users may be identified in one application program (e.g. multiple IMS transactions executed in one program scheduling). Each user has at least one unit of recovery. Many units of recovery may be executed serially in one thread. In TSO and IMS a thread corresponds to the execution of one application program. In CICS many application programs may serially use the same thread.

COMMIT

When an application program positions itself on a table row the page containing that row is locked. This guarantees integrity (the current row cannot be modified).

When an application program updates a table row the page containing that row is locked exclusively until the next synchronization point is reached.

When a synchronization point is reached all updates within the unit of recovery must be committed in DB2 and the connected subsystem. To assist in achieving this, a protocol called **two phase commit** is provided. The ability of the different external subsystems to guarantee unified commit depends on their implementation of the protocol.

Two Phase Commit

This protocol recognized the existence of two distinct phases of processing required to establish a new point of consistency (synchronization point). One of the connected subsystems is the "coordinator", the others are "participants". In connections with IMS and CICS, DB2 is a participant. In connections with TSO/Batch, DB2 is the coordinator and no participant outside of DB2 exists.

Phase 1 Commit preparation and vote collection

The coordinator assumes the role of the vote collector and notifies all participants. Each participant must declare whether or not it agrees to continue with the commit process. If one participant disagrees, it returns a negative vote and the unit of recovery must be rolled back by all the subsystems involved. Each

participant that agrees to commit must prepare to do the second phase (e.g. by completing its loggingduring phase 1), but still be capable of reversing the changes if a negative vote is cast by some other participant.

Phase 2 Must-complete processing

The coordinator, having received all positive votes from phase 1, notifies the participants to start phase 2. Each participant then commits the unit of recovery and makes the new committed form of the objects (e.g. data base records) accessible to other applications. There are no voting options in phase 2.

If any vote from phase 1 was negative the coordinator notifies all participants to abort (rollback the changes).

If any subsystem abends after phase 2 commit has been started, that subsystem must be able to recover to the synchronization point with the unit of recovery completed, that is, the commit phase 2 must be executed during the restart.

TSO/BATCH ATTACHMENT FACILITY

The TSO attach package is initiated via the TSO Terminal Monitor Program (TMP). It attaches a task which passes the authorization-ID (logon-ID or batch USER-ID) to DB2. A single thread is created, and the user program is invoked through OS LINK. The user program communicates with DB2 through a language interface link edited with the program.

The application program has access to DB2 resources and MVS services. Requests for DB2 resources must only be made from one MVS task, either the task which was current when the application received control, or one of its subtasks. Work is committed when the application returns control to the attachment facility, or through intermediate SQL COMMIT statements. The application program may request a rollback of all the data base changes through an SQL ROLLBACK statement.

The TSO/Batch attachment facility does not participate in any commit, rollback or restart scenarios with resources outside of the DB2 domain. An application may access recoverable resources outside of DB2, however, there is no DB2 support for coordination and synchronization of the consistency of those resources.

In this chapter we look at DB2 facilities in the areas of:

- **Optimization**

  What are the parameters of DB2 which allow us to influence the performance characteristics of the system ?

- **Monitoring**

  How can we measure DB2 performance ? What statistical facilities are available ?

- **Accounting**

  What accounting information is available on a user and application basis ?

OPTIMIZATION

Performance of a DB2 system is influenced by a number of parameters. Some of them have a **system** wide function, and others may be changed for individual **data base** objects like tables spaces. The data base parameters may be changed dynamically when the system is running, whereas most of the system parameters are defined at DB2 installation time.

System Parameters

Following is a list of some of the system parameters which are defined at **installation time**. They are all contained in a parameter module (DSNZPARM), and when DB2 is started a version of the parameter module may be selected through the START DB2 command. New versions of the parameter module may be generated at any time, but will not be active until DB2 is stopped and started again.

Logging Selection of total number of log buffers, number of full buffers before write, archive log blocksize, single or dual active log, optional offload function, and all the attributes of archive logs. See section "The DB2 Log" on page 130 in "Chapter 11. Operation and Recovery" for details.

Operation Checkpoint frequency, optional SMF statistics and accounting information, size of instorage trace tables, optional security.

Buffer pool The minimum and maximum size of the DB2 buffer pools may be modified.

Data Base Parameters

The DB2 objects used to store and manipulate table data allow for a number of options which may influence the performance characteristics of a DB2 system. The list below is a selection of DB2 functions or options, which may be changed during execution. Some changes require a redefinition of DB2 objects and data may be unavailable during that time.

Indexes Indexes may be added on table columns which are often used in WHERE search conditions or joins of tables.

Index subpages The number of index subpages may vary between 1 and 16. The index must be redefined to change this.

Buffer pool The buffer pool assigned to a table space may be dynamically changed.

Locking The locking strategy assigned to a table space may be dynamically changed. Changing the size of the locks (page or table space) may influence other applications and therefore the performance characteristics of the system.

Close Table space and index data sets may be closed when users are not accessing the data.

Clustering A clustering index on a column which is often used for sorting (ORDER BY) should improve performance.

Bind consistency The bind isolation parameter defined as repeatable read or cursor stability for all the tables used by an application.

Authorization Performance may be improved when authorization can be checked at bind time instead of delaying it to execution.

Rebind Rebinding application plans may improve performance when new access paths are evaluated based on current statistics (e.g. number of pages) and available indexes.

MONITORING

DB2 provides a number of facilities which assist the installation in monitoring subsystem activities and performance.

Display Commar

The DISPLAY command can be used to display information about current usage of DB2 data bases, and about users (or application programs) currently running DB2 work.

The display command shown below is used to list the current usage of a table space:

```
-DISPLAY DB    (data-base-name)
         ACTIVE
         SPACE (table-space-name)
```

The output contains status information about the DB2 data base and the named table space, as well as subsystem names (called connection-ID) and correlation-ID's of users or applications currently using the named table space. A correlation-ID is a TSO logon-ID, IMS region and PSB-name, or a CICS transaction name.

To list all the users or programs currently executing work using DB2 use the display command shown below:

```
-DISPLAY THREAD (*)
```

The output lists the connection-ID (subsystem name), correlation-ID (userid, PSBNAME, or transaction name), and application plan name of all currently active users of DB2.

STOSPACE Utili

The STOSPACE utility provides information about the actual space allocated for storage groups, table spaces, and indexes. The information is gathered from the VSAM catalog. Actual table data is not accessed and does not have to be available.

The STOSPACE is invoked through the TSO SPF utilities panel (shown in Figure 25 on page 135) using the following utility statement:

```
STOSPACE STOGROUP (storage-group-name-list)
```

The output of the STOSPACE utility is stored in the DB2 catalog; it includes:

- The number of kilobytes allocated to each table space and index associated with the named storage group(s). This value is stored in the SPACE column of the catalog tables SYSTABLESPACE and SYSINDEX.

- The total number of kilobytes allocated for the storage group (column SPACE), and the run date (column SPCEDATE) is stored in the SYSTOGROUP catalog table.

RUNSTATS Utility

The RUNSTATS utility collects **statistical information** for one table space and its associated indexes. The output of the utility is stored in the DB2 catalog.

The statistical data stored in the catalog is used by the bind process to select the best access path for SQL data manipulation. It includes:

- For each index: the number of levels, the number of leaf pages, the number of distinct key values, and if the table space is clustered by this index.
- For the table space: the number of active pages.
- For each table of the table space: the number of rows, and the number and percentage of pages in the table space that contain rows of that table.
- Reorganization information about the number of rows whose RID position is in a page different from its record.

The RUNSTATS utility is invoked as a stand-alone utility using the TSO SPF utility panel with a utility statement as shown below:

```
RUNSTATS TABLESPACE table-space-name
         INDEX      (ALL | index-names)
         SHRLEVEL   REFERENCE | CHANGE

RUNSTATS INDEX      index-names
```

TABLESPACE and INDEX: The utility is run for one table space and associated indexes, or for one or more indexes alone.

