

DB2

Instructor Notes:

DB2

Lesson 00:

Capgemini 

DB2

Instructor Notes:

Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
05-Nov-2009	1.0	DB2	Arjun Singh	Content Revamp
11-Nov-2009	1.1	DB2	CLS team	Review
30th June'11	2	DB2	Rajita Dhumal	Content Revamp
9th Aug 2016	2.1	DB2	Sarika Katyare	Content Revamp Post integration

Instructor Notes:

Course Goals and Non Goals

➤ Course Goals

- To understand the DB2 relational database management system

➤ Course Non Goals

- NA



Instructor Notes:

Pre-requisites

- COBOL
- MVS
- JCL



Instructor Notes:

Intended Audience

- Developers



Instructor Notes:

Day Wise Schedule

➤ Day 1

- Lesson 1: Getting Started with Database
- Lesson 2: Basics of SQL
- Lesson 3: Data Query Language
- Lesson 4: Aggregate (GROUP) Functions

➤ Day 2

- Lesson 5: SQL (Single-row)
- Lesson 6: Joins and Subqueries



Instructor Notes:

Day Wise Schedule

➤ Day 3

- Lesson 7: DB2 Objects
- Lesson 8: Data Manipulation Language

➤ Day 4

- Lesson 9: Components of DB2
- Lesson 10: Embedded SQL
- Lesson 11: Indicator Variable
- Lesson 12: DB2 Interactive Interface



Instructor Notes:

Day Wise Schedules

➤ Day 5

- Lesson 13: Cursors
- Lesson 14: Errors / Exception Handling
- Lesson 15: Transaction Processing
- Lesson 16: Concurrency and Locking
- Lesson 17: Stored procedure
- Lesson 18: Tools and Utilities for DB2



Instructor Notes:

Table of Contents

➤ **Lesson 1: Getting Started with Database**

- About Database
- Characteristics of DBMS
- DBMS models
- Relational DBMS
- Introduction to DB2
- How does DB2 organize and access information?

➤ **Lesson 2: Basics of SQL**

- SQL, rules for SQL Statements, standard SQL Statement groups
- SPUFI- DB2 Interactive function



Instructor Notes:

Table of Contents

➤ Lesson 3: Data Query Language

- The SELECT statement
- The WHERE clause
- The DISTINCT clause
- The Comparison, Arithmetic, and Logical operators
- The ORDER BY clause

➤ Lesson 4: Aggregate (GROUP) Functions

- Aggregate (Group) functions:
- GROUP BY clause
- HAVING clause



Instructor Notes:

Table of Contents

➤ Lesson 5: SQL (Single-row) Functions

- Number functions
 - Character functions
 - Date functions
 - Conversion functions

➤ Lesson 6: Joins and Subqueries

- Join
 - Inner Join
 - Outer join
 - Self Join
 - Sub-queries
 - UNION operator



Instructor Notes:

Table of Contents

➤ Lesson 7: DB2 Objects

- Storage structure:
 - Storage groups
 - Database
 - Index space
 - Index
 - Synonym
 - View
 - Alias
 - Catalog



Instructor Notes:

Table of Contents

➤ Lesson 8: Data Manipulation Language

- Concept of Data Manipulation Language
- Inserting rows into a table
- Deleting rows from a table
- Updating rows in a table

➤ Lesson 9: Components of DB2

- System structure
- System services
- Locking services
- Database services



Instructor Notes:

Table of Contents

- Lesson 10: Embedded SQL
 - Static SQL
 - Dynamic SQL
 - Embedded SQL
 - Host variables
- Lesson 11: Indicator Variable
 - Indicator variable
- Lesson 12: DB2 Interactive Interface
 - DCLGEN



Instructor Notes:

Table of Contents

➤ Lesson 13: Cursors

- What are cursors?
- Declaring a cursor
- Fetching a row from cursor
- Closing a cursor

➤ Lesson 14: Errors / Exception Handling

- Error Handling
- SQL Communication Area



Instructor Notes:

Table of Contents

➤ Lesson 15: Transaction Processing

- Transactions
- COMMIT and ROLLBACK

➤ Lesson 16: Concurrency and Locking

- Concurrency problems:
- Lost update problem.
- Uncommitted dependency problem.
- Inconsistent analysis problem.
- How DB2 solves these problems.
- Deadlocks



Instructor Notes:

Table of Contents

- **Lesson 17: Stored procedure**
 - Introduction to stored procedure
 - Implementing DB2 Stored Procedures
 - Creating Stored Procedures
 - Executing a Stored Procedure

- **Lesson 18: Utilities and tools for DB2**
 - Loading of DB2 tables
 - Unloading of DB2 tables
 - File Aid for DB2
 - File manager for DB2



Instructor Notes:

References

- A Guide to DB2; by C.J. Date
- The DB2 Experts Guide; by Bruce Larson
- IBM DB2 Application Programming & SQL Guide
- IBM Database 2 General Information
- DB2 Developers Guide; by Craig Mullins
- DB2 for the COBOL programmer; by Curtis Garvin, Steve Eckols



DB2

Lesson 1: Getting
Started with Database

Capgemini

Lesson Objectives

➤ In this lesson, you will learn about:

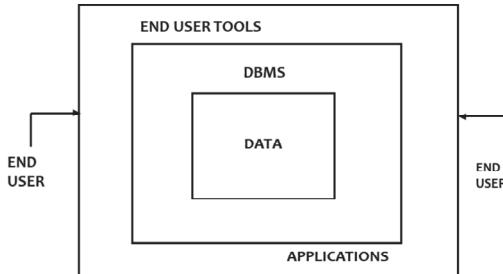
- Database
- Characteristics of DBMS
- DBMS models
- Relational DBMS
- Introduction to DB2
- How does DB2 organize and access information?



1.1: Introduction to Database

Explanation

- **Database:** It is a set of inter-related data.
- **DBMS:** It is a software that manages the data.
- **Schema:** It is a set of structures and relationships, which meet a specific need.



Introduction to Database:

A set of inter-related data is known as "database". The software that manages the database is known as "Database Management System" or "DBMS".

Hence DBMS can be described as "a computer-based record keeping system which consists of software for processing a collection of interrelated data".

A set of structures and relationships that meet a specific need is called as a "schema".

1.2: Characteristics of DBMS

Explanation

➤ Given below are the characteristics of DBMS:

- Control of Data Redundancy
 - Traditionally, same data is stored in a number of places
 - Gives rise to data redundancy and its disadvantages
 - DBMS helps in removing data redundancies by providing means of data- integration.
- Sharing of Data
 - DBMS allows many applications to share the data.
- Maintenance of Integrity
 - DBMS maintains the correctness, consistency, and interrelationship of data with respect to the application, which uses the data.

Characteristics of DBMS:

Some of the characteristics of the DBMS are given below:

Control of Data Redundancy

When the same data is stored in a number of files, it results in data redundancy. In such cases, if the data is changed at one place, the change has to be duplicated in each of the files.

The main disadvantages of data redundancy are:

- Storage space is wasted.
- Processing time may be wasted as more data needs to be handled.
- Inconsistencies may creep in.

DBMS helps in removing redundancies by providing means of integration.

Sharing of Data

DBMS allows many applications to share the data.

Maintenance of Integrity

Integrity of data refers to the correctness, consistency and interrelationship of data with respect to the application that uses the data. Some of the aspects of data integrity are:

- Many data items can only take a restricted set of values.
- Certain field values cannot be duplicated across records. Such restrictions, called primary key constraints, can be defined to the DBMS.
- Data integrity, which defines the relationships between different files, is called referential integrity rule, which can also be specified to the DBMS.

contd.

1.2: Characteristics of DBMS

Explanation

- Support for Transaction Control and Recovery
 - DBMS ensures that updates physically take place after a logical Transaction is complete.
- Data Independence
 - In DBMS, the application programs are transparent to the physical organization and access techniques.
- Availability of Productivity Tools
 - Tools like query language, screen and report painter, and other 4GL tools are available.

Characteristics of DBMS (contd.):

Support for Transaction Control and Recovery

Multiple changes to the database can be clubbed together as a single “logical transaction”.

The DBMS ensures that the updates take place physically, only when the logical transaction is complete.

Data Independence

In conventional file based applications, programs need to know the “data organization” and “access technique” to be able to access the data.

This means that if you make any change in the manner the data is organized, then you have to make changes to the application programs that apply to the data.

In DBMS, the application programs are transparent to the “physical organization” and “access techniques”.

Availability of Productivity Tools

Tools like query language, screen and report painter, and other 4GL tools are available.

These tools can be utilized by the end-users to query, print reports, etc. SQL is one such language, which has emerged as standard.

contd.

1.2: Characteristics of DBMS

Explanation

- Control over Security
 - DBMS provides tools with which the DBA can ensure security of the database.
- Hardware Independence
 - Most DBMS are available across hardware platforms and operating systems.

Characteristics of DBMS (contd.): **Security**

DBMSes provide tools, which can be used by the DBA to ensure security of the database.

Hardware Independence

Most DBMSes are available across hardware platforms and operating systems.

Thus the application programs need not be changed or rewritten when the “hardware platform” or “operating system” is changed or upgraded.

1.3: Levels of Abstraction

Explanation

➤ There are three levels of database abstraction:

- Conceptual Level:
 - The overall integrated structural organization of the database.
- Physical Level:
 - The information about how the database is actually stored in the disk.
- View / External Level:
 - The user view of the database. It is different for different users based on application requirement.



1.4: The Data Models

What is a Data Model?

➤ The “Data model” defines the range of data structures supported and the availability of data handling languages.

- It is a collection of conceptual tools to describe:
 - Data
 - Data relationships
 - Constraints
- There are different data models:
 - Hierarchical Model
 - Network Model
 - Relational Model

What is a Data Model?

A “Data model” is a conceptual representation of the data structures that are required by a database. The data structures include:

- the data objects
- the associations between data objects, and
- the rules which govern operations on the objects

As the name implies, the “Data model” focuses on the data that is required, and how it should be organized rather than the operations that will be performed on the data.

The DBMS MODELS

The range of “data structures” that are supported, and the availability of data handling languages depend on the model of DBMS on which it is based. The models are:

- The hierarchical model
- The network model
- The relational model

1.4: The Data Models

Why is Data Modeling Important?

➤ Why is Data Modeling important?

- The goal of the “data model” is to ensure that all the data objects required by the database are completely and accurately represented.
- The “data model” uses easily understood notations and natural language. Hence, it can be reviewed and verified as correct by the end-users.

Why is Data Modeling Important?

The “data model” is also detailed enough to be used, by the database developers, as a “blueprint” for building the physical databases. The information contained in the “data model” will be used to define the relational tables, primary and foreign keys, stored procedures, and triggers.

Poorly designed databases require more time in the long-term. Without careful planning you may create a database that:

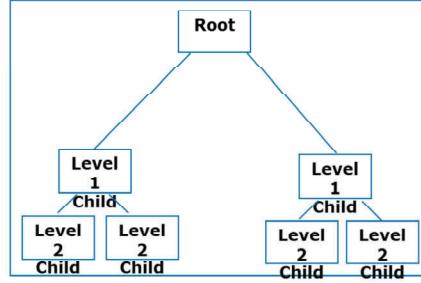
- Omits data required to create critical reports.
- Produces results that are incorrect or inconsistent.
- Is unable to accommodate changes in the user requirements.

1.5: The Hierarchical Model

Explanation

➤ The Hierarchical model:

- In this model, data is represented by a simple tree-structure.
 - Relationships between entities are represented as parent-child.
 - Many-to-many relationships are not allowed.
 - Parents and children are tied together by links called "pointers".



1.5: The Hierarchical Model

Example

➤ Example:

- Consider a student course - marks database.
- In the Hierarchical model a student can register for many courses and gets marks for each course.

- A parent can have many children
- A child cannot have more than one parent
- No child can exist without its parent

Ccode	Cname	Scode	Sname	Marks
C1	Physics	S1	A	65
C2	Chemistry			78
C3	Maths	S2	B	83
C4	Biology			85

Ccode	Cname	Marks
C3	Maths	83
C4	Biology	85

Example of a Hierarchical model:

Consider a student course - marks database. In the Hierarchical model a student can register for many courses, and get marks for each course.

The student record is called as "root". It has got a course - marks record that is called as "child record".

In general:

- A parent can have many children.
- A child cannot have more than one parent.
- No child can exist without its parent.

1.5: The Hierarchical Model

Possibilities

➤ Possibilities in a Hierarchical model:

- **INSERT**
 - Insertion of Dummy student is required to introduce a new course.
- **DELETE**
 - Deleting a student - the only one to take the course deletes course information.
- **UPDATE**
 - To change the course name of one course, the whole database has to be searched. This may result in data inconsistency.

Possibilities in a Hierarchical model:

In the Hierarchical model, following possibilities exist:

INSERT

Since no child record can exist without its parent, it is not possible to insert the new course details without introducing a dummy student record.

DELETE

If a course is selected by only one student, then deleting that student will automatically delete all information about the course.

UPDATE

To change the course name of one course, the whole database has to be searched. This may result in data inconsistency.

1.6: The Network Model



Explanation

➤ The Network model:

- The Network model solves the problem of data redundancy by representing relationships in terms of "sets" rather than "hierarchy".
- A record occurrence may have any number of immediate superiors.
- The Network model supports many-to-many relationships.
- There is no restriction on number of parents.

1.6: The Network Model



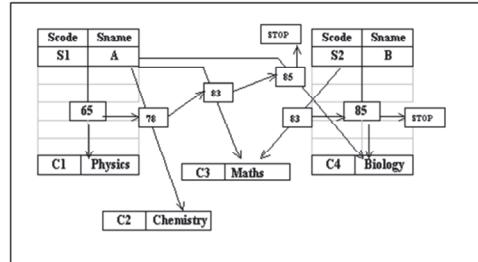
Explanation

- A record type can have a number of parent and child record types.
- It is more complex than the Hierarchical model because of links.
- It is a superset of the Hierarchical model.

1.6: The Network Model

Example

- In the example of student course – marks, “student record” and “course record” is linked together through “marks record”.



Example of Network model:

In the Network model, the “student record” and “course record” is linked together through a “marks record”.

There are no restrictions on number of parents.

A record type can have any number of “parent” and “child” record types.

The Network model is more complex than the Hierarchical model because of it's links.

The Network model can represent any structure that is designed in the Hierarchical model. Hence, it is a superset of the Hierarchical model.

1.6: The Network Model

Possibilities

➤ Possibilities in a Network model:

- **INSERT**
 - Inserting a “course record” or “student record” poses no problems. They can exist without any connectors till a student takes the course.
- **DELETE**
 - Deleting any record automatically adjusts the chain.
- **UPDATE**
 - Update can be done only to a particular child record.

Possibilities in a Network model:

In Network model, following possibilities exist:

INSERT

Inserting a course record or student record poses no problems, as they can exist without any connectors till a student takes the course.

DELETE

Deleting any record automatically adjusts the chain.

UPDATE

Update can be done only to a particular child record.



1.7: The Relational Model



Explanation

➤ The Relational model:

- The Relational model developed out of the work done by Dr. E. F. Codd at IBM in the late 1960s. He was looking for ways to solve the problems with the existing models.
- At the core of the Relational model is the concept of a "table" (also called a "relation"), which stores all data.
- Each "table" is made up of:
 - "records" (i.e. horizontal rows that are also known as "tuples"), and
 - "fields" (i.e. vertical columns that are also known as "attributes")

1.7: The Relational Model

Example

- Examples of RDBMS:
 - DB2
 - Oracle
 - Informix
 - Sybase
- Because of lack of linkages, the Relational model is easier to understand and implement.

Student Table	
Scode	Sname
S1	A
S2	B

Course Table	
Ccode	Cname
C1	Physics
C2	Chemistry
C3	Maths
C4	Biology

Marks Table

Ccode	Scode	Marks
C1	S1	65
C2	S1	78
C3	S1	83
C4	S1	85
C3	S2	83
C4	S2	85

1.7: The Relational Model



Possibilities

➤ Possibilities in a Relational model:

- INSERT
 - Inserting a “course record” or “student record” poses no problems because tables are separate.
- DELETE
 - Deleting any record affects only a particular table.
- UPDATE
 - Update can be done only to a particular table.

1.8: Relational DBMS

Examples

➤ Examples of Relational tables:

Dept table		
Deptno	Dname	Loc
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Operations	Boston

↑
"column" or
"attribute"

Emp table				
Empno	Emplname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30
7566	Jones	Manager	7839	20
7839	King	President		10
7902	Ford	Analyst	7566	20

← "row" or
"tuple"

{ "Intension"
"Extension" }

Relational DBMS (RDBMS):

The Relational model presents an orderly, predictable, and intuitive approach for:

- Organizing data,
- Manipulating data, and
- Viewing data

RDBMS Terminology

Relational data consists of relations.

A relation (or relational table) is a “two dimensional” table with special properties.

A relational table consists of:

- a set of named columns, and
- an arbitrary number of rows

The columns are called as “attributes” or “fields”. The rows are called as “tuples” or “records”.

Each “attribute” is associated with a “domain”.

A “domain” is a set of values that may appear in one or more columns.

1.8: Relational DBMS

Properties

➤ Properties of Relational Data Entities:

- Tables must satisfy the following properties to be classified as relational:
 - Entries of attributes should be single-valued.
 - Entries of attributes should be of the same kind.
 - No two rows should be identical.
 - The order of attributes is unimportant.
 - The order of rows is unimportant.
 - Every column can be uniquely identified.

Properties of Relational Data Entities:

Relational tables have six properties, which must be satisfied for any table to be classified as Relational. These are :

1. Entries of attributes are single valued:
Entry in every row and column position in a table must be single-valued. This means columns do not contain repeating groups.
2. Entries of attribute are of the same kind:
Entries in a column must be of same kind. A column supposed to store “sal” of a employee should not store “comm”.
3. No two rows are identical:
Each row should be unique. This uniqueness is ensured by the values in a specific set of columns called the “Primary key”.
4. The order of attributes is unimportant:
There is no significance attached to order in which columns are stored in the table. A user can retrieve columns in any order.
5. The order of rows is unimportant:
There is no significance attached to the order in which rows are stored in the table. A user can retrieve rows in any order.
6. Every column can be uniquely identified:
Each column is identified by it's “name” and not it's “position”. A column name should be unique in the table.

1.9: Data Integrity

Definition

- “Data Integrity” is the assurance that data is consistent, correct, and accessible throughout the database.

Data Integrity:

Data Integrity refers to the wholeness and soundness of the database.

1.9: Data Integrity

Types

➤ Some of the important integrities are:

- Entity Integrity:
 - It ensures that no "records" are duplicated, and that no "attributes" that make up the primary key are NULL.
 - It is one of the properties that is necessary to ensure the consistency of the database.
- Foreign Key and Referential Integrity
 - The Referential Integrity rule: If a Foreign key in table A refers to the Primary key in table B, then every value of the Foreign key in table A must be null or must be available in table B.

Data Integrity:

Some of the most important integrities are given below.

Domain Constraints

A "domain" is a set of values that are permitted to appear in one or more columns. Once a "domain" is specified and a "column" is associated with the "domain", then the "column" can take only those values that are permitted by the "domain".

Primary Key and Entity Integrity

"Primary key" is a "column" or "set of columns" in a table which uniquely identifies a "row" in a table.

No two rows of the table can have the same values for the Primary key.

Entity integrity is maintained by ensuring that none of the columns that make up the Primary key can take "NULL" (unknown) values.

Foreign Key and Referential Integrity

In the EMP table the deptno field can be considered as a 'foreign key'. This means that the EMP table cannot contain a value for the deptno, which does not exist in the table DEPT.

The Referential Integrity rule specifies that if a Foreign key in table A refers to the Primary key in table B, then every value of the Foreign key in table A must be NULL or must be available in table B.

1.9: Data Integrity

Types

- Unique Constraint:
 - It is a single field or combination of fields that uniquely defines a tuple or row.
 - It ensures that every value in the specified key is unique.
 - A table can have any number of unique constraints, with at most one unique constraint defined as a Primary key.
 - A unique constraint can contain NULL value.

Data Integrity (contd.):

Delete Restrict referential integrity

This means that no Primary key value can be deleted if there are some Foreign key values dependent on it. That is to say, Dept no 10 cannot be deleted because there are some employees assigned to deptno 10.

Delete Cascade referential integrity

This means that if a Primary key value is deleted, then all Foreign key values dependent on it will be deleted. That is to say, if Dept no 10 is deleted, then all rows (employees) having dept no 10 will be deleted.

Update Cascade referential Integrity

This means that if a Primary key value is updated, then all Foreign key values dependent on it will be updated to the new value of Primary key. That is to say, if we change the deptno 10 to 50, then all the employees in deptno 10 will be shifted, as well, to deptno 50 (column deptno will be automatically updated).

Column Constraints

These are the constraints, which specify restrictions on the values that can be taken by a column. These restrictions may be defined with or without other values in the same row.

1.9: Data Integrity

Types

- Column Constraint:
 - It specifies restrictions on the values that can be taken by a column.

DEPT table		
Deptno	Dname	Loc
10	Accounting	New York
20	Research	Dallas

EMP table				
Empno	Emplname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30



1.10: Introduction to DB2



Explanation

- DB2 is the acronym for IBM DATABASE2.
- It is a subsystem of the MVS OS.
- It is IBM's relational DBMS for MVS.
- It allows any number of MVS users to access any number of relational databases by means of the well known relational language that is SQL.
- In Jun 1983, IBM announced the MVS Relational System that is DB2.

Introduction to DB2:

DB2 is a Relational Database Management System (RDBMS) for mainframe operating system. It is a system that allows **MVS** users to build, access, and maintain relational databases, using the well-known relational language **SQL** ("Structured Query Language).

1.10: Introduction to DB2

Explanation

➤ Prior to DB2, IBM's product line included:

- DBMS for MVS operating system namely **IMS**
- RDBMS for VM and VSE namely **SQL/DS**

Introduction to DB2:

IBM's product line prior to DB2 included a non relational DBMS for MVS, namely **IMS**, and a relational DBMS for VM and VSE, namely **SQL/DS**. However, it did not include a relational offering for MVS.

In June 1983, however, the MVS relational product DB2 was announced.

How does DB2 organize and access information?

DB2 is a Database Management System. A more complete description is that DB2 is a Relational Database Management System that uses the industry standard **Structured Query Language (SQL)** as a central component.

A relational database system presents all information in the form of tables. A **table** is just a **two dimensional array** with horizontal rows and vertical columns.

The intersection of a row and a column is a **value**. A value may be a text string, a number, or nothing (null value).

The name **relational database** comes from the technical term for a table, namely a **relation**. In addition, the formal names for the elements of a relation are **tuple** (row) and **attribute** (column). However, these are not the only terms. A row is called a **record**, and a column is called a **field**.

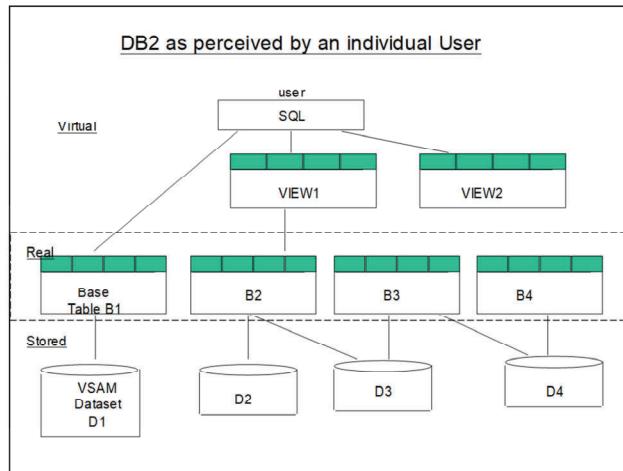
Although **files** and **tables** have similar structures, they are processed differently.

In standard file processing, the unit of processing is an **individual record**.

In DB2, the unit of processing is an **entire table**, which may have one, many, or no rows.

1.11: DB2 as perceived by an Individual User

Explanation



DB2 as Perceived by an Individual User:

The figure demonstrates a pictorial representation of DB2 as it is visible to the users.

Users can be either an **end-user** at an online terminal, or an **application programmer** writing in one of the DB2-supported host languages.

There will be many users all operating on the database at the same time.

DB2 will ensure that one user's update cannot cause another user's operations to produce an incorrect result.

A **base table** is a real table that physically exists, in the sense that there exists physically stored records, and possibly physical indexes, in one or more MVS files.

A **view** is a virtual table, that is a table that does not physically exist, but is visible to the user as if it did. Views act as different ways of looking at base tables.

1.12: Database Administrator

Definition

➤ A Database Administrator (DBA) is the database architect.

- DBA is responsible for the design and implementation of new databases, and:
 - centrally manages the database.
 - decides on the type of data, internal structures, and their relationships
 - ensures the security of the database
 - controls access to the data through user codes and passwords
 - can restrict the views or operations that the users can perform on the database

Database Administrator (DBA)

The database is centrally managed by a person known as the “Database Administrator” or the “DBA”.

The DBA initially studies the System. He decides accordingly the types of data to be used, then the structures to be used to hold the data, and the interrelationships between the data structures. He then defines data to the DBMS.

The DBA also ensures the security of the database. The DBA usually controls access to the data:

- by using the user codes and passwords, and
- by restricting the views or operations that the users can perform on the database

Summary

➤ In this lesson, you have learnt:

- What is a Database?
- Characteristics of DBMS
- The Data models, including:
 - the Hierarchical model
 - the Network model
 - the Relational model
- Relational DBMS (RDBMS)
- What is Data Integrity?
- The concepts of IBM Database 2, known as DB2



Review Questions

➤ Question 1: A DBA ____.

- Option 1: ensures the security of the application server.
- Option 2: controls access to the data through the user codes and passwords.
- Option 3: manages users.



➤ Question 2: The Physical Level is ____.

- Option 1: the overall structural organization of the d/b.
- Option 2: the information about how the database is actually stored in the disk.
- Option 3: the user view of the database.

Review Questions

- Question 3: There are different data models such as _____.
- Question 4: In Network model, each table is made up "tuples" and "fields".
 - True / False
- Question 5: A table can have any number of "Unique constraints".
 - True / False



Review Question – Match the Following

1. Hierarchical model
2. Network model
3. Data redundancy
4. In DBMS

- a. Inconsistencies may creep in.
- b. Many-to-many relationships are not allowed.
- c. It is a superset of the Hierarchical model.
- d. Application programs are transparent to the physical organization and access techniques.



DB2

Lesson 2: Basics of SQL

Capgemini

Lesson Objectives

➤ To understand the following topics:

- SQL, rules for SQL Statements, standard SQL Statement groups
- SPUFI- DB2 Interactive function



2.1: SQL

What is SQL?



➤ SQL:

- SQL stands for Structured Query Language.
- SQL is used to communicate with a database.
- Statements are used to perform tasks such as update data on a database, or retrieve data from a database.
- Benefits of SQL are:
 - It is a Non-Procedural Language.
 - It is a language for all users.
 - It is a unified language.

What is SQL?

SQL stands for Structured Query Language.

SQL is a set of commands that allows you to access a Relational Database.

SQL is an ANSI standard computer language.

SQL can execute queries against a database.

SQL can retrieve data from a database.

SQL can insert new records in a database.

SQL can delete records from a database.

SQL can update records in a database.

SQL is easy to learn.

Almost all modern Relational Database Management Systems like MS SQL Server, Microsoft Access, MSDE, Oracle, DB2, Sybase, MySQL, Postgres, and Informix use SQL as standard database language.

However, although all those RDBMS use SQL, they use different SQL dialects.

For example: MS SQL Server specific version of the SQL is called T-SQL, Oracle version of SQL is called PL/SQL which also supports Procedural Language features, MS Access version of SQL is called JET SQL, etc.

SQL is a non-procedural, English-like language that processes data in “groups of records” rather than processing one record at a time.

SQL provides automatic navigation to the data.

2.2: Rules for SQL Statements

Explanation



➤ Rules for SQL statements:

- SQL keywords are not case sensitive. However, normally all commands (SELECT, UPDATE, etc) are upper-cased.
- “Variable” and “parameter” names are displayed as lower-case.
- New-line characters are ignored in SQL.
- Many DBMS systems terminate SQL statements with a semi-colon character.
- “Character strings” and “date values” are enclosed in single quotation marks while using them in WHERE clause or otherwise.

2.3: Standard SQL Statement Groups

Tabular Representation

- Given below are the standard SQL statement groups:

Groups	Statements	Description
DQL	SELECT	DATA QUERY LANGUAGE – It is used to get data from the database and impose ordering upon it.
DML	DELETE INSERT UPDATE	DATA MANIPULATION LANGUAGE – It is used to change database data.
DDL	DROP CREATE ALTER	DATA DEFINITION LANGUAGE – It is used to manipulate database structures and definitions.
TCL	COMMIT ROLLBACK	TCL statements are used to manage the transactions.
DCL (Rights)	REVOKE GRANT	They are used to remove and provide access rights to database objects.

Standard SQL Statement groups:

“SQL commands” are “instructions” used to communicate with the database to perform “specific tasks” that work with data.

A database is a collection of structures with appropriately defined authorizations and accesses. The tables, indexes are structures in the database and are called as “objects” in the database.

The names of tables, indexes, and those of columns are called “identifiers”.

SQL commands can be used not only for searching the database, but also to perform various other functions.

For example: You can create tables, add data to tables, or modify data, drop the table, set permissions for users, etc.



Standard SQL Statement groups (contd.)

SQL commands are grouped into four major categories depending on their functionality:

Data Definition Language (DDL): These SQL commands are used for creating, modifying, and dropping the structure of database objects. These SQL statements define the structure of a database, including rows, columns, tables, indexes, and database specifics such as file locations, etc. The commands are CREATE, ALTER, DROP,etc

A CREATE statement in SQL creates an object inside a Relational Database Management System (RDBMS) such as table, index, constraints, etc. The types of objects that can be created depend on which RDBMS is being used. However, most support creation of Tables, Indexes, Users, and Databases.

An ALTER statement in SQL changes the properties of an object inside a Relational Database Management System (RDBMS).

Data Manipulation Language (DML): These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

Transaction Control Language (TCL): These SQL commands are used for managing changes that affect the data. These commands are COMMIT, ROLLBACK.

Data Control Language (DCL): These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

GRANT is used to allow specified users to perform specified tasks.

REVOKE is used to cancel previously granted or denied permissions.

2.1: DB2 Interactive Interface (DB2I)



Introduction

➤ The DB2 Interactive interface (DB2i) provides the following:

- Ability to execute SQL statements interactively
- Ability to invoke prewritten application programs
- Ability to issue operator commands
- Ability to invoke database utilities
- Ability to prepare application programs for execution



2.1: DB2 Interactive Interface (DB2I)

Invoking DB2

- Log on to TSO and enter ISPF.
- From ISPF menu, select DB2I. The DB2I main menu will offer following options.
 - SPUFI
 - DCLGEN
 - Program preparation
 - Precompile
 - Bind/rebind/free
 - Run
 - DB2 commands
 - Utilities

2.2: SPUFI

Concept of SPUFI

- SPUFI supports the interactive execution of SQL statements from a TSO terminal.
- You can create a text file containing one or more SQL statements (using the ISPF editor), then execute that file via SPUFI. Subsequently, you can use "ISP Browse" to browse through the results of those statements (which will have been written to another text file).
- It is intended primarily for application programmers who wish to test the SQL portions of their programs.

SPUFI stands for SQL processing using file input



Note: In ISPF primary option menu, select option 8, that is DB2I.



Note: In DB2I primary option menu, select option 1, that is SPUFI.
On SPUFI screen, enter the input data set name (Can be sequential or partitioned)
Specify processing options.
Write a query and enter 'END' command to execute.

Summary

➤ In this lesson, you have learnt:

- What is SQL?
 - Rules for SQL statements
 - Standard SQL statement groups
- SPUFI-DB2 Interactive function



Review Question

➤ Question 1: SQL ____.

- Option 1: cannot execute queries against a database.
- Option 2: can manipulate data from a database.
- Option 3: cannot retrieve data from a database.
- Option 4: can insert new records in a database.
- Option 5: can delete records from a database.



Review Question

➤ Question 2: SQL categories are ____.

- Option 1: DDL
- Option 2: DML
- Option 3: DSL
- Option 4: DQL
- Option 5: TCL
- Option 6: TDL



DB2

Lesson 3: Data Query
Language (The Select
Statement)

Capgemini

Lesson Objectives

➤ To understand the following topics:

- The SELECT statement
 - The WHERE clause
 - The DISTINCT clause
 - The Comparison, Arithmetic, and Logical operators
 - The ORDER BY clause



3.1: The SELECT Statement



Explanation

- The SELECT command is used to retrieve rows from a single table or multiple Tables or Views.
 - A query may retrieve information from specified columns or from all of the columns in the Table.
 - It helps to select the required data from the table.

```
SELECT [ALL | DISTINCT] { * | col_name,...}
  FROM table_name alias, ...
    [ WHERE expr1 ]
    [ GROUP BY expr4 ] [ HAVING expr5 ]
    [ UNION SELECT ... ]
    [ ORDER BY expr | ASC | DESC ];
```

The SELECT Statement:

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set). The statement begins with the SELECT keyword. The basic SELECT statement has three clauses:

```
SELECT
  FROM
  WHERE
```

The SELECT clause specifies the table columns that are retrieved.

The FROM clause specifies the tables accessed.

The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

Note:

Each clause is evaluated on the result set of a previous clause. The final result of the query will be always a “result table”.

Only FROM clause is essential. The clauses WHERE, GROUP BY, HAVING, ORDER BY, UNION are optional.

All the examples that follow are based on EMP and DEPT tables that are already available.

To select all rows from table dept.

It is NOT necessary to retrieve all the COLUMNS of the table. If LOC is not required, then the following query can be used.

```
SELECT deptno,dname,loc
FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATION	BOSTON

It is NOT necessary to list the column names in the same order as done at the time of table creation. If dept name is to be displayed followed by dept number, then the following query can be used.

```
SELECT deptno,dname FROM dept;
```

If all columns are to retrieved ,then instead of listing the column names, the following query can be used.

```
SELECT dname,deptno FROM dept;
```

When rows are retrieved and displayed, the “column heading” is same as the “column name”. To change the heading from LOC to LOCATION, can be used.

```
SELECT * FROM dept;
```

To list deptno from EMP table.

```
SELECT dname,loc "LOCATION" FROM dept;
```

```
SELECT deptno FROM emp ;
```

DEPTNO
20
30
30
20
30
30
10
20
10
14 rows selected

3.2: The WHERE Clause



Explanation

- The WHERE clause is used to specify the criteria for selection.
 - For example: List all employees, who belong to department number 20.
- ```
SELECT empno,ename,deptno FROM Emp
WHERE deptno=20;
```

| EMPNO | ENAME | DEPTNO |
|-------|-------|--------|
| 7369  | SMITH | 20     |
| 7566  | JONES | 20     |
| 7788  | SCOTT | 20     |
| 7876  | ADAMS | 20     |
| 7902  | FORD  | 20     |

### The WHERE Clause:

The WHERE clause is used to perform “selective retrieval” of rows. It follows the FROM clause, and specifies the search condition.

The result of the WHERE clause is the row or rows retrieved from the Tables, which meet the search condition.

The clause is of the form:

WHERE <search condition>

### Comparison Predicates:

The Comparison Predicates specify the comparison of two values.

It is of the form:

< Expression > < operator > < Expression >  
                   < Expression > <operator> <subquery>

The operators used are:

|    |                          |
|----|--------------------------|
| =  | Equal to                 |
| <> | Not equal to             |
| <  | Less than                |
| >  | Greater than             |
| <= | Less than or Equal to    |
| >= | Greater than or Equal to |

contd.

3.3: Comparison, Mathematical, and Logical Operators



## Explanation

- Mathematical Operators:

- Examples: +, -, \*, /

- Comparison Operators:

| Operator | Meaning                  |
|----------|--------------------------|
| =        | Equal to                 |
| >        | Greater than             |
| >=       | Greater than or Equal to |
| <        | Less than                |
| <=       | Less than or Equal to    |
| <>       | Not Equal to             |

- Logical Operators:

- Examples: AND, OR, NOT

### Operators:

Operators are used in “expressions” or “conditional statements”. They show equality, inequality, or a combination of both.

Operators are of three types:

mathematical

logical

range (comparison)

These operators are mainly used in the WHERE clause, HAVING clause in order to filter the data to be selected.

Mathematical operators:

These operators add, subtract, multiply, divide, and compare equality of numbers and strings. They are +, -, \*, /

Comparison Operators:

These operators are used to compare the column data with specific values in a condition. “Comparison Operators” are also used along with the “SELECT statement” to filter data based on specific conditions. The table in the slide describes each Comparison operator. Comparison operators indicate how the data should relate to the given search value.

Logical Operators:

There are three Logical Operators namely AND, OR and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output or not. When retrieving data by using a SELECT statement, you can use logical operators in the WHERE clause. This allows you to combine more than one condition.

3.3: Comparison, Mathematical, and Logical Operators



## Other Comparison Operators

| Other Comparison operators | Description                                                                                                                                                                                                                                                                                                                        |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [NOT] IN(x,y,...)          | Is similar to the OR logical operator. Can search for records which meet at least one condition contained within the parentheses.<br>For example: Pubid IN (1, 4, 5), only books with a publisher id of 1, 4, or 5 will be returned. The optional NOT keyword instructs DB2 to return books not published by Publisher 1, 4, or 5. |
| [NOT] LIKE                 | Can be used when searching for patterns if you are not certain how something is spelt.<br>For example: title LIKE 'TH%'. Using the optional NOT indicates that records that do contain the specified pattern should not be included in the results.                                                                                |
| IS[NOT]NULL                | Allows user to search for records which do not have an entry in the specified field.<br>For example: Shipdate IS NULL.<br>If you include the optional NOT, it would find the records that do not have an entry in the field.<br>For example: Shipdate IS NOT NULL.                                                                 |



3.3: IN Operator

## Explanation

- The IN operator matches a value in a specified list.

- The List must be in parentheses.
- The Values must be separated by commas.