SHRLEVEL: Specifies if other applications may use the data at read-only (REFERENCE) or update (CHANGE) level during the execution of the RUNSTATS utility.

Statistics Facility

You can have statistical information written to a System Management Facility (SMF) data set by specifying the appropriate parameter at installation time.

Two type of SMF records are generated and written at DB2 checkpoints. They contain subsystem summary data sections of all the components of DB2.

1. SMF record type 100 subtype 0

   - Subsystem Service Component (SSSC): sign-on, connect, commit, abort, etc
   - Command: counts per DB2 command
   - Latch manager: serialization contention
   - Storage manager: getmain, freemain, pool expansion
   - Trace data: on/off counts
   - Log manager: count of buffers filled, wait for buffer, timer exit, log reads from active/archive log, etc

2. SMF record type 100 subtype 1

   - SQL data: count per SQL statement type
   - Bind data: count of bind invocation and affected plans
   - Buffer manager: count of page requests, I/O's, pool expansions, paging required for buffer read/write, etc
   - Lock usage data: count of wait, deadlock, timeout

Note: All the counts in these SMF records are accumulated totals. The values are only reset to zero when DB2 is started.

ACCOUNTING FACILITY

The accounting facility collects data on an authorization-ID and work basis, and writes them as SMF type 101 subtype 0 records. A record is written each time a user (or subsytem application program) disconnects from DB2.

Accounting records are identified by authorization-ID (userid), connection-ID (subsystem name), correlation-ID (e.g. an IMS PSTnumber and PSBNAME), and plan-ID (IMS program name).

The SMF record contains

- The reason the record was written (normal termination, abend)
- Connection time (elapsed)
- CPU time
- SQL data: counters per SQL statement type
- Commit and abort counts

- Buffer manager: page requests, buffer expansions, updated rows

- Lock usage: wait, deadlock, timeout

Installations have to write there own programs to process SMF data for their accounting.

AUDITING

DB2 provides no explicit support relative to an installation developing and utilizing audit procedures. However, the DB2 recovery log contains useful audit information.

Installations may write their own programs to merge information from the DB2 log and data base image copies to reconstruct which user updated certain tables.

Log Access Services

DB2 provides log access services which allow user written programs to access the DB2 log, without explicit allocation of active and/or archive log data sets by the user.

- With a log OPEN request the user specifies the range of log RBA's which he wants to access. DB2 will dynamically allocate the required log data sets (active and/or archive).

- The user program then uses log GET requests to sequentially access all the log records of the specified range.

- A log CLOSE request dynamically deallocates the log data sets.

These simple log macros allow users to efficiently access log records, with all the control and management done by DB2. The only data set required in user JCL is the Boot Strap Data Set.

In this chapter we look at how application programs running under IMS[3] may access DB2 data, and how an IMS terminal user may interact with DB2.

OVERVIEW

An IMS system is structured into multiple address spaces. The **control region** communicates with the terminals, queues transactions and responses, schedules application programs in dependent regions, and provides recovery logging and checkpoint/restart. Application programs run in **dependent regions** and are called message processing programs (MPP), batch message programs (BMP), and Fast Path programs (IFP).

An IMS application program may access DL/I data bases and/or DB2 tables. Older programs might access DL/I data bases only, whereas new programs may require access to both DL/I and DB2 data, or a program might access DB2 tables only.

The overall structure of an IMS system connected to DB2 is shown in Figure 31 on page 164.

The IMS Resource Lock Manager (IRLM) provides the resource locking function for DB2 and optionally also for IMS.

Note: DL/I batch regions, which operate independently from the control region, may not access DB2 data.

IMS ATTACHMENT OF DB2

DB2 table data is accessed from IMS application programs through the IMS attachment of DB2. This attachment facility allows:

- an application program (MPP, BMP, or IFP) to issue SQL statements to access DB2 data (data manipulation), to define DB2 objects (data definition), to grant/revoke security capabilities, and
- a terminal user to enter DB2 commands

---

[3] Information Management System/Virtual Storage, Program Number 5740-XX2

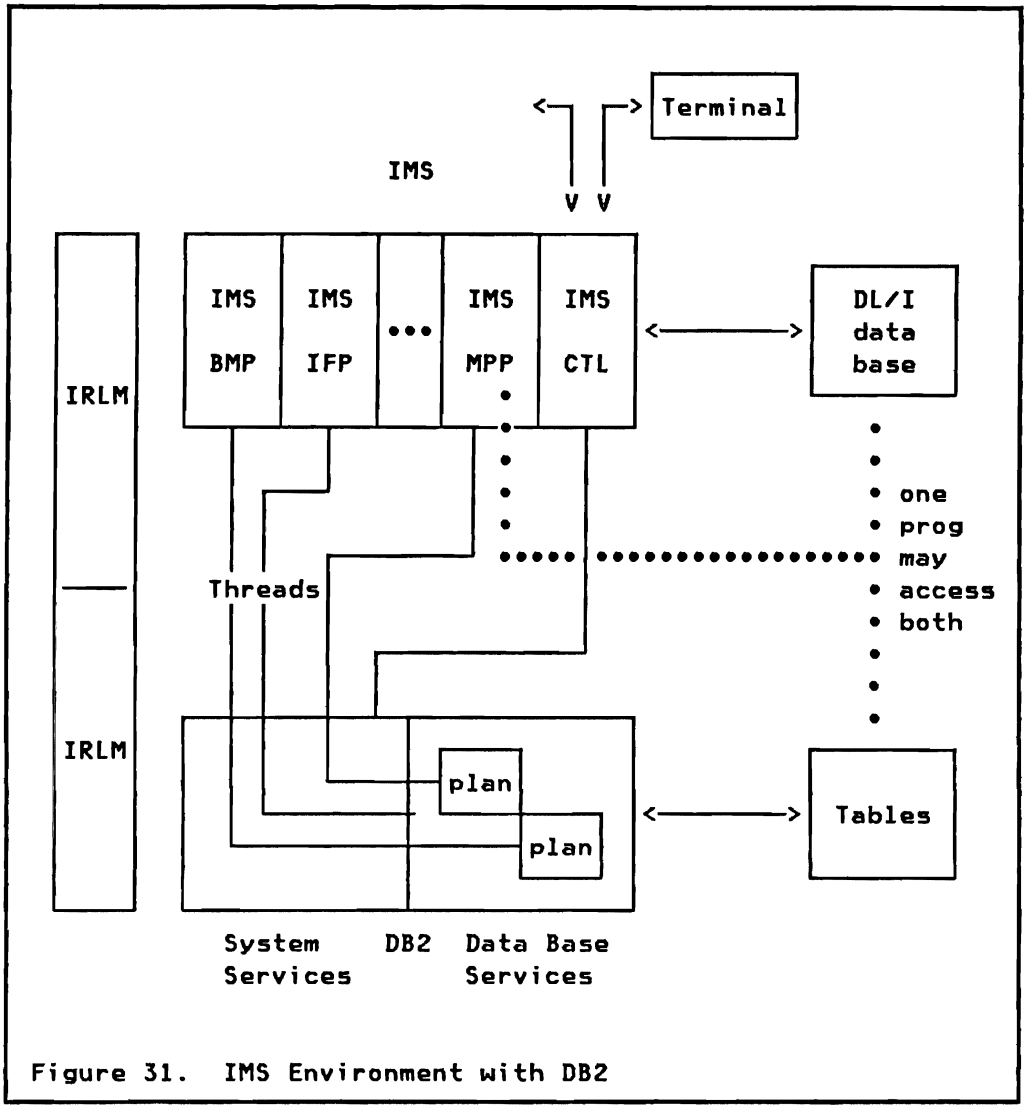


Figure 31. IMS Environment with DB2

The IMS attachment consists of multiple connections between IMS and DB2:

- The IMS control region must be connected to DB2 if any application program accesses DB2 data.
- Each dependent region running application programs which use DB2 data must be connected to DB2.

IMS Control Region

A connection is established between the IMS control region and DB2 at startup, or later through an IMS command. The connection-ID associated with the connection is the "IMSID" of the IMS stage 1 IMSCTRL macro.

The connection is used for subsystem coordination, commit synchronization, and to pass DB2 commands entered from an IMS terminal to DB2, and receive the responses to the command from DB2.

Attachment control parameters are specified in a **subsystem member (SSM)** in the IMS PROCLIB. The name of this member is the concatenation of the IMSID with a suffix specified in the EXEC JCL statement. The member contains a record with the following values:

- DB2 subsystem name
- Language interface token used in the IMS language interface module (DFSLI000), normally SYS1.
- Name of the attachment initialization module, must be DSNMIN10.
- Name of a resource translation table. This (optional) table may be used to map an application load module name to a DB2 application plan if they are not the same.
- Error option if an application program wants to access a non-operational DB2 system: return code, abend with/without queueing the input message.
- Command recognition character (CRC) which identifies DB2 commands entered from an IMS terminal.

An IMS system may be connected to multiple DB2 systems. Each DB2 system will be identified in a separate record of the above SSM member.

IMS Dependent Region

Each dependent region running application programs which use DB2 data must be connected to that DB2 system. The connections are by default the same as for the control region, unless a separate SSM member is specified in the EXEC JCL statement of the dependent region to activate a subset of the available connections.

- A connection from the control region must exist in order that a dependent region may communicate with a DB2 system.
- A dependent region may be connected to multiple DB2 systems, but a specific application program may only access data from one DB2 system. The DB2 system to be connected is identified by a token in the language interface module and the DB2 subsystem name in the corresponding SSM.

A thread is created for each application program (which issues SQL calls) being scheduled in the dependent region. The thread is

terminated at program end. For MODE=SNGL programs, each transaction is a unit of recovery, whereas multiple transactions may be in one unit of recovery for MODE=MULT programs. A unit of recovery is terminated by an IMS synchronization point in the dependent region. The protocol between an IMS dependent region and DB2 was shown in Figure 30 on page 154.

Authorization

The **Authorization-ID** for an IMS transaction is any of the following:

- The USER-ID if the application or command is invoked from a terminal which has used IMS SIGN-ON.
- The IMS logical terminal name (LTERM) if the application or command is invoked from a terminal without previous IMS SIGN-ON.
- For non-message driven programs the USER-ID on the job card (ASXBUSER).
- The PSBNAME for non-message driven programs where the USER-ID field (ASXBUSER) is null.

DB2 checks the authorization-ID before accessing an application-plan or executing a DB2 command. A user must be authorized (GRANTED) to the application plans he is accessing. DB2 authorization must also be granted for users entering DB2 commands from IMS terminals. DB2 cannot validate the authorization-ID itself, no password is available to DB2. IMS security checking must be used for this purpose.

APPLICATION PROGRAMMING

An application program in the IMS environment uses the techniques presented in "Chapter 7. Application Programming Environment." Accesses to DL/I data may be intermixed with accesses to DB2 data. Application programs must be link edited with a new IMS/VS Release 1.3 language interface module (DFSLI000) if they access DB2 data.

The two SQL statements COMMIT and ROLLBACK cannot be used in an IMS application program:

- **COMMIT** is automatically invoked at normal program termination, through a DL/I checkpoint call (CHKP), or in single mode when the next transaction is received (via a GU call).
- **ROLLBACK** is automatically invoked at abnormal program termination, or through the DL/I rollback calls (ROLL and ROLB).

An application program will receive an SQL return code when DB2 data is not available (data base stopped), or when authorization does not allow the access. These situations may be reflected back to the terminal by the application program.

An application program might be subject to **deadlock** abend. A deadlock may occur within DL/I or DB2 data only, or a combination of DL/I and DB2 data requests may be the cause for a deadlock. One of the contestants is selected and its changes rolled back. A message driven program is abended and requeued; a non-message driven gets a SQL return code of -911.

Transaction Processing

Communication with an IMS terminal uses the DL/I mechanism. Transaction input is received using the DL/I 'GU' or 'CHKP' call, and messages are sent to terminals using 'ISRT' and 'PURG' calls.

The IMS end user at the terminal should note no differences if a transaction accesses DB2 data. No message relating to DB2 are sent to the end user by IMS.

Application programs running transactions which use DB2 data may be displayed using the DB2 DISPLAY THREAD command:

```
-DISPLAY THREAD (imsid)
```

Commit

As presented above COMMIT of changes of DB2 resources is invoked through an IMS synchronization point. IMS uses the **two phase commit** protocol to synchronize committing of DL/I and DB2 changes. This protocol was introduced in "Two Phase Commit" on page 155(IMS is the coordinator, DB2 is a participant).

After IMS has decided to commit changes and commit phase 2 is started, both IMS and DB2 will commit the changes and make them available to other users. At the instant of the decision an IMS type '37' log record is written. If any system should abend during commit phase 2 then commit is finished at restart time.

COMMIT ends a unit of recovery. Data which was changed is made available to other applications and users. Note, however, if locking is at the level of the table space (see section "Defining a Table Space" on page 47 in "Chapter 5. Data Definition"), locks are freed only when the application program is terminated.

In-doubt Threads

A thread is **in-doubt** from the moment DB2 has voted positively in commit phase 1, until the notification is received from IMS to execute commit phase 2. If during that time interval either IMS or DB2 fails the thread remains in-doubt until both systems are up again.

The thread is in-doubt because DB2 does not know what decision was taken by IMS. IMS may already have committed or rolled back the DL/I changes, or will finish its process when being restarted after a failure. When both systems are up again they communicate about these in-doubt threads, and IMS will notify DB2 about the action to be taken. DB2 will then either commit or roll back the changes.

In-doubt threads can be displayed through:

```
-DISPLAY THREAD (imsid) TYPE (INDOUBT)
```

In case of an IMS cold start it may be required that the system administrator resolves in-doubt threads through a DB2 command:

```
-RECOVER INDOUBT (imsid) ACTION (COMMIT|ABORT)
         ID (correlation-ID)
```

Application Testing

The SQL portion of a program may be tested using SPUFI from a TSO SPF terminal. This may even be done before the program is coded. Individual SQL statements may be submitted, or streams of SQL requests may be passed to DB2 through SPUFI. Different versions of the same SQL function may be tested against each other, and the best then selected for actual application coding.

For detailed testing both DB2 and IMS must be up. IMS application programs may not be tested in batch if they access DB2 data. SQL cursor operations cannot be tested using SPUFI.

OPERATION

An IMS system connected to DB2 is controlled by the IMS master terminal operator or another authorized terminal users. The connection with DB2 is controlled through IMS commands, and once established may be used to route DB2 commands from IMS to DB2.

IMS Commands

The connection between IMS and DB2 is established automatically when IMS is started and DB2 is up already, or when DB2 is started and IMS is up already.

The following commands may be used to control the connection ("sysid" is the DB2 subsystem name):

/DIS SUBSYS sysid Displays the connection status and activity of the control region and each dependent region to the named DB2 system.

/STO SUBSYS sysid Stops the connection from IMS to the named DB2 system.

/STA SUBSYS sysid Starts the connection from IMS to the named DB2 system. Used after a /STO SUBSYS command or when automatic connection failed for some reason.

The usage of these IMS commands may be controlled through the standard IMS security mechanism. No authorization checking by DB2 is involved.

Issuing DB2 Commands from IMS

Once a connection between IMS and DB2 is established authorized terminal users and programs (by IMS security) may use the IMS **/SSR command** to route a DB2 command to the DB2 system.

```
/SSR -DB2-command
```

The DB2 command is routed to DB2 for processing. All DB2 commands except START DB2 are available. DB2 checks if the user (or program) is authorized to issue the DB2 command. Responses are routed back to the originating IMS user. The correct command recognition character (CRC) must be used to route the command to the destination DB2 system. The default CRC is the "-" sign. It must be specified in the SSM entry.