```
SQL> 2 SELECT title, pubid
 FROM books
 3 WHERE pubid IN (1, 2
 TITLE, 5);
```

```
REVENGE OF MICKEY 1
BUILDING A CAR WITH TOOTHPICKS 2
E-BUSINESS THE EASY WAY 2
PAINLESS CHILD-REARING 5
BIG BEAR AND LITTLE DOVE 5
HOW TO MANAGE THE MANAGER 1
SHORTEST POEMS 5
7 rows selected.
```

**IN predicate:**

It is of the form:

<Expression> IN <LIST>  
<Expression> IN <SUBQUERY>

The data types should match.

To get all the employees from deptno 10 and 20:

```
SELECT empno, ename, sal, deptno FROM emp
WHERE deptno in (10, 20);
```

| EMPNO | ENAME  | SAL  | DEPTNO |
|-------|--------|------|--------|
| 7369  | SMITH  | 800  | 20     |
| 7566  | JONES  | 2975 | 20     |
| 7782  | CLARK  | 2450 | 10     |
| 7788  | SCOTT  | 3000 | 20     |
| 7839  | KING   | 5000 | 10     |
| 7876  | ADAMS  | 1100 | 20     |
| 7902  | FORD   | 3000 | 20     |
| 7934  | MILLER | 1300 | 10     |



## 3.3: LIKE Operator

## Explanation

➤ The LIKE operator performs pattern searches.

- The LIKE operator is used with wildcard characters.
  - Underscore (\_) for exactly one character in the indicated position
  - Percent sign (%) to represent any number of characters

```
SQL> SELECT isbn.title
 2 FROM books
 3 WHERE isbn LIKE '_4%0';
ISBN TITLE
```

```

3437212490 COOKING WITH MUSHROOMS
2491748320 PAINLESS CHILD-REARING
```

LIKE predicate:

It is of the form:

<COLUMN > LIKE < PATTERN >

The pattern contains a search string along with other special characters % and \_. The % character represents a string of any length where as \_ (underscore) represents exactly one character.

A pattern %XYZ% means search has to be made for string XYZ in any position. A pattern '\_XYZ%' means search has to be made for string XYZ in position 2 to 4.

To list all employees whose name begins with "J":

```
SELECT ename FROM emp WHERE ename LIKE 'J%';
```

| ENAME |
|-------|
| JONES |
| JAMES |



## 3.4: Logical Operators

## Explanation

➤ Logical operators are used to combine conditions.

- Logical operators are NOT, AND, OR.
  - NOT reverses meaning.
  - AND both conditions must be true.
  - OR at least one condition must be true.

```
SQL> SELECT title, pubid, category
2 FROM books
3 WHERE pubid = 3
4 AND category = 'COMPUTER';

TITLE 3 PUBID CATEGORY
 4 AND category = 'COMPUTER';

DATABASE IMPLEMENTATION 3 COMPUTER
HOLY CRAL OF DB2 3 COMPUTER
HANDCRANKED COMPUTERS 3 COMPUTER
```

One More Example:

```
SQL> SELECT title, pubid, category
2 FROM books
3 WHERE pubid = 3
4 AND category = 'COMPUTER';
```

Combining Predicates by using Logical Operators:

The predicates can be combined by using logical operators like AND, OR, NOT. The evaluation proceeds from left to right and order of evaluation is:

\* Enclosed in parenthesis

AND

OR

To list employees working in dept NUMBER 10 or 20, give the following command:

```
SELECT ename "NAME",deptno
 FROM emp
 WHERE deptno = 10 OR deptno = 20;
```

3.4: Logical Operators



## Using AND or OR Clause

### ■ Use of AND or OR clause:

- For example: Display the employees from department number 10 and department number 20 who are 'CLERKS'.

```
SELECT empno, ename, job, deptno FROM Emp
WHERE job = 'CLERK' AND (deptno=10 OR deptno=20)
```

| EMPNO | ENAME  | JOB   | DEPTNO |
|-------|--------|-------|--------|
| 7369  | SMITH  | CLERK | 20     |
| 7876  | ADAMS  | CLERK | 20     |
| 7934  | MILLER | CLERK | 10     |

The AND operator displays a record if both the first condition and the second condition is true.

The OR operator displays a record if either the first condition or the second condition is true.

You can also combine AND and OR as shown in above example. (use parenthesis to form complex expressions).

Note : The “parenthesis” specifies the order in which the operators should be evaluated. Without parenthesis, the AND operator has a stronger binding than the OR operator.

To list employees hired after 01/09/81 and working in dept NUMBER 10, give the following command:

```
SELECT ename ,hiredate,deptno FROM emp
WHERE deptno = 10 AND hiredate >'01-SEP-81';
```

3.4: Logical Operators



## Using NOT Clause

- The NOT operator finds rows that do not satisfy a condition.

- For example: List employees working in depts other than 10.

```
SELECT empno, ename, deptno FROM emp
WHERE NOT deptno = 10;
```

| EMPNO | ENAME | DEPTNO |
|-------|-------|--------|
| 7369  | SMITH | 20     |
| 7566  | JONES | 30     |
| 7788  | SCOTT | 20     |
| 7876  | ADAMS | 20     |
| 7902  | FORD  | 30     |

To list employees working in department other than 10, use

```
SELECT ename FROM emp
WHERE NOT deptno = 10;
```

Note: NOT is a negation operator.

## 3.4: Logical Operators



## Multiple Logical Operators

- In case there are multiple Logical operators, then they are resolved in the order of AND, OR, NOT.

```
SQL> SELECT * FROM books
2 WHERE category ='FAMILY LIFE'
3 OR pubid = 4
4 AND cost>15;
```

| ISBN       | TITLE                         | PUBDATE   | PUBID | COST  | RETAIL | CATEGORY    |
|------------|-------------------------------|-----------|-------|-------|--------|-------------|
| 1059831198 | BODYBUILD IN 10 MINUTES A DAY | 21-JAN-01 | 4     | 18.75 | 30.95  | FITNESS     |
| 0401140733 | REVENGE OF MICKEY             | 14-DEC-01 | 1     | 14.2  | 22     | FAMILY LIFE |
| 2491748320 | PAINLESS CHILD-REARING        | 17-JUL-00 | 5     | 48    | 89.95  | FAMILY LIFE |
| 0200282519 | THE VON WAY TO COOK           | 11-SEP-00 | 4     | 10    | 28.75  | COOKING     |
| 0132149871 | HOW TO GET FASTER PIZZA       | 11-NOV-02 | 4     | 17.85 | 29.85  | SFIFHFI P   |

## 3.4: Logical Operators



## Multiple Logical Operators

➤ In case there are multiple Logical operators, you can use parentheses to override order of evaluation.

```
SQL> SELECT * FROM books
2 WHERE (category = 'FAMILY LIFE'
3 OR pubid = 4)
4 AND cost>15;
```

| ISBN       | TITLE                         | PUBDATE    | PUBID | COST  | RETAIL | CATEGORY    |
|------------|-------------------------------|------------|-------|-------|--------|-------------|
| 1059831198 | BODYBUILD IN 10 MINUTES A DAY | 21-JAN-014 | 18.75 | 30.95 |        | FITNESS     |
| 2491748320 | PAINLESS CHILD-REARING        | 17-JUL-005 | 48    | 89.95 |        | FAMILY LIFE |
| 0299282519 | THE VOK WAY TO COOK           | 11-SEP-00  | 4     | 19    | 28.75  | COOKING     |
| 0132149871 | HOW TO GET FASTER PIZZA       | 11-NOV-02  | 4     | 17.85 | 29.85  | SELF HELP   |

3.4: Logical Operators



## Treatment of NULL Operators

- NULL is the absence of data.
- Treatment of this scenario requires use of IS NULL operator.

```
SQL> SELECT order#
 2 FROM orders
 3 WHERE shipdate IS NULL;
ORDER#
```

```
1012
1015
1016
1018
1019
1020
6 rows selected.
```

NULL predicate:

The NULL predicate specifies a test for NULL values. The form for NULL predicate is:

< COLUMN SPECIFICATION > IS NULL.

< COLUMN SPECIFICATION > IS NOT NULL.

< COLUMN SPECIFICATION > IS NULL returns TRUE only when column has NULL values.

<COLUMN> = NULL cannot be used to compare null values.

To list all employees not entitled for commission:

```
SELECT ename, sal, comm FROM emp WHERE
comm IS NULL;
```

To list all employees who receive commission:

```
SELECT ename,sal,comm FROM emp WHERE comm
IS NOT NULL;
```

To list the BIG BOSS of the company:

Except for BIG BOSS, all employees have a value in the MGR column.

```
SELECT ename, job,salary FROM emp WHERE
mgr IS NULL;
```



3.5: Operator Precedence

## Tabular Representation

- Operator precedence is decided in the following order:

| Levels | Operators                                                          |
|--------|--------------------------------------------------------------------|
| 1      | * (Multiply), / (Division), % (Modulo)                             |
| 2      | + (Positive), - (Negative), + (Add), (+ Concatenate), - (Subtract) |
| 3      | =, >, <, >=, <=, <>                                                |
| 4      | NOT                                                                |
| 5      | OR                                                                 |
| 6      | AND                                                                |
| 7      | ALL, ANY, IN, LIKE                                                 |

### Operator Precedence:

When a complex expression has multiple operators, the operator precedence (or order of execution of operators) determines the sequence in which the operations are performed.

The order of execution can significantly affect the resulting value.

The operators have the precedence levels as shown in the table given in the slide.

An operator on higher levels is evaluated before an operator on lower level.



## 3.6: The DISTINCT Clause

## Explanation

- The SQL DISTINCT clause is used to eliminate duplicate rows.
  - For example: List all jobs in the EMP table. However, job should be displayed only once, even if duplicate values exist for the job.

```
SELECT DISTINCT job FROM Emp;
```

| JOB       |
|-----------|
| ANALYST   |
| CLERK     |
| MANAGER   |
| PRESIDENT |
| SALESMAN  |

The DISTINCT clause:

In the examples discussed so far, some of the values have been repeated. However, by default, all values are retrieved. If you wish to remove duplicate values, then use a query of the following type:

```
SELECT DISTINCT deptno FROM emp;
```

**Retrieval of Constant values by using Dummy Table**

A “SYSDUMMY1” is a table, which is created by DB2 along with the catalogs. It consists of exactly one column, whose name is IBMREQD, and one record. The value of that record is Y.

```
Sql>Select * from SYSIBM.SYSDUMMY1
IBMREQD

Y
```

For example, you can use it for math:

```
SQL>SELECT (319/212)+10 FROM
SYSIBM.SYSDUMMY1;
```

3.7: The ORDER BY Clause



## Explanation

- The ORDER BY clause presents data in a sorted order.
  - It uses an “ascending order” by default.
  - You can use the DESC keyword to change the default sort order.
  - It can process a maximum of 255 columns.
- In an ascending order, the values will be listed in the following sequence:
  - Numeric values
  - Character values
  - NULL values
- In a descending order, the sequence is reversed.

### The Order By Clause:

A query with its various clauses (FROM, WHERE, GROUP BY, HAVING) determines the rows to be selected and the columns. The order of rows is not fixed unless an ORDER BY clause is given.

An ORDER BY clause is of the form:

ORDER BY < Sort list> ASC/DESC

The columns to be used for ordering are specified by using the “column names” or by specifying the “serial number” of the column in the SELECT list.

The sort is done on the column in “ascending” or “descending” order. By default the ordering of data is “ascending” order.

contd.



3.7: The ORDER BY Clause

## Examples

- For example: Display empno, job, deptno ordered on ename in a descending order.

```
SELECT empno, ename, job, deptno FROM Emp
ORDER BY ename DESC;
```

| EMPNO | ENAME  | JOB       | DEPTNO |
|-------|--------|-----------|--------|
| 7934  | MILLER | CLERK     | 10     |
| 7839  | KING   | PRESIDENT | 10     |
| 7900  | JAMES  | CLERK     | 30     |
| 7902  | FORD   | ANALYST   | 20     |
| 7698  | BLAKE  | MANAGER   | 30     |
| 7499  | ALLEN  | SALESMAN  | 30     |
| 7876  | ADAMS  | CLERK     | 20     |

To select all employees sorted department-wise in ascending order, and within the department, salary-wise in descending order:

```
SELECT deptno,ename,sal FROM emp
order by deptno, sal desc;
```

| DEPTNO | ENAME  | SAL  |
|--------|--------|------|
| 10     | KING   | 5000 |
| 10     | CLARK  | 2450 |
| 10     | MILLER | 1300 |
| 20     | SCOTT  | 3000 |
| 20     | FORD   | 3000 |
| 20     | JONES  | 2975 |
| 20     | ADAMS  | 1100 |
| 20     | SMITH  | 800  |
| 30     | BLAKE  | 2850 |
| 30     | ALLEN  | 1600 |
| 30     | TURNER | 1500 |
| 30     | WARD   | 1250 |
| 30     | MARTIN | 1250 |
| 30     | JAMES  | 950  |

**The ORDER BY Clause (contd.):**

To select all employees sorted dept wise in ascending order and within dept salary-wise in descending order for deptno 10 and 20.

```
SELECT deptno,ename,sal FROM emp
WHERE deptno=10 or deptno = 20
ORDER BY deptno,sal DESC;
```

| DEPTNO | ENAME  | SAL  |
|--------|--------|------|
| 10     | KING   | 5000 |
| 10     | CLARK  | 2450 |
| 10     | MILLER | 1300 |
| 20     | SCOTT  | 3000 |
| 20     | FORD   | 3000 |
| 20     | JONES  | 2975 |
| 20     | ADAMS  | 1100 |
| 20     | SMITH  | 800  |

To select all employees along with their annual salary sorted on the basis of the annual salary.

```
SELECT ename, sal * 12 "Annual Salary" FROM emp
order by 2 desc;
```

| ENAME  | ANNUAL SALARY |
|--------|---------------|
| KING   | 60000         |
| SCOTT  | 36000         |
| FORD   | 36000         |
| JONES  | 35700         |
| BLAKE  | 34200         |
| CLARK  | 29400         |
| ALLEN  | 19200         |
| TURNER | 18000         |
| MILLER | 15600         |
| WARD   | 15000         |
| MARTIN | 15000         |
| ADAMS  | 13200         |
| JAMES  | 11400         |
| SMITH  | 9600          |

## 3.8: Tips and Tricks in SELECT Statements



## Quick Guidelines

- It is necessary to always include a WHERE clause in your SELECT statement to narrow the number of rows returned.
  - If you do not use a WHERE clause, then DB2 will perform a table scan of your table, and return all the rows.
  - By returning data you do not need, you cause the SQL engine to perform I/O it does not need to perform, thus wasting SQL engine resources.



### Tips and Tricks in SELECT Statements:

It is necessary to always include a WHERE clause in your SELECT statement to narrow the number of rows returned.

In some case you may want to return all rows. Then not using a WHERE clause is appropriate in this case.

However, if you don't need all the rows to be returned, use a WHERE clause to limit the number of rows returned.

Another negative aspect of a table scan is that it will tend to flush out data pages from the cache with useless data. This reduces ability of the DB2 to reuse useful data in the cache, which increases disk I/O and decreases performance.

## 3.8: Tips and Tricks in SELECT Statements



## Quick Guidelines

- In addition, the above scenario increases network traffic, which can also lead to reduced performance.
- And if the table is very large, a table scan will lock the table during the time-consuming scan, preventing other users from accessing it, and will hurt concurrency.

➤ In your queries, do not return column data that is not required.

- For example:
  - You should not use `SELECT *` to return all the columns from a table if all the data from each column is not required.
  - In addition, using `SELECT *` prevents the use of covered indexes, further potentially decreasing the query performance.

## 3.8: Tips and Tricks in SELECT Statements



## Quick Guidelines

➤ Carefully evaluate whether the SELECT query requires the DISTINCT clause or not.

- The DISTINCT clause should only be used in SELECT statements.
  - This is mandatory if you know that "duplicate" returned rows are a possibility, and that having duplicate rows in the result set would cause problems with your application.
  - The DISTINCT clause creates a lot of extra work for SQL Server.
    - The extra load reduces the "physical resources" that other SQL statements have at their disposal.
  - Hence, use the DISTINCT clause only if it is necessary.



Tips and Tricks in SELECT Statements (contd.):

Some developers, as a habit, add the DISTINCT clause to each of their SELECT statements, even when it is not required.

This is a bad habit that should be stopped.



## 3.8: Tips and Tricks in SELECT Statements

## Quick Guidelines

➤ In a WHERE clause, the various "operators" that are used, directly affect the query performance.

- Given below are the key operators used in the WHERE clause, ordered by their performance. The operators at the top produce faster results, than those listed at the bottom.
  - =
  - >, >=, <, <=
  - LIKE
  - <>
- Use "=" as much as possible, and "<>" as least as possible.



### Tips and Tricks in SELECT Statements (contd.):

#### Use simple operands

Some operators tend to produce speedy results than other operators. Of course, you may not have choice of using an operator in your WHERE clauses, but sometimes you do have a choice.

Using simpler operands, and exact numbers, provides the best overall performance.

If a WHERE clause includes multiple expressions, there is generally no performance benefit gained by ordering the various expressions in any particular order.

This is because the Query Optimizer does this for you, saving you the effort. There are a few exceptions to this, which are discussed further in the lesson.

contd.

**Tips and Tricks in SELECT Statements (contd.):****Don't include code that does not do anything**

This may sound obvious. However, this scenario is seen in some off-the-shelf applications.

For example, you may see code which is given below:

```
SELECT column_name FROM table_name
WHERE 1 = 0
```

When this query is run, no rows will be returned. It is just wasting SQL Server resources.

By default, some developers routinely include code, which is similar to the one given above, in their WHERE clauses when they make string comparisons.

For example:

```
SELECT column_name FROM table_name
WHERE LOWER(column_name) = 'name'
```

**Any use of text functions in a WHERE clause decreases performance.**

If your database has been configured to be case-sensitive, then using text functions in the WHERE clause does decrease performance. However, in such a case, use the technique described below, along with appropriate indexes on the column in question:

```
SELECT column_name FROM table_name
WHERE column_name = 'NAME' or column_name =
'name'
```

This code will run much faster than the first example.

## 3.8: Tips and Tricks in SELECT Statements



## Quick Guidelines

➤ If you use LIKE in your WHERE clause, try to use one or more leading character in the clause, if at all possible.

- For example: Use LIKE 'm%' not LIKE '%m'



➤ Certain operators in the WHERE clause prevents the query optimizer from using an Index to perform a search.

- For example: "IS NULL", "<>", "!=","!>","!<","NOT", "NOT EXISTS", "NOT IN", "NOT LIKE", and "LIKE '%500'"



### Tips and Tricks in SELECT Statements (contd.):

If you use a leading character in your LIKE clause, then the Query Optimizer has the ability to potentially use an Index to perform the query. Thus speeding performance and reducing the load on SQL engine.

However, if the leading character in a LIKE clause is a “wildcard”, then the Query Optimizer will not be able to use an Index. Here a table scan must be run, thus reducing performance and taking more time.

The more leading characters you use in the LIKE clause, it is more likely that the Query Optimizer will find and use a suitable Index.

## 3.8: Tips and Tricks in SELECT Statements



## Quick Guidelines

- Do not use ORDER BY in your SELECT statements unless you really need to use it.
- Whenever SQL engine has to perform a sorting operation, additional resources have to be used to perform this task.



Tips and Tricks in SELECT Statements (contd.):

Don't use ORDER BY in your SELECT statements unless you really need to:

The ORDER BY clause adds a lot of extra overhead.

For example: Sometimes it may be more efficient to sort the data at the client than at the server. In other cases, the client does not even need sorted data to achieve its goal. The key here is to remember that you should not automatically sort data, unless you know it is necessary. Whenever SQL Server has to perform a sorting operation, additional resources have to be used to perform this task. Sorting often occurs when any of the following Transact-SQL statements are executed:

ORDER BY  
GROUP BY  
SELECT DISTINCT  
UNION  
CREATE INDEX (generally not as critical as happens much less often)

In many cases, these commands cannot be avoided. On the other hand, there are few ways in which sorting overhead can be reduced, like:

Keep the number of rows to be sorted to a minimum. Do this by only returning those rows that absolutely need to be sorted.

Keep the number of columns to be sorted to the minimum. In other words, do not sort more columns than required.

Keep the width (physical size) of the columns to be sorted to a minimum.

Sort column with number datatypes instead of character datatypes.

## Summary

➤ In this lesson, you have learnt:

- What is SELECT statement?
- Usage of the following:
  - The WHERE clause
  - The DISTINCT clause
  - The Comparison, Arithmetic, and Logical operators
  - The AND or OR clause
  - The NOT clause
  - The ORDER BY clause



## Review Questions

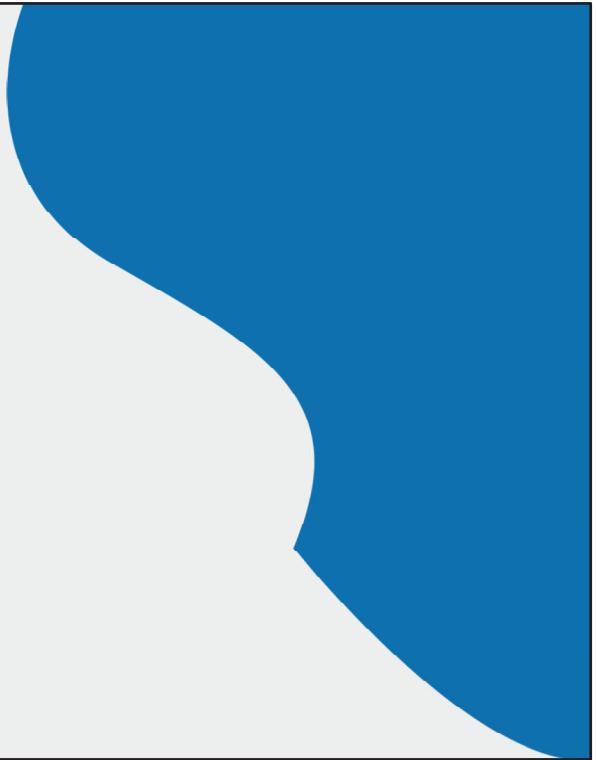
- Question 1: The \_\_\_ table consists of exactly one column, whose name is "dummy".
- Question 2: The LIKE operator comes under the \_\_\_ category.
  - Option 1: mathematical
  - Option 2: comparison
  - Option 3: logical
- Option 3: The \_\_\_ specifies the order in which the operators should be evaluated.



## Review Questions

- Question 4: The NOT NULL operator finds rows that do not satisfy a condition.
  - True / False
  
- Question 5: More than one column can also be used in the ORDER BY clause.
  - True / False





DB2

Lesson 4: Aggregate  
(GROUP) Functions

Capgemini

## Lesson Objectives

➤ To understand the following topics:

- Aggregate (Group) functions:
  - GROUP BY clause
  - HAVING clause



**Functions:**

Functions can be used to manipulate data values in a variety of ways.

Functions help in making the basic query block more powerful.

Functions may be used to perform calculations on data, alter data formats for display, convert data types, etc.

Functions are similar to operators in that they manipulate data items and return a result.

Functions differ from operators in the format in which they appear with their arguments. This format allows them to operate on zero, one, two, or more arguments:

function(argument, argument, ...)

There are two types of functions: SQL functions and User-defined functions.

**SQL functions:** SQL functions are built into most DBMS and are available for use in various appropriate SQL statements. SQL functions are further categorized as:

**Single-Row functions:** Single-row functions return a single result row for every row of a queried Table or View.

**For example:** ABS, ROUND etc.

**Object Reference functions:** Object functions manipulate REFs, which are references to objects of specified object types.

**For example:** REF, VALUE, etc.

**Aggregate functions:** Aggregate functions return a single row based on groups of rows, rather than on single rows. **For example:** MAX, MIN, COUNT, etc.

**User-defined functions:** You can write user-defined functions in PL/SQL or Java to provide functionality that is not available in SQL or SQL functions. User-defined functions can appear in a SQL statement wherever SQL functions can appear, that is, wherever an expression can occur.

**For example:** User-defined functions can be used in the following:

The select list of a SELECT statement

The condition of a WHERE clause

CONNECT BY, START WITH, ORDER BY, and GROUP BY clauses

The VALUES clause of an INSERT statement

The SET clause of an UPDATE statement



4.1: The Group Function

## Explanation

- The Group functions are built-in SQL functions that operate on “groups of rows”, and return one value for the entire group.
- The results are also based on groups of rows.

### Aggregate (Group) Functions:

SQL provides a set of “built-in” functions for producing a single value for an entire group. These functions are called as “Set functions” or “Aggregate (Group) functions”.

These functions can work on a “normal result table” or a “grouped result table”.

If the result is not grouped, then the aggregate will be taken for the whole result table.

4.1: The Group Function

## Explanation

### ➤ Syntax

```
SELECT [column,] aggregate
 function(column),,
 FROM table
 [WHERE condition]
 [GROUP BY column]
 [HAVING condition]
 [ORDER BY column] ;
```





4.2: Group Functions supported by SQL

## Tabular Representation

- Given below is a list of Group functions supported by SQL:

| Function     | Value returned                                                                                                                                 |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| SUM (expr)   | Sum value of expr, ignoring NULL values.                                                                                                       |
| AVG (expr)   | Average value of expr, ignoring NULL values.                                                                                                   |
| COUNT (expr) | Number of rows where expr evaluates to something other than NULL. COUNT(*) counts all selected rows, including duplicates and rows with NULLs. |
| MIN (expr)   | Minimum value of expr.                                                                                                                         |
| MAX (expr)   | Maximum value of expr.                                                                                                                         |

### Aggregate (Group) Functions supported by SQL:

All the above functions operate on a number of rows (for example, an entire table), and are therefore known as “Group (or Aggregate) functions”.

A Group function can be used on a subset of the rows in a table by using the WHERE clause.

The Aggregate functions ignore NULL values in the column.

To include NULL values, NVL function can be used with Aggregate functions.

Note :

Count(\*) : Returns the number of rows in the table, including duplicates and those with NULLs.

Count(<Expression>) : Returns the number of rows where expression is NOT NULL.

Example 1:

Display count of all employees, who have a reporting authority; i.e. who have empno in MGR field.

Select COUNT(mgr) from emp;

Example 2:

Display total count of distinct jobs in the organization

Select COUNT(DISTINCT job) from emp;

**Aggregate (Group) Functions supported by SQL (contd.):****SUM(COL\_NAME | EXPRESSION)**

SUM returns the total of values present in a particular “column” or a “number of columns” that are linked together in the expression. All the columns, which form the argument to SUM, must be numeric only.

To find the sum paid as salary to all employees every month, the following query is used:

```
SELECT SUM(sal) FROM emp;
SUM(SAL)
----- 29025
```

To find the yearly compensation paid to all SALESMEN, the following query is used:

```
SELECT SUM(12*sal) FROM emp
WHERE job = 'SALESMAN';
```

**AVG(COL\_NAME | EXPRESSION)**

AVG is similar to SUM. AVG returns the average of a NUMBER of values. The restrictions, which apply on SUM also, apply on AVG.

To find the average salary of all employees, the following query is used:

```
SELECT AVG(sal) FROM emp;
AVG(SAL)
----- 2073.21429
```

To find the average yearly compensation paid to SALESMEN, the following query is used:

```
SELECT AVG(12*sal) FROM emp
WHERE job = 'SALESMAN';
```

contd.

**Aggregate (Group) Functions supported by SQL (contd.):****COUNT(\*)**

COUNT returns the number of rows.

To find the total number of employees, the following query is used:

```
SELECT COUNT(*) FROM emp;
COUNT(*)
----- 14
```

It is possible to restrict the rows for the operation of COUNT.

To find the total number of CLERKS, the following query is used:

```
SELECT COUNT(*) FROM emp WHERE job = 'CLERK' ;
```

To find the total number of CLERKS hired after '13-jan-81', the following query is used:

```
SELECT COUNT(*) FROM emp
WHERE job = 'CLERK' AND hiredate >'13-jan-81' ;
```

When an aggregate function is used in a SELECT statement, column names cannot be used in SELECT unless GROUP BY clause is used.

**MIN(COL\_NAME | EXPRESSION)**

MIN returns the lowest of the values from the column. MIN accepts columns, which are NON-NUMERIC too.

To find the minimum salary paid to any employee, the following query is used:

```
SELECT MIN(sal) FROM emp;
MIN(SAL)
----- 800
```

To list the employee who alphabetically heads the list, the following query is used:

```
SELECT MIN(ename) FROM emp;
```

contd.

**Aggregate (Group) functions supported by SQL (contd.):****MAX(COL\_NAME | EXPRESSION)**

MAX is the reverse of MIN. MAX returns the maximum value from among the list of values.

To find the maximum salary paid to any employee, the following query is used:

```
SELECT MAX(sal) FROM emp ;
MAX(SAL)
----- 5
```

**Statistical Group functions:**

They are based on normal distribution, and include:

STDDEV  
VARIANCE



4.2: Group Functions supported by SQL

## Examples

- Example 1: Display the total number of employees.

```
SELECT COUNT(*) FROM Emp;
```

- Example 2: Display total salary of all employees.

```
SELECT SUM(SAL) FROM Emp;
```

Note:

The first query returns the value, which counts the number of rows fetched by it from the Emp table.

All the rows in the Emp table are treated as one group.



4.3: The GROUP BY Clause

## Explanation

- GROUP BY clause is used along with the Group functions to retrieve data that is grouped according to one or more columns.

- For example: Find out the number of employees in each department.

```
SELECT deptno, count(*) FROM Emp
GROUP BY deptno;
```

| DEPTNO | COUNT(*) |
|--------|----------|
| 10     | 3        |
| 20     | 5        |
| 30     | 6        |

The GROUP BY Clause:

The GROUP BY clause is of the form:

```
GROUP BY <column list>
```

The columns specified must be selected in the query. If a table is grouped, the SELECT list should have columns and expressions, which are single-valued for a group. These can be:

columns on which grouping is done

constants

aggregate functions on the other columns on which no grouping is done

contd.

**The GROUP BY Clause (contd.):****Note:**

The GROUP BY clause should contain all the columns in the SELECT list, except those used along with the Group functions .

When an Aggregate (Group) function is used in a SELECT statement, the column names cannot be used in SELECT, unless GROUP BY clause is used.

**For example 1:** To calculate the average salary for every department.

```
SELECT deptno, AVG(sal) FROM Emp
 GROUP BY deptno;
```

| DEPTN<br>O | AVG(SAL<br>)  |
|------------|---------------|
| 10         | 2916.666<br>7 |
| 20         | 2175          |
| 30         | 1566.666<br>7 |

**For example 2:** Find the total number of CLERKS hired after “13-jan-81”.

```
SELECT COUNT(*) FROM emp
 WHERE job = 'CLERK' AND hiredate >'13-jan-81' ;
```

**Usage of GROUP BY and HAVING clauses:**

All the SELECT statements we have used until now have acted on data as if the data is in a “single group”. But the rows of data in some of the tables can be thought of as being part of “different groups”.

**For example:** The EMP table contains employees who are CLERKS, SALESMEN, etc. If we wish to find the minimum salary of each group of employees, then none of the clauses that we have seen until now are of any use.

The GROUP BY clause is used to group the result table derived from earlier FROM and WHERE clauses. Further, a HAVING clause is used to apply search condition on these groups.

When a GROUP BY clause is used, each row of the resulting table will represent a group having same values in the column(s) used for grouping.

Subsequently, the HAVING clause acts on the resulting grouped table to remove the row that does not satisfy the criteria in the HAVING search condition.



## 4.4: The HAVING Clause

## Explanation

- HAVING clause is used to filter data based on the Group functions.
  - HAVING clause is similar to WHERE condition. However, it is used with Group functions.
- Group functions cannot be used in WHERE clause. However, they can be used in HAVING clause.

The HAVING Clause:

A HAVING clause is of the form:

HAVING <search condition>

The HAVING search condition applies to “each group”. It can be:

formed using various predicates like between, in, like, null, comparison, etc combined with Boolean operators like AND, OR, NOT

Since the search condition is for a “grouped table”. The predicates should be:

on a column by which grouping is done.

on a set function (Aggregate function) on other columns.

The aggregate functions can be used in HAVING clause. However, they cannot be used in the WHERE clause.

contd.



## 4.4: The HAVING Clause

## Examples

- For example: Display all department numbers having more than three employees.

```
SELECT deptno ,COUNT(*) FROM Emp
GROUP BY deptno HAVING COUNT(*) >3;
```

| DEPTNO | COUNT(*) |
|--------|----------|
| 20     | 5        |
| 30     | 6        |

The HAVING clause (contd.):

When WHERE, GROUP BY, and HAVING clauses are used together in a SELECT statement:

The WHERE clause is processed first in order.

Subsequently, the rows that are returned after the WHERE clause is executed are grouped based on the GROUP BY clause.

Finally, any conditions, on the Group functions in the HAVING clause, are applied to the grouped rows before the final output is displayed.

**More examples of GROUP BY and HAVING Clauses:**

Example 1: To find out Average, Maximum, Minimum salary of departments, where average salary is greater than 2000.

```
SELECT deptno,AVG(sal),MIN(sal),MAX(sal) FROM emp
GROUP BY deptno HAVING AVG(sal) > 2000;
```

| DEPTNO | AVG(SAL)   | MIN(SAL) | MAX(SAL) |
|--------|------------|----------|----------|
| 10     | 2916.66667 | 1300     | 5000     |
| 20     | 2175       | 800      | 3000     |

Only the column names which have been used in GROUP BY clause and aggregate columns can be used in SELECT clause.

Grouping can be done on multiple columns, as well.

To find the minimum salary of various categories of employees in various departments, the following query is used:

```
SELECT deptno,job,MIN(sal) FROM emp
GROUP BY deptno,job;
```

The output is sorted on the basis of deptno. And within each deptno, the output is sorted on the order of job. You can reverse the order of deptno and job and check the changes to the output.

Example 2: In this example, list the minimum salary of various categories of employees, in a department-wise manner, such that minimum salary is greater than 1500.

```
SELECT job,deptno,MIN(sal) FROM emp
GROUP BY job,deptno HAVING MIN(sal) >1500;
```

| JOB       | DEPTNO | MIN(SAL) |
|-----------|--------|----------|
| ANALYST   | 20     | 3000     |
| MANAGER   | 10     | 2450     |
| MANAGER   | 20     | 2975     |
| MANAGER   | 30     | 2850     |
| PRESIDENT | 10     | 5000     |

## 4.5: Tips and Tricks



## Quick Guidelines

➤ Suppose your SELECT statement contains a HAVING clause. Then write your query such that the WHERE clause does most of the work (removing undesired rows) instead of the HAVING clause doing the work of removing undesired rows.



➤ Use the GROUP BY clause only with an Aggregate function, and not otherwise.

- Since in other cases, you can accomplish the same end result by using the DISTINCT option instead, and it is faster.



### Tips and Tricks:

By appropriately using the WHERE clause, you can eliminate unnecessary rows before they reach the GROUP BY and HAVING clause. Thus saving some unnecessary work, and boosting performance.

For example: In a SELECT statement with WHERE, GROUP BY, and HAVING clauses, the query executes in the following sequence.

First, the WHERE clause is used to select the appropriate rows that need to be grouped.

Next, the GROUP BY clause divides the rows into sets of grouped rows, and then aggregates their values.

And last, the HAVING clause then eliminates undesired aggregated groups.

If the WHERE clause is used to eliminate as many of the undesired rows as possible, then the GROUP BY and the HAVING clauses will have to do less work. Thus boosting the overall performance of the query.

contd.

**Tips and Tricks (contd.):**

The GROUP BY clause can be used with or without an Aggregate function.

However, if you want optimum performance, do not use the GROUP BY clause without an Aggregate function. This is because you can accomplish the same end result by using the DISTINCT option instead, and it is faster.

**For example:** You can write your query two different ways:

```
SELECT OrderID
FROM [Order Details]
WHERE UnitPrice > 10
GROUP BY OrderID
or
```

```
SELECT DISTINCT OrderID
FROM [Order Details]
WHERE UnitPrice > 10
```

Both of the above queries produce the same results, but the second one will use less resources and perform faster.

## Summary

➤ In this lesson, you have learnt about:

- Aggregate (Group functions)
  - GROUP BY clause
  - HAVING clause





## Review Questions

➤ Question 1: Identify the various group functions from the list given below:

- Option 1: maximum
- Option 2: stddev
- Option 3: sum
- Option 4: count
- Option 5: minimum





## Review Questions

- Question 2: The AVG function ignores NULL values in the column.
  - True / False
- Question 3: Count(\*) returns the number of rows in the table, including duplicates and those with NULLs.
  - True / False



## DB2

### Lesson 5: SQL (Single-row) Functions

Capgemini 

## Lesson Objectives

➤ To understand the following topics:

- SQL (single-row) functions
  - Number functions
  - Character functions
  - Date functions
  - Conversion functions



## 5.1: SQL Functions



## Explanation

➤ Single-row functions return a single result row for every row of a queried Table or View.

- Single-row functions can appear in SELECT lists, WHERE clauses and HAVING clauses.
- Different categories of single valued functions are:
  - Numerical functions
  - Character functions
  - Date and Time functions
  - Conversion functions
  - Collection functions
  - Miscellaneous Single-row functions

### SQL Functions:

So far, we have seen “Aggregate functions”, which operate against a “collection of values”, however return a “single value”.

Now we shall see “scalar functions” which operate against a “single value”, and return a “single value” based on the input value.

5.2: Scalar Functions



## Explanation

- Scalar Functions Operates on individual rows
- Produces Single value
- Scalar function may be nested
- Column functions may be used as argument of scalar functions
- Scalar functions may be used as arguments of column functions

5.2: Scalar Functions



## Explanation

### ➤ LENGTH

- Returns the length of its arguments

### ➤ SUBSTR

- Returns a substring of a string

### ➤ UPPER

- Converting to uppercase

### ➤ LOWER

- Converting to lowercase

### ➤ VALUE

- The function returns the first argument that is not null

## 5.3: Date and Time Functions

## Explanation



### ➤ DATE

- Return the Date from a valid date argument

### ➤ DAY

- Return the Day from a valid date argument

### ➤ DAYS

- Days functions returns an integer representation of a date

### ➤ MONTH

- Return the Month from a valid date argument

### ➤ YEAR

- Return the Year from a valid date argument

### ➤ TIME

- The TIME function returns a time derived from its argument

**DAY :**

It extracts the day portion of a date or timestamp.

**DAYS :**

It converts a date or timestamp to a number of days.

**DECIMAL :**

It converts a number to decimal representation.

**DIGITS :**

It converts a number (decimal or integer) to a character string representation = CHAR.

**FLOAT :**

It converts a number to a floating point representation.

**HEX :**

It converts a scalar value to a character string representing the internal Hex code.

**HOUR :**

It extracts the hours portion of a time or timestamp.

5.3: Date and Time Functions



## Explanation

### ➤ **TIMESTAMP**

- This function returns a time stamp derived from its argument

### ➤ **MINUTE**

- Return the minutes from valid date

### ➤ **HOUR**

- Return the hour

### ➤ **CHAR**

- Returns string representation of date time value or a decimal value

## 5.3: Date and Time Functions



## Data Type Conversions

### ➤ DECIMAL

- Allows any numeric data type to be specified as the first operand and converted to a decimal value

### ➤ INTEGER

- Returns an integer representation of its numeric argument

### ➤ FLOAT

- Returns a floating-point representation of its numeric argument

### ➤ DIGITS

- Returns a character string representation of the digits in the data item

### ➤ HEX

- Returns a character string result which shows the hexadecimal representation of each byte of the argument

5.4: SUBSTR Function



## Explanation

### ➤ Syntax

```
SUBSTR(STRINGNAME,START,LENGTH)
```

### ➤ Sample Query 1

```
SELECT DEPTNAME,SUBSTR(DEPTNAME,1,4) FROM Q.ORG
```

### ➤ Result

| DEPTNAME       |      |
|----------------|------|
| HEAD OFFICE    | HEAD |
| NEW ENGLAND    | NEW  |
| MID ATLANTIC   | MID  |
| SOUTH ATLANTIC | SOUT |
| GREAT LAKES    | GREA |

### CONCATENATION(||) :

It can be used to concatenate two character strings or two graphic strings (INITIALS || LASTNAME).

### CHAR :

It converts a date, time, or timestamp to its character string representation.

### DATE :

It converts a scalar value to a date.

5.4: SUBSTR Function



## Explanation

### ➤ Sample Query 2

```
SELECT DEPTNUMB,DIVISION FROM Q.ORG WHERE SUBSTR(DIVISION,3)
='STERN'
```

### ➤ Result

| DEPTNUMB | DIVISION |
|----------|----------|
| 15       | EASTERN  |
| 20       | EASTERN  |
| 38       | EASTERN  |
| 66       | EASTERN  |
| 84       | EASTERN  |

### ➤ NOTES

- If length not specified, sub string is to end of the string
- If start is beyond the end of the string - - Error
- If start + length is beyond the end of the string - - Error
- Sub string of null is null

5.5: LENGTH Function



## Explanation

### ➤ Syntax

```
LENGTH(ARGUMENT)
```

### ➤ Sample Query 3

```
SELECT NAME, LENGTH(NAME), JOB, LENGTH(JOB) FROM Q.STAFF
```

### ➤ Result

| NAME     | JOB |       |   |
|----------|-----|-------|---|
| SANDERS  | 7   | MGR   | 5 |
| PERNAL   | 6   | SALES | 5 |
| MARENghi | 8   | MGR   | 5 |

### ➤ NOTE

- If argument is null, length is null

5.6: VALUE Function



## Explanation

### ➤ Syntax

```
VALUE(arg1, arg2, argn)
```

➤ Result is first non-null value in the argument list

➤ Sample Query 4

```
SELECT ID, COMM, VALUE(COMM, 0) FROM Q.STAFF
```

### ➤ Result

| ID | COMM   |        |
|----|--------|--------|
| 10 | -      | 0.00   |
| 20 | 612.45 | 612.45 |
| 1  | -      | 0.00   |
| 40 | 846.55 | 846.55 |

## 5.7: CONVERSIONAL Functions

## Explanation

**➤ DECIMAL, FLOAT, INTEGER**

- Convert Numeric Data

**➤ DIGITS**

- Character Representation of Numeric Value

**➤ HEX**

- Character Representation of Hexadecimal digits

**➤ FLOOR**

- Returns the largest integer value that is less than or equal to the argument

**➤ Sample query 5**

```
SELECT FLOOR(MAX(SALARY)/12)
FROM DSN8710.EMP;
SELECT DECIMAL(SAL,9,1), INTEGER(SAL)
FROM EMP WHERE ID=55;
```

**INTEGER :**

It converts a number to integer represent.

**LENGTH :**

It computes the length of the scalar value in bytes.

**MICROSECOND :**

It extracts the microsecond portion of a timestamp.

**MINUTE :**

It extracts the minute portion of a time or timestamp.

**TIME :**

It converts a scalar value to a time.

**TIMESTAMP :**

It converts either single scalar value or a pair of scalar values (representing a date and time resp) to a timestamp.

**VALUE :**

It converts a null into a non null value.

```
Select custno, fname, lname, value('H : || Homeph, 'W : || Workph, 'No
Phone Number');
```

**VARGRAPHIC :**

It converts a character string in a graphic string.

**YEAR**

It extracts the year portion of a date or timestamp.

## 5.7: Date/Time Scalar Functions



## Explanation

### ➤ Date/Time Scalar Functions

- DAY
- MONTH
- YEAR
- HOUR
- MINUTE
- SECOND
- MICROSECOND

### ➤ Sample Query 6

```
SELECT STARTDT, DAY(STARTDT), MONTH(STARTDT),
 YEAR(STARTDT) FROM DT WHERE YER(STARTD) > 1988
```

| STARTDT    | DAY | MONTH | YEAR |
|------------|-----|-------|------|
| 2001-02-12 | 12  | 02    | 2001 |
| 2002-11-28 | 28  | 11    | 2002 |

## 5.7: Date/Time Scalar Functions



## Explanation

- CHAR FUNCTION to control external format of Date/Time data
- Sample Query 7

```
SELECT TTIME, CHAR(TTIME , USA) FROM DT;
```

| TTIME    |          |
|----------|----------|
| -----    | -----    |
| 12.45.22 | 12:45 PM |
| 02.45.22 | 02:45 AM |
| 15.45.22 | 03:45 PM |

### ➤ Sample

### Query

8

| DT         |            |
|------------|------------|
| -----      | -----      |
| 2006-02-12 | 02/12/2006 |
| 2006-12-12 | 12/12/2006 |
| 2006-12-12 | 12/12/2006 |

```
SELECT DT, CHAR(DT, USA) FROM ZONE;
```

## 5.7: Days Functions



## Explanation

➤ Days functions returns the total days between the specified date and 0001-01-01

```
SELECT DT, DAYS(DT) FROM TEST11;
```

| DT         |        |
|------------|--------|
| -----      | -----  |
| 2006-02-12 | 732354 |
| 2006-12-12 | 732657 |
| 2006-12-12 | 732657 |

```
SELECT DT, DAYS(DT), 'DAYS DIFFERENCE FROM 0001-01-01' FROM TEST11;
```

| DT         |                                        |
|------------|----------------------------------------|
| -----      | -----                                  |
| 2006-02-12 | 732354 Days Difference From 0001-01-01 |
| 2006-12-12 | 732657 Days Difference From 0001-01-01 |
| 2006-12-12 | 732657 Days Difference From 0001-01-01 |

5.7: STRIP Function

## Explanation



➤ The STRIP Function removes blanks or another specified character from the end, the beginning, or at both end of a string expression.

➤ Query 1:

```
SELECT EMPNO ,STRIP(EMPNO,L,'*')"STRIPED LEADING *" FROM
TEST13;
```

➤ Output for Query 1

| EMPNO       | STRIPED LEADING * |
|-------------|-------------------|
| *****A001   | A001              |
| # #####A001 | #####A001         |
| A001*****   | A001*****         |
| ***A001***  | A001***           |

5.7: STRIP Function

## Explanation



### ➤ Query 2

```
SELECT EMPNO , STRIP(EMPNO,T,'*'), 'STRIPED TRAILING' FROM
TEST13;
```

### ➤ Output For Query 2

| EMPNO       | STRIPED TRAILING * |
|-------------|--------------------|
| *****A001   | *****A001          |
| # #####A001 | # #####A001        |
| A001*****   | A001               |
| ***A001***  | ***A001            |
| ***A001# ## | ***A001# ##        |



5.7: STRIP Function

## Explanation

### ➤ Query 3

```
SELECT EMPNO, STRIP(EMPNO, B, '*'), 'STRIPED BOTH *' FROM
TEST13;;
```

### ➤ Output For Query 3

| EMPNO      | STRIPED BOTH * |
|------------|----------------|
| *****A001  | A001           |
| #####A001  | #####A001      |
| A001*****  | A001           |
| ***A001*** | A001           |
| ***A001### | A001###        |

## 5.8: Current Date and Time Value



## Explanation

➤ CURRENT DATE

- Today's Date

➤ CURRENT TIME

- Current Time of Day

➤ CURRENT TIMESTAMP

- Current Date and Time Converted to Time stamp

➤ Example:

- SELECT CURRENT DATE FROM SYSIBM.SYSDUMMY1;

```
-----+-----+-----+-----+-----+-----
```

```
-----+-----+-----+-----+-----+-----
```

2011-06-06

**Numeric Functions (contd.):****FLOOR(n)**

It returns largest integer equal to or less than n.

```
SELECT FLOOR(15.7) "Floor" FROM SYSIBM.SYSDUMMY1;
L;
```

Floor

-----

15

**ABS(n)**

It returns the absolute value of n.

```
SELECT ABS(-15) "Absolute" FROM SYSIBM.SYSDUMMY1;
```

Absolute

-----

15

**POWER function**

It returns m raised to n<sup>th</sup> power ----- power(m,n)

It is of the form:

Power(m,n)

Raised

9

```
SELECT POWER (3,2) "Raised" FROM SYSIBM.SYSDUMMY1;
```

5.9: SQL Functions



## Examples

### ➤ Example 1:

```
SELECT ABS(-15) "Absolute" FROM SYSIBM.SYSDUMMY1; (1);
```

Absolute  
15

### ➤ Example 2:

```
SELECT POWER(3,2) "Raised" FROM SYSIBM.SYSDUMMY1; ;
```

Raised  
9

### Examples of Number Functions:

SYSIBM.SYSDUMMY1; table, which is shown in the slide, is a table owned by SYS.

SYS owns the “data dictionary”, and SYSIBM.SYSDUMMY1; is part of the data dictionary.

SYSIBM.SYSDUMMY1; is a small work-table, which consists of only one row and one column, and contains the value “x” in that column. Besides arithmetic calculations, it also supports “date retrieval” and it’s “formatting”. Often a simple calculation needs to be done. A SELECT must have a table name in its FROM clause, else it fails.

To facilitate such calculations via a SELECT, the SYSIBM.SYSDUMMY1; dummy table is provided.

The structure of the SYSIBM.SYSDUMMY1; table can be viewed by using the SQL statement:

```
DESC SYSIBM.SYSDUMMY1; ;
```



## Examples

➤ Example 3: ROUND(n,m): Returns n rounded to m places

```
SELECT ROUND(17.175,1) "Number" FROM
SYSIBM.SYSDUMMY1; ;
```

Number  
17.2

Examples of Number (numeric) Functions (contd.):  
Round(n,m):

```
SELECT ROUND(17.175,-1) "Number" FROM
SYSIBM.SYSDUMMY1; ;
```

O/P : 20

TRUNC(n,m):

```
SELECT TRUNC(15.81,1) "Number" FROM
SYSIBM.SYSDUMMY1; ;
```

O/P : 15.8

```
SELECT TRUNC(15.81,-1) "Number" FROM
SYSIBM.SYSDUMMY1; ;
```

O/P : 10

## 5.10: Tips and Tricks



## Quick Guidelines

➤ If possible, try avoiding the SUBSTRING function in the WHERE clauses.

- Depending on how it is constructed, using the SUBSTRING function can force a table scan instead of allowing the Optimizer to use an Index (assuming there is one).
- Instead, use the LIKE condition, for better performance.
  - For example: Use the second query instead of using the first query.



```
WHERE SUBSTRING(column_name,1,1) = 'b'
```

```
WHERE column_name LIKE 'b%'
```

### Tips and Tricks:

If possible, try avoiding the SUBSTRING function in your WHERE clauses.

Depending on how it is constructed, using the SUBSTRING function can force a table scan instead of allowing the Optimizer to use an Index (assuming there is one).

If the substring you are searching for does not include the first character of the column you are searching for, then a table scan is performed.

## Summary

➤ In this lesson, you have learnt:

- SQL (single-row) functions
- Character functions
- Number functions
- Date functions
- Conversion functions





## Review Questions

➤ Question 1: Single row functions can be broadly classified as \_\_\_\_.

- Option 1: character functions
- Option 2: numeric functions
- Option 3: Date functions
- Option 4: all the above

➤ Question 2: The function which returns the value after capitalizing the first character is \_\_\_\_.



## DB2

Lesson 6: Joins and Sub queries

Capgemini

## Lesson Objectives

➤ To understand the following topics:

- Join
  - Inner Join
  - Outer join
  - Self Join
- Sub-queries
  - Co-related sub-query
- UNION operator



6.1: Joins

## Explanation



➤ If we require data from more than one table in the database, then a join is used.

- Tables are joined on columns, which have the same “data type” and “data width” in the tables.
- The JOIN operator specifies how to relate tables in the query.
  - When you join two tables a Cartesian product is formed, by default.
- Different types of joins are:
  - Inner / Equi-Join
  - Non-equijoin
  - Outer Join
  - Self Join

### Joins:

JOINS make it possible to select data from more than one table by means of a single statement.

The joining of tables is done in SQL by specifying the tables to be joined in the FROM clause of the SELECT statement.

When you join two tables a Cartesian product is formed.

The conditions for selecting rows from the product are determined by the predicates in the WHERE clause.

All the subsequent WHERE, GROUP BY, HAVING, ORDER BY clauses work on this product.

If the same table is used more than once in a FROM clause then “aliases” are used to remove conflicts and ambiguities. They are also called as “co-relation names” or “range variables”.

contd.

**Joins (contd.):**

Assume two tables:

**TABLE F1**

| <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|
| A           | 1           |
| B           | 2           |
| C           | 3           |

**Table F2**

| <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|
| X           | 100         |
| Y           | 200         |

The statement `SELECT * FROM F1,F2;` results in:

| <u>COL1</u> | <u>COL2</u> | <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|-------------|-------------|
| A           | 1           | X           | 100         |
| B           | 2           | X           | 100         |
| C           | 3           | X           | 100         |
| A           | 1           | Y           | 200         |
| B           | 2           | Y           | 200         |
| C           | 3           | Y           | 200         |

6 rows selected

The statement `SELECT * FROM F1 First_T, F1 Second_T;` results in:

| <u>COL1</u> | <u>COL2</u> | <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|-------------|-------------|
| A           | 1           | A           | 1           |
| B           | 2           | A           | 1           |
| C           | 3           | A           | 1           |
| A           | 1           | B           | 2           |
| B           | 2           | B           | 2           |
| C           | 3           | B           | 2           |
| A           | 1           | C           | 3           |
| B           | 2           | C           | 3           |
| C           | 3           | C           | 3           |

9 rows selected

## 6.1: Joins

## Obtaining data from multiple tables

**EMP table**

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|-------------|-----------|---------------|
| 100         | King      | 90            |
| 101         | Kochhar   | 90            |
| ...         |           |               |
| 202         | Fay       | 20            |
| 205         | Higgins   | 110           |
| 206         | Gietz     | 110           |

**DEPT table**

| DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|---------------|-----------------|-------------|
| 10            | Administration  | 1700        |
| 20            | Marketing       | 1800        |
| 50            | Shipping        | 1500        |
| 60            | IT              | 1400        |
| 80            | Sales           | 2500        |
| 90            | Executive       | 1700        |
| 110           | Accounting      | 1700        |
| 190           | Contracting     | 1700        |

| EMPLOYEE_ID | DEPARTMENT_ID | DEPARTMENT_NAME |
|-------------|---------------|-----------------|
| 200         | 10            | Administration  |
| 201         | 20            | Marketing       |
| 202         | 20            | Marketing       |
| ...         |               |                 |
| 102         | 90            | Executive       |
| 205         | 110           | Accounting      |
| 206         | 110           | Accounting      |

## Obtaining data from multiple tables:

Sometimes you need to use data from more than one table. In the example given in the slide, the report displays data from two separate tables.

Employee IDs exist in the EMP table.

Department IDs exist in both the EMP and DEPT tables.

Location IDs exist in the DEPT table.

To produce the report, you need to link the EMP and DEPT tables and access data from both of them.

6.2: Inner Join / Equijoin



## Explanation

- In an Equijoin, the WHERE statement generally compares two columns from two tables with the equivalence operator “=”.
- This JOIN returns all rows from both tables, where there is a match.

6.2: Inner Join / Equijoin

## Explanation



### ➤ Syntax

```
SELECT <col1>, <col2>,...
FROM <table1>,<table2>
Where <table1>.<col1>=<table2>.<col2>
[AND <condition>] [ORDER BY <col1>, <col2>,...]
```

```
SELECT <col1>, <col2>,...
FROM <table1> LEFT/RIGHT/FULL OUTER JOIN/JOIN <table2>
ON <table1>.<col1>=<table2>.<col2>
[AND <condition>] [ORDER BY <col1>, <col2>,...]
```

### Equijoin:

In the syntax given in the slide:

Column1 in Table1 is usually the Primary key of that table.

Column2 in Table2 is a Foreign key in that table.

Column1 and Column2 must have the same data type, and for certain data types, they should have same size, as well.

6.2: Inner Join / Equijoin

## What is an Equijoin?

**EMP table**

| EMPLOYEE_ID | DEPARTMENT_ID |
|-------------|---------------|
| 200         | 10            |
| 201         | 20            |
| 202         | 20            |
| 124         | 50            |
| 141         | 50            |
| 142         | 50            |
| 143         | 50            |
| 144         | 50            |
| 103         | 60            |
| 104         | 60            |
| 107         | 60            |
| 149         | 80            |
| 174         | 80            |
| 176         | 80            |

...

**DEPT table**

| DEPARTMENT_ID | DEPARTMENT_NAME |
|---------------|-----------------|
| 10            | Administration  |
| 20            | Marketing       |
| 20            | Marketing       |
| 50            | Shipping        |
| 60            | IT              |
| 60            | IT              |
| 60            | IT              |
| 80            | Sales           |
| 80            | Sales           |
| 80            | Sales           |

...

**Foreign key****Primary key****Equijoins:**

To determine an employee's department name, you compare the value in the DEPARTMENT\_ID column in the EMP table with the DEPARTMENT\_ID values in the DEPT table.

The relationship between the EMP and DEPT tables is an "Equijoin", that is values in the DEPARTMENT\_ID column on both tables must be equal.

Frequently, these type of JOIN involves PRIMARY and FOREIGN key complements.

Note: Equijoins are also called simple joins or inner joins.

6.2: Inner Join / Equijoin



## Example

➤ Here is an example of Equijoin:

- In the table EMP, only the numbers of the departments are stored, and not their name.
- In the given example, we now want to retrieve:
  - the name as well as the number for each salesman, and
  - the name of the department where he is working

6.2: Inner Join / Equijoin

## Example

| ENAME  | DEPTNO | DNAME |
|--------|--------|-------|
| ALLEN  | 30     | SALES |
| WARD   | 30     | SALES |
| MARTIN | 30     | SALES |
| TURNER | 30     | SALES |

```
SELECT ename, e.deptno, dname
 FROM Employee e, Department d
 WHERE e.deptno = d.deptno AND job=
 'SALESMAN';
```

Ansi SQL:1999 statndard syntax is as mentioned below for the above query:

```
SELECT ename, e.deptno, dname
 FROM Employee e JOIN Department d
 ON e.deptno = d.deptno AND job= 'SALESMAN';
```

### Equijoin (contd.)

- Typically the tables are joint to get meaningful data.
- Suppose we want to get the department name of all employees along with their names.
  - The employee name is in the employee table and dept name is present in dept table. Hence we have to take the join of two tables on the basis of the column common between these two tables i.e deptno column.

```
SQL> SELECT ename, dname, sal FROM emp,dept
 WHERE emp.deptno = dept.deptno;
```

| ENAME  | DNAME      | SAL  |
|--------|------------|------|
| SMITH  | RESEARCH   | 800  |
| ALLEN  | SALES      | 1600 |
| WARD   | SALES      | 1250 |
| JONES  | RESEARCH   | 2975 |
| MARTIN | SALES      | 1250 |
| BLAKE  | SALES      | 2850 |
| CLARK  | ACCOUNTING | 2450 |
| SCOTT  | RESEARCH   | 3000 |
| KING   | ACCOUNTING | 5000 |
| TURNER | SALES      | 1500 |
| ADAMS  | RESEARCH   | 1100 |
| JAMES  | SALES      | 950  |
| FORD   | RESEARCH   | 3000 |
| MILLER | ACCOUNTING | 1300 |

14 rows selected

- The join is based on the equality of column values in the two tables (emp.deptno= dept.deptno) and therefore is called an Equijoin.
- The Equijoin is also used to provide summary information. Suppose we want to know the details of all departments along with number of employees working in it.
  - Since the number of employees can be found out only through the emp table we have to take a join.
  - Also since we want number of employees per dept, we have to group the product of the join.

```
SQL> SELECT dept.deptno, dept.dname, dept.loc, count(*) "NO of
 employees" FROM dept,emp
 WHERE emp.deptno=dept.deptno
 GROUP BY dept.deptno,dept.dname,dept.loc;
```

| DEPTNO | DNAME      | LOC      | NO of Employees |
|--------|------------|----------|-----------------|
| 10     | ACCOUNTING | NEW YORK | 3               |
| 20     | RESEARCH   | DALLAS   | 5               |
| 30     | SALES      | CHICAGO6 |                 |

- NULL is an unknown and not a value, NULL values never satisfy Equi-join condition.

6.2: Inner Join / Equijoin



## Joining more than two tables

- To join together “n” tables, you need a minimum of “n-1” JOIN conditions.
- For example: To join three tables, a minimum of two joins is required.

**EMP table**

| LAST_NAME | DEPARTMENT_ID |
|-----------|---------------|
| King      | 90            |
| Kochhar   | 90            |
| De Haan   | 90            |
| Hunold    | 60            |
| Ernst     | 60            |
| Lorentz   | 60            |
| Mourgos   | 50            |
| Rajs      | 50            |
| Davies    | 50            |
| Matoe     | 60            |
| Vargas    | 50            |
| Zlotkey   | 80            |
| Abel      | 80            |
| Taylor    | 80            |
| ***       |               |

20 rows selected.

**DEPT table**

| DEPARTMENT_ID | LOCATION_ID |
|---------------|-------------|
| 10            | 1700        |
| 20            | 1800        |
| 50            | 1500        |
| 60            | 1400        |
| 80            | 2500        |
| 90            | 1700        |
| 110           | 1700        |
| 190           | 1700        |

8 rows selected.

**LOC table**

| LOCATION_ID | CITY                |
|-------------|---------------------|
| 1400        | Southlake           |
| 1500        | South San Francisco |
| 1700        | Seattle             |
| 1800        | Toronto             |
| 2500        | Oxford              |

### Additional Search Conditions:

Sometimes you may need to join more than two tables.

For example: To display the last name, the department name, and the city for each employee, you have to join the EMP, DEPT, and LOC tables.

```
SELECT e.last_name, d.department_name, l.city
 FROM employees e, departments d, locations l
 WHERE e.department_id = d.department_id
 AND d.location_id = l.location_id;
```

| LAST_NAME | DEPARTMENT_NAME | CITY                |
|-----------|-----------------|---------------------|
| Hunold    | IT              | Southlake           |
| Ernst     | IT              | Southlake           |
| Lorentz   | IT              | Southlake           |
| Mourgos   | Shipping        | South San Francisco |
| Rajs      | Shipping        | South San Francisco |
| Davies    | Shipping        | South San Francisco |
| ***       |                 |                     |

19 rows selected.

## 6.3: Non-equijoins

# Explanation

**EMP table**

| LAST_NAME         | SALARY |
|-------------------|--------|
| King              | 24000  |
| Kochhar           | 17000  |
| De Haan           | 17000  |
| Hunold            | 9000   |
| Ernst             | 6000   |
| Lorentz           | 4200   |
| Mourgos           | 5800   |
| Rajs              | 3500   |
| Davies            | 3100   |
| Matos             | 2600   |
| Vargas            | 2500   |
| Zlotkey           | 10500  |
| Abel              | 11000  |
| Taylor            | 8600   |
| ...               |        |
| 20 rows selected. |        |

**JOB\_GRADES tables**

| GRA | LOWEST_SAL | HIGHEST_SAL |
|-----|------------|-------------|
| A   | 1000       | 2999        |
| B   | 3000       | 5999        |
| C   | 6000       | 9999        |
| D   | 10000      | 14999       |
| E   | 15000      | 24999       |
| F   | 25000      | 40000       |

Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB\_GRADES table.

**Non-equijoins:**

A Non-equijoin is a JOIN condition containing something other than an equality operator.

The relationship between the EMP table and the JOB\_GRADES table has an example of a Non-equijoin.

A relationship between the two tables is that the SALARY column in the EMP table must be between the values in the LOWEST\_SALARY and HIGHEST\_SALARY columns of the JOB\_GRADES table.

The relationship is obtained using an operator other than equal (=).

**Non-equiijoins:**

- When the comparison operator used in joining columns is other than equality, the join is called a Non Equi-join (Theta Join).
- Suppose we want to find the different pairs of employees doing the same job but belonging to different departments, the following query is used:

```
SQL> SELECT first.ename, first.deptno, second.ename,
 second.deptno,first.job
 FROM emp first , emp second
 WHERE first.empno != Second.empno
 AND first.deptno != second.deptno
 AND first.job=second.job
 ORDER BY 1;
```

| <u>ENAME</u> | <u>DEPTNO</u> | <u>ENAME</u> | <u>DEPTNO</u> | <u>JOB</u> |
|--------------|---------------|--------------|---------------|------------|
| ADAMS 20     | MILLER 10     |              |               | CLERK      |
| ADAMS 20     | JAMES 30      |              |               | CLERK      |
| BLAKE 30     | JONES 20      |              |               | MANAGER    |
| BLAKE 30     | CLARK 10      |              |               | MANAGER    |
| CLARK 10     | JONES 20      |              |               | MANAGER    |
| CLARK 10     | BLAKE 30      |              |               | MANAGER    |
| JAMES 30     | SMITH 20      |              |               | CLERK      |
| JAMES 30     | MILLER10      |              |               | CLERK      |
| JAMES 30     | ADAMS 20      |              |               | CLERK      |
| JONES 20     | CLARK 10      |              |               | MANAGER    |
| JONES 20     | BLAKE 30      |              |               | MANAGER    |
| MILLER10     | SMITH 20      |              |               | CLERK      |
| MILLER10     | ADAMS 20      |              |               | CLERK      |
| MILLER10     | JAMES 30      |              |               | CLERK      |
| SMITH 20     | MILLER10      |              |               | CLERK      |
| SMITH 20     | JAMES 30      |              |               | CLERK      |

16 rows selected

6.4: Outer Join



## Explanation

- If a row does not satisfy a JOIN condition, then the row will not appear in the query result.
- The missing row(s) can be returned if an OUTER JOIN operator is used in the JOIN condition.

### Outer Join:

Outer Joins are similar to Inner Joins. However, they give a bit more flexibility when selecting data from related tables. This type of join can be used in situations where it is desired to select “all rows from the table on the left (or right or both)”, regardless of whether the other table has values in common, and (usually) enter NULL where data is missing.

Outer Join is an exclusive “union” of sets (whereas normal joins are intersection). OUTER JOINs can be simulated using UNIONs.

In a JOIN of two tables an Outer Join may be for the first table or the second table. If the Outer Join is taken on, say the DEPT table, then each row of DEPT table will be selected at least once whether or not a JOIN condition is satisfied.

An Outer Join does not require each record in the two joint tables to have a matching record in the other table. The joint table retains each record — even if there is no other matching record.

Outer Joins subdivide further into “left outer joins”, “right outer joins”, and “full outer joins”, depending on which table(s) one retains the rows from (left, right, or both). contd.



6.4: Outer Join

## Explanation

### ➤ Syntax

```
SELECT <col1>, <col2>,...
FROM <table1> LEFT/RIGHT/FULL OUTER JOIN <table2>
ON <table1>.<col1>=<table2>.<col2>
[AND <condition>]
```

### Outer Join (contd.):

Left Outer Join: A Left Outer Join returns all the values from the left table, plus matched values from the right table (or NULL in case of no matching join predicate).

Right Outer Join: A Right Outer Join returns all the values from the right table and matched values from the left table (or NULL in case of no matching join predicate).

Full Outer Join: A Full Outer Join combines the results of both Left and Right Outer Joins. The joint table will contain all records from both tables, and fill in NULLs for missing matches on either side.

To join "n" tables together, you need a minimum of "n-1" join conditions.

For example: To join three tables, a minimum of two joins is required.

6.4: Outer Join  
**Illustration**

**DEPT table**

| DEPARTMENT_NAME | DEPARTMENT_ID |
|-----------------|---------------|
| Administration  | 10            |
| Marketing       | 20            |
| Shipping        | 50            |
| IT              | 60            |
| Sales           | 80            |
| Executive       | 90            |
| Accounting      | 110           |
| Contracting     | 190           |

8 rows selected.

**EMP table**

| DEPARTMENT_ID | LAST_NAME |
|---------------|-----------|
| 90            | King      |
| 90            | Kochhar   |
| 90            | De Haan   |
| 60            | Hunold    |
| 60            | Ernst     |
| 60            | Lorentz   |
| 50            | Moungos   |
| 50            | Rajs      |
| 50            | Davies    |
| 50            | Matos     |
| 50            | Vargas    |
| 80            | Zlotkey   |

20 rows selected.

**There are no employees in department 190.**



Returning Records with No Direct Match with Outer Joins:

If a row does not satisfy a JOIN condition, the row will not appear in the query result.

For example: In the Equijoin condition of EMP and DEPT tables, employee Grant is not displayed because there is no department ID recorded for her in the EMP table. Instead of displaying 20 employees in the result set, 19 records are displayed.

```
SELECT e.last_name, e.department_id,
 d.department_name
 FROM employees e, departments d
 WHERE e.department_id = d.department_id;
```

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|-----------------|
| Whalen    | 10            | Administration  |
| Hartstein | 20            | Marketing       |
| Fay       | 20            | Marketing       |
| Moungos   | 50            | Shipping        |

...  
19 rows selected.

6.4: Outer Join



## Example

➤ Given below is an example of Outer Join:

- List the department names, whose employees are there in the employee table, and
- List the name of the department, whose employees are not present in the employee table

**Outer Join (contd.):**

- Consider the query for getting all the department details along with the number of employees in it. In an Equijoin, the details of deptno 40 were not shown because there are no employees assigned to it in the EMP table.
- This can be avoided by taking an Outer Join. Outer Join is an exclusive union of sets (whereas normal joins are intersection). Outer Joins can be simulated using UNIONS.
- In a join of two tables an outer join may be for the first table or the second table. If the outer join is taken on say the dept table then each row of dept table will be selected at least once whether or not a join condition is satisfied.
- To get the details of all departments with number of employees assigned to it (even 0 employees)

```
SELECT dept.deptno, dept.dname, dept.loc, count(empno) "No. of
employees" FROM dept LEFT OUTER JOIN emp
ON emp.deptno =dept.deptno
GROUP BY dept.deptno,dept.dname,dept.loc;
```

| <u>DEPTNO</u> | <u>DNAME</u> | <u>LOC</u> | <u>No. of employees</u> |
|---------------|--------------|------------|-------------------------|
| 10            | ACCOUNTING   | NEW YORK   | 3                       |
| 20            | RESEARCH     | DALLAS     | 5                       |
| 30            | SALES        | CHICAGO    | 6                       |
| 40            | OPERATIONS   | BOSTON     | 0                       |

- Suppose there are any employees without deptno assigned to it. If we take the Equijoin, then employees without any dept will not be selected because JOIN condition will not be satisfied for NULL values. To solve this problem we have to take an Outer Join on the DEPT table.
- The query to get all the names, salary, department names of all employees we have to take an Outer Join on EMP table.

```
SELECT ename, sal,dname FROM dept RIGHT OUTER
JOIN emp ON
dept.deptno=emp.deptno;
```

6.5: Self Join



## Explanation

➤ In Self Join, two rows from the "same table" combine to form a "resultant row".

- It is possible to join a table to itself, as if they were two separate tables, by using table labels (aliases).
- This allows joining of rows in the same table.

### Self Join:

To join a table to itself, "two copies" of the same table have to be opened in the memory.

Hence in the FORM clause, the table name needs to be mentioned twice.

Since the table names are the same, the second table will overwrite the first table.  
In effect, this will result in only one table being in memory.

This is because a table name is translated into a specific memory location.  
To avoid this, each table is opened using an "alias".

These two table aliases will cause two identical tables to be opened in different memory locations.

This will result in two identical tables to be physically present in the computer memory.

6.5: Self Join  
**Illustration**

**EMP (WORKER)**

| EMPLOYEE_ID | LAST_NAME | MANAGER_ID |
|-------------|-----------|------------|
| 100         | King      |            |
| 101         | Kochhar   | 100        |
| 102         | De Haan   | 100        |
| 103         | Hunold    | 102        |
| 104         | Ernst     | 103        |
| 107         | Lorentz   | 103        |
| 124         | Mourgos   | 100        |

...

**EMP (MANAGER)**

| EMPLOYEE_ID | LAST_NAME |
|-------------|-----------|
| 100         | King      |
| 101         | Kochhar   |
| 102         | De Haan   |
| 103         | Hunold    |
| 104         | Ernst     |
| 107         | Lorentz   |
| 124         | Mourgos   |

...



**MANAGER\_ID in the WORKER table is equal to EMPLOYEE\_ID in the MANAGER table.**

Joining a Table to itself:

Sometimes you need to join a table to itself.

To find the name of manager of each employee, you need to join the EMP table to itself, or perform a self join.

For example: To find the name of Whalen's manager, you need to:

Find Whalen in the EMP table by looking at the LAST\_NAME column.

Find the manager number for Whalen by looking at the MANAGER\_ID column. Whalen's manager number is 101.

Find the name of the manager with EMPLOYEE\_ID 101 by looking at the LAST\_NAME column. Kochhar's employee number is 101, so Kochhar is Whalen's manager.

In the above process, you look twice in the EMP table.

The first time you look in the table to find Whalen in the LAST\_NAME column and MANAGER\_ID value of 101.

The second time you look in the EMPLOYEE\_ID column to find 101 and the LAST\_NAME column to find Kochhar.

**Self-Join (contd.)**

- In a self-join a Cartesian product is taken on two sets of rows of the same table.

**TABLE F1**

| <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|
| A           | 1           |
| B           | 2           |
| C           | 3           |

```
SELECT * FROM f1 first_f1,f1 second_f1
WHERE first_f1.col1 = second_f1.col1;
```

| <u>COL1</u> | <u>COL2</u> | <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|-------------|-------------|
| A           | 1           | A           | 1           |
| B           | 2           | B           | 2           |
| C           | 3           | C           | 3           |

- To get the names and salaries of all employees, who have managers, and managers name and managers salary.