Examples are:

```
/SSR -DISPLAY DB (data-base-name) SPACE(*)
/SSR -START   DB (data-base-name)
/SSR -DISPLAY THREAD
```

MONITORING THE IMS ATTACHMENT

The following facilities are available to monitor IMS Attach related events:

- The **IMS DC Monitor** facility records DB2 subsystem exit activity (interactions with DB2) if the monitor has been started by the IMS master terminal operator.
- The **IMS DISPLAY Command** provides information on subsystem connections and active threads.
- The **IMS LOG** contains records of all indoubt situations, and may also contain IMS TRACE data relating to DB2 events.
- The **DB2 TRACE** facility monitors events within DB2 and writes records to in-storage tables or GTF. The trace facility is presented in "Chapter 16. Installation and Servicing" section "Trace" on page 190.
- The **DB2 Accounting** facility collects statistics on an authorization-ID and work basis and writes them to SMF. See "Chapter 13. Monitoring and Accounting" section "Accounting Facility" on page 161.
- The **DB2 Statistics** facility collects system wide statistics about thread connections and usage. These are useful when only one subsystem is connected to DB2. See "Chapter 13. Monitoring and Accounting" section "Statistics Facility" on page 160.
- The **IMS TRACE** command.

CHAPTER 15. CICS ENVIRONMENT

In this chapter we look at how application programs running under CICS[4] may access DB2 data, and how a CICS terminal user may interact with DB2.

OVERVIEW

A CICS system runs in one address space. Transactions are started as CICS subtasks within this address space.

A CICS application program may access DL/I data bases, VSAM data sets, ISAM and DAM data, and DB2 tables. Older programs might access DL/I data bases and other CICS data sets only, whereas new programs may require access to DL/I, VSAM, and DB2 data, or a program might access DB2 tables only.

The overall structure of a CICS system connected to DB2 is shown in Figure 32 on page 172.

A CICS subsystem can only be connected to one DB2 system at any one time. A connection may be started and stopped, and a new connection may be established to a different DB2 system.

CICS MRO (multi-region operation) and ISC (inter system communication) may be used to execute transactions in another CICS address space. These transactions may access DB2 data through a connection between their CICS address space and DB2. Individual SQL calls may not be shipped from one CICS system to another. Batch programs using the CICS shared data base support may not access DB2 data.

CICS ATTACHMENT OF DB2

DB2 table data is accessed from CICS application programs through the CICS attachment facility of DB2. This facility allows:

- an application program to issue SQL statements to access DB2 data (data manipulation) and to define DB2 objects (data definition), and
- an authorized terminal user to enter DB2 commands

---

[4] Customer Information Control System / Operating System / Virtual Storage, Program Number 5740-XX1

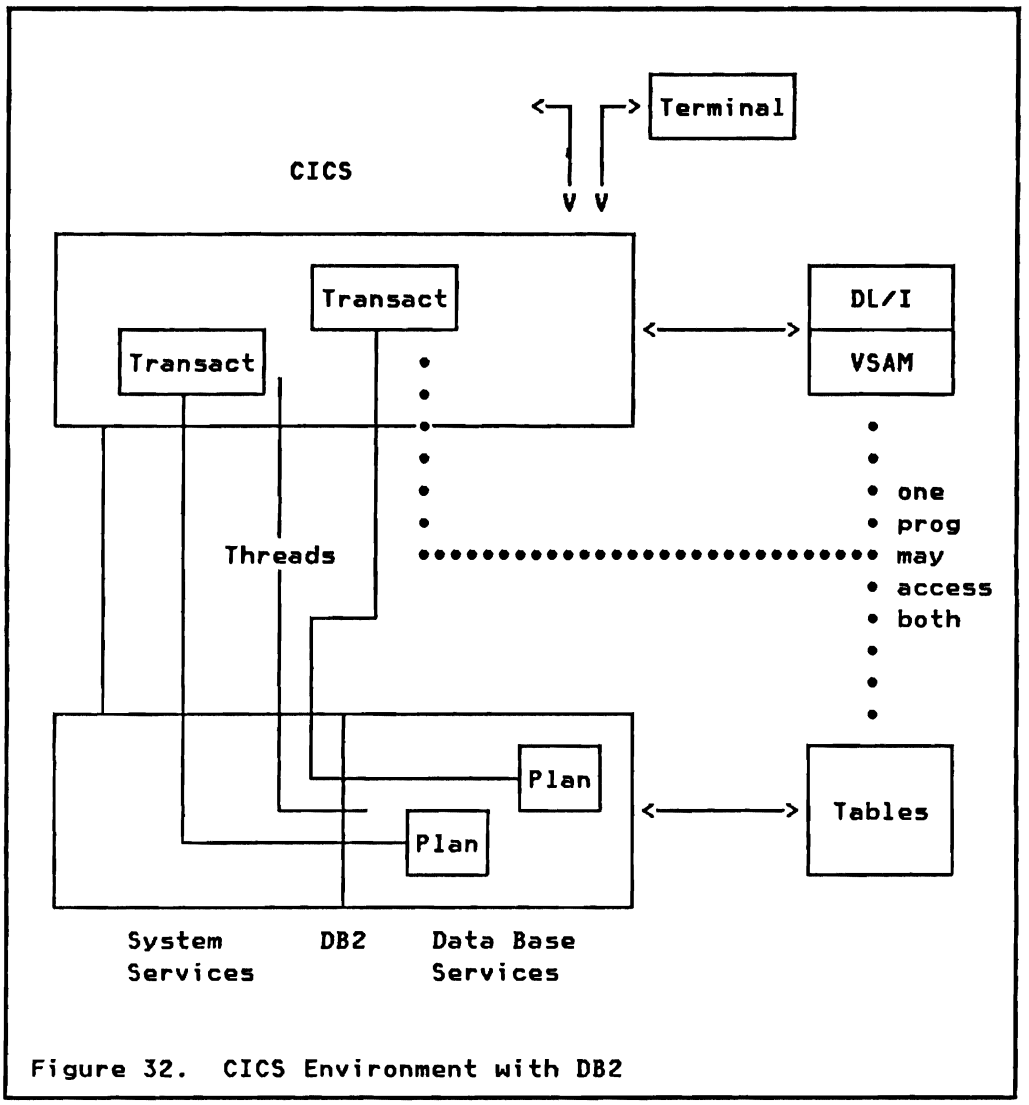


Figure 32. CICS Environment with DB2

The CICS attachment facility consists of multiple connections between CICS and DB2:

- One connection is used to pass operator commands from CICS to DB2.
- A predefined number of connections are used for application programs which use DB2 data.

CICS Resource Control Table

The CICS DB2 Resource Control Table (RCT) predefines the maximum number of connections (threads) which may exist at any one time between CICS and DB2. The table is set up through macros provided with the CICS attachement facility and is structured into three parts:

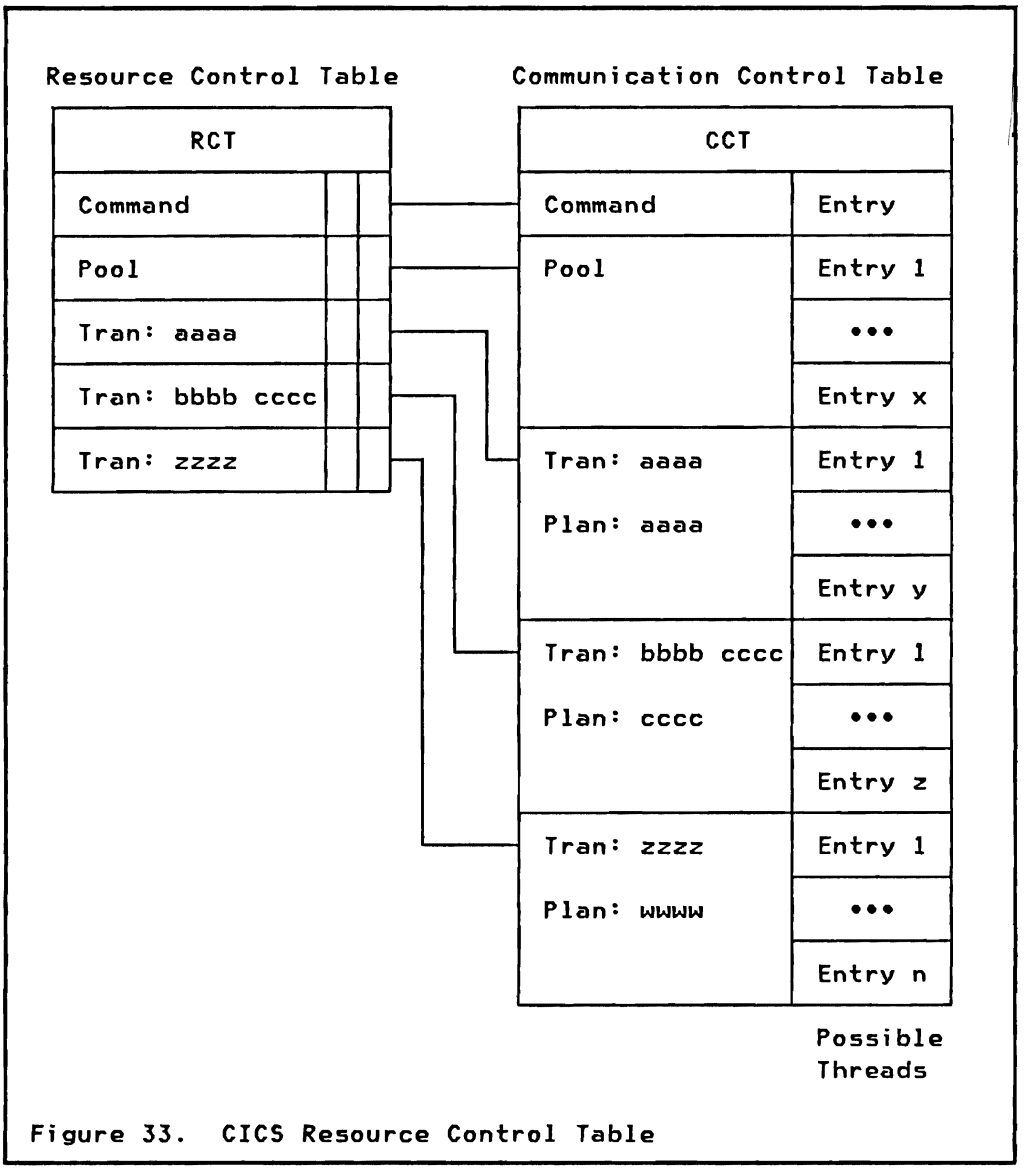


Figure 33. CICS Resource Control Table

Command One thread is reserved for command usage. For heavy command usage pool threads are additionally used.

Pool Shared threads in a pool to be used by all transactions not having dedicated threads. Only one pool with a user specified number of threads is defined.

Dedicated Groups of threads, each servicing one or more predefined transactions using the same application plan. Each group consists of a user specified number of threads. All the threads of the group will use the same application plan. Pool threads will be used in addition if all threads of a group are busy and more transactions of the group are active. The user may define as many groups as needed.

When the CICS Attach is started it builds the **Communication Control Table (CCT)** from the Resource Control Table (RCT). Each entry in the CCT is a possible thread. The two tables are illustrated in Figure 33 on page 173.

A selection of parameters which are specified in the Resource Control Table follows:

1. Subsystem wide values:

   - The CICS subsystem name (connection-ID) under which the CICS system is known to DB2.
   - The name of the DB2 subsystem to which CICS is connected.
   - Maximum number of active threads. Each active thread is an MVS subtask within the CICS address space.
   - The CICS destination for attachment related messages and statistics reports.

2. Values per RCT entry:

   - Name of the application plan, and one or more transaction-ID's for transaction dedicated RCT entries.
   - Number of threads (entries in CCT), and how many of them should be initialized as MVS subtasks at CICS startup. Additional tasks will be set up as required.
   - Dispatching priority limit for thread subtasks. May be specified as higher, equal, or lower than the CICS priority. CICS does its own subtask management for transactions, and will not give up control until all scheduled work is done, unless threads have higher priority than CICS.
   - Action to be taken when no thread is available: wait, try pool, or abend.
   - Authorization-ID to be used for the thread. This may be specified as USER (the sign-on-ID of a terminal user), TERM (the terminal-ID), TXID (transaction-ID), CICS-ID, or a constant.

A thread to DB2 is created when an entry in the CCT is selected. The thread remains active as long as the same application plan is accessed, or until inactive threads must be terminated because the maximum number of threads has been reached.

Authorization

The **Authorization-ID** for a CICS transaction is any of the following:

- User-ID specified in the CICS sign-on transaction
- Terminal-ID
- Transaction-ID
- CICS-ID
- Constant

A different authorization-ID may be selected for each entry of the Resource Control Table. A list like (USER,TERM,TXID) may be specified to select an authorization-ID to be used. In this case the USER-ID is used if the terminal user has used CICS sign-on, the terminal-ID is used otherwise, and the transaction-ID is used when the transaction is not connected to a terminal.

DB2 checks the authorization-ID before accessing an application-plan or executing a DB2 command. A user must be authorized (GRANTED) to all the application plans he is accessing. DB2 authorization must also be granted for users entering DB2 commands from CICS terminals. CICS should check the authorization-ID, DB2 cannot validate it since no password is available to DB2.

APPLICATION PROGRAMMING

An application program in the CICS environment uses the techniques presented in "Chapter 7. Application Programming Environment." Accesses to DL/I and other CICS data may be intermixed with accesses to DB2 data. DB2 data is available only through the command level interface using EXEC SQL statements.

The two SQL statements COMMIT and ROLLBACK must not be used in a CICS application program:

- **COMMIT** is automatically invoked at normal program termination, or through a CICS synchronization point call (SYNCPOINT).

- **ROLLBACK** is automatically invoked at abnormal program termination, or through a CICS rollback call (ROLLBACK).

An application program will receive an SQL return code when DB2 data is not available (data base stopped), or when authorization does not allow the access. These situations may be reflected back to the terminal by the application program.

Application programs must be link-edited with a DB2 language interface module (DSNCLI) if they access DB2 data; this in addition to the DL/I language interface module if they access DL/I data.

Transaction Processing

Communication with a CICS terminal uses the normal CICS mechanism. Transaction input is received using the CICS RECEIVE function, and messages are sent to terminals using the CICS SEND function.

The CICS end user at the terminal should note no differences if a transaction accesses DB2 data. No messages relating to DB2 are sent to the end user by CICS.

Application programs running transactions which use DB2 data may be displayed using the DB2 DISPLAY THREAD command:

```
-DISPLAY THREAD (cicsid)
```

Commit

As presented above, COMMIT of changes of DB2 resources is invoked through a CICS synchronization point. CICS uses the **two phase commit** protocol to synchronize committing of DL/I changes (or other CICS data) and DB2 changes. This protocol was introduced in "Two Phase Commit" on page 155 (CICS is the coordinator, DB2 is a participant).

After CICS has decided to commit changes and commit phase 2 is started, both CICS and DB2 will commit the changes and make them available to other users. If any system should abend during commit phase 2 then commit is finished at restart time.

COMMIT ends a unit of recovery. Data which was changed is made available to other applications and users. Note, however, if locking is at the level of the table space (see section "Defining a Table Space" on page 47 in "Chapter 5. Data Definition"), locks are freed only when the application program is terminated.

In-doubt Threads

A thread is **in-doubt** from the moment DB2 has voted positively in commit phase 1, until the notification is received from CICS to execute commit phase 2. If during that time interval either CICS or DB2 fails the thread remains in-doubt until both systems are up and reconnected again.