```
SQL> SELECT a.ename name , a.sal salary , b.ename mgr, b.sal " Mgr
Salary" FROM emp a , emp b WHERE A.mgr = B.empno;
```

| <u>NAME</u> | <u>SALARY</u> | <u>MGR</u> | <u>Mgr Salary</u> |
|-------------|---------------|------------|-------------------|
| SMITH       | 800           | FORD       | 3000              |
| ALLEN       | 1600          | BLAKE      | 2850              |
| WARD        | 1250          | BLAKE      | 2850              |
| JONES       | 2975          | KING       | 5000              |
| MARTIN      | 1250          | BLAKE      | 2850              |
| BLAKE       | 2850          | KING       | 5000              |
| CLARK       | 2450          | KING       | 5000              |
| SCOTT       | 3000          | JONES      | 2975              |
| TURNER      | 1500          | BLAKE      | 2850              |
| ADAMS       | 1100          | SCOTT      | 3000              |
| JAMES       | 950           | BLAKE      | 2850              |
| FORD        | 3000          | JONES      | 2975              |
| MILLER      | 1300          | CLARK      | 2450              |

13 rows selected. \* King is not selected because he does not have a manager.

6.5: Self Join



## Example

➤ Here is an example of Self Join:

- List the names of all employees together with the name of their manager:

```
SELECT e1.ename, e2.ename FROM Employee e1, Employee
e2
WHERE e1.mgr = e2.empno;
```

| ENAME  | ENAME |
|--------|-------|
| SCOTT  | JONES |
| FORD   | JONES |
| ALLEN  | BLAKE |
| WARD   | BLAKE |
| JAMES  | BLAKE |
| MILLER | CLARK |
| ADAMS  | SCOTT |

### Example of Self Join:

In the example shown in the slide, the data is required for all the employees and the names of their respective managers.

This data is available in the EMP table. This table holds the employee number, employee names, and their respective manager numbers, who in turn are employees in the same table.

Thus the EMPNO is the PRIMARY KEY. And the MGR column is used to refer to the employee details in the same table.

Thus MGR is a FOREIGN KEY mapping to the EMPNO, which is the primary key of the table.

It is possible to extract the manager number to which employee reports by using the EMP table. However, to extract the manager name (i.e. employee's name), a reference has to be made to the same table. This can be done by using a Self Join.

6.6: Sub-queries

## Explanation



- A sub-query is a form of an SQL statement that appears inside another SQL statement.
  - It is also called as a “nested query”.
- The statement, which contains the sub-query, is called the “parent statement”.
- The “parent statement” uses the rows returned by the sub-query.

### Sub-queries:

As mentioned earlier, since a basic SQL query returns a relation, it can be used to construct composite queries.

Such a SQL query, which is nested within another higher level query, is called a “sub-query”.

This kind of a nested sub-query is useful in cases, where we need to select rows from tables based on a condition, which depends on the data stored in the table itself.

6.6: Sub-queries

## Explanation



➤ Subqueries can be used for the following purpose :

- To insert records in a target table.
- To create tables and insert records in the table created.
- To update records in the target table.
- To create views.
- To provide values for conditions in the clauses, like WHERE, HAVING, IN, etc., which are used with SELECT, UPDATE and DELETE statements.

6.6: Sub-queries



## Examples

➤ Example 1: To display name and salary of employees who have the same job as the employee 7896.

- Method 1:

```
SELECT job FROM emp WHERE empno=7896;
```

- O/P : CLERK

```
SELECT ename,sal FROM emp WHERE job='CLERK';
```

6.6: Sub-queries



## Examples

➤ Example 1 (contd.):

- Method 2: By using sub-query

```
SQL> SELECT ename, sal FROM emp
 WHERE job = (Select job from emp
 where empno = 7896);
```

➤ Example 2: To display names of all employees, who do the same job as SMITH.

```
SELECT ename FROM emp
WHERE job =
 (SELECT job FROM emp WHERE
ename='SMITH');
```

Examples:

Example 3: To display the details of the employees working in BOSTON.

```
SELECT ename FROM emp
WHERE deptno =(SELECT deptno FROM dept
WHERE loc='BOSTON');
```

**Sub-queries (contd.):**

- When the WHERE clause needs a set of values which can be only obtained from another query, the Sub-query is used. In the WHERE clause it can become a part of the following predicates
  - COMPARISON Predicate
  - IN Predicate
  - ANY or ALL Predicate
  - EXISTS Predicate.
- It can be also used as a part of the condition in the HAVING clause.
- To list the names of all employees working in SALES dept, assuming that deptno of SALES dept is not known.

```
SELECT ename,sal FROM emp
WHERE deptno = (SELECT deptno
 FROM dept
 WHERE dname = 'SALES');
```

| <u>ENAME</u> | <u>SAL</u> |
|--------------|------------|
| ALLEN        | 1600       |
| WARD         | 1250       |
| MARTIN       | 1250       |
| BLAKE        | 2850       |
| TURNER       | 1500       |
| JAMES        | 950        |

- The second SELECT statement is called INNER QUERY or SUB-QUERY. All the rules, which apply to the main query, also apply to the sub-query. The inner query is executed FIRST and ONLY ONCE. The inner query returns a single value, namely 30. This value is used to evaluate the main query. Notice that the table used in inner query can be different from the table used in main (also called OUTER) query.
  - The sub-query cannot have an ORDER BY clause except for inline views and UNION of selects.
  - The sub-query may select only one column except in the case of EXISTS predicate.
- There are two types of sub-queries: Co-related and Non Co-related Sub-queries.
  - If the Sub-query uses any data from the FROM clause of the outer query, then the sub-query is evaluated for each row of the outer query. The Sub-query thus has to be repeated for each row of the outer query and is called as “co-related sub-query”.
  - When no data is needed from the outer query, the sub-query is evaluated only once. The Sub-query is executed, and the result set is obtained. Such a sub-query is called as “non co-related sub-query”.

6.6: Sub-queries

## Sub-queries using Comparison operators

- Let us discuss sub-queries that use comparison operators.
- Some comparison operators are:

| Operator | Description                                                         |
|----------|---------------------------------------------------------------------|
| IN       | Equals to any member of                                             |
| NOT IN   | Not equal to any member of                                          |
| *ANY     | compare value to every value returned by sub-query using operator * |
| *ALL     | compare value to all values returned by sub-query using operator *  |

### Sub-queries by using Comparison operators:

Sub-queries are divided as “single row” and “multiple row” sub-queries. While single row comparison operators ( $=$ ,  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $\neq$ ) can only be used in single row sub-queries, multiple row sub-queries can use IN, ANY or ALL.

For example: The assignment operator ( $=$ ) compares a single value to another single value. In case a value needs to be compared to a list of values, then the IN predicate is used. The IN predicate helps reduce the need to use multiple OR conditions.

The NOT IN predicate is the opposite of the IN predicate. This will select all rows where values do not match the values in the list.

The FOR ALL predicate is evaluated as follows:

True if the comparison is true for every value of the list of values.

True if sub-query gives a null set (No values)

False if the comparison is false for one or more of the list of values generated by the sub-query.

The FOR ANY predicate is evaluated as follows

True if the comparison is true for one or more values generated by the sub-query.

False if sub-query gives a null set (No values).

False if the comparison is false for every value of the list of values generated by the sub-query.

6.6: Sub-queries



## Examples

- Example 1: To list names and jobs of all employees in dept 20 who have the same job as someone in the sales dept.

```
SELECT ename , job FROM emp
WHERE deptno = 20 AND job IN
 (SELECT job FROM emp WHERE deptno =
 (SELECT deptno FROM dept WHERE dname =
 'SALES')) ;
```

**Sub-queries by using Comparison operators (contd.):**

Example 2: To display details of employees, who have salary greater than the lowest salary in dept 30.

```
SELECT * FROM emp
WHERE sal > (SELECT min(sal) FROM emp
Or
WHERE deptno = 30);
```

Example 3: To display details of employees who are not clerks but whose salary is less than some clerks.

```
SELECT * FROM emp
WHERE sal > ANY (SELECT sal FROM emp
WHERE deptno = 30);
```

Example 4: To display employees who have salary greater than the highest sal in dept 30.

```
SELECT * FROM emp
WHERE job <> 'CLERK' AND
sal < ANY (SELECT sal FROM emp
WHERE job = 'CLERK');
```

```
SELECT * FROM emp WHERE sal > ALL
(SELECT sal FROM emp WHERE deptno = 30);
```

### Sub-queries in Predicates

Comparison

<EXPRESSION> < OPERATOR > < SUBQUERY>

- The sub-query should result in one value of the same data type as the left-hand side.
- To get all the employees whose salary is greater than the average salary of the company.

```
SQL> SELECT ename,sal FROM emp
2 WHERE sal > (SELECT AVG(sal) FROM emp);
```

| <u>ENAME</u> | <u>SAL</u> |
|--------------|------------|
| CLARK        | 2450       |
| BLAKE        | 2850       |
| JONES        | 2975       |
| SCOTT        | 3000       |
| FORD         | 3000       |
| KING         | 5000       |

- To select all employees who do the same job as that of “SCOTT”:

```
SQL> SELECT empno,ename, sal FROM emp
2 WHERE JOB=(SELECT job FROM emp WHERE ename='SCOTT');
```

| <u>EMPNO</u> | <u>ENAME</u> | <u>SAL</u> |
|--------------|--------------|------------|
| 7788         | SCOTT        | 3000       |

### **IN predicate in Sub-queries**

<EXPRESSION> IN < SUBQUERY>

- The list of values generated by the sub-query should be of same datatype as the left-hand side.
- To see all details of departments who have employees working in it

```
SQL> SELECT * FROM dept
WHERE deptno IN (SELECT distinct deptno FROM emp);
```

| <u>DEPTNO</u> | <u>DNAME</u> | <u>LOC</u> |
|---------------|--------------|------------|
| 10            | ACCOUNTING   | NEW YORK   |
| 20            | RESEARCH     | DALLAS     |
| 30            | SALES        | CHICAGO    |

### ALL, ANY PREDICATES

- These predicates are called as Quantified Predicates. They allow testing of a single value against all members of the set.
- The general form is  

$$< \text{EXPRESSION} > < \text{OPERATOR} > < \text{ANY / ALL} > < \text{SUB-QUERY} >$$
- The datatypes must be comparable to the left and right hand side. The sub-query generates a list of values of the same type as the <EXPRESSION> on the left and the operator is any comparison operator.
- FOR ALL the predicate is evaluated as follows:
  1. True if the comparison is true for every value of the list of values.
  2. True if sub-query gives a null set ( No values)
  3. False if the comparison is false for one or more of the list of values generated by the sub-query.
- FOR ANY the predicate is evaluated as follows:
  1. True if the comparison is true for one or more values generated by the sub-query.
  2. False if sub-query gives a null set ( No values)
  3. False if the comparison is false for every value of the list of values generated by the sub-query.

#### Examples of ANY, ALL

- To find names and salary details of all employees whose salary is equal to the salary given to an employee working in deptno 30

```
SQL> SELECT ename,sal FROM emp
2 WHERE sal = ANY (SELECT sal FROM emp WHERE
deptno=30);
```

| <u>ENAME</u> | <u>SAL</u> |
|--------------|------------|
| JAMES        | 950        |
| WARD         | 1250       |
| MARTIN       | 1250       |
| TURNER       | 1500       |
| ALLEN        | 1600       |
| BLAKE        | 2850       |

6 rows selected.

- To select the employees whose sal is greater than the sal of all the employees working in deptno 30.

```
SQL> SELECT ename,sal FROM emp
2 WHERE sal > ALL (SELECT sal FROM emp WHERE
deptno=30);
```

| <u>ENAME</u> | <u>SAL</u> |
|--------------|------------|
| JONES        | 2975       |
| SCOTT        | 3000       |
| KING         | 5000       |
| FORD         | 3000       |



6.7: Co-related Sub-query

## Explanation

- A sub-query becomes “co-related”, when the sub-query references a column from a table in the “parent query”.
  - A co-related sub-query is evaluated once for each row processed by the “parent statement”, which can be either SELECT, UPDATE, or DELETE statement.
  - A co-related sub-query is used whenever a sub-query must return a “different result” for each “candidate row” considered by the “parent query”.

### Co-related Sub-query:

A “co-related sub-query” is a form of query used in SELECT, UPDATE, or DELETE commands to force the DBMS to evaluate the query once “per row” of the parent query, rather than once for the “entire query”.

A co-related sub-query is used to answer “multi-part questions”, whose answers depend on the values in each row of the parent query.

A co-related sub-query is one way of reading every row in a table, and comparing values in each row against related data.

6.7: Co-related Sub-query

## Examples

- Example 1: To list all those employees, who are working in the same department as their manager:

| EMPNO | ENAME  | JOB      | MGR  | DEPTNO |
|-------|--------|----------|------|--------|
| 7499  | ALLEN  | SALESMAN | 7698 | 30     |
| 7521  | WARD   | SALESMAN | 7698 | 30     |
| 7654  | MARTIN | SALESMAN | 7698 | 30     |
| 7782  | CLARK  | MANAGER  | 7839 | 10     |
| 7844  | TURNER | SALESMAN | 7698 | 30     |
| 7876  | ADAMS  | CLERK    | 7788 | 20     |

```
SELECT empno, ename, job, mgr, deptno FROM Employee e1
WHERE deptno IN
(SELECT deptno FROM Employee e
WHERE e.empno = e1.mgr);
```



6.7: Co-related Sub-query

## Examples

- Example 2: To display details of employees whose salary is greater than the average salary in their own department:

```
SELECT * FROM emp X
WHERE sal >(SELECT avg(sal) FROM emp Y
WHERE Y.deptno = X.deptno) ;
```

Examples:

Example 3: To display details of employees who earn highest salary in their respective departments.

```
SQL > SELECT * FROM emp X
WHERE sal = (SELECT max(sal) FROM emp Y
WHERE Y.deptno = X.deptno) ;
```

**Co-related Sub-queries (contd.)**

- In the inner query one may refer to the identifiers available from the outer query with proper identification.

```
SQL> SELECT ename, mgr, hiredate FROM emp a
2 WHERE a.hiredate < (SELECT b.hiredate FROM emp b
WHERE b.empno=a.mgr);
```

| ENAME | MGR  | HIREDATE  |
|-------|------|-----------|
| SMITH | 7902 | 17-DEC-80 |
| ALLEN | 7698 | 20-FEB-81 |
| WARD  | 7698 | 22-FEB-81 |
| JONES | 7839 | 02-APR-81 |
| BLAKE | 7839 | 01-MAY-81 |
| CLARK | 7839 | 09-JUN-81 |

6 rows selected.

- Here two sets of rows of EMP table will be created one for the outer query and one for the inner query. In the first pass all the rows of the second set will be compared with first row of first set. This process will be repeated for all the rows of the first set. If for each comparison the condition becomes true then the row from the first set will be selected.

```
SELECT ename, sal, job from emp a
Where sal > (select avg (sal) from emp b where b.job =
a.job);
```

| ENAME  | SAL  | JOB      |
|--------|------|----------|
| SMITH  | 1600 | SALESMAN |
| JONES  | 2975 | MANAGER  |
| BLAKE  | 2850 | MANAGER  |
| TURNER | 1500 | SALESMAN |
| MILLER | 1300 | CLERK    |

6.8: EXISTS / NOT EXISTS Operator

## Explanation



➤ The EXISTS / NOT EXISTS operator enables to test whether a value retrieved by the Outer query exists in the result-set of the values retrieved by the Inner query.

- The EXISTS / NOT EXISTS operator is usually used with a co-related sub-query.
  - If the query returns at least one row, the operator returns TRUE.
  - If the value does not exist, it returns FALSE.
- The NOT EXISTS operator enables to test whether a value retrieved by the Outer query is not a part of the result-set of the values retrieved by the Inner query.

6.8: EXISTS / NOT EXISTS Operator

## Examples



- Example 1: To display details of employees who have some other employees reporting to them.

```
SQL > SELECT * FROM emp X
 WHERE EXISTS (SELECT * FROM emp Y
 WHERE Y.mgr = X.empno) ;
```

- Example 2: To display details of departments which have employees working in it.

```
SQL > SELECT * FROM dept
 WHERE EXISTS (SELECT * FROM emp
 WHERE emp.deptno = dept.deptno)
;
```

Examples:

Example 3: To display details of departments which have no employees working in it:

```
SQL > SELECT * From dept
 WHERE NOT EXISTS (SELECT * FROM emp
 WHERE emp.deptno = dept.deptno) ;
```

**Exists (contd.)**

- The Exists Predicate is of the form  
<Query> WHERE  
EXISTS < sub-query>
- Query will be evaluated if EXISTS takes a value TRUE. It takes the Value TRUE if the <sub-query> result table is non null and FALSE if it is Null. It is mostly used with Co-Related Subqueries. EXISTS does not check for a particular value. It checks whether subquery returns rows or not.
- To list all details of dept which has atleast one employee assigned to it.

```
SQL> SELECT * FROM dept a
2 WHERE EXISTS (SELECT * FROM EMP b
WHERE b.deptno = a.deptno);
```

| DEPTNO | DNAME      | LOC      |
|--------|------------|----------|
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESEARCH   | DALLAS   |
| 30     | SALES      | CHICAGO  |

- Here there are 4 rows in dept table so the Inner Query will be executed 4 times. Since there are no employees for deptno 40 the sub-query will return no rows i.e a null set. Hence EXISTS will become false and the outer query will not select the row in dept table with value 40.

6.9: Union Operator



## Explanation

- Each query executes separately and produces a separate result table
- UNION merges the results tables and eliminates duplicate rows
- UNION ALL will not eliminate duplicate rows

### Union rules

Same number of columns must be selected from each table

Corresponding columns must have identical Data type

Width

Decimal places

Treatment of nulls

Use place holders if no values will be selected for one of the queries

ORDER BY will sequence the result. Must specify relative column sequence.

ORDER BY 4 DESC

6.9: Union Operator

## Example

- List all managers and applicants with some college experience

```
SELECT LASTNAME, EMPNO, 'MANAGER'
 FROM EMP
 WHERE JOB = 'MANAGER'
UNION ALL
SELECT NAME, ' ', 'APPLICANT'
 FROM APPLICANT
 WHERE EDLEVEL > 12
```

Example1:

```
SELECT EMP.EMPNO, EMP.EMPNAME,
EMP.WORKDEPT, DEPT.DEPTNAME
 FROM EMP, DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
 UNION
SELECT EMP.EMPNO, EMP.EMPNAME, ' ', ''
 FROM EMP
```

## Summary

➤ In this lesson, you have learnt:

- Join
  - Inner Join
  - Outer join
  - Self Join
- Sub-queries
  - Co-related sub-query
- UNION operator



## Review Questions

➤ Question 1: To perform an Inner join, which of the following is true.

- Option 1: Two tables in the from clause
- Option 2: Join conditions have to included
- Option 3: A table must join to itself.



➤ Question 2: A sub-query is termed as \_\_\_, as well.

➤ Question 3: A sub-query, which executes for every row of the outer query, is \_\_\_\_.

DB2

Lesson 7: DB2 Objects

Capgemini

## Lesson Objectives

➤ In this lesson, you will learn about:

- Storage structure:
  - Storage groups
  - Database
  - Index space
  - Index
  - Synonym
  - View
  - Alias
  - Catalog



  
7.1: Storage Structure

## Concept

- The total collection of stored data is divided up into a number of disjoint databases.
- Each database is divided into a number of disjoint spaces – several table spaces and index spaces.
  - A space is a dynamically extendable collection of pages, where a page is a block of physical storage.
    - The pages are given size of either 4k or 32 k.

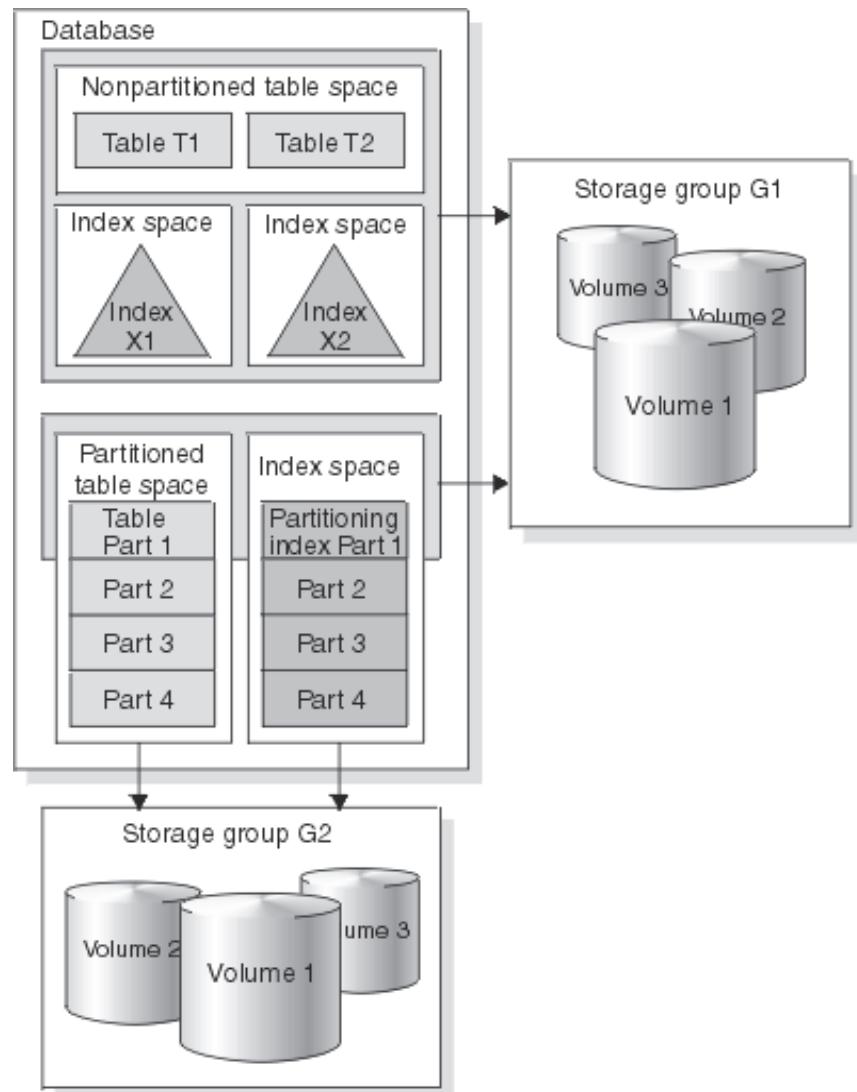
### Storage Structure:

The total collection of stored data is divided up into a number of disjoint databases

Each database is divided into a number of disjoint spaces – several table spaces and index spaces.

A space is a dynamically extendable collection of pages, where a page is a block of physical storage (it is a unit of i/o that is the unit transferred between primary and secondary storage in a physical i/o operation).

The pages are given size of either 4k or 32 k. Each table space contains one or more stored tables. A stored table is a physical representation of a base table.

**Storage Structure:**

7.2: Database

## Concept

- Database is a named collection of logically related objects:
  - tables
  - their associated indexes
- Various spaces contain those tables and indexes.
- It is an administrative/operational grouping of DB2 objects.
- The system operator can make the given database available or unavailable for processing via an appropriate start or stop command.

A database is a collection of structures with appropriate authorizations and accesses that are defined.

The structures in the database like tables, indexes, etc. are called as objects in the database.

All objects that belong to the same user are said to be the “schema” for the particular user.

## 7.3: Storage Groups

## Concept

- Storage Groups are uniquely named collections of DASD volumes.
  - All are of the same device type, that is used by DB2.
  - They allocate space for new or expanding table spaces and/or index spaces of a database.
- A DASD volume may belong to more than one storage group.
- Within each storage group, spaces and partitions are stored using VSAM linear data sets.

### Storage Groups:

A DB2 storage group is a set of volumes on direct access storage devices (DASD). The volumes hold the data sets in which tables and indexes are actually stored. The description of a storage group names the group and identifies its volumes and the VSAM catalog that records the data sets.

7.4: Table Space

## Concept



- Table space is a logical address space on secondary storage that is used to hold one or more tables.
- It is divided into equal-sized units called pages.
- It is the storage unit for recovery and reorganization.
- There are three types of table spaces:
  - simple
  - partitioned
  - segmented

### STORAGE GROUPS

7.4: Table Space



## Simple Table Space

- A simple table space can contain one or more tables.
- Intermix rows of interdependent values for performance (can be useful while using joins).
- Rows from multiple tables can be interleaved on a page under the DBA's control and maintenance.
- Only one table per tablespace is recommended, because:
  - Scanning rows in one table requires scanning the entire tablespace
  - Updating rows in one table causes the entire tablespace to be located

7.4: Table Space



## Segmented Table Space

- The segmented table space was introduced to rectify the limitations of simple table space.
- It does not allow records for different stored tables to be interleaved on a single page.
- Available space is divided into group of pages called segments.
- Each segment is a group of 4 to 64 pages.
- A segment consists of records pertaining to only one table.
- If a particular table grows in size to fill all segments allocated to it, then a new segment will be obtained for that table.
- Using a segmented table space, you get:
  - improved performance
  - concurrency
  - space management for multiple tables in a single table space

7.4: Table Space



## Partitioned Table Space

- The partitioned table space is used for very large tables and may contain only one table.
- A partitioned table space contains exactly one stored table which is very large.

7.5: Data Types



## Explanation

- Each literal or column value manipulated by the RDBMS has a Data type.
- A value's Data type associates a fixed set of properties with the value.
- These properties cause the RDBMS to treat values of one data type differently from values of another.

When you create a table, you must specify a Data type for each of its columns.

Each value subsequently placed in a column assumes the column's Data type.

7.4: Data Types



## Numeric Data Type

- **INTEGER** (Exact Value)
- **DECIMAL** (Exact Value)
- **NUMERIC** (Exact Value)
- **SMALLINT** (Exact Value)
- **FLOAT** (Approximate Value)
- **REAL** (Approximate Value)
- **DOUBLE** (Approximate Value)

7.5: Data Types



## String Data Types

- CHARACTER(CHAR)
- VARCHAR
- GRAPHIC
- VARGRAPHIC
- LONG VARGRAPHIC
- LONG VARCHAR

7.5: Data Types

## Date and Time Data Type

- DATE
- TIME
- TIMESTAMP

SQL has “primitive” date and time data types which can help avoid the complexity of the encoding process.

DATE: Values consists of three parts for year,month and day. The range of values is 0001-01-01 to 9999-12-31

Data is represented in unsigned packed decimal digits occupying four bytes.

TIME: Time is represented in unsigned packed decimal digits(HH:MM:SS),occupying three bytes,permitted

values are legal times in the range midnight to midnight (000000 – 240000)

TIMESTAMP: Combination of DATE and TIME is accurate to the nearest microseconds

represented in unsigned packed decimal digits occupying ten bytes

Format : yyyyymmddhhmmssnnnnnn

7.5: Data Types

## SMALLINT Data Type



- Two byte binary integer
- Range of values -32768 to +32767

7.5: Data Types



## Integer Data Type

- Binary integer of 32 bits
- Range of values -2147483648 to +2147483647 Internal
- Storage 4 Bytes

7.5: Data Types



## Float (n) or Real Data Type

- Single precision floating point number 'n' is in the range of 1 to 21. If n is omitted the column has double precision. The number is defined using 32 bits
- Range of magnitude is 5.4E-79 to 7.2+75

7.5: Data Types



## Float (n) or Double Data Type

- Double precision floating point number. 'n' is in the range of 22 to 53
- The default is 53.
- The number is defined using 64 bits
- Range of magnitude is 5.4E-79 to 7.2E+75

7.5: Data Types

## Numeric (p, s) Data Type

- Number having a precision p and scale s
- Maximum value for 'p' is 31
- Internal representation is packed decimal



7.5: Data Types



## Char (n) or Character (n) Data Type

- Fixed length character strings represented using EBCDIC .
- 'n' is the range of 1 to 254.
- All values in the column are of the same length.
- If 'n' is omitted,a value of 1 is assumed.

7.5: Data Types



## VarChar (n) Data Type

- Variable length character string represented using EBCDIC.
- The maximum value of n is 4046 .
- The values in the column may have different lengths not to exceed a maximum of 'n'
- VARCHAR field have length field as well as text fields.
- First two bytes is allocated to hold the length of the text

7.5: Data Types



## Long VarGraphic and VarChar Data Type

- A long string can not be indexed,not referenced in WHERE clause,not used with GROUP BY or ORDER BY clause
- A long string cannot appear in PRIMARAY KEY and FOREIGN KEY representations

## 7.6: Constants and Literals



## List

- **Integer:** It is a signed or unsigned decimal integer with no decimal point.
  - For example: 4 -15 +364 0
- **Decimal:** It is a signed or unsigned decimal number with decimal point.
  - For example: 40 -95.7 +364.05 0.0007
- **Float:** It is written as a decimal constant, followed by a character E, followed by an integer constant.
  - For example: 4E3 -95.7E46 +364E.5 07E1
- **Character string:** It is written either as a string of characters enclosed in a single quotes or a string of pairs of hexadecimal digits enclosed in single quotes preceded by letter X.
  - For example:
    - '123MAIN St.'
    - 'PIGON'
    - X 'F1F2F3F40D156247'

## 7.6: Constants and Literals



## List of Constants and Literals

- Graphic string: It is written as a string of double byte characters preceded by the character '<' and followed by '>' the whole enclosed in single quotes and preceded by letter G.
  - For example: G '<STRING>'
- Date: It is written in the form 'mm/dd/yyyy' enclosed in single quotes.
  - For example:
    - '1/18/1941'
    - '12/25/1889'
- Time: It is written in the form hh:mm AM or hh:mm PM and enclosed in the single quotes.
  - For example:
    - '10:00AM'
    - '9:30PM'

7.6: Constants and Literals

## List of Constants and Literals



➤ **Timestamp:** It is written in the form yyyy-mm-dd-hh-m.ss.nnnnn and enclosed in single quotes.

- For example:
  - '1996-4-28-12.00.00.000000'
  - '1994-10-17-18.30.45'



### 7.7: Special Register

## Concept

- Special register is used to store information that can be referenced in SQL statements.
- DB2 supports a number of special registers.
  - USER : It returns the 'primary authorization ID'.
  - CURRENT SQLID : It returns the current authorization ID.
  - CURRENT SERVER : It returns the ID of the current server.
  - CURRENT PACKAGESET : It returns the ID of the collection currently in use.
  - CURRENT TIME : It returns the current time.
  - CURRENT DATE : It returns the current date.
  - CURRENT TIMESTAMP : It returns the current timestamp.

### Special Register:

Special register is used to store information that can be referenced in SQL statements.

The special registers currently defined are USER, CURRENT SOLID, CURRENT DATE, CURRENT TIME, CURRENT TIMESTAMP, and CURRENT TIMEZONE.

### Note:

Users are known to DB2 by an authorization identifier.

Each individual user is assigned an authorization ID. That ID is used to sign on to the system, and serves as the primary id for the user in question.

Tables and other objects that are purely private to that user will typically be created under the control of and hence be owned by the primary id.

Each functional area (for example: each department) in the organization is also assigned an authorization id. However, that id is typically not given the sign-on authority.

Users sign on to the system under their primary id. Once signed on, the users can operate under their primary id or using the SQL statement (SET CURRENT SQLID) they can switch to a secondary id.

An external subsystem such as IBM's RACF keeps track of the secondary id(s) that can legitimately be used by a given primary id.

A given primary id can have any number of secondary ids. Furthermore, the same secondary id can be used by any number of primary ids.

7.8: Table

## Explanation

- Tables are objects, which store the user data.
- Use the CREATE TABLE statement to create a table, which is the basic structure to hold data.
  - For example:

```
CREATE TABLE DEPT
(Deptno integer(2),
 Dname varchar(14),
 Loc varchar(13)) in <DBName.TablespaceName>;
```

Creating tables is done with the create table command. You can add rows to a table with the INSERT statement, after creating a table.

The above create table command does the following:

Defines the table name

Defines the columns in the table and the datatypes of those columns

In above example, we create a table called DEPT which has 3 columns. The first column is DEPTNO which is a NUMBER datatype. This means we will be storing numbers in this column. The second and third columns are of VARCHAR datatype. We will be storing textual data in these columns

Syntax:

```
CREATE TABLE table_name
(
 {col_name.col_datatype [[CONSTRAINT
 const_name][col_constraint]]},...
 [table_constraint],...
)
[AS query]
```

If a table is created as shown in the slide, then there is no restriction on the data that can be stored in the table.

However, if we wish to put some restriction on the data, which can be stored in the table, then we must supply some “constraints” for the columns. We will see the Constraints as next topic”

7.9: Data Integrity



## Explanation

### ➤ Data Integrity:

- "Data Integrity" allows to define certain "data quality requirements" that must be met by the data in the database.
- Oracle uses "Integrity Constraints" to prevent invalid data entry into the base tables of the database.
  - You can define "Integrity Constraints" to enforce the business rules you want to associate with the information in a database.
  - If any of the results of a "DML statement" execution violate an "integrity constraint", Oracle rolls back the statement and returns an error.

### Data Integrity: Integrity Constraint

An Integrity Constraint is a declarative method of defining a rule for a column of a table.

#### Example of Data Integrity:

Assume that you define an "Integrity Constraint" for the SAL column of the EMP table.

This Integrity Constraint enforces the rule that "no row in this table can contain a numeric value greater than 10,000 in this column".

If an INSERT or UPDATE statement attempts to violate this "Integrity Constraint", database rolls back the statement and returns an "information error" message.

7.9: Data Integrity



## Advantages

### ➤ Advantages of Integrity Constraints:

- Integrity Constraints have advantages over other alternatives. They are:
  - Enforcing “business rules” in the code of a database application.
  - Using “stored procedures” to completely control access to data.
  - Enforcing “business rules” with triggered stored database procedures.

7.9: Data Integrity



## Types of Data Integrity

➤ Let us see the types of Data Integrity:

- Nulls
- Unique Column Values
- Primary Key Values
- Referential Integrity

### Types of Data Integrity:

Oracle supports the following Integrity Constraints:

NOT NULL constraints for the rules associated with nulls in a column.

“Null” is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a “null” value (the absence of a value) in that column.

UNIQUE key constraints for the rule associated with unique column values. A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).

PRIMARY KEY constraints for the rule associated with primary identification values. A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.

FOREIGN KEY constraints for the rules associated with referential integrity. A “Referential Integrity” rule defined on a key (a column or set of columns) in one table guarantees that “the values in that key, match the values in a key in a related table (the referenced value)”. Oracle currently supports the use of FOREIGN KEY integrity constraints to define the referential integrity actions, including:

- update and delete No Action
- delete CASCADE
- delete SET NULL

CHECK constraints for complex integrity rules

## 7.10: Types of Data Integrity

## NOT NULL constraint

- The user will not be allowed to enter null value.

- Example:

```
CREATE TABLE EMP
 (empno INTEGER not null,
 ename varchar(20) not null,
 ) in PATNIDB.DSRP035S;
```

- A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
  - A NULL value can be inserted into a column of any data type.

### NOT NULL constraint:

Often there may be records in a table that do not have values for every field.

This could be because the information is not available at the time of the data entry or because the field is not applicable in every case.

If the column is created as **NONNULLABLE**, in the absence of a user-defined value the DBMS will place a **NULL** value in the column.

A **NULL** value is different from a blank or a zero. It is used for a quantity that is "unknown".

"**Null**" is a rule defined on a single column that allows or disallows, inserts or updates on rows containing a "null" value (the absence of a value) in that column. A **NULL** value can be inserted into a column of any data type.

#### Principles of **NULL** values:

Setting a **NULL** value is appropriate when the "actual value" is unknown, or when a value is not meaningful.

A **NULL** value is not equivalent to the value of "zero" if the data type is number, and it is not equivalent to "spaces" if the data type is character.

A **NULL** value will evaluate to **NULL** in any expression

For example: **NULL** multiplied by 10 is **NULL**.

**NULL** value can be inserted into columns of any data type.

If the column has a **NULL** value, Oracle ignores any **UNIQUE**, **FOREIGN KEY**, and **CHECK** constraints that may be attached to the column.

## 7.10: Types of Data Integrity



## UNIQUE constraint

➤ The keyword UNIQUE specifies that no two records can have the same attribute value for this column.

- For Example:

```
CREATE TABLE DEPT
(DEPTNO INTEGER NOT NULL,
 DNAME VARCHAR(14) ,
 LOC VARCHAR(13),UNIQUE (DEPTNO)) IN
PATNIDB.DSRP001S;
```

### UNIQUE constraint:

The UNIQUE constraint does not allow duplicate values in a column.

If the UNIQUE constraint encompasses two or more columns, then two equal combinations are not allowed.

However, if a column is not explicitly defined as NOT NULL, then NULLS can be inserted multiple number of times.

A UNIQUE constraint can be extended over multiple columns.

A “unique value” rule defined on a column (or set of columns) allows inserting or updating a row only if it contains a “unique value” in that column (or set of columns).

## 7.10: Types of Data Integrity



## PRIMARY KEY constraint

➤ The Primary Key constraint enables a unique identification of each record in a table.

- For Example:

```
CREATE TABLE EMP
 (empno integer constraint pk_emp primary
 key,
 );
```

### PRIMARY KEY constraint:

On a technical level, a PRIMARY KEY combines a UNIQUE constraint and a NOT NULL constraint.

Additionally, a table can have at the most one PRIMARY KEY.

After creating a PRIMARY KEY, it can be referenced by a FOREIGN KEY.

A “primary key” value rule defined on a key (a column or set of columns) specifies that “each row in the table can be uniquely identified by the values in the key”.

7.10: Types of Data Integrity

## CHECK constraint



- CHECK constraint allows users to restrict possible attribute values for a column to admissible ones.

- For Example:

```
CREATE TABLE EMP
 (empno number(4) not null,
 ename varchar(40)
 constraint check_name check(ename =
 upper(ename)),
 ) IN PATNIDB.DSRP001S; ;
```

CHECK constraint:

A CHECK constraint allows to state a minimum requirement for the value in a column.

If more complicated requirements are desired, an INSERT trigger must be used.

7.10: Types of Data Integrity

## FOREIGN KEY constraint



- The FOREIGN KEY constraint specifies a "column" or a "list of columns" as a foreign key of the referencing table.
- The referencing table is called the "child-table", and the referenced table is called "parent-table".
  - For Example:

```
CREATE TABLE EMP
 (empno integer(4) constraint pk_emp primary key,
 ,
 deptno integer(3) constraint fk_deptno
 references dept(deptno));
```

### FOREIGN KEY Constraint Keywords:

Given below are a few Foreign Key constraint keywords:

FOREIGN KEY: Defines the column in the child table at the table constraint level.

REFERENCES: Identifies the table and column in the parent table.

ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted.

ON DELETE SET NULL: Converts dependent FOREIGN KEY values to NULL.

You can query the USER\_CONSTRAINTS table to view all constraint definitions and names.

You can view the columns associated with the constraint names in the USER\_CONS\_COLUMNS view.

In EMP table, for deptno column, if we want to allow only those values that already exist in deptno column of the DEPT table, we must enforce what is known as REFERENTIAL INTEGRITY. To enforce REFERENTIAL INTEGRITY, declare deptno field of DEPT table as PRIMARY KEY, and deptno field of EMP table as FOREIGN KEY as follows

**FOREIGN KEY Constraint Keywords:**

- In the given example, FOREIGN KEY has been declared as a Table constraint.

```
CREATE TABLE Dept
(
Deptno integer(2) CONSTRAINT DEPTNO_P_KEY
PRIMARY KEY,
Dname varchar(14) NOT NULL,
Loc varchar(13) NOT NULL
);
CREATE TABLE Emp
(
Empno integer(4) CONSTRAINT P_KEY PRIMARY
KEY,
Ename varchar(10) CONSTRAINT
ENAME_NOT_NULL NOT NULL,
Deptno integer(2),
Job CHAR(9) CONSTRAINT JOB_ALL_UPPER
CHECK (Job =UPPER(Job)),
Hiredate DATE DEFAULT SYSDATE,
CONSTRAINT DEPTNO_F_KEY FOREIGN KEY
(Deptno)
REFERENCES Dept(Deptno)
);
```

**FOREIGN KEY Constraint Keywords:**

- Same can be given as a COLUMN constraint as shown in the example given below:

```
CREATE TABLE emp
(
 Empno integer(4) CONSTRAINT P_KEY PRIMARY
 KEY,
 Ename varchar(10) CONSTRAINT ENAME_NOT_NULL
 NOT NULL,
 Deptno integer(2) CONSTRAINT DEPTNO_F_KEY
 REFERENCES
 Dept(Deptno),
 Job CHAR(9) CONSTRAINT JOB_ALL_UPPER
 CHECK(job=UPPER(job)),
 Hiredate DATE DEFAULT SYSDATE
);
```

7.11: CREATE TABLE

## Example

➤ The CREATE TABLE example with some constraints as:

```
CREATE TABLE emp (
 empno integer(4) CONSTRAINT P_KEY PRIMARY KEY,
 ename varchar(10) CONSTRAINT ENAME_NOT_NULL
 NOT NULL, deptno integer(2), job CHAR(9) CONSTRAINT
 JOB_ALL_UPPER CHECK(job = UPPER(job)), hiredate DATE
 DEFAULT SYSDATE);
```

Creation of database objects: Tables (contd.):

A table can have a maximum of 1000 columns. Only one column of type LONG is allowed per table.

|                                |                                                                                                                                                                                                                                                                                                                           |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Table_name,col_name,const_name | A string upto 30 characters length.<br>Can be made up to A-Z,0-9,\$,_,#<br>Must begin with a non-numeric ORACLE data characters.                                                                                                                                                                                          |
| Col_datatype                   | One of the previously mentioned types.                                                                                                                                                                                                                                                                                    |
| Col_constraint                 | A restriction on the column can be of following types:<br>PRIMARY KEY, NOT NULL, UNIQUE,<br>FOREIGN KEY, CHECK, Can be named. Only one<br>PRIMARY KEY is allowed per table.                                                                                                                                               |
| Table_constraint               | A restriction on single or multiple columns. The types<br>are same as in col_constraint. (NULL constraint is<br>not allowed here).                                                                                                                                                                                        |
| AS query                       | Query is an SQL statement using SELECT<br>command. SELECT command returns the rows from<br>tables. It is useful if the table being created is based<br>on an existing table. New table need not have all the<br>columns of the old table.<br>Names of columns can be different. All or part of the<br>data can be copied. |

Any object created by a user is accessible to the user and the DBA only.

To make the object accessible to other users, the creator or the DBA must explicitly give permission to others.

## 7.11: Examples of CREATE TABLE

## Create new table based on existing table

- Constraints on an “old table” will not be applicable for a “new table”.

```
CREATE TABLE NewEmp(Emp#,Emp_Name,Hire_Date)
AS
(
 SELECT empno,ename,hiredate FROM emp
 WHERE hiredate >'01-jan-82'
);
```

Creating new table based on an existing table:

The example in the slide shows how to create a new table based on an existing table.

When the new table will be created, it will contain empno, ename, hiredate of all employees hired after 01-Jan-1982.

Constraints on an old table will not be applicable for the new table except NOT NULL constraint.

7.12: Examples of ALTER TABLE

## Example



➤ Given below is an example of ALTER TABLE:

```
ALTER TABLE table_name
 [ADD (col_name col_datatype col_constraint ,...)]|
 [ADD (table_constraint)]|
 [DROP CONSTRAINT constraint_name]|
 [DROP COLUMN existing_col_name]
```

### Examples of ALTER TABLE:

Table\_name must be an existing table.

A column cannot be removed from an existing table by using ALTER TABLE.

The uses of modifying columns are:

- Can increase the width of a character column, any time.
- Can increase the number of digits in a number, any time.
- Can increase or decrease the number of decimal places in a number column, any time. Any reduction on precision and scale can be on empty columns only.
- Can add only NOT NULL constraint by using Column constraints. All other constraints have to be specified as Table constraints.

7.12: Examples of ALTER TABLE

## ALTER Table – Add clause

➤ The “Add” keyword is used to add a column or constraint to an existing table.

- For adding three more columns to the emp table, refer the following example:

```
ALTER TABLE emp
 ADD (sal integer(7,2) CONSTRAINT SAL_GRT_0
CHECK(sal >0),
 mgr integer(4), comm integer(9,2));
```

### ALTER TABLE – Add clause:

The uses of adding a column to the table by using Add clause are:

You can add any column without a NOT NULL specification.

You can add a NOT NULL column in three steps:

Add a column without NOT NULL specification.

Fill every row in that column with data.

Modify the column to be NOT NULL.

For adding constraint, refer the following example:

```
ALTER TABLE Dept ADD (dname varchar(10) NOT NULL);
```

7.12: Examples of ALTER TABLE

## ALTER Table – Add clause

➤ Example :

- For adding Referential Integrity on "mgr" column, refer the following example:

```
ALTER TABLE emp
ADD CONSTRAINT FK FOREIGN KEY (mgr)
REFERENCES emp(empno);
```



7.12: Examples of ALTER TABLE

## ALTER Examples

```
ALTER TABLE EMPLOYEE ALTER LASTNAME
SET DATA TYPE VARCHAR(30)
```

```
ALTER TABLE EMPLOYEE DROP PRIMARY KEY
```

```
ALTER TABLE EMPLOYEE ADD RESTRICT ON DROP
```

```
ALTER TABLE PARTS ADD CONSTRAINT CHK
FOREIGN KEY(Prod#) REFERENCES
PRODUCTS ON DELETE SET NULL;
```



7.12: Examples of ALTER TABLE

## ALTER TABLE – DROP clause

- The DROP TABLE command is used to remove the definition of a table from the database.
  - For example:

```
DROP TABLE Emp;
DROP TABLE Dept CASCADE CONSTRAINTS;
```

### Deleting Database Objects: Tables

Deleting objects that exist in the database is an easy task. Just say:

```
DROP Obj_Type obj_name;
```

For example:

```
DROP TABLE EMP;
```

A table that is dropped cannot be recovered. When a table is dropped, dependent objects such as indexes are automatically dropped. Synonyms and views created on the table remain, but give an error if they are referenced.

You cannot delete a table that is being referenced by another table. To do so use the following:

```
DROP table-name CASCADE CONSTRAINTS;
DROP TABLE Dept CASCADE CONSTRAINTS;
DROP SYNONYM new_emp;
```

If new\_emp is a synonym for a table, then the table is not affected in any way. Only the duplicate name is removed.

7.13: Index



## Explanation

- Index is a database object that functions as a "performance-tuning" method for allowing faster retrieval of records.
- Index creates an entry for each value that appears in the indexed columns.
- The absence or presence of an Index does not require change in wording of any SQL statement.

An index is an ordered set of pointers to the data in a DB2 table.

It is based on the values of data in one or more columns of a table.

Index is used to:

- Locate the row(s) that contain a given value
- Gain an efficient and faster direct access to data

Index has an entry for every value in the column(s), with an RID for each row.

Multiple indexes can be defined on a table.

DB2 decides whether or not to use any index to access the table data - when and how to use.

7.13: Index



## Examples

- Example 1: The index shown below can be very useful when you query for comparing revenue - cost.

```
CREATE INDEX sales_margin_index
 ON sales(revenue - cost)
```

- Example 2

```
CREATE INDEX uppercase_idx ON emp (UPPER(empname));
```

7.14: Synonym

## Explanation

- A "Synonym" is an "alias" that is used for any table, view, materialized view, sequence, procedure, function, or package.
  - Since a Synonym is simply an alias, it does not require storage except for storage of its definition in the data dictionary.
  - Synonyms are often used for "security" and "convenience".
  - Synonyms are useful in hiding ownership details of an object.

### ➤ Syntax

```
CREATE SYNONYM another_name FOR existing_name
```

### ➤ where:

➤ Existing\_name is the name of a table, view, or sequence.

Synonym:

A Synonym simplifies access to objects. A Synonym is simply another name for an object.

With Synonyms, you can:

Ease referring to a table owned by another user.

Shorten lengthy object names.

7.14: Synonym

## Example

➤ Here is an example for synonym:

- Suppose a procedure "proc1" is created in a schema "scott". While calling this procedure, if the user refers it as "scott.proc1", then a synonym is created as:

```
Create synonym prc1 for scott.proc1;
```

Creating and Removing Synonyms:

You can create a shortened name for the SALGRADE table as shown in the following example:

```
CREATE SYNONYM s
FOR salgrade
Synonym Created.
```

You can create a shortened name for the DEPT\_SUM\_VU view as shown in the following example:

```
CREATE SYNONYM d_sum
FOR dept_sum_vu;
Synonym Created.
```

You can drop a Synonym as shown in the following example:

```
DROP SYNONYM d_sum;
Synonym dropped.
```

7.15: View

## Explanation

- A View can be thought of as a "stored query" or a "virtual table", i.e. a logical table based on one or more tables.
  - A View can be used as if it is a table.
  - A View does not contain data.
  - View gets catalogued in to SYSIBM.SYSVIEWS.

### ➤ Syntax

```
CREATE [OR REPLACE] VIEW view [(alias[, alias]...)]
AS subquery
```

Why do we use Views?

Views are used:

To restrict data access.

To make complex queries easy.

To provide data independence.

To present different views of the same data.

Features of Simple and Complex Views:

| Features                       | Simple View | Complex View |
|--------------------------------|-------------|--------------|
| Number of tables               | One         | One or more  |
| Contains functions             | No          | Yes          |
| Contains groups of data        | No          | Yes          |
| DML operations through a View. | Yes         | Not always   |

**View:****Creating a View:**

You can embed a sub-query within the CREATE VIEW statement.

The sub-query can contain complex SELECT syntax.

**Characteristics of a View:**

View is a logical table that is based on one or more Tables.

View can be used as if it is a Table.

View does not contain data.

Whenever a View is accessed, the query is evaluated. Thus a View is dynamic.

Any changes made in the View affects the Tables on which the View is based.

View helps to hide the following from the user:

Ownership details of a Table, and

Complexity of the query used to retrieve the data

7.15: View

## Example

➤ Given below is an example of a simple View:

```
CREATE VIEW newempview
AS
SELECT * FROM emp
WHERE hiredate >'01-jan-82';
```

Restrictions while using Views:

Updating / Inserting rows in the base table is possible by using Views, however, with some restrictions. This is possible only if the View is based on a single table. The restrictions are :

Updation / Insertion not possible if View is based on two tables. However, this can be done in ORACLE 8.

Insertion is not allowed if the underlying table has any NOT NULL columns, which are not included in the View.

Insertion / Updation is not allowed if any column of the View referenced in UPDATE / INSERT contains “functions” or “calculations”.

Insertion / Updation / Deletion is not allowed if View contains GROUP BY or DISTINCT clauses in the query.

7.15: View

## Example

### ➤ Creating a Complex View:

- As shown in the example given below, create a Complex View that contains group functions to display values from two tables.

```
CREATE VIEW dept_sum_vu(name, minsal, maxsal, avgsal)
AS SELECT d.department_name, MIN(e.salary),
MAX(e.salary),AVG(e.salary)
FROM employees e, departments d
WHERE e.department_id = d.department_id
GROUP BY d.department_name;
```

View created.

### Modifying a View:

As shown in the example given below, modify the View “EMPVU80” by using the CREATE OR REPLACE VIEW clause. Add an alias for each column name.

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS SELECT employee_id, first_name || ' ' || last_name,
salary, department_id
FROM employees
WHERE department_id = 80;
View created.
```

Column aliases in the CREATE VIEW clause are listed in the same order as the columns in the sub-query.

7.15: View

## Rules for performing operations on a View

- You can perform “DML operations” on simple Views.
- You cannot remove a row if the View contains the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword

### Rules for performing DML Operations on a View:

You can perform “DML operations” on data “through a View” only if those operations follow certain rules.

You can remove a row from a view, unless it contains any of the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudocolumn ROWNUM keyword

### Removing a View:

You can remove a View by using the following syntax:

```
DROP VIEW view;
```

7.15: View

## Inline View

- “Inline view” is not a schema object like a regular View. However, it is a temporary query with an alias.

```
SELECT DNAME, ENAME, SAL FROM EMP A,
 (SELECT DNAME, DEPTNO FROM DEPT) B
 WHERE A.DEPTNO = B.DEPTNO
```

### Inline View:

In the example shown in the slide, the portion of the query, which is highlighted, is the Inline view.

7.16: Deleting Database Objects

## Explanation



➤ Use the following syntax for deleting database objects:

```
DROP Obj_Type obj_name;
```

- Example 1: Given below is an example of deleting a table:

```
DROP TABLE emp;
```

- Example 2:

```
DROP SYNONYM new_emp;
```

- If new\_emp is a Synonym for a table, then the Table is not affected in any way. Only the duplicate name is removed.

### Examples:

A table, which is dropped, cannot be recovered.

Dependent objects such as Indexes are automatically dropped.

Synonyms and Views remain intact. However, they give an error when referenced.

7.17: Catalog

## Concept

- A catalog contains information about every DB2 object that is maintained by DB2 system including tablespace, indexes, tables, copies of tablespaces and indexes, storage groups and so forth.
- For every DDL or DCL an entry is made in the appropriate Catalog table.
- Catalog table can be accessed using SQL, however they cannot be updated.
- To retrieve information from the DB2 catalog, the select privilege on the catalog tables is needed.
  - Some of the catalog tables are :
    - SYSTABLES: It maintains a list of all DB2 tables.
    - SYSSTOGROUP: It stores information about every storage group in the database.
    - SYSTABLESPACE: It describes table spaces in the database.
    - SYSINDEXES: Every row defines one index of the database.

7.17: Catalog  
**Concept**

**Catalog:**

A catalog contains information about every DB2 object that is maintained by it. Whenever a DB2 object is created, dropped or altered by any DDL statement or whenever control over the object is changed by any DCL statement of SQL, an entry is made in the appropriate catalog table.

Catalog table can be accessed using SQL. However, they cannot be updated by a user (but are updated by the system).

To retrieve information from the DB2 catalog, the select privilege on the catalog is needed. Some of the catalog tables are given below:

SYSTABLES  
SYSSTOGROUP  
SYSTABLESPACE

## Summary

➤ In this lesson, you have learnt:

- Storage structure of DB2
- Various objects used to store data in DB2
- Basic Data Types
- Data Integrity
- Different types of Database Objects:
- Modification of Database Objects
- Deleting Database Objects



## Review Questions

- Question 1: Indexes can be either created \_\_\_\_ or \_\_\_\_.
- Question 2: Synonyms are useful in hiding ownership details of an object.
  - True/False
- Question 3: Which of the following table space can be used to store large tables?
  - Option 1: Simple table space
  - Option 2: Partitioned table space
  - Option 3: Segmented table space
- Question 4 : A ----- is a named collection of direct access volumes
  - Option 1: Storage group
  - Option 2: Table space



## DB2

### Lesson 8: Data Manipulation Language

Capgemini

## Lesson Objectives

➤ To understand the following topics:

- Concept of Data Manipulation Language
- Inserting rows into a table
- Deleting rows from a table
- Updating rows in a table





8.1: Data Manipulation Language

## Explanation

- Data Manipulation Language (DML) is used to perform the following routines on database information:
- Topic Details - Line 2
  - Retrieve
  - Insert
  - Modify
- DML changes data in an object. If you insert a row into a table, that is DML.
- All DML statements change data, and must be committed before the change becomes permanent.

## 3.2: Addition Of Data Into Tables

## Explanation



### ➤ INSERT command:

- INSERT is a DML command. It is used to add rows to a table.
- In the simplest form of the command, the values for different columns in the row to be inserted have to be specified.
- Alternatively, the rows can be generated from some other tables by using a SQL query language command.

Addition of Data into Tables:

Requisites for using INSERT command:

If values are specified for all columns in the order specified at creation, then col\_names could be omitted.

Values should match “data type” of the respective columns.

Number of values should match the number of column names mentioned.

All columns declared as NOT NULL should be supplied with a value.

Character strings should be enclosed in quotes.

Date values should be enclosed in quotes.

Values will insert one row at a time.

Query will insert all the rows returned by the query.

The table\_name can be a “table” or a “view”. If table\_name is a “view”, then the following restrictions apply:

The “view” cannot have a GROUP BY, DISTINCT, UNION, clause or a join.



## 8.2: Addition Of Data Into Tables

## Inserting Rows Into A Table

➤ Inserting by specifying values:

```
INSERT INTO table_name[(col_name1,col_name2,...)]
{VALUES (value1,value2,...);}
```

- Example: To insert a new record in the DEPT table

```
INSERT INTO Dept VALUES (10, 'ACCOUNTING',
'MUMBAI');
```

Inserting Rows into a Table:

Example:

Inserting a row in EMP table giving some values.

This row will be created if all the constraints like NOT NULL are satisfied.

```
INSERT INTO EMP (empno, ename, sal)
VALUES (200, 'Archer', 40000);
```



## 8.2: Addition Of Data Into Tables

## Example

➤ Inserting rows in a table from another table using Subquery:

- Example: The example given below assumes that a new\_emp\_table exists. You can use a subquery to insert rows from another table.

```
➤ INSERT INTO new_emp_table
 SELECT * FROM emp WHERE emp.hiredate > '01-jan-82';
```



8.3: Deletion of Data from Tables

## Explanation

- The DELETE command is used to delete one or more rows from a table.

- The DELETE command removes all rows identified by the WHERE clause.



```
DELETE [FROM] {table_name | alias }
[WHERE condition];
```

### Deletion of Data from Tables

The table\_name can be a “table” or a “view”.

The DELETE command is used to delete one or more rows from a table.

The DELETE statement removes all rows identified by the WHERE clause.

This is another DML, which means we can rollback the deleted data, and that to make our changes permanent.

If WHERE clause is omitted, all rows from the table are removed. Else all rows which satisfy the condition are removed.

FROM clause can be omitted without affecting the statement.

## 3.3: Deletion of Data from Tables

## Examples



- Example 1: If the WHERE clause is omitted, all rows will be deleted from the table.

```
DELETE FROM Emp;
```



- Example 2: If we want to delete all information about department 10 from the Emp table:



```
DELETE FROM Emp WHERE deptno=10;
```

## Deletion of Data from Tables

## Example 3:

```
DELETE Emp WHERE ename = 'JONATHAN SEAGULL';
```

## 3.4: Modifying / Updating existing Data in a Table

## Explanation



➤ Use the UPDATE command to change single rows, groups of rows, or all rows in a table.

- In all data modification statements, you can change the data in only “one table at a time”.

```
UPDATE table_name
SET col_name = value|
 col_name =
 SELECT_statement_returning_single_value|
 (col_name,...) = SELECT_statement
 [WHERE condition];
```

Modifying / Updating existing Data in a Table:

The table\_name can be a “table” or a “view”.

The “value” can be a value, an expression, or a query, which returns a single value.

The UPDATE command provides automatic navigation to the data.

Note: If the WHERE clause is omitted, all rows in the table will be updated by a value that is currently specified for the field. Else only those rows which satisfy the condition will be updated.



8.4: Modifying / Updating existing Data in a Table

## Examples

- Example 1: To UPDATE the column "dname" of a row, where deptno is 10, give the following command:

```
UPDATE Dept
SET dname= 'Information Technology'
WHERE deptno=10;
```



## 3.4: Modifying / Updating existing Data in a Table

## Examples

- Example 2: To UPDATE the columns "dname" and "loc" of row(s) where deptno is 10, give the following command:

```
UPDATE Dept
SET dname= 'Information Technology' , loc= 'California'
WHERE deptno=10;
```

3.4: Modifying / Updating existing Data in a Table

## Using a Subquery to do an Update



- For making salary of SMITH equal to that of emp no 7369, use the following command:

```
UPDATE emp
SET SAL = (SELECT SAL FROM emp WHERE empno = 7369)
WHERE ename = 'SMITH';
```

### Example:

All employees other than the PRESIDENT will have the same job, salary, and commission as that of ALLEN. You can use a subquery to do an update.

```
UPDATE emp
SET (job,sal,comm) = (SELECT job,sal,comm FROM emp
WHERE ename = 'ALLEN')
WHERE NOT job = 'PRESIDENT';
```

## Summary

➤ In this lesson, you have learnt:

- The concept of Data Manipulation Language
- Inserting rows into a table
- Deleting rows from a table
- Updating rows in a table



## Review Questions



- Question 1: All DML statements are autocommitted.
  - True / False
  
- Question 2: In a transaction, DDL statement after DML statement commits the changes done by DML.
  - True/False



## DB2

### Lesson 9: Components of DB2

Capgemini

## Lesson Objectives

➤ In this lesson, you will learn about:

- System structure
- System services
- Locking services
- Database services
  - Precompiler
  - Bind
  - Runtime Supervisor
  - Data manager
  - Buffer manager



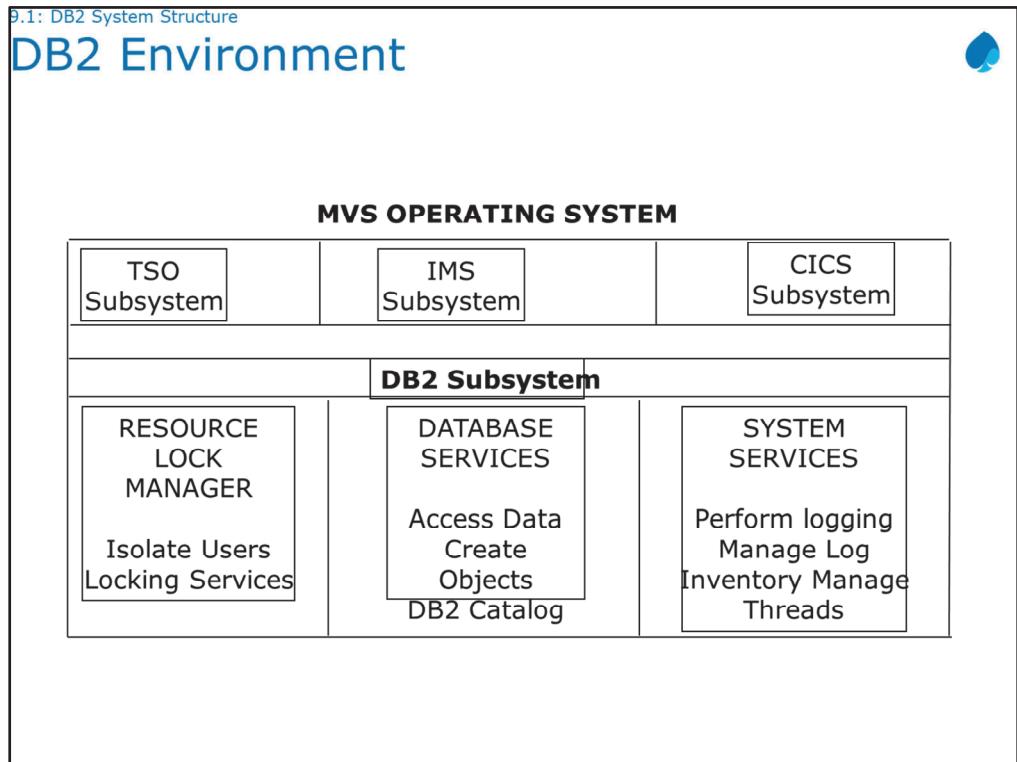
9.1: DB2 System Structure



## Important Services

➤ Major components of DB2 are:

- System Services:
  - It supports system operation.
  - It provides for Operator communication.
  - It supports logging and similar functions.
- Locking Service:
  - It provides the necessary controls for managing concurrent access to data.
- Database Services:
  - It supports definition, retrieval and update of user and system data.



DB2 System Structure – Major Components:

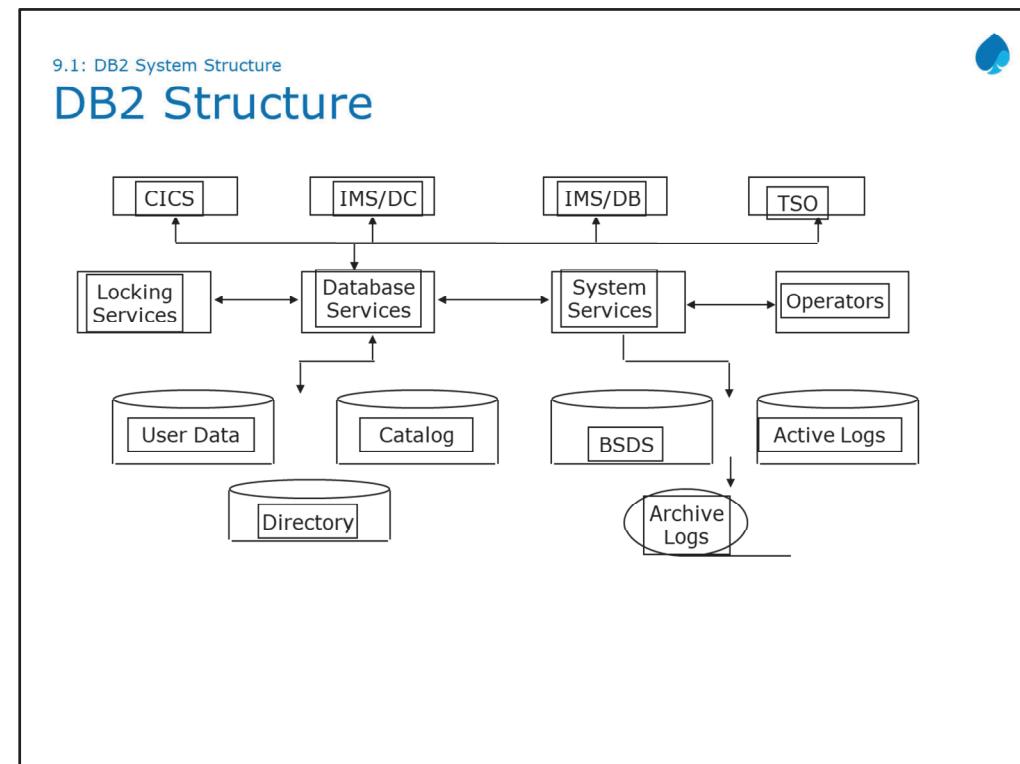
DB2 Environment Overview:

DB2 requires MVS as the operating system.

There are three MVS host environments through which applications can access DB2 resources. They are TSO, IMS, and CICS.

TSO is the work environment for DB2's interactive facilities

DB2 data can be accessed from application programs written in COBOL, PL/I, or C.



9.2: System Services



## Concept Of System Services

➤ The system services component handles all system-wide tasks like:

- Controlling connections to other MVS subsystems
  - CICS, IDMS/DC and DB, TSO
- Handling system start up, shutdown, and operator communication

9.2: System Services

## Concept Of System Services

- Managing the system log
  - System log is a set of predefined disk data sets that are used to record information for recovering user and system data in the event of a failure
  - Information regarding the log data sets (Active and Archive) is recorded in the BOOT STRAP DATA SET (BSDS).
- Gathering system-wide statistics, performance, auditing and accounting information

### System Services:

This component handles all system wide tasks including connections to other MVS subsystems.

It also handles system startup and shut down operator communication.  
It manages the system log.

The system log is a set of predefined disk data sets that are used to record information for recovering user and system data in the event of a failure.

When an active log data set becomes full (or on operator command ), DB2 switches to a new data set and copies the old one to an archive log data set on disk or tape.

When the active data sets are full, they are recycled. Information regarding the data sets themselves is recorded in the BOOT STRAP DATA SET (BSDS).

Gathering system wide statistics, performance and accounting information.

9.3: Locking Services



## Concept Of Locking Services

- Locking Services are provided by an MVS subsystem called the IMS Resource Lock Manager (IRLM).
- IRLM is a general-purpose lock manager.
- It is used by DB2 to:
  - Control concurrent access to data
- IMS may or may not be present in the system.

9.4: Database Services



## Concept Of Database Services

➤ The purpose of Database Services is:

- to support the definition, retrieval, and update of database data

➤ Its sub-components are as follows:

- Precompiler (PRECOM)
- Bind (BIND)
- Runtime Supervisor (RS)
- Data Manager (DM)
- Buffer Manager (BM)

➤ These components allow:

- Preparation of applications programs for execution
- Subsequent execution of those programs

9.4: Database Services

## Precompiler



- Precompiler is a preprocessor for the host language.
- Functions of Precompiler are:
  - Analyzing the host language source module
  - Stripping out all the SQL statements it finds
  - Replacing them by host language CALL statements
  - From SQL statements it encounters, the precompiler constructs a Database Request Module (DBRM) which becomes input to the BIND component

### Precompiler:

Precompiler is the compiler for the SQL statements. It reads the SQL statements from the DBRMs and produces a mechanism to access data as directed by the SQL statements being bound. The bind plan accepts one or more DBRMs, which are produced from the previous DB2 program precompilation, as input. The output of the bind plan is an application plan containing executable logic representing optimized access paths to DB2 data. An application plan is executable only with a corresponding module. Before you can run a DB2 program, regardless of the environment, an application plan name must be specified.

The major functions of bind are given below:

- Parsing and syntax checking
- Optimization
- Authorization
- Checking that DB2 tables and columns being accessed conform to the corresponding DB2 catalog information

9.4: Database Services



## Precompiler

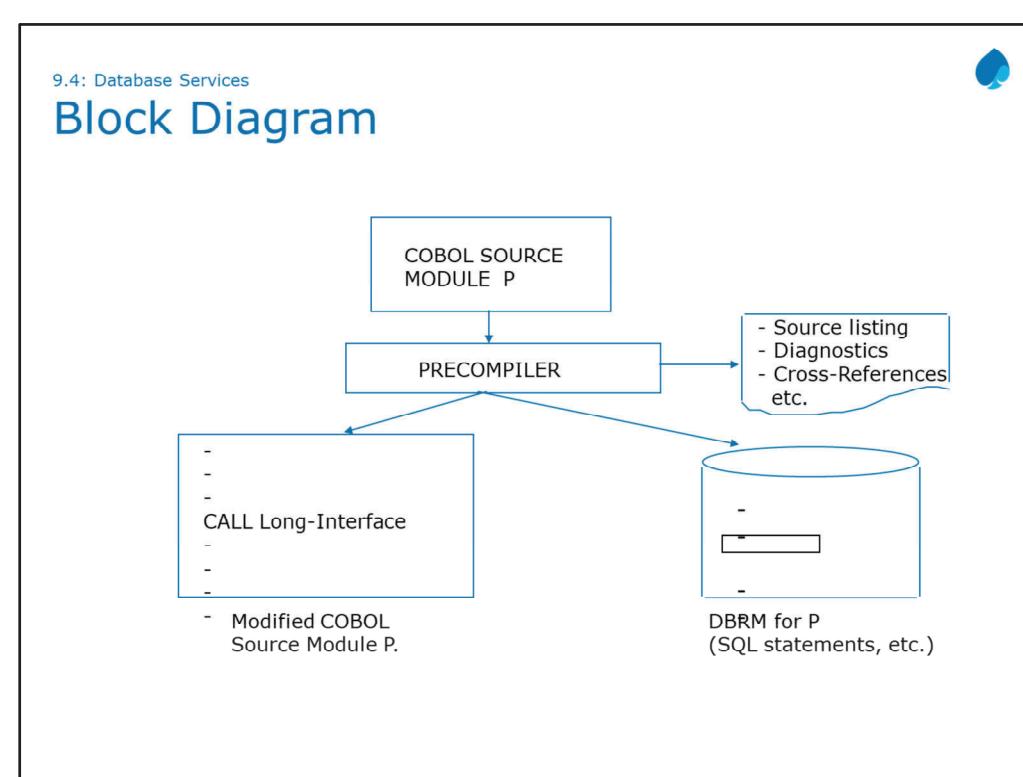
- A precompiler searches for and expands DB2 related include members.
- It searches for SQL statements in the body of the program's source code.
- It creates a modified version of the source program in which every SQL statement in the program is commented out.
- A call to the DB2 runtime interface module, along with applicable parameters, replaces each original SQL statement.
- Precompiler extracts all SQL statements from the program and places them in a database request module.

9.4: Database Services



## Precompiler

- It places a timestamp token in the modified source and the DBRM to ensure that these two items are inextricably tied.
- It reports on the success or failure of the pre-compile process.
- The pre-compiler searches for SQL statements embedded in EXEC and END EXEC SQL.
- A precompiler uses the SQL statements to build a DBRM for P, which it stores away as a member of an MVS partitioned data set.
- DBRM contains a copy of the original SQL statements, together with additional information.



9.4: Database Services



## Concept Of Bind

➤ The function of bind component is twofold:

- It is used to bind a given DBRM to produce what is called a package.
- It is used to bind together a list of packages to produce what is called a application plan.