The thread is in-doubt because DB2 does not know what CICS's final decision was relative to the commit process. CICS may already have committed or rolled back the DL/I changes, or will finish its process when being restarted after a failure. When both systems are up again they communicate to resolve the in-doubt situations. DB2 will then either commit or roll back the changes based on the CICS decision.

In-doubt threads can be displayed through:

```
-DISPLAY THREAD (cicsid) TYPE (INDOUBT)
```

In case of a CICS cold start it may be required that the system administrator resolves in-doubt threads through a DB2 command:

```
-RECOVER INDOUBT (cicsid) ACTION (COMMIT|ABORT)
         ID (correlation-ID)
```

Application Testing

The SQL portion of a program may be tested using SPUFI from a TSO SPF terminal. This may even be done before the program is coded. Individual SQL statements may be submitted, or streams of SQL requests may be passed to DB2 through SPUFI. Different combinations of the same SQL function may be tested against each other, and the best then selected for actual application coding.

For detailed testing both DB2 and CICS must be up. CICS application programs may not be tested in batch if they access DB2 data.

Execution Diagnostic Facility

The Execution (Command-Level) Diagnostic Facility (EDF) enables an application programmer to test a command-level application program online without making any modifications to the source program or the program preparation procedure. The facility intercepts execution of the program at various points and displays information about the program at these points. Also displayed are any screens sent by the user program, so that the programmer can converse with the application program during testing just as a user would on the production system.

The EDF may thus be used to trace all the SQL calls issued by a CICS application program. Information will be displayed at the terminal before each SQL call is passed to DB2, and after DB2 has executed the SQL call.

OPERATION

A CICS system connected to DB2 is controlled by an authorized terminal operator. The connection with DB2 is controlled through CICS commands, and once established may be used to route DB2 commands from CICS to DB2.

The connection between CICS and DB2 requires the activation of the CICS attachment facility. This activation can either be automatic, using the CICS Program List Table (PLT); or manual, using the DSNC STRT command (described below).

The following commands may be used to control the connection between CICS and DB2:

DSNC DISP PLAN plan	Displays the threads being used to access an application plan. The plan name, transaction name, status (active, inactive), and the authorization-ID are displayed.
DSNC DISP TRAN tran	Same as DISP PLAN for a transaction name.
DSNC DISP STAT	Displays statistics collected for each entry in the RCT. These include the number of SQL calls, COMMITs, ROLLBACKs, authorizations, and the number of times more threads were requested than available for this RCT entry. The output of any of the above DISP commands may be routed to a specified destination of the CICS system.
DSNC DISP xxxx dest	The output of the above DSNC DISPlay commands may be routed to another CICS terminal ("dest").
DSNC DISC plan	Disconnects all the threads which access an application plan as soon as no transaction is using the thread. New threads are allowed to be started.
DSNC STOP QUIESCE	Stops the CICS attachment when all currently running transactions have terminated. FORCE in place of QUIESCE will terminate the connection to DB2 immediately regardless of running transactions.
DSNC STRT x	Requests the CICS attachment to be started. The RCT named DSNCRCTx will be loaded, where "x" is the suffix given in the STRT command (default 0). By specifying a suffix a connection to a selected DB2 system may be started.
DSNC MODI TRAN tran n	Changes the maximum number of active threads for the named transaction (RCT entry) to the new value "n".

DSNC MODI DEST old new Changes the CICS destination for attachment related messages and statistics.

The usage of these CICS commands may be controlled through the standard CICS security mechanism. No authorization checking by DB2 is involved.

Issuing DB2 Commands from CICS

Once a connection between CICS and DB2 is established authorized terminal users (by CICS security) may use the CICS **DSNC command** to route a DB2 command to the DB2 system.

```
DSNC -DB2-command
```

The DB2 command is routed to DB2 for processing. DB2 checks if the user is authorized to issue the DB2 command. Responses are routed back to the originating CICS user. The command recognition character (CRC) "-" must be used to route the command to the destination DB2 system.

Examples are:

```
DSNC -DISPLAY DB (data-base-name) SPACE(*)
DSNC -START   DB (data-base-name)
DSNC -DISPLAY THREAD
```

MONITORING THE CICS ATTACHMENT

The following facilities are available to monitor CICS Attach related events:

- The **CICS Auxiliary Trace** facility and the **CICS Monitoring** facility (CMF) may be used to trace and monitor SQL calls issued by a specific CICS application program.
- The **DB2 TRACE** facility monitors events within DB2 and writes records to in-storage tables or GTF. The trace facility is presented in "Chapter 16. Installation and Servicing" section "Trace" on page 190.
- The **CICS DISPLAY Command** provides information on subsystem connections and active threads.
- The **DB2 Accounting** facility collects statistics on an authorization-ID and work basis and writes them to SMF. See "Chapter 13. Monitoring and Accounting" section "Accounting Facility" on page 161.
- The **DB2 Statistics** facility collects system wide statistics about thread connections and usage. These are useful when

only one subsystem is connected to DB2. See "Chapter 13. Monitoring and Accounting" section "Statistics Facility" on page 160.

In this chapter we look at the DB2 installation process, how system parameters may be modified after installation, and at how the installed product is serviced. Special consideration is given to the IMS and CICS attachments.

PREREQUISITES

The prerequisites listed below are required for DB2 execution.

Hardware

Processor DB2 operates on 308x, 303x, S/370 158/168, or 4300 systems given sufficient real storage to satisfy the combined requirements of DB2, the host operating system (MVS or MVS/XA), appropriate Data Facility Product, access methods, batch requirements and other customer-required applications. The use of System/370 Dual Address Space hardware is recommended, since the software simulation of this feature may severly impact performance.

DASD DB2 is independent of DASD device type. Any disk devices supported by DFP (Data Facility Product) access methods may be used. DB2 uses disk devices for the active recovery logs, BSDS, and Data base data sets, optionally also for archive logs, image copies, and utility work data sets.

Tape DB2 is independent of tape device type. Tape (or MSS) may be used for archive logs, image copies, unloaded tables, and utility work data sets.

Communication An MVS supported console is used for messages. DB2 operation may be controlled from TSO, IMS, or CICS terminals.

Software

Required

- MVS/SP V1 R3 or a subsequent release for the MVS/370 environment
- MVS/SP V2 R1 or a subsequent release for the MVS/XA environment

- DFP/370 (Data Facility Product) R1 for MVS
- DFP/XA (Data Facility Product) R1.1 for MVS/XA
- TSO/E (Extension) R1 or TSO Command Package for MVS
- TSO/E (Extension) R1 for MVS/XA
- OS/VS Sort/Merge R5

Optional (recommended)

- ISPF (Interactive System Productivity Facility) for installation and interactive access, both the Dialog Manager and the Program Development Facility
- RACF (Resource Access Control Facility) R5
- QMF (Query Management Facility)
- DXT (Data Extract)
- IMS V1 R3 or CICS V1 R6 for transaction oriented applications

TAPE CONTENT

DB2 is distributed on four standard label tapes in SMP4 format:

- DB2 tape
- IMS attach tape
- CICS attach tape
- IRLM tape

Each tape contains the following files:

1. SMP control statements and system library allocation JCL.
2. Unloaded libraries containing source of macros and samples. The DB2 tape also contains TSO CLISTs, ISPF panels, and initialization data. No such file on the IRLM tape. **No source of DB2 modules is provided.**
3. Unloaded library containing DB2 object modules.
4. Unloaded library containing TSO HELP information (DB2 tape).

SMP CONSIDERATIONS

DB2 can share SMP data sets with IMS/VS V1 R3 and subsequent releases, unless otherwise specified. (However, DB2 cannot use the same SMP data sets as MVS). If the installation is not using IMS/VS or is using a version prior to IMS/VS V1 R3, unique SMP data sets are required.

INSTALLATION TASKS

The DB2 installation process consists of a series of tasks:

1. Allocate the system data sets: BSDS, active logs, DB2 catalog, directory, and temporary data base.
2. Initialize the catalog and the directory.
3. Initialize the BSDS.
4. Customize MVS for DB2.
5. Assemble and link edit the DB2 initialization module (DSNZPARM) which contains the user specified system parameters (see "Chapter 13. Monitoring and Accounting").
6. Image copy the catalog and the directory.
7. Create a default storage group.