9.4: Database Services



## Concept Of Bind

➤ Application plan:

- It contains a set of internal control structures.
- It represents compiled form of the original SQL statements from which the DBRMS were built.
- It includes calls on the Data Manager Components.

9.4: Database Services



## Concept Of Bind

- It servers as a compiler for SQL statements.
- It reads SQL statements from DBRMs.
  - It produces an access mechanism.
- The output of the Bind plan command is an application plan:
  - It contains executable logic representing optimized access paths to DB2 data.
  - An application plan is executable only with a corresponding load module.

9.4: Database Services



## Concept Of Bind

- Before you can run a DB2 program, regardless of the environment, an application plan name must be specified.
- DBRMs can be bound into a package. Subsequently packages can be bound into an application plan.
- Advantages of packages is that it is not necessary to recompile the entire plan.

9.4: Database Services



## Concept Of Bind

- A bind converts high-level DB requests (in effect, SQL statements) into optimized internal form.
- Major functions of bind are as follows:
  - Parsing, syntax checking
  - Optimization (Access path selection) Plan generation
  - Authorization checking
  - Check that the DB2 tables and columns being accessed conform to the corresponding DB2 catalog information

9.4: Database Services



## Bind Parameters

➤ Sample Bind Card:

```
//BIND EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN
BIND PLAN(DEPTPLAN)
VALIDATE(BIND)
ISOLATION(CR)
RELEASE(COMMIT)
OWNER(DSRP001)
LIB('DSRP001.DB2.DBRMLIB')
/*
```

9.4: Database Services

## Bind Parameters



### ➤ MEMBER and LIBRARY:

- DBRM is given in MEMBER and the partitioned dataset containing the DBRM is given in LIB.
- During BIND package, if an existing package is used to create the new package then, COPY should be mentioned instead of MEMBER and LIBRARY.
- COPY(collection-id. Package-id)

9.4: Database Services



## Bind Parameters

- Collection id: Packages have three qualifiers. Location id which is used in distributed environment. Collection id is used for grouping of packages.
- Naming syntax: location id. collection-id. package-name.
- Ex. PKLIST(COLL1.\*) bound all the packages with collection id COLL1 to the PLAN.
- PKLIST is a Bind parameter of BIND PLAN. Packages to be connected with PLAN are named here.

9.4: Database Services



## Bind Parameters

### ➤ ACTION:

- Package or plan can be an addition or replacement.
- Default is replacement.
- REPLACE will replace an existing package with the same name, location and collection.
- ADD will add the new package to SYSIBM.SYSPACKAGE.
- Syntax: ACTION(ADD/REPLACE)

9.4: Database Services



## Bind Parameters

### ➤ ISOLATION LEVEL:

- It is used to describes to what extent a program bound to this package can be isolated from the effects of other programs running
- This determines the duration of the page lock.
- Cursor stability(CS): As the cursor moves from the record in one page to the record in next page, the lock over the first page is released . It avoids concurrent updates or deletion of row that is currently processing.
- Repeatable Read(RR): All the locks acquired are retained until commit point.

9.4: Database Services



## Bind Parameters

### ➤ ISOLATION LEVEL:

- Uncommitted Read(UR): It can be applied only for retrieval SQL. There are no locks during READ and so it may read the data that is not committed.

9.4: Database Services



## Bind Parameters

### ➤ ACQUIRE and RELEASE:

- ACQUIRE(USE) and RELEASE(COMMIT): DB2 imposes table or table space lock when it executes an SQL statement that references a table in the table space and it releases the acquired lock on COMMIT or ROLLBACK.
- ACQUIRE(USE) and RELEASE(DEALLOCATE): locks table and table spaces on use and releases when the plan terminates.
- ACQUIRE(ALLOCATE) and RELEASE(DEALLOCATE): when DB2 allocates the program thread, it imposes lock over all the table and table spaces used by the program.
- ACQUIRE(ALLOCATE) and RELEASE(DEALLOCATE) : This is not allowed. It increases the frequency of deadlocks.

9.4: Database Services



## Bind Parameters

### ➤ VALIDATE:

- It controls the handling of 'object not found' and 'not authorized' errors.
- Default is VALIDATE(RUN): If all objects are found and all privileges are held, then don't check once again at execution time. If there is any privilege issue or object not found error, then produce warning messages, bind the package and check authorization and existence at execution time.
- VALIDATE(BIND): If there is any privilege or object not found issue, the produce error message. The statement in error will not be executable .



#### Bind Plan versus Bind Package:

In some cases, however, a DBRM may not be bound directly into a plan. Instead it may first be bound into a package, and then finally the packages may be bound into a plan. There are certain disadvantages of directly binding the DBRMs into an application plan. They are:

If an individual DBRM needs to be recompiled for any reason (for example: some index was dropped), then the entire plan has to be recompiled and rebound.

If multiple plans involve the same DBRM, that same DBRM has to be compiled multiple times, and if that DBRM ever needs to be recompiled, then all relevant plans have to be recompiled and rebound in their entirety. Adding a new DBRM to an existing plan required (again) a recompilation and rebind of the entire plan.

Partly as a consequence of the aforesaid points, bind and rebind times are becoming unacceptably high in some DB2 installations and availability was suffering as a result.

The package concept was introduced to remedy these deficiencies. If a given DBRM needs to be recompiled, then all that has to be done is an appropriate package bind – it is not necessary to recompile the entire plan. Indeed, it may not be necessary to do a new plan bind to incorporate the new package either.

9.4: Database Services



## Runtime Supervisor (RS)

- Runtime Supervisor (RS) is resident in main storage when the application program is executing.
- Its job is to oversee that execution.
  - When application program requests some DB operation to be performed (wishes to execute some SQL), the control goes first to the RS.
  - The RS then uses the control information in the application plan to request the appropriate operations on the part of the Data Manager.

9.4: Database Services



## Data Manager

- The Data Manager performs all the normal access method functions such as search, retrieval, and update.
- In short, this component manages the physical database(s).

### Data Manager:

The Data Manager can be thought of as a very sophisticated access method. It performs all of the normal access method functioning such as search, retrieval, update, index maintenance, and so on. Broadly speaking, the Data Manager is the component that manages the physical database(s). It invokes other system components as necessary in order to perform detailed functions such as locking, logging, I/O operations, and so on during the performance of its basic task.

9.4: Database Services



## Buffer Manager

➤ Buffer Manager is responsible for:

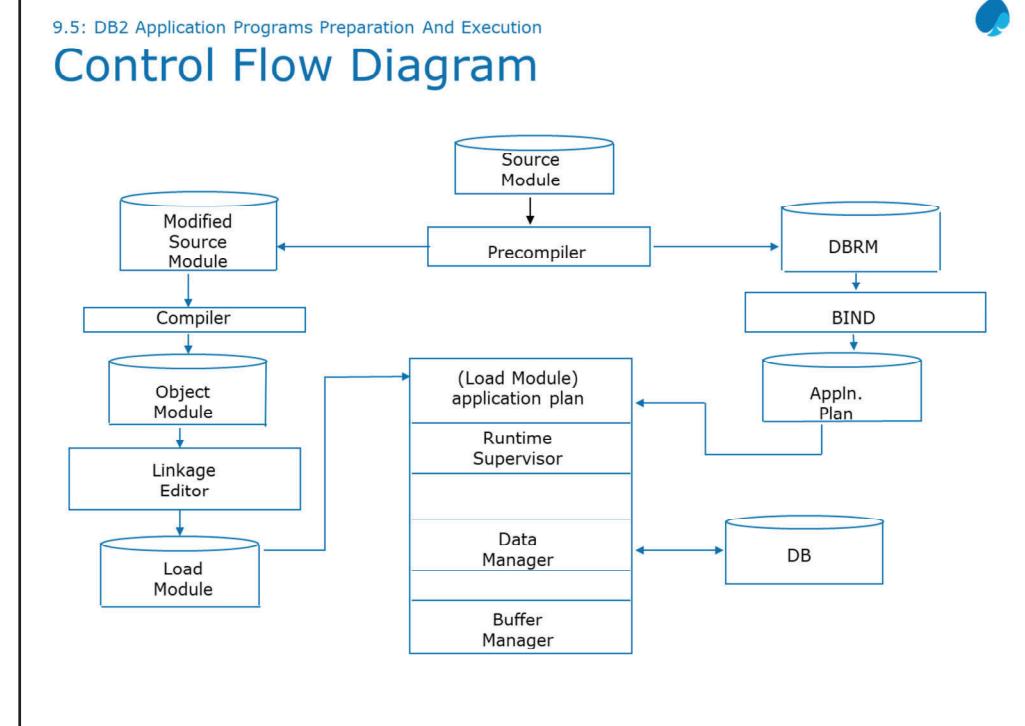
- Physically transferring data between the external medium and (virtual) storage.

➤ In effect, it performs the actual I/O operations.

Buffer Manager:

The Buffer Manager is the component responsible for physically transferring data between external storage and virtual memory.

It employs sophisticated techniques to get the best performance out of the buffer pools under its care and to minimize the amount of physical I/O actually performed.



### DB2 Application Programs Preparation and Execution:

The DB2 catalog stores information about the plan and package.

The DB2 directory stores the actual plan and package.

The Load module, DB2 catalog, and DB2 directory must all be available when you execute a program.

You can bind a program to a package or directly to a plan. However, you cannot run a program that is bound to a package until that package is bound to a plan. For efficiency, a program should be bound to a package that is bound to a plan.

## Summary



➤ In this lesson, you have learnt:

- Various components of DB2 that provide various services



Summary



## Review Questions

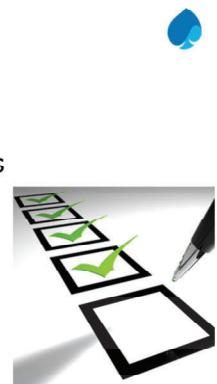
- Question 1: While execution, when application program requests some DB operation to be performed, first control goes to the -----
  - Runtime supervisor
  - Data manger
  - Buffer Manager
  
- Question 2: DBRM's can be bound into a package or directly into an application plan?
  - True/False



## Review Questions

➤ Question 3: Which of the following DB2 services is responsible for writing into log data sets?

- System services
- Database services
- Locking services



DB2

Lesson 10: Embedded SQL

Capgemini

## Lesson Objectives

➤ In this lesson, you will learn about:

- Static SQL
- Dynamic SQL
- Embedded SQL
- Host variables



10.1: Static SQL



## Concept Of Static SQL

- Static SQL is hard-coded into an application program.
- It cannot be modified during the program's execution except for changes to the values assigned to the host variables.
- Cursors are used to access set-level data (that is when a SQL SELECT returns more than 1 row)
- The general form is as shown below:

```
EXEC SQL
[SQL statements]
END-EXEC.
```

### Static SQL:

In static SQL, the SQL statements that are issued are coded, precompiled, compiled, and bound before execution time. With static SQL, the tables your SQL statements access and the functions your statements perform are fixed before the program that contains them is executed.

Although a static SQL statement performs predetermined functions, it does not mean that its operations are limited. Since, you can use host variables in a static SQL statement, its operation can vary. As a result, you can code application programs that issue static SQL statements to meet a wide variety of application requirements. In fact, static SQL lets you accomplish nearly all the tasks you need to perform in application programs.

10.2: Dynamic SQL

## Concept Of Dynamic SQL

- Statements can change throughout the program's execution.
- When the SQL is bound, the application plan or package that is created does not contain the same information as that for a static SQL program.
- The access paths cannot be determined before execution.

### Dynamic SQL:

In contrast, dynamic SQL lets you build and issue statements that can vary completely from one execution to another. Your program constructs a text string that contains the specifications for the statement and stores it in a host variable. Then, when you invoke the statement dynamically at run time, DB2 interprets the string, translates it into an executable SQL statement, binds it, and runs it.

The statements that you cannot issue dynamically fall into three groups:

    Cursor processing statements

    For example: OPEN, FETCH, CLOSE

    Precompiler directives

    For example: DECLARE, INCLUDE, WHENEVER

    Statements for Dynamic SQL processing dynamically

    For example: EXECUTE IMMEDIATE, PREPARE, EXECUTE, DECLARE

10.3: Embedded SQL

## Concept Of Embedded SQL



- Embedded SQL statements are prefixed by EXEC SQL and are terminated by END-EXEC.
- SQL statements can include references to host variables.
- Any tables (base tables or views) used in the program must be declared.
- After any SQL statement has been executed, the feedback information is returned to the program in an area called the SQL Communication Area (SQLCA).
- The embedded SQL SELECT statement requires an INTO clause specifying the host variables.

### Embedded SQL:

Any SQL statement that can be used at the terminal can also be used in an application program.

There are various differences of detail between a given interactive SQL statement and its corresponding embedded form.

## 10.4: Host Variables



## Concept of Host Variables

- Host variables are variables that are:
  - Declared in the working storage section
  - Used by DB2 when it moves data between your program and a table
- They are defined according to the rules of programming language.
- SQL statements can include references to host variables. Such references are prefixed with a colon to distinguish them from SQL column names.

## 10.4: Host Variables



## Concept of Host Variables

- Host variables can appear:
  - In SQL data manipulation statements.
- They are generally used for designating a target for retrieval.
- Such variables can appear in the following positions:
  - INTO clause in SELECT or FETCH (target for retrieval)
  - WHERE or HAVING clause (value to be compared)
  - SET clause in UPDATE (value to be assigned)
  - VALUES clause in INSERT (value to be inserted)

## 10.4: Host Variables

## Concept of Host Variables

➤ Let us see an example of Host Variables:

```
EXEC SQL
 SELECT STATUS , CITY INTO
 :STATUS , :CITY
 FROM EMPLOYEE
 WHERE SNO = :GIVEN-SNO
END-EXEC.
```



## Summary



➤ In this lesson, you have learnt:

- The method to embed your DB2 SQL statements in your application program
- The method to use host variables with your embedded SQL statements



Summary

## Review Questions

➤ Question 1: After any SQL statement has been executed, feedback information is returned to the program in an area called -----

- SQLCA
- SQLAREA
- SQLERR

➤ Question 2: Every SQL statements inside a COBOL-DB2 application program must be inside EXEC SQL and END-EXEC

- True/False



## DB2

Lesson 11: Indicator  
Variable

## Lesson Objectives

➤ In this lesson, you will learn about:

- Indicator variable



## 11.1: Indicator Variable

## Concept of Indicator Variable

- An indicator variable is used if there is a chance that the source of a retrieval operation might be null.
- The user should include an indicator variable in the INTO clause in addition to the normal target variable.

### Indicator Variable:

An indicator variable is defined in the working storage for a column that can have a NULL value. On a SELECT or FETCH statement, DB2 will place a negative value in the variable if the column has a NULL value. The corresponding COBOL data-name is not changed and therefore retains whatever value it received from a previous SELECT or FETCH statement. On an UPDATE or INSERT statement, the programmer places a negative value in the variable to indicate to DB2 that a NULL value needs to be used for the column. In this case, the column cannot be defined in the table as NOT NULL.

Unless an indicator variable is used for a field, on an input operation (SELECT, FETCH) there is an SQL error if the field does contain NULLs; on an output operation (INSERT, UPDATE). DB2 will not be able to insert or update NULLs into the field.

I1.1: Indicator Variable

## Concept of Indicator Variable



### ➤ Example of Indicator Variable:

EXEC SQL

```
SELECT STATUS, CITY
INTO :STATUS:STATUS-IND, :CITY:CITYIND
FROM S
WHERE SNO=:GIVEN-SNO
END-EXEC.
```

- For processing and finding out whether the fields were NULL or not, a check can be performed as:

- IF STATUS-IND < 0 THEN /\*STATUS WAS NULL \*/

11.2: Defining The Indicator Variable



## Explanation

- Each indicator variable is a half-word integer (PIC S9(4) COMP).
- If defined as an array (OCCURS clause), then:
  - It may be used for a list of columns with the first occurrence of the indicator variable corresponding to the first column in that list.
  - Using a single indicator variable, NULLS can be handled for all the fields of the table.

## 11.2: Defining The Indicator Variable



## Explanation

- The indicator variable will have a negative value if the select returns a null value.
- If the indicator variable returns a value  $> 0$ , then it indicates a truncated variable as the length of the character string before truncation.

## 11.2: Defining The Indicator Variable



## Explanation

➤ If the Indicator variables contain:

- A negative number, then it indicates that the column has been set to Null.
- The value -2, then it indicates that the column has been set to Null as a result of Data Conversion Error.
- A positive or a zero value, then it indicates that the column is not Null.

➤ Suppose a column defined as a char data type is truncated on retrieval because the host variable is not large enough. Then the indicator Var contains the original length of the truncated column.

11.2: Defining The Indicator Variable



## Illustration

➤ Let us see an example on defining the Indicator variable:

```
01 DEPT-INDICATORS
 10 DEPT-IND OCCURS 5 TIMES PIC S9(4) COMP.
```

```
EXEC SQL
 SELECT DEPTNO, DNAME, MGR, HO, LOC
 INTO :DCLDEPT: DEPT-IND
 FROM DEPT
 WHERE DEPTNO = 'A00'
END-EXEC
```

11.3: Using Indicator Variables on UPDATE and INSERT

## Illustration

- Example 1: Indicator variables can be used in the VALUES clause to insert NULL values.

```
IF COLORIND < 0 OR CITYIND < 0
EXEC SQL
 INSERT INTO P (PNO,COLOR, CITY)
 VALUES
 (:PNO, :PCOLOR:COLOR-IND, :PCITY:CITYIND)
 END-EXEC.
```

### Using Indicator Variables on UPDATE and INSERT:

On an UPDATE or INSERT statement, it is the programmer who indicates (by placing a negative value in the appropriate indicator variable) that we will use nulls for a column used in the UPDATE or INSERT statement. Naturally, such a column must not be defined with the NOT NULL attribute.

An example of using an indicator variable in an INSERT operation is shown below:

```
***** Set EMPNO, EMPNAME, EMPSALARY to the correct
*****values.
IF condition-1
 MOVE 0 TO EMPSALARY - IND
ELSE
 MOVE - 1 TO EMPSALARY-IND

EXEC SQL
 INSERT INTO EMPTABLE
 (:EMPNO, EMPNAME, EMPSALARY)
 VALUES (: EMPNO, :EMPNAME, EMPSALARY:
 EMPSALARY-IND)
 END - EXEC.
```

## 11.3: Using Indicator Variables on UPDATE and INSERT



## Illustration

- Example 2: Indicator variables can appear on the right hand side of an assignment in the SET clause to set a value to NULL. For example, see the following sequence:

```
IF STATUS-IND = -1

EXEC SQL
 UPDATE S
 SET STATUS = :STATUS:STATUS-IND
 WHERE CITY ='LONDON'
END-EXEC.
```

## Summary

➤ In this lesson, you have learnt:

- The method to use indicator variables to find, store, and update NULL values.



## Review Questions



➤ Question 1: What is the size of an indicator variable is a half-word integer (PIC S9(4) COMP).

- 01 IND1 PIC S9(8) COMP
- 01 IND1 PIC S9(4) COMP
- 01 IND1 PIC S9(4) COMP-1



➤ Question 2: Which of the following is / are function/s of an indicator variable?

- To insert null values
- To update a column containing null values
- To find the column containing not null values

DB2

Lesson 12: DB2 Interactive Interface

Capgemini

## Lesson Objectives



➤ In this lesson, you will learn about:

- DCLGEN



12.1: DCLGEN



## Concept of DCLGEN

- DCLGEN allows the user to invoke the declarations generator program.
- Program preparation includes the following steps :
  - precompilation
  - compilation or assembly
  - linkage editor processing
  - bind processing
  - program execution (TSO only)

12.1: DCLGEN

## Using DCLGEN



DB2I PRIMARY OPTION MENU                    SSID: DST2  
COMMAND ===> 2  
Select one of the following DB2 functions and press ENTER.  
1 SPUFI                                        (Process SQL statements)  
2 DCLGEN                                        (Generate SQL and source language declarations)  
3 PROGRAM PREPARATION (Prepare a DB2 application program to run)  
4 PRECOMPILE                                    (Invoke DB2 precompiler)

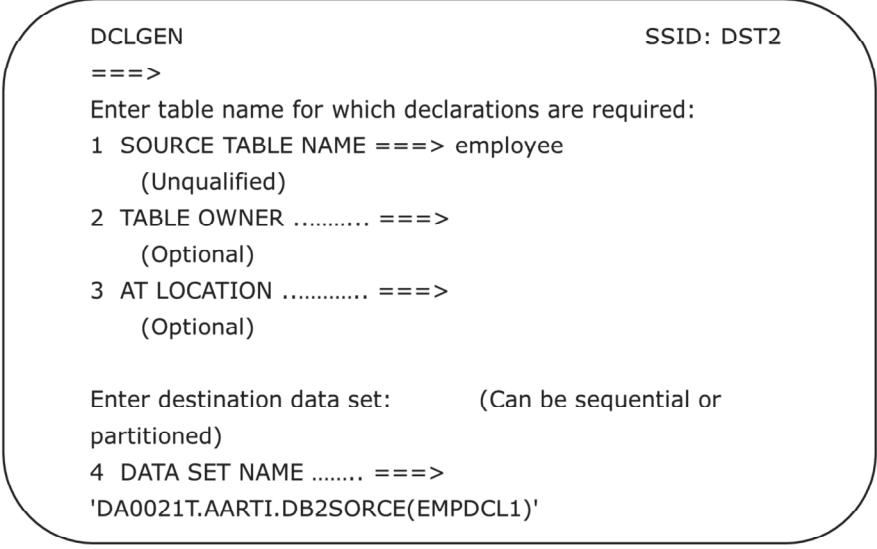
12.1: DCLGEN

## Using DCLGEN

- 5 BIND/REBIND/FREE (BIND, REBIND, or FREE plans or packages)
- 6 RUN (RUN an SQL program)
- 7 DB2 COMMANDS (Issue DB2 commands)
- 8 UTILITIES (Invoke DB2 utilities)
- D DB2I DEFAULTS (Set global parameters)
- X EXIT (Leave DB2I)

12.1: DCLGEN

## Using DCLGEN



DCLGEN  
====>  
Enter table name for which declarations are required:  
1 SOURCE TABLE NAME ===> employee  
(Unqualified)  
2 TABLE OWNER ..... ===>  
(Optional)  
3 AT LOCATION ..... ===>  
(Optional)

Enter destination data set: (Can be sequential or partitioned)  
4 DATA SET NAME ..... ===>  
'DA0021T.AARTI.DB2SORCE(EMPDCL1)'

12.1: DCLGEN

## Using DCLGEN



- 5 DATA SET PASSWORD ===> (If password protected)  
Enter options as desired:
- 6 ACTION ..... ==> REPLACE (ADD new or REPLACE old declaration)
- 7 COLUMN LABEL .... ==> YES (Enter YES for column label)
- 8 STRUCTURE NAME ... ==> (Optional)
- 9 FIELD NAME PREFIX...==> (Optional)
- 10 DELIMIT DBCS ..... ==> YES (Enter YES to delimit DBCS identifiers)

12.1: DCLGEN

## Using DCLGEN



11 COLUMN SUFFIX ..... ==> NO (Enter YES to append column name)  
12 INDICATOR VARS ..... ==> NO (Enter YES for indicator variables)

12.1: DCLGEN

## DCL Structure



- DCL structure is the COBOL equivalent structure declaration for the DB2 table
- This structure can be referenced in the COBOL program by using INCLUDE statement
- Eg.

```
EXEC SQL
INCLUDE EMPDCL
END-EXEC
```

- Here EMPDCL is the name of the Member in the DCL Library.

12.1: DCLGEN  
**DCL Structure**

```
01 DCLEMPLOYEE.
 10 EMPNO PIC X(6).
 10 FIRSTNME.
 49 FIRSTNME-LEN PIC S9(4) USAGE COMP.
 49 FIRSTNME-TEXT PIC X(12).
 10 MIDINIT PIC X(1).
 10 LASTNAME.
 49 LASTNAME-LEN PIC S9(4) USAGE COMP.
 49 LASTNAME-TEXT PIC X(15).
 10 WORKDEPT PIC X(3).
 10 PHONENO PIC X(4).
 10 HIREDATE PIC X(10).
 10 JOB PIC X(8).
 10 EDLEVEL PIC S9(4) USAGE COMP.
 10 SEX PIC X(1).
 10 BIRTHDATE PIC X(10).
 10 SALARY PIC S9(7)V9(2) USAGE COMP-3.
 10 BONUS PIC S9(7)V9(2) USAGE COMP-3.
 10 COMM PIC S9(7)V9(2) USAGE COMP-3.
 01 IEMPLOYEE.
 10 INDSTRUC PIC S9(4) USAGE COMP OCCURS 14 TIMES.
```

12.1: DCLGEN

## Sample SQL Embedded Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROCESS.
DATA DIVISION.
WORKING-STORAGE SECTION.
 EXEC SQL INCLUDE SQLCA END-EXEC.
 EXEC SQL INCLUDE EMPDCL END-EXEC.
 01 SQLNO PIC -9(3).
```

12.1: DCLGEN

## Sample SQL Embedded Program

```
PROCEDURE DIVISION.
0001PROCESS.
 EXEC SQL SELECT EMPNO,EMPNAME,PHONENO
 INTO :EMPNO, :EMPNAME,:PHONENO FROM EMPLOYEE
 WHERE EMPNO = 'EM0003'
 END-EXEC.
 IF SQLCODE = 0
 DISPLAY EMPNO " " EMPNAME " " PHONENO
 ELSE
 MOVE SQLCODE TO SQLNO
 DISPLAY 'SQLCODE :' SQLNO
 END-IF.
 STOP RUN.
```

## Review Question

➤ Question 1: Which of the following is used to generate host variables?

- SPUFI
- QMF
- DCLGEN



## DB2

### Lesson 13: Cursors

Capgemini

## Lesson Objectives

➤ In this lesson, you will learn:

- What are cursors?
- Declaring a cursor
- Fetching a row from cursor
- Closing a cursor



13.1: Cursors



## Concept of Cursor

- A cursor is a mechanism allowing an application program to retrieve a set of rows.
- Following steps have to be performed to use cursors:
  - Declare cursor: to define a cursor.
  - Open cursor: to create a result table.
  - Fetch cursor: to retrieve rows from cursor, one at a time to be executed in a loop.
  - Close cursor: to close the cursor.
- The cursor facility allows a COBOL program to gain addressability to individual row occurrences of a many-rows result table.

### Cursors:

A cursor is a mechanism allowing an application program to retrieve a set of rows. The cursor facility allows a COBOL program to gain addressability to individual row occurrences of a many-rows result table.

Following steps have to be performed to use cursors:

- Declare cursor: to define cursor.
- Open cursor: to create result table.
- Fetch cursor: to retrieve rows from cursor, one at a time to be executed in a loop.
- Close cursor: to close the cursor.

13.1: Cursors



## Defining the Cursor

### ➤ Declaring (defining) a cursor.

- It is done in the data division of your program.
- This is purely declarative in nature.

➤ Cursor name should begin with a letter. It must not exceed 18 characters.

### Defining the Cursor:

Declaring (defining) a cursor is done in the data division of your program.

This is purely declarative in nature. Therefore no information is retrieved from the database yet. The Cursor name should begin with a letter and must not exceed 18 characters.

When the OPEN statement is encountered, the SELECT in the cursor declaration is executed. The OPEN cursor statement not only executes the selection of data from the DB2 database, but also establishes the initial position of the cursor in the results table. One program can have multiple cursors.

13.1: Cursors



## Illustration

➤ Let us see an example on defining a cursor:

```
EXEC SQL
 DECLARE cursor CURSOR FOR SELECT col1,
 col2....
 FROM table
 WHERE condition [FOR UPDATE OF col1, col2 ..]
 END-EXEC
```

13.1: Cursors



## Opening a Cursor

➤ Let us see an example on opening a cursor:

```
EXEC SQL
 OPEN cursor
END-EXEC
```

13.1: Cursors



## Fetching a Row from Cursor

➤ Let us see an example on fetching a row from a cursor:

```
EXEC SQL
```

```
 FETCH cursor INTO :host var1, :host var2,...
```

```
END-EXEC
```

13.1: Cursors

## Closing a Cursor



➤ Let us see an example on closing a cursor:

```
EXEC SQL
 CLOSE cursor
END-EXEC
```

13.2:An Example Based On a Single Table



## Illustration

➤ Let us see an example based on a single table:

```
EXEC SQL
 DECLARE X CURSOR FOR
 SELECT S#, SNAME, STATUS
 FROM S WHERE CITY = :Y
 END-EXEC.

 EXEC SQL
 OPEN X
 END-EXEC.
```

13.2:An Example Based On a Single Table

## Illustration

```
LOOP UNTIL NO MORE ROWS OR ERROR
 EXEC SQL
 FETCH X INTO :S#, :SNAME.:SNAME-IND
 END-EXEC.
 PROCESSING STATEMENTS .
 EXEC-SQL
 CLOSE X
 END-EXEC.
```

13.3: Data Modification

## Concept of Data Modification



- Statements used for Data Modification are:
  - UPDATE
  - DELETE
- They operate on data a set at a time
- It is accomplished with a cursor.
  - Special clause is used, namely: WHERE CURRENT OF.
  - The cursor is declared with a special FOR UPDATE OF clause.

### Data Modification:

Often an application program must read data, and then based on its value, either update or delete data. One can use the UPDATE or DELETE SQL statements to modify and delete rows from DB2 tables.

These statements are similar to SELECT statements which operate on a set of data at any given point of time.

This is accomplished with a cursor and a special clause of the UPDATE and DELETE statements usable only by embedded SQL, namely WHERE CURRENT OF. The cursor is declared with a special FOR UPDATE OF CLAUSE.

A FOR UPDATE OF clause appears with:

SELECT statement to indicate what columns can be updated when retrieved

The columns to be updated must be listed in the FOR UPDATE OF CLAUSE or the DECLARE.

You do not have to select a column to update it.

13.3: Data Modification



## Concept of Data Modification

- A FOR UPDATE OF clause appears with:
  - SELECT statement to indicate what columns can be updated when retrieved.
- The columns to be updated must be listed in the FOR UPDATE OF CLAUSE or the DECLARE.
- You do not have to select a column to update it.

13.3: Data Modification



## Concept of Data Modification

### ➤ Rules for Update:

- The SELECT statement must be on a single table and not on a join.
- If the DECLARE cursor statement contains a subquery, then it must not be on the same table as the main query.
- You cannot use DISTINCT, GROUP BY, ORDER BY, or BUILT-IN functions.
- Only those columns are eligible for updation which are selected in the FOR.. UPDATE clause.

13.3: Data Modification

## Example: Current Forms of Update and Delete

➤ Let us see an example on current forms of UPDATE and DELETE:

```
EXEC SQL
 UPDATE table
 SET field = : exp [, field = : exp]
 WHERE CURRENT OF cursor
 END-EXEC.
```

```
EXEC SQL
 DELETE FROM table
 WHERE CURRENT OF cursor
 END-EXEC.
```

13.3: Data Modification

## Example of Data Modification



➤ Let us see an example on Data Modification:

```
EXEC SQL
 Declare C1 cursor for
 Select Deptno, Deptname, Mgrno From Dept
 Where ADMRDEPT = :ADMRDEPT
 for update of MGRNO
 END-EXEC
```

13.3: Data Modification

## Example of Data Modification

```
PROCEDURE DIVISION.
 MOVE 'A00' TO ADMRDEPT.
 EXEC SQL
 OPEN C1
 END-EXEC
 PERFORM 200-MODIFY-DEPT-INFO UNTIL NO-MORE-
 ROWS.
```

## 13.3: Data Modification

## Example of Data Modification

```
EXEC SQL
 CLOSE C1
END-EXEC.
200-Modify-Dept-Info.
EXEC SQL
 Fetch C1 into :deptno, :deptname, :mgrno
END-EXEC.
```

13.3: Data Modification

## Example of Data Modification

```
If sqlcode < 0
 GO TO 9999-error-paragraph.
If sqlcode = +100
 Move 'NO' to more-rows
Else
 EXEC SQL
 Update Dept Set MGRNO = '00000'
 Where current of C1
 END-EXEC.
```

### 13.3: Data Retrieval

## Example

➤ Let us see an example on Data Retrieval.

- A pseudo code for retrieving data from an application join, using cursors is shown below:

```
EXEC SQL
Declare Deptcur cursor for
 Select Deptno, Deptname
 From Dept
END-EXEC.
```

```
EXEC SQL
Declare Empcur cursor for
 Select empno, salary from emp
 where workdept = :hv-workdept
END-EXEC.
```



13.3: Data Retrieval

## Example

```
EXEC SQL
 Open Deptcur
END-EXEC
```

```
Loop until no more dept rows or error
EXEC SQL
 fetch deptcur into :deptno, :Deptname
END-EXEC.
```

```
Move deptno to HV-WORKDEPT
EXEC SQL
 OPEN Empcur
END-EXEC.
```

## 13.3: Data Retrieval

## Example

```
Loop until no more employee rows or error
EXEC SQL
 FETCH EMPCUR INTO
 :EMPNO, :SALARY
 END-EXEC.
 Process retrieved data
 End Loop (2)
End of Loop (1)
```



13.3: Data Retrieval



## With Hold Option

### ➤ Normal problem:

- With COMMIT - closing of the cursor.
- Re-positioning required.

### ➤ WITH HOLD:

- It is an optional specification on a cursor declaration.
- It does away with normal problems.
- It is illegal with DELETE and UPDATE CURRENT.
- COMMIT must be followed with a FETCH.

#### WITH HOLD Option:

WITH HOLD is an optional specification on a cursor declaration. The significance can be understood by considering what happens in its absence.

Suppose we need to process some large table, one row at a time by means of a cursor, and update a few of them as we go. It is often desirable to divide the work into batches and to make the processing of each batch into a separate transaction (by issuing a separate COMMIT at the end of each transaction). For example: A table of one million rows might be processed by a sequence of 10,000 transactions, each one dealing with just 100 rows. In this way, for example, if it becomes necessary to roll a given transaction back, then at most 100 updates will have to be undone, instead of potentially as many as a million.

However, the problem with this approach is that every time we issue a COMMIT, we implicitly close the cursor, thereby losing our position within the table. Therefore the first thing each transaction has to do is to execute some re-positioning code in order to get back to the row that is due to be processed next. This re-positioning code can often be quite complex, especially if the processing sequence is determined by a combination of several columns.

Suppose the cursor declaration specifies WITH HOLD, however COMMIT does not close the cursor, instead leaves it open, positioned such that the next FETCH will move it to the next row in sequence. Then possibly the complex code for repositioning is thus no longer required.

However it is important to note that the first operation on the cursor following the COMMIT must be FETCH. The UPDATE and DELETE CURRENT are illegal.

## Summary

➤ In this lesson, you have learnt:

- A cursor is a mechanism allowing an application program to retrieve a set of rows.
- The cursor facility allows to gain addressability to individual row occurrences of a many-rows result table.



## Review Questions

➤ Question 1: Which of the operations is / are declarative?

- DECLARE cursor
- OPEN cursor
- FETCH

➤ Question 2: If a column needs to be updated then it must be present in the 'for update of.' clause of DECLARE cursor?

- True/False



## DB2

### Lesson 14: Errors / Exception Handling

Capgemini

## Lesson Objectives

➤ In this lesson, you will learn about:

- Error Handling
- SQL Communication Area



14.1: Error Handling

## Introduction



➤ To facilitate error processing, DB2 provides:

- The SQL communication area
- A subprogram named DSNTIAR for reporting errors
- A WHENEVER statement

14.2: SQL communication area

## Explanation

- An SQLCA is a collection of variables that is updated at the end of the execution of every SQL statement
- SQL communications area structure is used within the DB2 program to return error information to the application program
- There are multiple fields inside SQLCA. For ex.
  - SQLCODE – most widely used in application programs
  - SQLERRD
  - SQLWARN

14.2: SQL communication area

## SQLCODE field

- 0 means successful execution
- A positive number means successful execution with one or more warnings. An example is +100 which means no rows found
- A negative number means unsuccessful with an error.
  - An example is -305 which means null value occurred, and no indicator variable was defined



Most of the program use only the SQLCODE field in the SQL communication area.

```
SQLCA.
05 SQLCAID PIC X(8).
05 SQLCABC PIC S9(9) COMP-4.
05 SQLCODE PIC S9(9) COMP-4.
05 SQLERRM.
 49 SQLERRML PIC S9(4) COMP-4.
 49 SQLERRMC PIC X(70).
05 SQLERRP PIC X(8).
05 SQLERRD OCCURS 6 TIMES
 PIC S9(9) COMP-4.
05 SQLWARN.
 10 SQLWARN0 PIC X.
 10 SQLWARN1 PIC X.
 10 SQLWARN2 PIC X.
 10 SQLWARN3 PIC X.
 10 SQLWARN4 PIC X.
 10 SQLWARN5 PIC X.
 10 SQLWARN6 PIC X.
 10 SQLWARN7 PIC X.
05 SQLEXT.
 10 SQLWARN8 PIC X.
 10 SQLWARN9 PIC X.
 10 SQLWARNA PIC X.
 10 SQLSTATE PIC X.
```

**The most useful fields in the SQL communication area:**

| Field      | Data type       | Description                                                                                |
|------------|-----------------|--------------------------------------------------------------------------------------------|
| SQLCODE    | Binary fullword | SQL return code                                                                            |
| SQLERRD(3) | Binary fullword | Number of rows affected by an INSERT, DELETE, or UPDATE statement                          |
| SQLERRD(5) | Binary fullword | Contains the column (position) fo the syntax error for a dynamic SQL statement             |
| SQLWARN0   | 1-byte string   | Contains W if any other SQLWARN field contains W                                           |
| SQLWARN1   | 1-byte string   | Contains W if a string was truncated when stored in a host variable                        |
| SQLWARN2   | 1-byte string   | Contains W if null values were excluded during the processing of a column function         |
| SQLWARN3   | 1-byte string   | Contains W if the number of columns and host variables don't match                         |
| SQLWARN4   | 1-byte string   | Contains W if an UPDATE or DELETE statement issued dynamically doesn't have a WHERE clause |
| SQLWARN6   | 1-byte string   | Contains W if an arithmetic operation produces an unusual date or timestamp                |
| SQLSTATE   | 5 byte string   | Contains a return code indicating the status of the most recent SQL statement              |

DB2 uses the SQLWARN fields to report some unusual conditions that aren't considered to be errors. The SQLSTATE field is similar to the SQLCODE field in that it contains a return code indicating the status of the most recent SQL statement. Although the SQLCODE field is unique to DB2 for MVS, the SQLSTATE field can be used across DB2 (and ANSI-compliant SQL) platforms.

14.3: WHENEVER

## Explanation



### ➤ Syntax

```
EXEC SQL
WHENEVER condition action
END-EXEC.
```

### ➤ where "condition" is one of the following :

- NOT FOUND means SQLCODE = 100
- SQLWARNING means SQLCODE >0 & NOT = 100
- SQLERROR means SQLCODE < 0

### ➤ "action" can be:

- CONTINUE statement or
- GO TO statement

|      |                                                                               |
|------|-------------------------------------------------------------------------------|
| 000  | Successful execution                                                          |
| +100 | Row not found                                                                 |
| -530 | Invalid foreign key value                                                     |
| -532 | Attempt to delete a row that is RESTRICTED due to referential integrity rules |
| -551 | You don't have the authority to perform a function on a DB2 object            |
| -803 | Duplicate value for a unique column                                           |
| -811 | SELECT returned > 1 row and you are not using a cursor                        |
| -911 | Deadlock - UOW is rolled back                                                 |
| -913 | Deadlock - Unsuccessful execution                                             |
| -922 | Authorization failure                                                         |
| -923 | DB2 connection failure                                                        |

14.3: WHENEVER



## Explanation

- WHENEVER checks the SQL code automatically.
  - It is based on the value it finds. It takes the action you specify.
  - If you omit the action for a WHENEVER statement, then the default of CONTINUE will apply for that condition.
- There is no limit to the number of WHENEVER statements you can use.

14.3: WHENEVER

## Illustration

➤ Let us see an example using 'WHENEVER':

```
1000-INQUIRY.
```

```
EXEC SQL
 WHENEVER SQLERROR
 GOTO 1000-UNDO
 END-EXEC.
```

```
EXEC SQL
 SELECT FLD1, FLD2 INTO :FLD1, :FLD2
 FROM EMP-TABLE WHERE CODE = 113
 END-EXEC.
```



14.3: WHENEVER

## Illustration (contd..)

```
1000-UNDO.
DISPLAY 'ERROR! CAN'T PROCEED'.

EXEC SQL
 ROLLBACK
END-EXEC.

1000-EXIT.
EXIT.
```

**SQL Warnings:**

- **SQLERRD** is an array of six full word items. The third of the six, contains useful information.
- After an INSERT, DELETE, or UPDATE statement, the SQLERRD(3) contains the number of rows that the statement has affected.
- For example:

```
EXEC SQL
 DELETE FROM EMP
 WHERE DNO IN
 (SELECT DNO FROM MASTER_DEPT)
END-EXEC.
Display 'SQLERRD(3)' ROWS WERE DELETED.
```

**DB2 Error Codes:**

| SQLCODE | Keyword   | Meaning                                                                                                                                                                                                                                              |
|---------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| +100    | SELECT    | Row not found during FETCH, SELECT, UPDATE, or DELETE                                                                                                                                                                                                |
| +304    | Program   | Value and host variable are incompatible                                                                                                                                                                                                             |
| -305    | Variables | Null value occurred, and no indicator variable was defined                                                                                                                                                                                           |
| -501    | Cursor    | Cursor named in FETCH or CLOSE is not open                                                                                                                                                                                                           |
| -551    | Authority | You lack the authority to access the named object, possibly because its name is not spelled correctly.                                                                                                                                               |
| -803    | Updating  | Duplicate keys not allowed                                                                                                                                                                                                                           |
| -805    | Plan      | DBRM not bound into this plan                                                                                                                                                                                                                        |
| -811    | SELECT    | Embedded SELECT or sub select returned more than one row                                                                                                                                                                                             |
| -818    | Plan      | Timestamps in load module and plan do not agree; program was probably re-precompiled without being rebound                                                                                                                                           |
| -901    | System    | Mysterious system error, permits running more SQL statements                                                                                                                                                                                         |
| -904    | System    | Unavailable resources; if resource name is a table, view, and so on, this was probably caused by contention and re-trying the operation may work; if resource name is a 44 character VSAM file name, the file has probably been archived or deleted. |
| -911    | System    | Deadlock or timeout, updates rolled back                                                                                                                                                                                                             |
| -913    | System    | Deadlock or timeout, updates not rolled back                                                                                                                                                                                                         |

14.4: DSNTIAR subprogram



## Explanation

- DSNTIAR is an error-reporting program that comes with DB2. It takes data from the communication area, adds explanatory text, and formats it in a readable form. Then your COBOL program can display the message or save it on disk.

14.4: DSNTIAR subprogram

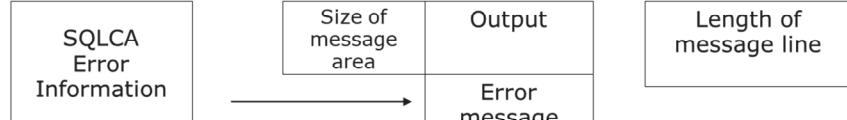
## Error Message Formatting Routine



### ➤ ERROR Message Formatting Routine :

```
CALL 'DSNTIAR' USING SQLCA,
 ERROR-MSG, ERROR-TXT-LENGTH
```

And pass it three areas :



### ➤ Notes:

- Attention should be paid not to invoke DSNTIAR unless an error condition is detected, because the module is dynamically loaded into storage when invoked. If you do this without care, you will waste a lot of resources. It is wise to use a general error routine that is called after each SQL statement.

14.4: DSNTIAR subprogram

## Error Message Formatting Routine

### ➤ Call statement in the Procedure Division

```
CALL 'DSNTIAR' USING SQLCA
 ERROR-MESSAGE
 ERROR-LINE-LENGTH.
```

### ➤ Related field definitions in working-storage

```
01 ERROR-MESSAGE.
 05 ERROR-MESSAGE-LENGTH PIC S9(4) COMP VALUE +800.
 05 ERROR-MESSAGE-LINE PIC X(80) OCCURS 10 TIMES
 INDEXED BY EML-INDEX.
01 ERROR-LINE-LENGTH PIC S9(9) COMP VALUE +80.
```

14.4: DSNTIAR subprogram



## Error Message Formatting Routine

### ➤ Description

- The DSNTIAR subprogram requires three arguments:
- the name of the SQL communication area
- the name of the data area that will receive the formatted message from DSNTIAR
- the name of a binary fullword field that contains the length of the message lines that DSNTIAR will return

14.4: DSNTIAR subprogram



## DSNTIAR Return Codes

➤ **Code      Meaning**

- 0              Successful execution.
- 4              More data was available than could fit into the provided message area.
- 8              The logical record length was not between 72 and 240, inclusive.
- 12             The message area was not large enough, or the message length was 240 or greater.
- 16             Error in TSO message routine.20Module DSNTIA1 couldn't be loaded.
- 24             SQLCA data error.

14.4: DSNTIAR subprogram



## DSNTIAR Output

### ➤ Sample DSNTIAR output after an unsuccessful SELECT statement

```
DSNT404I SQLCODE = 100, NOT FOUND: ROW NOT FOUND FOR FETCH,
 UPDATE, OR DELETE, OR THE RESULT OF A QUERY IS AN EMPTY TABLE
DSNT415I SQLERRP = DSNXRFCH SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD = 110 0 0 1 0 0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD = X'FFFFFFFFFF92' X'00000000' X'00000000' X'FFFFFFFF'
 X'00000000' X'00000000' SQL DIAGNOSTIC INFORMATION
```

### ➤ Description

- DSNTIAR takes data from the DB2 communication area, adds explanatory text, and formats it in a more readable form.
- DSNTIAR returns the message as a variable-length field.
- DSNTIAR can return a maximum of 10 message lines.

## Summary

➤ In this lesson, you have learnt:

- The use of WHENEVER statement in exception handling



## Review Questions



➤ Question 1: Which of the following fields of SQLCA contains number of rows affected by an INSERT, DELETE, or UPDATE statement?

- Option 1: SQLCODE
- Option 2: SQLERRD
- Option 3: SQLSTATE



➤ Question 2: Which of the following is an error-reporting program ?

- Option 1: SQLCA
- Option 2: DSNTIAR
- Option 3: WHENEVER

## DB2

### Lesson 15: Transaction Processing

Capgemini



## Lesson Objectives

➤ In this lesson, you will learn about:

- Transactions
- COMMIT and ROLLBACK



15.1: What is a Transaction?



## Explanation

### ➤ Transaction:

- Logical unit of work.
- Database may not be consistent between any two SQLs.
- Sequence of several such SQL operations that transform a consistent state of the database into another consistent state.

A transaction is a logical unit of work.

Consider the following example:

Suppose, for the sake of example the parts table contains an additional column totqty representing the total quantity for the part in question.

The value of the totqty for any part is supposed to be equal to the sum of all sp.qty values taken all over sp rows for that part. Now, consider the following sequence of operations, the intent of which is to add a new shipment (S5,P1,1000) to the database:

15.1: What is a Transaction?

## Sample Code for Transaction

```
EXEC SQL
 WHENEVER SQLERROR GOTO UNDO
END-EXEC.

EXEC SQL
 INSERT INTO SP VALUES ('S5'/P1,1000)
END-EXEC.

EXEC SQL
 UPDATE P
 SET TOTQTY = TOTQTY + 1000
 WHERE P# = 'P1'
END-EXEC.
```



15.1: What is a Transaction?

## Sample Code for Transaction

```
EXEC SQL
 COMMIT
END-EXEC.

GO TO FINISH.

UNDO :
EXEC SQL
 ROLLBACK
END-EXEC.

FINISH :
RETURN.
```

The insert adds the new shipment to the sp table, the update updates the totqty column for the part p1 appropriately.

The point of example is that what is presumably intended to be a single atomic transaction “create a new shipment” –infact involves 2 updates to the database. What is more is that the database is not consistent even between these two updates. Thus a logical unit of work is not necessarily just one SQL operation ; rather it is a sequence of several such operations, in general, that transforms a consistent state of the database into another consistent state, without necessarily preserving consistency at all intermediate points. Now it is clear that what must not be allowed to happen in the example is for one of the two updates to execute and the other not (because then the database would be in an inconsistent state). Ideally we would want that both the updates are done, but we cannot have such a guarantee: there is always a chance that things will go wrong. For example a system crash may occur between the two updates.

But a system that supports transaction processing does provide the next best thing to such a guarantee. Specifically, it guarantees that if the transaction executes some updates and then a failure occurs, for whatever reason before the transaction reaches its normal termination, the those updates will be undone. Thus the transaction either executes in its entirety or is totally cancelled.

15.1: What is a Transaction?



## Transaction Processing

➤ Transaction executes in any one of the following ways:

- Entirely
- Cancels Totally.

➤ Transaction manager provides this atomicity via:

- COMMIT
- ROLLBACK

The system component that provides this atomicity is known as the transaction manager and the COMMIT WORK and ROLLBACK WORK are the keys to the way it works.

The COMMIT WORK operation signals successful end-of-transaction :it tells that transaction manager that a logical unit of work has been successfully completed.

The ROLLBACK operation by contrast, signals unsuccessful end-of-transaction indicating an inconsistent state and all the updates made by the logical unit of work must be rolled back or undone.

## 15.2: COMMIT &amp; ROLLBACK



## Explanation

- COMMIT operation signals successful end-of-transaction.
- ROLLBACK operation signals unsuccessful end-of-transaction.
- SQL commit statement takes the following form:

```
EXEC SQL
 COMMIT [WORK]
END-EXEC.
```

15.2: COMMIT & ROLLBACK



## Explanation

- SQL Rollback Statement takes the following form:

```
EXEC SQL
 ROLLBACK [WORK]
END-EXEC.
```

- SYNCHRONIZATION POINT:

- Represents boundary point between two consecutive transactions.
- Program initiation, COMMIT and ROLLBACK each establish a synchpoint.

## Points to Note

- Every SQL operation in DB2 is executed within the context of some transaction.
- Transactions cannot be nested inside one another.
- Single program execution consists of a sequence of one or more transactions.
- Transactions are not only the unit of work but also the unit of recovery.



## Review Questions

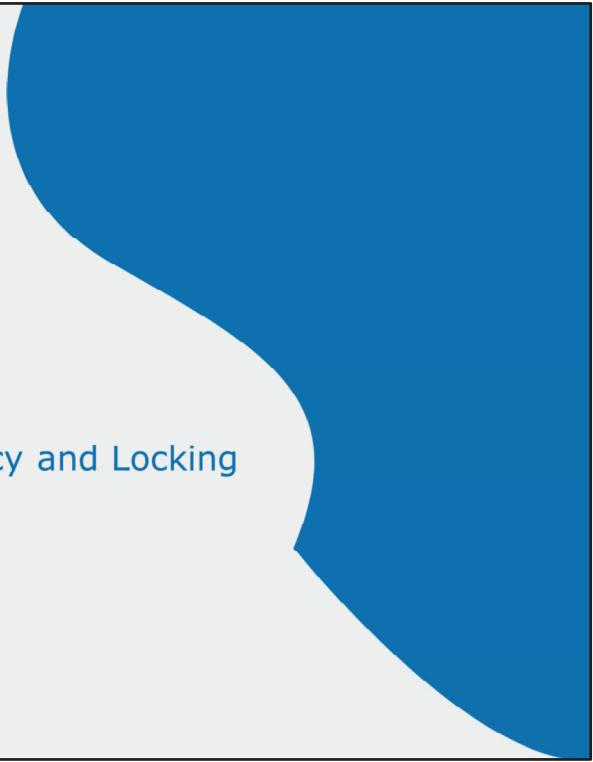
➤ Question 1: Which of the following operation signals unsuccessful end-of-transaction?

- Option 1: ROLLBACK
- Option 2: COMMIT



➤ Question 2: Single program execution consists of a sequence of one or more transactions.

- True / False



DB2

Lesson 16: Concurrency and Locking

Capgemini

## Lesson Objectives

➤ In this lesson, you will learn about:

- Concurrency problems:
  - Lost update problem.
  - Uncommitted dependency problem.
  - Inconsistent analysis problem.
- How DB2 solves these problems.
- Deadlocks





## Database Transaction

- Transaction is also known as unit of work
- It is a recoverable sequence of one or more SQL operations, grouped together as a single unit, usually within an application process.
- In single-user environments, each transaction runs serially and doesn't interfere with other transactions.
- In multi-user environments, transactions can (and often do) run simultaneously. As a result, each transaction has the potential to interfere with other active transactions.

## Database Transaction

### ➤ Problems with multiple user environments

- **Lost Update:** This occurs when two transactions read and then attempt to update the same data, and one of the updates is lost.
- **Uncommitted Read:** This occurs when a transaction reads data that has not yet been committed.
- **Non-repeatable Read:** This occurs when a transaction reads the same row of data twice, but gets different data values each time.
- **Phantom Read:** This occurs when a row of data that matches search criteria is not seen initially, but then seen in a later read operation.

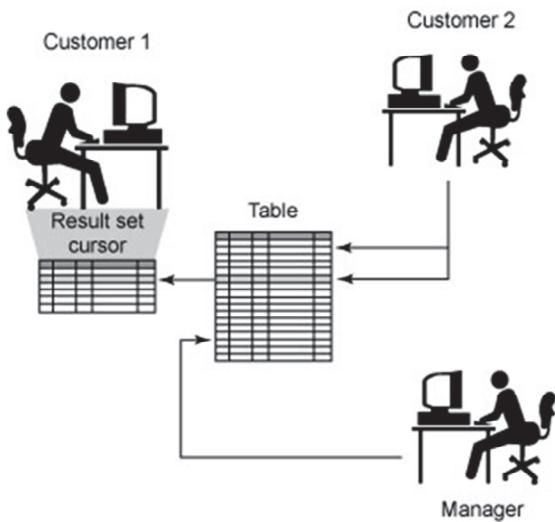




## Isolation levels to maintain concurrency

- Maintaining database consistency and data integrity, while allowing more than one application to access the same data at the same time, is known as concurrency.
- DB2 attempts to enforce concurrency is through the use of given four isolation levels, which determine how data used in one transaction is locked or isolated from other transactions while the first transaction works with it.
  - Repeatable read
  - Read stability
  - Cursor stability
  - Uncommitted read

## Uncommitted Read (UR)

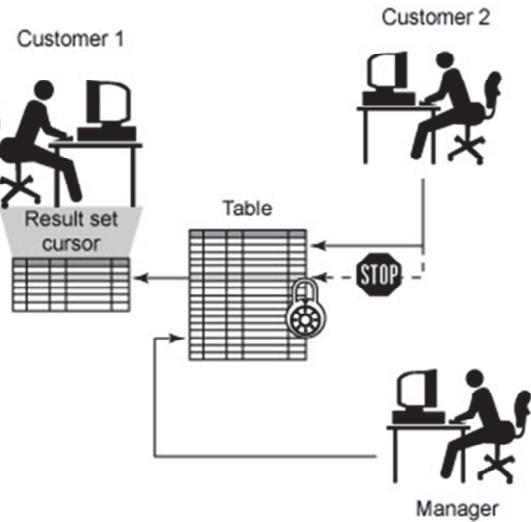




## Uncommitted Read (UR)

- Least restrictive isolation level available.
- Allows an application to access uncommitted changes of other applications.
- When this isolation level is used, rows retrieved by a transaction are only locked if the transaction modifies data associated with one or more rows retrieved or if another transaction drop or alter the table the rows were retrieved from.
- When this isolation level is used, dirty reads, non-repeatable reads, and phantoms can occur.
- Use it if you're executing queries on read-only tables/views/databases or if it doesn't matter whether a query returns uncommitted data values.

## Cursor Stability(CS)

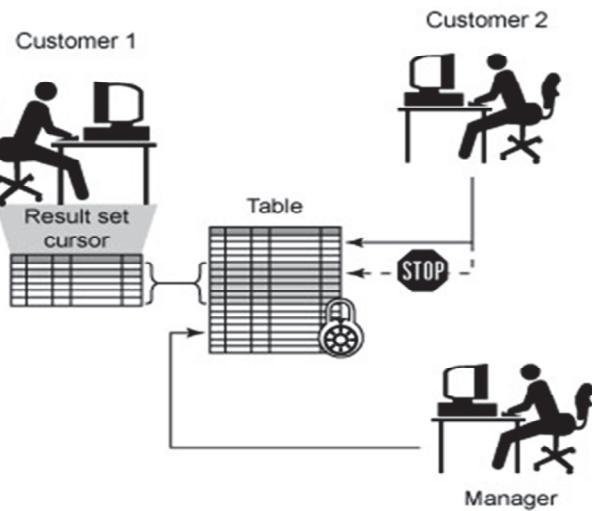




## Cursor Stability(CS)

- Default isolation level.
- This isolation level only locks the row that is currently referenced by a cursor that was declared and opened by the owning transaction. The lock remains in effect until next row is fetched or transaction is terminated.
- When this isolation level is used, lost updates and dirty reads cannot occur; non-repeatable reads and phantoms can and may be seen.
- Use the Cursor Stability isolation level when you want maximum concurrency between applications, yet you don't want queries to see uncommitted data.

## Read Stability(RS)

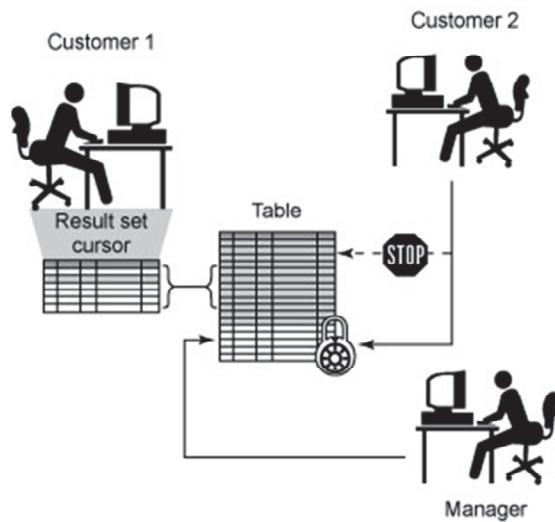




## Read Stability(RS)

- When this isolation level is used, only rows that are actually retrieved or modified by the owning transaction are locked.
- When this isolation level is used, lost updates, dirty reads, and non-repeatable reads cannot occur; phantoms, however, can and may be seen.
- Use the Read Stability isolation level when you want some level of concurrency between applications, yet you also want qualified rows to remain stable for the duration of an individual transaction.

## Repeatable Read(RR)





## Repeatable Read(RR)

- Most restrictive isolation level available
- If an entire table or view is scanned in response to a query, the entire table or all table rows referenced by the view are locked. This greatly reduces concurrency, especially when large tables are used.
- Lost updates, dirty reads, non-repeatable reads, and phantoms cannot occur.
- Use the Repeatable Read isolation level if you're executing large queries and you don't want concurrent transactions to have the ability to make changes that could cause the query to return different results if run more than once.



## Using isolation levels with queries

### ➤ Specifying isolation levels with queries

- Syntax:
  - SELECT statement to set a specific query's isolation level to Repeatable Read (RR), Read Stability (RS), Cursor Stability (CS), or Uncommitted Read (UR).
- Example:
  - SELECT \* FROM employee WHERE empid = 109124 WITH RR

| Isolation Level  | Phenomena    |             |                     |          |
|------------------|--------------|-------------|---------------------|----------|
|                  | Lost Updates | Dirty Reads | Nonrepeatable Reads | Phantoms |
| Repeatable Read  | No           | No          | No                  | No       |
| Read Stability   | No           | No          | No                  | Yes      |
| Cursor Stability | No           | No          | Yes                 | Yes      |
| Uncommitted Read | No           | Yes         | Yes                 | Yes      |



## Locking

- DB2's concurrency control mechanism is based on a technique called locking.
- When a transaction needs assurance that the object it is interested does not change in some unpredictable manner during execution, it acquires a lock on that object to prevent it from changing.
- A lock is a mechanism that is used to associate a data resource with a single transaction, for the sole purpose of controlling how other transactions interact with that resource while it is associated with the transaction that has it locked.

DB2's concurrency control mechanism is based on a technique called locking. The basic idea of locking is simple: when a transaction needs assurance that some of the objects it is interested in typically a database row - will not change in some unpredictable manner while its back is turned , it acquires a lock on that object and thereby prevent them from changing it. The first transaction is thus able to carry out its processing in the certain knowledge that the object in question will remain in stable state for as long as that transaction wishes it to

The two types of locks that can be placed are shared lock(S) and exclusive lock (X). We assume that if a transaction requests a lock that is currently not available , the transaction simply waits until it is. In practice, the installation can specify a maximum wait time ; then if any transaction ever reaches this threshold in waiting for a lock, it "times out" and the lock request fails ( a negative SQLCODE is returned)

## Locking

### APPLICATION 1

```
START APPLICATION
START TRANSACTION
SQL OPERATION
SQL OPERATION
SQL OPERATION
COMMIT
END TRANSACTION
```

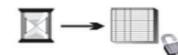
TABLE A



Transaction for Application 1 has locked Table A; Table A will not be unlocked until the transaction is terminated

### APPLICATION 2

```
START APPLICATION
START TRANSACTION
```



Transaction for Application 2 has started but cannot continue until the lock on Table A is released

### APPLICATION 1

```
END APPLICATION
```

Application 1 has completed processing and can end gracefully

### APPLICATION 2

```
SQL OPERATION
SQL OPERATION
SQL OPERATION
COMMIT
END TRANSACTION
END APPLICATION
```

Transaction for Application 2 has locked Table A and can continue processing

16.2: How Does DB2 Solve the Three Concurrency Problems?



## Explanation: Locking

- The effect of the lock is to lock other transactions out of the object and thereby prevent it from changing.
- Important locks are
  - Shared locks (S locks)
  - Exclusive locks (X locks)
- Shared locks (S locks)
- Exclusive locks (X locks)

16.2: How Does DB2 Solve the Three Concurrency Problems?



## Explanation: Locking

- An application pgm may access a row in a db2 table in order to perform an update.
- Application pgm takes a period of time to process the row information, move new values to columns to be updated, issue the statement to update the database and commit the data.
- While data is changed, but not yet committed, DB2 places a lock on the data in the table.

16.2: How Does DB2 Solve the Three Concurrency Problems?

## Explanation: Compatibility Matrix

|   |   |   |   |
|---|---|---|---|
|   | A | X | S |
| B | X | N | N |
|   | S | N | Y |

From the compatibility matrix , two inferences can be drawn:

If transaction A holds an X lock on row R , then a request from transaction B for a lock of either type on R causes B to go into a wait state. B will wait until A's lock is released.

If transaction A hold a shared lock (S) lock on row R , then A request from transaction B for an X lock on R causes B to go into a wait state (and B will wait until A's lock is released).

A request from transaction B for an S lock on R is granted (that is , B also holds an S lock on R).

Transaction requests for row locks are always implicit. When a transaction successfully fetches a row , it automatically acquires an S lock on that row. When a transaction successfully updates a row, it automatically acquires and X lock on that row. If it already holds an S lock on that row, as it will in FETCH/UPDATE or FETCH/DELETE SEQUENCE, then the update or delete “promotes the S lock to X level.

16.2: How Does DB2 Solve the Three Concurrency Problems?

## Handling Lost Update Problem

| TRAN A                            | TIME | TRAN B                                |
|-----------------------------------|------|---------------------------------------|
| -                                 | -    | -                                     |
| FETCH R<br>(acquires S lock on R) | T1   | -                                     |
| -                                 | -    | -                                     |
| -                                 | T2   | -                                     |
| -                                 | -    | -                                     |
| UPDATE R<br>(request X lock on R) | T3   | -                                     |
| -                                 | -    | -                                     |
| Wait                              | -    | -                                     |
| Wait                              | T4   | UPDATE R<br>(request for X lock on R) |
| Wait                              | -    | Wait                                  |

The above figure shows what would happen to the interleaved execution under the locking mechanism of DB2. As one can see, transaction A's update at time t3 is not accepted because it is an implicit request for an X lock on R, and such a request conflicts with the S lock already held by transaction B; so A goes into a wait state. For analogous reasons, B goes into a wait state at time t4. Now both transactions are unable to proceed, so there is no question of any update being lost. DB2 thus solves the lost update problem by reducing it to another problem but at least it does solve the original problem. This problem is called the deadlock problem, discussed later.

16.2: How Does DB2 Solve the Three Concurrency Problems?

## Handling Uncommitted Dependency Problem

| TRAN A                               | TIME | TRAN B                                |
|--------------------------------------|------|---------------------------------------|
| -                                    | T1   | -                                     |
| -                                    |      | UPDATE R<br>(X lock on R)             |
| -                                    |      | -                                     |
| FETCH R<br>(request for S lock on R) | T2   | -                                     |
| Wait                                 |      | -                                     |
| Wait                                 | T3   | SYNCPOINT<br>(Release X lock<br>on R) |
| Wait                                 |      | -                                     |
| resume : FETCH R<br>(X lock on R)    | T4   |                                       |

Transaction A's operation at time t2 is not accepted , because it is an implicit request for a lock on R, and such a request conflicts with the X lock already held by B; so A goes into a wait state and remains so until B reaches a synchpoint (either commit or rollback), when B's lock is released and A is able to proceed; and at that point A sees a committed value (either the pre – B value, if B terminates and with a ROLLBACK , or the post – b value otherwise. Either way, A is no longer dependent on an uncommitted update.

16.3: Deadlock

## Explanation



| TRAN. A           | TIME | TRAN. B |
|-------------------|------|---------|
| -                 | -    | -       |
| -                 | T1   | -       |
| Lock R1 in X mode |      | -       |
| -                 | T2   | R2 IN X |
| -                 |      | -       |
| R2 IN X           | T3   | -       |
| Wait              |      | R1 IN X |
| Wait              |      | Wait    |
|                   | T4   | Wait    |



16.2: Deadlock

## Explanation

- If a deadlock occurs, the system detects and break it.
- To break a deadlock, choose one of the deadlocked transactions as the victim by either rolling it back automatically or requesting it to roll itself back.
- Either way, the transaction releases its lock and allows some other transaction to proceed.

A 'deadlock' occurs when program A locks page X exclusively and attempts to lock page Y, while program B has already locked page Y exclusively and attempts to lock page X. Neither program can continue without the required lock. DB2 cancels one of the processes (the one with the fewest log records) with a time out code. To avoid this situation, you can:

Use the same sequence of update. Both programs should advance along the tables in ascending key order; this makes it less likely that they will cross each other's paths and attempt to lock the same page. Both programs should access various tables in the same sequence, for the same reason.

Avoid clashing updates through different paths (if possible). The problem with such updates can be exemplified as follows:

User A is updating page 1 using index X for access to the data.

User B is updating page 2 using index Y for access to the data.



However, each index also references the data on the other page (i.e., Index X references data on page 2 and index Y references data on page 1).

Therefore, after the data is reached by means of one index, and after that data is updated, each query must modify the other index so it would reflect the new, updated data. However, that other index is still locked by the other query.

Such deadlocks are hard to avoid. You may watch out for some conditions which make this situation more likely:

Large number of indexes on a table, with frequent deletes or updates.  
Multi-row updates and deletes.

Large value used for SUBPAGES for the index.

Use frequent COMMITS. The chance that a page needed by program A will already be held by program B is thereby reduced.

Use cursor with FOR UPDATE OF ... This technique locks with INTENT UPDATE, ahead of time, all the pages which you will need. This is a more relaxed lock than an exclusive lock, but it still prevents other programs from locking one of your pages exclusively. Thus, less contention, fewer deadlocks and fewer time outs will occur.

16.3: Deadlock



## Explanation

- In general, any executable SQL operation may be rejected with a negative SQLCODE.
  - This indicates that the transaction has just been selected as the victim in a deadlock situation.
  - It has either been rolled back or is requested to do so.
- Application programs may need to include explicit code to deal with the problem of deadlock if it arises.



## Lock Attributes

- Resource being locked is called object.
- Objects which can be explicitly locked are databases, tables and table spaces.
- Objects which can be implicitly locked are rows, index keys, tables. Implicit locks are acquired by DB2 according to isolation level and processing situations.
- Object being locked represents granularity of lock.
- Length of time a lock is held is called duration and is affected by isolation level.

## How locks are acquired

- DB2 provides certain explicit facilities such as:
  - LOCK TABLE (SQL statement),
  - Isolation parameter ON BIND command
  - Tablespace LOCKSIZE parameter
  - Acquire or release parameters ON BIND





16.3: Lock Table

## Explanation

- Depending on the versions, this command either locks a table or an entire table space.
- Syntax

```
LOCK TABLE table IN mode MODE
```

- Here "mode" is SHARE or EXCLUSIVE and "table" must designate a base table and not a view.
- Once a lock is acquired no other transaction is able to access any part of the table in any way - until the original lock is released

16.3: Lock Table



## Explanation

- Time, the original lock is released depends on the RELEASE parameter on BIND.
- Lock table statement can be used to control locks from within a DB2 application program.
- Every individual page lock uses system resources during storage and processing.
- A single table lock reduces storage and processing time required by many small locks.
  - Thus system resources are saved.

## 16.3: The LOCKSIZE Parameter



## Explanation

- Physically, DB2 locks data in terms of pages or tables or table spaces, depending on what was specified as the LOCKSIZE for the relevant tablespace in the CREATE or ALTER TABLESPACE operation.
- For a given table space, the LOCKSIZE can be specified as PAGE, TABLE, TABLESPACE or ANY



## 16.3: The LOCKSIZE Parameter

## Explanation

### ➤ TABLESPACE

- All locks acquired on data in the tablespace are at the tablespace level.

### ➤ TABLE

- Locks acquired on data in the tablespace are at the table level.

### ➤ PAGE

- Locks acquired on data in the tablespace are at the page level whenever possible.

### ➤ ANY

- (Which is the default) DB2 itself decides the appropriate physical unit of locking for the tablespace for each plan.
- Defaults to a page lock
  - If the number of locked pages exceeds an installation default, DB2 does a lock escalation and automatically locks a larger unit.

16.3: Acquire/Release Parameters on Bind



## Explanation

- While Binding, DB2 allows us to specify transaction-level lock acquiring and releasing parameters.
- **ACQUIRE**
  - ACQUIRE (use):
    - Tells DB2 to take locks at the time of SQL statements execution. This is the DB2 default.
  - ACQUIRE(allocate):
    - Tells DB2 to take all necessary locks at the start of the transaction.

16.3: Acquire/Release Parameters on Bind



## Explanation

➤ **RELEASE:** Similar to the DB2 acquire parameters, there are 2 release parameters:

- Release (commit):
  - Tells DB2 to release all locks of transaction commit time. This is the DB2 default.
- Release(deallocate):
  - Tells DB2 to release all locks only when the program ends (i.e. the thread is deallocated).

16.3: Acquire/Release Parameters on Bind



## Explanation

### ➤ ACQUIRE and RELEASE:

- All combinations except ACQUIRE (Allocate) and RELEASE (commit) are allowed.
- For more concurrency, use ACQUIRE (use) and RELEASE (commit). This is the DB2 default.
- For better performance, use ACQUIRE (ALLOCATE) and RELEASE (DEALLOCATE).

## Review Questions

➤ Question 1: What should be isolation level parameter so that other transactions could change the data which your transaction has already read?

- Option 1: RR
- Option 2: CS
- Option 3: SS



➤ Question 2: Which of the combinations should be used for better performance?

- Option 1: ACQUIRE (ALLOCATE) and RELEASE (DEALLOCATE)
- Option 2: ACQUIRE (Allocate) and RELEASE (commit)
- Option 3: ACQUIRE (use) and RELEASE (commit)

## DB2

### Lesson 17: Stored Procedure

Capgemini



## Lesson Objectives

➤ In this lesson, you will learn about:

- Introduction to stored procedure
- Implementing DB2 Stored Procedures
- Creating Stored Procedures
- Executing a Stored Procedure



17.1: Stored Procedure



## What Is a Stored Procedure?

- Stored procedures are specialized programs that are executed under the control of the relational database management system.
- A stored procedure must be directly and explicitly invoked before it can be executed.
- Stored procedures can access flat files, Virtual Storage Access Method (VSAM) files, and other files, as well as DB2 tables.
- A stored procedure provides a common piece of code that is written only once and is maintained in a single instance that can be called from several different applications.
- DB2 provides some stored procedures but you can also create your own.

17.1: Stored Procedure



## Use Of Stored Procedures?

- Reusability
- Consistency
- Maintenance
- Business Rules
- Data Integrity
- Remote Access
- Performance

**Reusability:** The predominant reason for using stored procedures is to promote code reusability. Instead of replicating code on multiple servers and in multiple programs, stored procedures allow code to reside in a single place—the database server. Stored procedures then can be called from client programs to access DB2 data.

**Consistency:** An additional benefit of stored procedures is increased consistency. If every user with the same requirements calls the same stored procedures, the DBA can be assured that everyone is running the same code. If each user uses his or her own individual, separate code, no assurance can be given that the same logic is being used by everyone. In fact, you can be almost certain that inconsistencies will occur.

**Maintenance:** Stored procedures are particularly useful for reducing the overall code maintenance effort. Because the stored procedure exists in one place, you can make changes quickly without propagating the change to multiple workstations.

**Business Rules:** By implementing your business rules in stored procedures, the rules can be accessed by multiple programs and take advantage of the reusability, consistency, and maintenance benefits discussed in the previous three bullets.

**Data Integrity:** Additionally, you can code stored procedures to support database integrity constraints. You can code column validation routines into stored procedures, which are called every time an attempt is made to modify the column data.

**Remote Access:** Stored procedures can be developed to offer a simple way for calling remote programs.

**Performance:** Another common reason to employ stored procedures is to enhance performance. In a client/server environment, stored procedures can reduce network traffic because multiple SQL statements can be invoked with a single execution of a procedure instead of sending multiple requests across the communication lines.

17.2: Stored Procedure



## Implementing DB2 Stored Procedures

- The programming languages supported by DB2 for z/OS include C, C++, COBOL, z/OS Assembler, PL/I, REXX, SQL Procedures language (both external and native), and Java.
- Parameters are essential to the effective use of stored procedures.
- Parameters allow data to be sent to and received from a stored procedure.
- DB2 stored procedures are registered and managed within DB2 like other DB2 objects, using standard DDL statements—ALTER, CREATE, and DROP.

17.3: Stored Procedure



## Creating Stored Procedures