To ease the installation DB2 provides a CLIST which creates jobstreams that perform all of the above tasks. During invocation of the CLIST the user must change the supplied default values to suit installation requirements.

Installation using ISPF

A series of ISPF panels may be used for a tailoring installation of DB2. All the installation parameters are presented on full screen menus for easy selection, and the installation job stream is generated accordingly.

The tailoring session flows through the following panels:

1. **Installation main panel** for specification of data set prefixes and names. Selection of long session (all menus), short session (only selected main parameters, rest default to system supplied values or values specified in an earlier session), or individual panels (for updating). The short session should be adequate for an initial installation.

2. **Installation data parameters** defines the VSAM catalog and volumes for the DB2 subsystem data sets.

3. **Installation sizes** defines a rough number of DB2 objects to be used to calculate the sizes of the DB2 catalog, directory, temporary data base, and region size for the DB2 address spaces.

4. **Installation log data sets** defines the active log processing options (dual, number of data sets, buffer size, etc).

5. **Installation archive log** defines the archive log processing options (dual, device type, space, blocksize, etc).

6. **Installation Boot Strap Data Set** defines the options for the BSDS (dual, dsname, volume).

7. **Installation operator functions** defines the console routing codes, SMF statistics and accounting, trace table size, and system checkpoint frequency.

8. **Installation IRLM** defines the name and options for the IMS Resource Lock Manager. DB2 and IMS data sharing options may be specified.

9. **Installation MVS PARMLIB updates** defines member names and options to be included in the MVS PARMLIB (e.g. DB2 subsystem name, SRC subsystem recognition character, etc).

10. **Installation protection** defines passwords for data set password protection, optionally enables DB2 security, and defines the authorization-ID of the system administrator.

11. **Installation data bases to be started automatically** defines a list of DB2 data bases, table spaces, and indexes to be started automatically when DB2 comes up. This panel is only presented in an update session.

The jobstream generated by this installation process consists of a number of jobs which are stored as members of a partitioned data set. The jobs are run after the tailoring session is ended.

Post Installation Tasks

Post installation tasks include:

- Setting up system security using either RACF or MVS data set password protection.
  - A RACF user-ID for the two DB2 start procedures provides automatic protection of new DB2 data base data sets.
  - Adequate authorization for users of log utilities.
  - Automatic protection of image copies.
- Changing the DB2 supplied user-ID exit, which sets the user-ID used by DB2 to the job card user-ID or TSO logon-ID if RACF is not used.
- Installation of DB2 data base exits for validation or editing of table data.
- SMF conditioning to allow DB2 to generate SMF records for statistics and accounting.

Enabling the TSO/Batch Attachment

The TSO/Batch attachment is an integral part of DB2 that provides access to DB2 data from TSO terminals and batch address spaces. The attachment is enabled by providing:

- access to DB2 provided TSO ISPF panels, CLISTs, and help information.

DB2 authorization must be coordinated with TSO logon-ID's and batch user-ID's (job card).

Batch programs accessing DB2 data must follow the application development cycle presented in "Chapter 7. Application Programming Environment" on page 77. The DB2 TSO language interface module (DSNELI) must be link edited with the program.

IRLM Considerations

The IMS Resource Lock Manager (IRLM) release 2 is used by DB2 for its operation. DB2 uses the IRLM to provide a locking function for DB2 resources.

IMS may also use the IRLM for block level sharing, and if used the IRLM will also provide local locking for IMS. Alternately IMS provides its own locking mechanism (i.e. program isolation). IMS 1.3 requires IRLM release 2 for block level sharing.

- One IRLM may serve both DB2 and IMS for all locking purposes.
- Separate IRLM's may each individually serve DB2 or IMS.
- IRLM may be used for DB2 only, and IMS provides its local locking through program isolation.

When DB2 and the IRLM are installed, IRLM release 2 will replace any IRLM already installed for IMS (unless separate SMP datasets are used for DB2 and IMS).

IMS Considerations

The following tasks must be performed to install and enable the IMS attachment of DB2:

- IMS address spaces must have access to DB2 libraries for the attachment modules which execute in the IMS address spaces.
- Create the subsystem members (SSM) which define the connections between DB2 and IMS.
- Add the SSM parameter to IMS EXECs.
- The application plan names in DB2 are assumed to be identical to the IMS program names. If not, a resource translation table must be created.
- Coordinate DB2 and IMS security. The authorization-ID used by DB2 is the sign-on user-ID or logical terminal name for message driven programs, and the job card USER or PSBNAME for batch programs (BMP).
- Optionally restrict the usage of the IMS /SSR command to route DB2 commands from an IMS terminal to DB2.
- Define programs and transaction using DB2 data to IMS. Application programs must be link edited with the IMS/DB2 language interface module (DFSLI000).
- Test the IMS attachment during Installation Verification (see below).

CICS Considerations

The following tasks must be performed to install and enable the CICS attachment of DB2:

- The CICS address space must have access to DB2 libraries for the attachment modules which execute in the CICS address spaces.
- Define the connections between DB2 and CICS in the Resource Control Table (RCT). The application plan names in DB2 are given in the RCT or default to the transaction name.
- Coordinate DB2 and CICS security. The authorization-ID used by DB2 is specified in the RCT and may be the sign-on user-ID, terminal-ID, transaction-ID, or a constant.
- Increase the OSCORE value in the CICS SIT since each thread is an MVS subtask and requires approximately 1500 bytes.
- Optionally restrict the usage of the CICS DSNC command to route DB2 commands from a CICS terminal to DB2.
- Define the DB2 attachment facilities in the CICS PPT and PCT.
- Define programs and transaction using DB2 data to CICS. Application programs must be link edited with the CICS/DB2 language interface module (DSNCLI).
- Test the CICS attachment during Installation Verification (see below).

Updating Installation Parameters

An update process may be required after DB2 installation to:

- Tune the system
- Change operation/monitoring options
- Change authorization
- Change system data sets
- Change system data bases (catalog)

An ISPF session provides individual panels to modify installation parameters, and generates a job stream to be executed. A new version of the system initialization module may be generated and activated the next time DB2 is started.

INSTALLATION VERIFICATION

The objective of DB2 Installation Verification is to verify the correct installation of DB2. Installation Verification consists of a set of jobs which are intended to be run by an administrator or programmer after installation of DB2. The installer is not required to become familiar with DB2. The jobs are run only to verify the successful installation. The jobs can be run with minimal online script entry. Results can be verified by comparing the output against the installation publications.

Installation Verification is composed of six phases. Most of the phases are batch jobs. The last phase resets the catalog and data base back to their initial state at the end of phase 1. This permits the user to invoke the whole process again. Installation Verification is simply a fixed path through the sample application shipped with DB2.

The six Installation Verification phases are:

- Phase 1 prepares the system. Sample libraries are allocated, DB2 storage groups, data bases, tables, indexes, and views are created. Edit and validation exits are compiled and link-edited. The tables are loaded, and the DB2 utilities for statistics, reorganization, and image copy are executed.

- Phase 2 demonstrates the application development cycle for batch programs written in Assembler Language, COBOL, FORTRAN, or PL/I. Each program runs through precompile, compile, link-edit, bind, and execution.

- Phase 3 is a SPUFI session. Temporary tables are created and loaded. SQL SELECT's and UPDATE's are executed on temporary tables and on the system catalog.

- Phase 4 validates the IMS attachment. Programs, PSB's, transactions, and MFS formats must be defined in IMS. The IMS transactions may then be executed online.

- Phase 5 validates the CICS attachment. Programs, transactions, and maps must be defined in CICS. The CICS transactions may then be executed online.

- Phase 6 restores the DB2 system and catalog. DB2 objects used by the process are dropped, and the status of the system after phase 1 is rebuilt.

SAMPLE APPLICATION

A sample application is shipped with DB2. It consists of several tables, and programs which retrieve and manipulate data in these tables. Subsets of two of the tables (EMPLOYEE and DEPARTMENT) were introduced in "Chapter 3. Relational Data Model" and subsequently used in many examples throughout this book.

Application programs in PL/I, FORTRAN, and COBOL demonstrate many different techniques used to access the tables through SQL statements. Some of the programs are executed during the Installation Verification.

The objectives of the sample application are:

- To provide a common resource for application and system programmers, and all others who are interested in how application programs work in DB2.
- To assist verification that DB2 works after it has been installed.

The sample application consists of three actual applications:

1. The **organization application** manages information about a sample company's employees, departments, and department structure. Two tables for employees and departments are used.
2. The **project application** manages information about the company's projects. This includes project structures, project activities, project staffing by employees, and project estimates and processing. Four tables holding projects, activity types, project - activities, and project - employee assignments are used.
3. The **phone application** manages individual employees phone numbers.

The applications may be used interactively from IMS and CICS, or in batch under TSO. The ISPF interactive support (SPUFI) may be used to retrieve and manipulate data in the sample tables. An additional consistency checking program is provided to match information in the above sample tables. It tells the user if data stored about departments, employees, projects, or activities is missing or appears inconsistently in the sample applications tables.

SERVICEABILITY

DB2 is consistent with the MVS philosophy of establishing recovery exits at each major functional interface. These recovery exits are designed to handle unexpected error conditions. **Functional Recovery Routines (FRR)** are established by all components at initial entry. These routines gain control from MVS in case of an error in that component. The FRR calls a **Procedure Recovery Routine (PRR)** which provides error diagnostics and attempts to recover the component. This mechanism guarantees a specialised routine for each error or exceptional condition.

Problem Determination Tools

DB2 provides a number of problem determination tools to be used in analyzing certain error conditions.

Formatted Dump

Two types of formatted dumps are produced by DB2:

1. The online format dump program executes as an exit of the MVS SNAP/DUMP facility and produces a summarized output of the major DB2 control blocks and data structures. These dumps are recorded in SYSABEND/SYSUDUMP data sets.

2. The DB2 formatting exit of the MVS dump format program (AMDPRDMP) allows the user to format DB2 information from a dump recorded in a SYS1.DUMP data set. A control statement provided to the MVS format program allows to select information to be formatted.

Trace

DB2 has a TRACE facility that traces the control flow at an internal component (resource manager) level. The scope of information traced may be specified as GLOBAL (whole DB2 subsystem), or LOCAL (events associated with certain application plans). Tracing can be done in-storage in a wrap-around table (RESident), or via GTF (GLOBAL only). In-storage tables are formatted by the DB2 formatted dump facility, each entry is 32 bytes long.

The DB2 trace is started through the DB2 start command, and stopped through the stop command (with the same options):

```
-START TRACE (GLOBAL)  DEST (RES | GTF)

-START TRACE (LOCAL)   PLAN (plan-name,...)
```

Trace facilities are also available for IRLM and the attaches.

LOGREC Recording

When DB2 detects a subsystem error the following information is recorded in SYS1.LOGREC:

- Failure type
- DB2 subsystem and component name
- Resource manager and recovery routine name
- Load module name
- Possibly additional information like control blocks, etc.

APPENDIX A. COMPATIBILITY BETWEEN DB2 AND SQL/DS

SQL/DS is a data base product directed at the DOS and VM user. It will normally be run in conjunction with CICS or CMS. A SQL/DS application program interfaces to both CICS or CMS (for system services) and SQL/DS (for data base services) through a SQL interface.

The SQL/DS interface for data base services approximates the SQL supported by DB2. The major differences between the SQL supported by the two products are described below.

Migrating an application program from SQL/DS to DB2 will require little or no modifications if the program uses only the SQL statements supported by both systems. All migration will involve precompilation and recompilation of the program. An installation will choose either:

1. To run the application under CICS which will communicate with DB2. SQL statements which are environmental under SQL/DS, and not part of the common subset, will have to be converted to CICS statements.

2. To convert the application to run as an IMS application and issue DL/I calls for system services and communication. This approach involves more changes than the first approach.

In either of the above approaches the data must be converted from SQL/DS format to DB2 format. The DB2 load utility (see "Load Utility" on page 71) is prepared to accept data from a SQL/DS "unload" data set. In addition the data base declarative information must be entered through the DB2 definitional process. No utilities for descriptor conversion from SQL/DS to DB2 are provided.

SQL/DS Statements and Options not found in DB2

- SQL/DS definitional statements which do not apply to DB2 objects will be diagnosed as errors by the DB2 precompiler:

  - ACQUIRE DBSPACE
  - ALTER DBSPACE
  - DROP DBSPACE
  - LOCK DBSPACE
  - DROP PROGRAM

- Options on GRANT and REVOKE statements which do not apply to DB2 are:

  - RUN privilege to run a program
  - DBA data base administrative privilege
  - RESOURCE privilege to create tables, etc
  - CONNECT privilege to connect program to SQL

- DB2 assumes that environmental operations are carried out via interfaces from the application program to the transaction manager (IMS or CICS). SQL/DS provides SQL statements for these operations. DB2 will generate diagnostics (precompiler) and warning return codes (execution) for these statements:

  - BEGIN DECLARE SECTION
  - END DECLARE SECTION
  - CONNECT
  - UPDATE STATISTICS
  - WHENEVER ... STOP
  - RELEASE option of COMMIT/ROLLBACK

- The following statements are only supported in the TSO/Batch environment of DB2. A return code is generated when issued in IMS or CICS environments:

  - COMMIT
  - ROLLBACK

- DB2 does not allow the user to specify or change the amount of free space within tables or indexes:

  - CREATE INDEX .. PCTFREE
  - ALTER DBSPACE .. PCTFREE

Statements that have different Effects in SQL/DS and DB2

- DROP TABLE drops synonyms in DB2.

- COMMIT and ROLLBACK drops prepared statements in DB2.

- LOCK TABLE locks a table space in DB2. SQL/DS allows the locking of individual tables.

- CREATE INDEX supports different maximum key lengths.

- Comparison of varying length strings of different length. DB2 always pads with blanks, SQL/DS does not. In SQL/DS if the shorter string is identical to the longer string (in the positions of the shorter string) then the shorter string is assumed to be "less" in comparison.

- UPDATE: SQL/DS allows a string literal to update a numeric field.

- The maximum row size is approximately 32K in DB2, but 255 x 32K in SQL/DS.

- In SQL/DS multiple row INSERTs, UPDATEs, and DELETEs are left inserted, updated, or deleted when an error is encountered during execution. The user must use ROLLBACK or WHENEVER STOP to roll back the work. In DB2 the table is unchanged when an error is detected during execution of such a multiple row statement.

- Different number and contents of system catalog tables.

- Implementation dependent codes will be used in the SQLCA feedback area. The layout of the SQLCA is identical.

- Incompatibilities in SQL return codes. Successful execution of SQL statements is highly compatible, but error return codes might be different due to differences in the underlying data management structure and system services.

GG24-1582-00 READER'S COMMENT FORM

IBM DATABASE 2 Concepts and Facilities Guide

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

What is your occupation?

_____

Number of latest TNL applied:

_____

Thank you for your cooperation.

If you wish a reply, please give your name and mailing address:

_____

_____

_____

_____

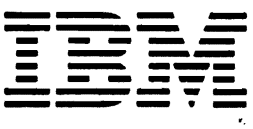
Reader's Comment Form

Cut or Fold Along Line

Fold

IBM INTERNATIONAL SYSTEMS CENTER
Department 471
Building F27
555 Bailey Avenue
P. O. Box 50020
San Jose, California 95150
U.S.A.

Fold



IBM DATABASE 2 Concepts and Facilities Guide GG24-1582 Printed in the U.S.A.

GG24-1582