```
>CREATE PROCEDURE SYSPROC.PROCNAME(INOUT
CHAR(20))
LANGUAGE COBOL
EXTERNAL NAME LOADNAME
PARAMETER STYLE GENERAL
NOT DETERMINISTIC
MODIFIES SQL DATA
WLM ENVIRONMENT WLMNAME
STAY RESIDENT YES
DYNAMIC RESULT SETS 1;
```

- This statement creates a stored procedure named PROCNAME in the SYSPROC schema using an external load module name of LOADNAME.
- The stored procedure is written in COBOL and runs under the control of WLM(work load manager).
- It returns one result set.

17.3: Stored Procedure



## Creating Stored Procedures

- A native SQL stored procedure includes the executable SQL text of the stored procedure.
- For example, the following statement creates a new SQL stored procedure to update an employee salary:

```
CREATE PROCEDURE UPD_SALARY
(IN EMP_NBR CHAR(10), IN PCT DECIMAL(6,2))
LANGUAGE SQL MODIFIES SQL DATA
UPDATE DSN81010.EMP
SET SALARY = SALARY * PCT
WHERE EMPNO = EMP_NBR
```

17.3: Stored Procedure



## Creating Stored Procedures

### ➤ Configuring Parameter Lists

- The parameters to be used by DB2 stored procedures must be specified in parentheses after the procedure name in the CREATE PROCEDURE statement. You can define three types of parameters:
  - IN: An input parameter
  - OUT: An output parameter
  - INOUT: A parameter that is used for both input and output

➤ The type of parameter must be predetermined and cannot be changed without dropping and re-creating the stored procedure.

17.4: Stored Procedure



## Executing a Stored Procedure

- To run a stored procedure, you must explicitly issue a CALL statement.
- For example, the following statement calls a stored procedure named SAMPLE, sending a literal string as a parameter:

```
➤ EXEC SQL CALL SAMPLE('ABC') END-EXEC.
```
- To issue a CALL statement for a stored procedure requires the EXECUTE privilege on the stored procedure
- Except for native SQL stored procedures, DB2 runs stored procedure code isolated from the core DB2 code

## Review Questions

- Question 1: \_\_\_\_\_ allow data to be sent to and received from a stored procedure.
- Question 2: \_\_\_\_\_ this statement creates a stored procedure.





DB2

Lesson 18: Tools and Utilities for DB2

Capgemini

## Lesson Objectives

➤ In this lesson, you will learn about:

- Loading of DB2 tables
- Unloading of DB2 tables
- File Aid for DB2
- File manager for DB2



18.1: Utilities for Db2



## UnLoading DB2 tables

- The Unload utility is used to unload data from a table to a sequential data set.
- To use the Unload utility, the definitions of the tablespace and tables must be available on the system.
- The data set used for the unload operation can be saved both on disk and tape.
- Unload work only with tables, and cannot be used with views.
- We can use the unloaded data sets to create a backup database using either an existing database or creating a new database with a new database schema prefix.

18.2: Utilities for DB2



## Loading DB2 tables

- The Load utility is used to load data into a table of a tablespace.
- It enables you to load records into the tables and builds or extends any indexes defined on them.
- If the table space already contains data, you can either add the new data, or replace the existing data with the new data.
- Because the Load utility operates at a table space level, to run it you must have the required authority for all the tables of the table space.
- The data set used for the Load utility can be read from both disk and tape.
- Load work only with tables, and cannot be used with views.

18.2: Utilities for DB2



## What are COPY and CHECK Pending?

- COPY PENDING and CHECK PENDING status on the table space restricts any updates on table space.
- If you LOAD or REORG the table space with the option LOG NO, then the table space get into COPY PENDING status. The meaning is, an image copy is needed on table space.
- If LOAD the table space with ENFORCE NO option, then the table space get into CHECK PENDING status. The meaning is table space is loaded without enforcing constraints.
- CHECK utility needs to be run on the table space.

COPY utility-it is used to create image copy backup dataset for a complete table space or a single partition of a table space. It can be of type FULL OR incremental.

CHECK utility-these are data consistency utilities. They are used to monitor, control and administer data consistency errors.

## 18.3: Tools for DB2



## File Manager for DB2

- File manager for DB2 provides a comprehensive, user-friendly set of tools for working with DB2 data.
- These tools include the familiar view, edit, copy and print utilities found in ISPF, enhanced to meet the needs of application developers.
- It also provides utilities for listing DB2 objects, managing DB2 privileges, generating JCL to run DB2 standalone utilities, exporting and importing DB2 tables to or from VSAM data sets, creating data to populate DB2 tables, and prototyping SQL SELECT statements.
- FM/DB2 uses templates to provide a formatted view of your data, enabling you to view, edit, and manipulate data according to the columns and data types in the table you are working with.

An FM/DB2 template is a collection of information that you can use to select and format tables and columns in a DB2 object. If you use an FM/DB2 function that interfaces with non-DB2 data, the corresponding template describes the records and fields in the data set.



18.3: Tools for DB2

## Starting and exiting FM/DB2

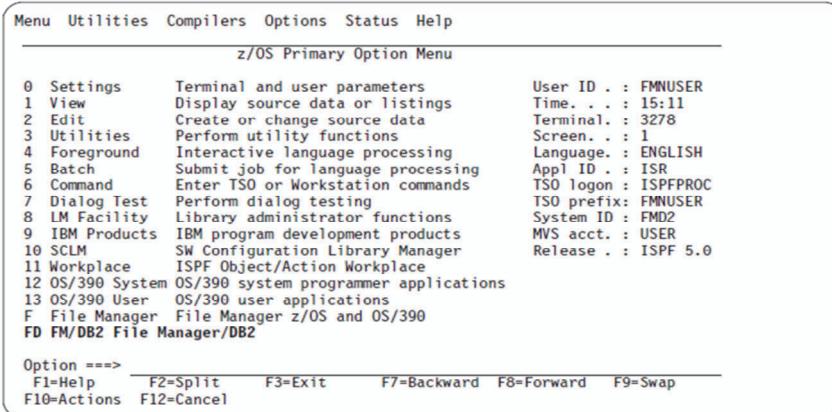
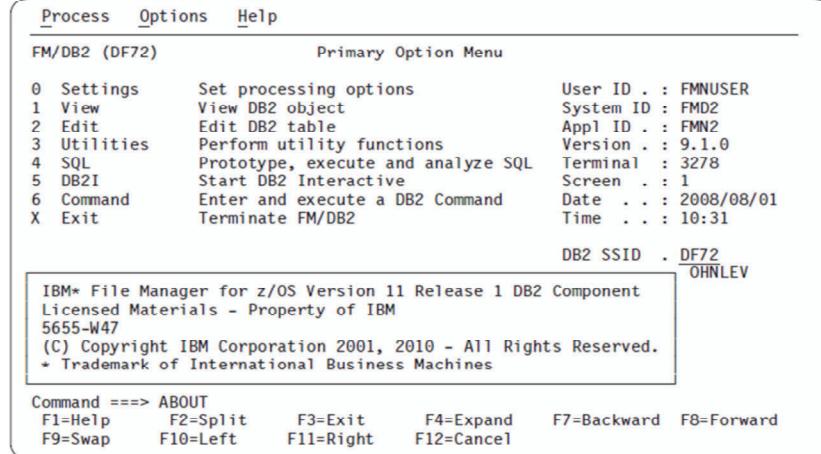


Figure 2. z/OS Primary Option Menu panel showing FM/DB2 option

18.3: Tools for DB2

## Starting and exiting FM/DB2



18.3: Tools for DB2



## Starting and exiting FM/DB2

### ➤ Exiting from FM/DB2

- You can exit from FM/DB2 from the Primary Options Menu panel in any of the following ways:
- Press the Exit function key (F3).
- Enter X (or EXIT or END) on the command line.
- Select Process> Exit FM/DB2 from the Action Bar.
- To exit the application from a panel within FM/DB2:
- Enter =X on the command line.

## 18.4: Tools for DB2



## File Aid for DB2

- File-AID for DB2 makes browsing and editing table data as simple as working with data under ISPF.
- Developers can edit an entire table or build selection criteria that can be saved and reused to:
  - select specific columns
  - determine the order in which columns are displayed
  - specify the sort order of the data in each column
  - select rows based on WHERE clauses
  - select data using pre-existing SQL statements.
- File-AID for DB2 always maintains the integrity of your database.



## 18.4: Tools for DB2

## File Aid for DB2

➤ The File-AID for DB2 EDIT function lets developers view data in either

➤ row or table mode.

The screenshot shows a terminal window titled "File-AID for DB2 Edit" with the command "DSN8910.EMP". The window displays a table of employee data with columns: COLUMN NAME, TYPE (LEN), KEY, and COLUMN VALUE. The data includes fields like EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT, PHONE, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, and COMM. The terminal also shows system parameters: SSID: D901, SCROLL ==> PAGE, and MAXCOL 80. At the bottom, it says "\*\*\*\*\* BOTTOM OF DATA \*\*\*\*\*".

| COLUMN NAME   | TYPE (LEN) | KEY  | COLUMN VALUE |
|---------------|------------|------|--------------|
| 001 EMPNO     | CHAR(6)    | PRIM | 000010       |
| 002 FIRSTNAME | VC(12)     |      | CHRISTINE    |
| 003 MIDINIT   | CHAR(1)    |      | I            |
| 004 LASTNAME  | VC(15)     |      | HAAS         |
| 005 WORKDEPT  | CHAR(3)    |      | R&D          |
| 006 PHONE     | CHAR(4)    |      | 3978         |
| 007 HIREDATE  | DATE       |      | 1965-01-01   |
| 008 JOB       | CHAR(8)    |      | PRES         |
| 009 EDLEVEL   | SMALLINT   |      | 18           |
| 010 SEX       | CHAR(1)    |      | F            |
| 011 BIRTHDATE | DATE       |      | 1933-08-14   |
| 012 SALARY    | DEC(9,2)   |      | 52750.00     |
| 013 BONUS     | DEC(9,2)   |      | 1000.00      |
| 014 COMM      | DEC(9,2)   |      | 4220.00      |

## 18.4: Tools for DB2



## File Aid for DB2

- With File-AID for DB2, developers can create, populate, customize, refresh and, where appropriate, authorize DB2 objects.
- File-AID for DB2 creates an effective DB2 environment — without coding SQL
- File-AID for DB2 gets into the test cycle faster, letting developers validate and explain program SQL before beginning the compile and bind process.
- File-AID for DB2 supports new features incorporated into recent DB2 releases. These include: Distinct Type, Stored Procedures, Triggers, Identity Column, ROWID and User Defined Functions.



## 18.4: Tools for DB2

## File Aid for DB2

- The CREATE/DROP/ALTER DB2 OBJECTS function creates a table using ISPF-like screens instead of coding DDL statements.
- By modeling the table after one created by a DBA, developers can quickly set up their own test tables.
- With File-AID *for DB2*, tables are populated by:
  - copying data from an existing table
  - cutting and pasting data



## Review Questions

➤ Question 1: \_\_\_\_\_ utility is used to load data into a table of a tablespace.

- Option 1: RECOVER
- Option 2: LOAD
- Option 3: CHECK



➤ Question 2: \_\_\_\_\_ uses templates to provide a formatted view of your data, enabling you to view, edit, and manipulate data according to the columns and data types in the table you are working with.

# DB2

## Lab Book

## Document Revision History

| Date                      | Revision No. | Author        | Summary of Changes               |
|---------------------------|--------------|---------------|----------------------------------|
| 10-Nov-2009               | 4.0          | Arjun Singh   | Content creation                 |
| 16-Nov-2009               |              | CLS Team      | Review                           |
| 30-Jun-2011               | 5.0          | Rajita Dhumal | Revamped                         |
| 8 <sup>th</sup> -Aug-2012 | 5.1          | Rajita Dhumal | Revamped after Assignment Review |

## Table of Contents

|                                                                                   |    |
|-----------------------------------------------------------------------------------|----|
| Document Revision History .....                                                   | 2  |
| Table of Contents .....                                                           | 3  |
| Getting Started .....                                                             | 4  |
| Overview.....                                                                     | 4  |
| Pre-requisite .....                                                               | 4  |
| Requirements .....                                                                | 4  |
| Lab 1. Execution of SQL queries using SPUFI.....                                  | 5  |
| Lab 2. Data Query Language .....                                                  | 8  |
| Lab 3. Single Row Functions .....                                                 | 10 |
| Lab 4. JOINS AND SUBQUERIES.....                                                  | 12 |
| Lab 5. Set Operators .....                                                        | 15 |
| Lab 6. DB2 Objects.....                                                           | 17 |
| Lab 7. Data Manipulation Language.....                                            | 21 |
| Lab 8. Using and execution of embedded SQL (on IBM Mainframes).....               | 24 |
| 8.1: Create table definitions using DCLGENs.....                                  | 24 |
| Lab 9. Using and execution of embedded SQL (on IBM Mainframes).....               | 33 |
| Write a COBOL programs: .....                                                     | 33 |
| Lab 10.Using and execution of embedded SQL (on IBM Mainframes) .....              | 36 |
| Lab 11. Problem statements based on Analysis, Debugging and Enhancement tasks.... | 39 |
| Appendices .....                                                                  | 41 |
| Appendix A: JCL's .....                                                           | 41 |
| Appendix B: DB2 datatypes and the COBOL equivalent Picture .....                  | 45 |
| clauses.....                                                                      | 45 |
| Appendix C: SQLCA.....                                                            | 47 |
| Appendix D :List of SQL codes .....                                               | 48 |
| Lab 12. Using and execution of embedded SQL (CA-Realia Workbench ) .....          | 49 |

## Getting Started

### Overview

This lab book is a guided tour for learning DB2. It comprises assignments.

### Pre-requisite

It is expected that participants know the following:

- Good knowledge of COBOL, JCL and SQL
- Using the ISPF editor
- Various options available on ISPF such as dataset allocation, copy, member list of PDS, how to compress PDS, etc.
- How to compile and view compilation errors, correct those errors
- How to submit jobs and view job log (=S.ST), SYSOUT

### Requirements

You need three PDS, one each for storing DB2 source code, JCL code and loadlib. The convention followed for naming the PDS is as follows:

IBMID.NAME.X where IBMID is IBM login id used by the participant, NAME is name of the participant and X is either DB2SORCE or DB2JCL or DBMRLIB or LOADLIB.

E.g. the participant using IBM login id DA0002T and name as ANNETTE will have three PDS as follows:

DA0000T.ANNETTE.DB2SORCE – for storing DB2 source programs as PDS members  
DA0000T.ANNETTE.DB2JCL – for storing the DB2 JCL's as PDS members  
DA0000T.ANNETTE.DBMLIB – for storing the DBRM modules  
DA0000T.ANNETTE.LOADLIB – for storing load modules of programs as PDS members

If these PDSs do not exist, then these are to be allocated as described below.

After logging on to IBM mainframe, on the ISPF main menu, type 3.2 against Option as shown in the figure below.

## Lab 1. Execution of SQL queries using SPUFI

|              |                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>Understand how to execute queries using SPUFI.</li> </ul> |
|              | 15 Min.                                                                                          |

### Objective:

Query the employee and dept tables created by DA0021T.

### Input :

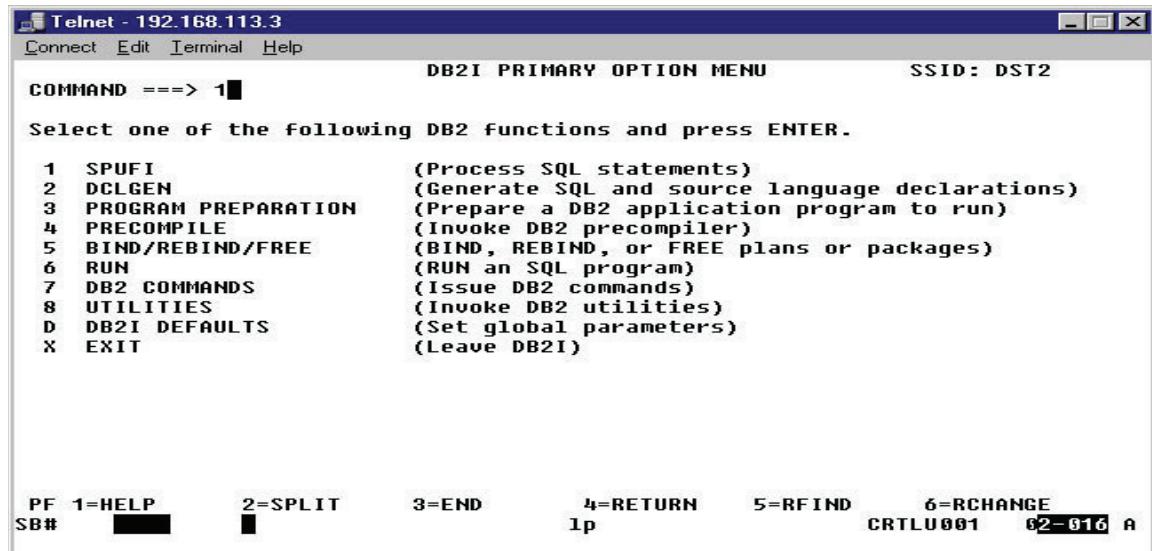
SELECT statement(s).

### Program Specifications:

Display all the records from DA0021T.EMPLOYEE (Employee table) and DA0021T.DEPT (Department table).

### Steps

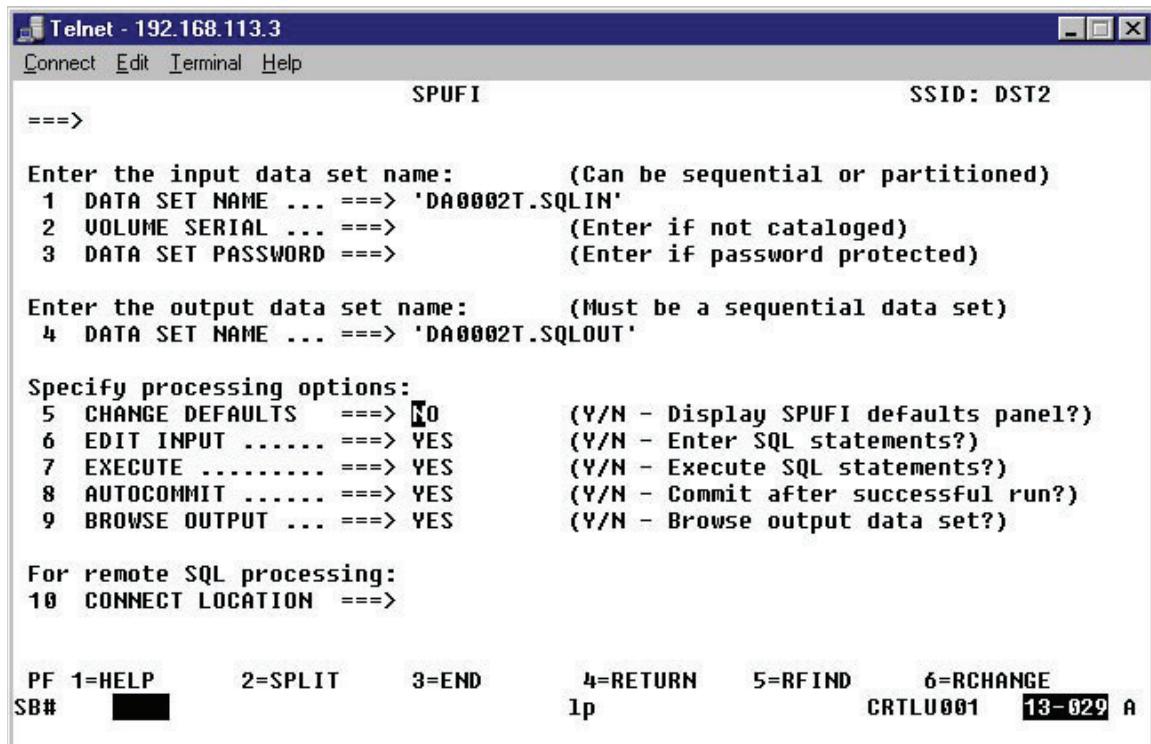
- Allocate a sequential dataset named SQLIN. This dataset will be required to specify the queries.
- Go to DB2I from ISPF options.
- Select SPUFI from the DB2I PRIMARY OPTION MENU.



**Figure 11**

In the SPUFI screen you will specify the name of the sequential dataset we have allocated. The name of the output data set will have to be specified also. But this dataset need not be allocated.

Note : Edit input is YES. The moment you hit the Enter key the input dataset will be opened in the ISPF editor.



```

Telnet - 192.168.113.3
Connect Edit Terminal Help
SPUFI SSID: DST2
===>

Enter the input data set name: (Can be sequential or partitioned)
1 DATA SET NAME ... ==> 'DA0002T.SQLIN'
2 VOLUME SERIAL ... ==> (Enter if not catalogued)
3 DATA SET PASSWORD ==> (Enter if password protected)

Enter the output data set name: (Must be a sequential data set)
4 DATA SET NAME ... ==> 'DA0002T.SQLOUT'

Specify processing options:
5 CHANGE DEFAULTS ==> NO (Y/N - Display SPUFI defaults panel?)
6 EDIT INPUT ==> YES (Y/N - Enter SQL statements?)
7 EXECUTE ==> YES (Y/N - Execute SQL statements?)
8 AUTOCOMMIT ==> YES (Y/N - Commit after successful run?)
9 BROWSE OUTPUT ... ==> YES (Y/N - Browse output data set?)

For remote SQL processing:
10 CONNECT LOCATION ==>

PF 1=HELP 2=SPLIT 3=END 4=RETURN 5=RFINDD 6=RCHANGE
SB# [REDACTED] 1p CRTL U001 13-029 A

```

Key in the following query statements :

```

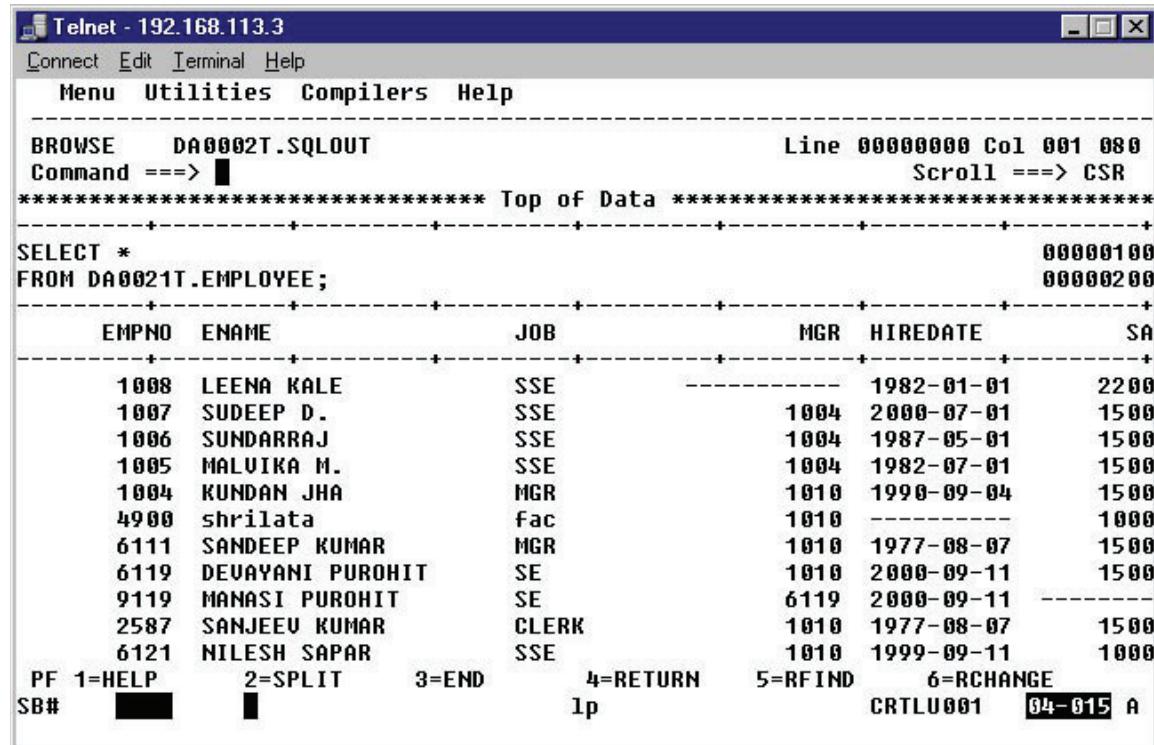
SELECT *
FROM DA0021T.EMPLOYEE;
SELECT *
FROM DA0021T.DEPT;

```

After keying in the queries, exit from the editor by entering END against Command or hitting the PF3 key. This will take you back to the SPUFI screen.

Note : An \* appears against Edit input . Hit Enter. This will cause the queries in the input data set to be executed and the output will be directed to the output dataset.

Once the queries have been executed the output data set is automatically opened in the editor. You can scroll through the contents of this output dataset.



```

Telnet - 192.168.113.3
Connect Edit Terminal Help
Menu Utilities Compilers Help
BROWSE DA0002T.SQLOUT Line 00000000 Col 001 080
Command ==>

***** Top of Data *****

SELECT * 00000100
FROM DA0021T.EMPLOYEE; 00000200
+-----+-----+-----+-----+-----+-----+
| EMPNO | ENAME | JOB | MGR | HIREDATE | SA |
+-----+-----+-----+-----+-----+-----+
1008	LEENA KALE	SSE		1982-01-01	2200
1007	SUDEEP D.	SSE	1004	2000-07-01	1500
1006	SUNDARRAJ	SSE	1004	1987-05-01	1500
1005	MALVINKA M.	SSE	1004	1982-07-01	1500
1004	KUNDAN JHA	MGR	1010	1990-09-04	1500
4900	shrilata	Fac	1010	-----	1000
6111	SANDEEP KUMAR	MGR	1010	1977-08-07	1500
6119	DEUAYANI PUROHIT	SE	1010	2000-09-11	1500
9119	MANASI PUROHIT	SE	6119	2000-09-11	-----
2587	SANJEEV KUMAR	CLERK	1010	1977-08-07	1500
6121	NILESH SAPAR	SSE	1010	1999-09-11	1000
+-----+-----+-----+-----+-----+-----+
PF 1=HELP 2=SPLIT 3=END 4=RETURN 5=RFIND 6=RCHANGE
SB# CRTLU001 04-015 A

```

Key in END against Command or use the PF3 key to exit from this screen.  
Incase no more queries are to be keyed in exit from each screen till you come to the ISPF main menu.

## Lab 2. Data Query Language

|              |                                                             |
|--------------|-------------------------------------------------------------|
| <b>Goals</b> | • Understand how to use various clauses of SELECT statement |
| <b>Time</b>  | 45 min                                                      |

1. List Name, Sal, Comm, and deptno of the employees who are working in department 20, 30 and 40. **(To Do)**

2. List the details of the employees with elaborated Column headers.

```
Select Empno as "Employee Number",
 Ename as "Employee Name",
 Job as "Designation",
 Mgr as "Manager Number",
 Hiredate as "Date of Joining",
 Sal as "Salary",
 Comm as "Commission",
 DeptNo as "Department Number"
from emp;
```

Example 1: Sample Code

3. List the Department name, Location Name, number of employees, and the average salary for all employees in that department, and label the columns Count(\*) as Number of People.

```
Select D.dname,D.Loc,
 count(*) "Number of People",
 avg (sal) "Salary"
from Emp E, Dept D wheree.deptno=d.deptno
group by dname,loc
```

Example 2: Sample Code

4. List the details of the employees whose designations are either CLERK or SALESMAN. **(To Do)**

5. List the empno, name, and Department No of the employees who have experience of more than 18 years. **(To Do)**

6. List the name and Designations of the employees who have joined before Jan 1982.

7. List the name, designation, and income for 10 years annual income of the employees who are working in departments 10 and 30. **(To Do)**
8. List the name and experience (in years) of employees who are working as CLERKS. **(To Do)**
9. List the details of the Department where the Location of the department is in 'BOSTON'. **(To Do)**
10. List the Name and Salary of the employees who are earning between 1000 and 3000. Sort them based on their salaries and name. **(To Do)**
11. Display employees who do not have manager. **(To Do)**
12. Display details of all employees whose job is Clerk or Analyst, and whose salary is not equal to \$2500, \$3500, or \$7000. **(To Do)**
13. Get the Department number, and sum of Salary of all non managers and president in that department, where the sum is greater than 2000. Order the output with the highest total at the top.
14. Get the employee name, salary, and annual salary for only those employees whose annual salary is greater than 20000. Order the result in increasing order of annual salary.

## Lab 3. Single Row Functions

|              |                                                                                           |
|--------------|-------------------------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>• Learning various Single Row Functions</li> </ul> |
| <b>Time</b>  | 45 min                                                                                    |

1. Find the total experience of all employees present in EMP table. **(To Do)**
2. List the details of the employees, whose names start with 'A' and end with 'S'. **(To Do)**
3. List the name and job of the employees whose names should contain N as the second or third character, and ending with either 'N' or 'S'. **(To Do)**
4. List the name of the Employees, who have joined in the year 1982. **(To Do)**
5. List the names of the Employees having % character in their name. **(To Do)**
6. List the details of the employees who have joined in December (irrespective of the year). **(To Do)**
7. List the name, experience in Years, and commission of employees whose job is PRESIDENT. If Commission exists, then display it, otherwise display '0' in Commission column. **(To Do)**
8. Show employee's names with the respective numbers of asterisk from Emp table. **(To Do)**
9. Show employee's names with the respective numbers of asterisk from Emp table except first and last characters. For example: KING will be replaced with K\*\*G. **(To Do)**
10. Show all employees who were hired in the first half of the month. **(To Do)**

- 11.** Display the ename, hiredate and day of the week on which the employee joined. Order the results by the day of the week starting with Monday. (To Do)

## Lab 4. JOINS AND SUBQUERIES

|              |                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>Solving the complex queries using Joins and Subqueries</li> </ul> |
| <b>Time</b>  | 90 Min                                                                                                   |

1. List name, job, and deptno of the employees who are having the same job as that of any employee of department no 20. Sort them in an ascending order by deptno. **(To Do)**
  
2. List lowest paid employees working for each manager. Sort the output by salary. **(To Do)**
  
3. List the three least paid employees in EMP table. Sort the result by salary in an ascending order.
  
4. Using a sub query, list the employee name and job of employees who are not from CHICAGO location. **(To Do)**
  
5. List the employee name and department name of all employees working in a department, which has at least three employees working in it. **(To Do)**
6. List all grades and the number of employees in each grade. **(To Do)**
  
  
9. Observe the following tables, and pick up the employee, who is a 3GLS capable and stays in CHICAGO. Write the SQL query to retrieve the Employee details. **(To Do)**

### EMP

|           |             |
|-----------|-------------|
| EMPNO     | INTEGER     |
| HOTELNAME | VARCHAR(10) |

### COURSEDONE

|          |         |
|----------|---------|
| EMPNO    | INTEGER |
| COURSENO | INTEGER |

### COURSEMASTER

COURSENO, INTEGER  
COURSENAME VARCHAR(10)

**HOTEL**

HOTELNAME VARCHAR(10)  
ADDRESS VARCHAR(10)

**DETAILS IN EMP TABLE**

| EMPNO | HOTELNAME |
|-------|-----------|
| 101   | RAMADA    |
| 102   | WELCOME   |
| 103   | TAJ       |
| 104   | OBEROX    |
| 105   | WELCOME   |

**DETAILS IN COURSEDONE TABLE**

| EMPNO | COURSENO |
|-------|----------|
| 101   | 2        |
| 102   | 1        |
| 103   | 3        |
| 101   | 4        |
| 102   | 5        |
| 104   | 1        |

**DETAILS IN COURSEMASTER TABLE**

| COURSENO | COURSENAME |
|----------|------------|
| 1        | 3GLS       |
| 2        | COBOL      |
| 3        | 4GLS       |
| 4        | PASCAL     |
| 5        | ORACLE     |

**DETAILS IN HOTEL TABLE**

HOTELNAME ADDRESS

---

|         |           |
|---------|-----------|
| RAMADA  | NEWYORK   |
| WELCOME | ILLINOIS  |
| TAJ     | NEWJZRSEY |
| OBEROX  | CHICAGO   |

- 10.** Write the query to display the details of employees, who belong to such a department whose (departments) sum of salary is maximum. **(To Do)**
- 11.** Show all employees who were hired on the day of the week on which highest number of employees were hired. **(To Do)**
- 12.** Write a query to display the details of such a department, where no sales representatives work. **(To Do)**
- 13.** List all employees with their names, salary, and incremented salary at 10% of the average salary. **(To Do)**
- 14.** Get the names of all employees, who are earning the maximum salary in their respective departments. **(To Do)**
- 15.** Write a query to get the details of oldest and the newest employees in the organization, in terms of experience in the company. **(To Do)**

## Lab 5. Set Operators

|              |                                                                            |
|--------------|----------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>• Learning Set Operators</li> </ul> |
| <b>Time</b>  | 15 min                                                                     |

Observe the following queries and do the given assignments.

```
create table previous_products (
 pid int,
 name varchar(40),
 unit varchar(40),
 price int,
 stock int);
```

Example 3: Sample Code

```
create table current_products (
 pid int,
 name varchar(40),
 unit varchar(40),
 price int,
 stock int);
```

Example 4: Sample Code

```
insert into previous_products values(7,'Uncle Bob''s Organic Dried Pears','12 - 1 lb
pkgs.',30.00,15);
insert into previous_products values(8,'Northwoods Cranberry Sauce','12 - 12 oz
jars',40.00,6);
insert into previous_products values(1,'Chang','24 - 12 oz bottles',19.00,17);
insert into previous_products values(3,'Aniseed Syrup','12 - 550 ml bottles',10.00,13);
insert into previous_products values(4,'Chef Anton''s Cajun Seasoning','48 - 6 oz
jars',22.00,53);
insert into previous_products values(5,'Chef Anton''s Gumbo Mix','36 boxes',21.35,0);
insert into previous_products values(6,'Grandma''s Boysenberry Spread','12 - 8 oz
jars',25.00,120);
```

Example 5: Sample Code

```
insert into current_products values(7,'Uncle Bob''s Organic Dried Pears','12 - 1 lb
pkgs.',30.00,15);
```

```
insert into current_products values(8,'Northwoods Cranberry Sauce','12 - 12 oz jars',40.00,6);
insert into current_products values(9,'Mishi Kobe Niku','18 - 500 g pkgs.',97.00,29);
insert into current_products values(10,'Ikura','12 - 200 ml jars',31.00,31);
insert into current_products values(11,'Queso Cabrales','1 kg pkg.',21.00,22);
insert into current_products values(5,'Chef Anton''s Gumbo Mix','36 boxes',21.35,0);
insert into current_products values(6,'Grandma''s Boysenberry Spread','12 - 8 oz jars',25.00,120);
```

**Example 6: Sample Code**

- 1.** Get the details of all products irrespective of the fact whether they are in previous set or current set. **(To Do)**
  
- 2.** Get the details of all products along with the repetition of those that were present both in the previous and current sets. **(To Do)**

## Lab 6. DB2 Objects

|              |                                                                                                        |
|--------------|--------------------------------------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>Learning various commands to define Database Objects</li> </ul> |
| <b>Time</b>  | 90 min                                                                                                 |

**1.** Create the Customer table with the following columns. (**To Do**)

```

Customerid INTEGER(5)
CustomerName INTEGER(10)
Address1 VARCHAR(30)
Address2 VARCHAR(30)

```

**2.** Modify the Customer table CustomerName column of datatype with VARCHAR(30). CustomerName should not accept Nulls. (**To Do**)

**3.** Add the following Columns to the Customer table. (**To Do**)

```

Sex VARCHAR(1)
Age INTEGER(3)
PhoneNo INTEGER(10)
Income INTEGER(10,2)

```

**4.** Insert rows with the following data in to the Customer table.

Insert into customer values: (1000, 'Allen', '#115 Chicago', '#115 Chicago', 'M', '25, 7878776', 10000)

In similar manner, add the below records to the Customer table (**To Do**):

- 1000, Allen, #115 Chicago, #115 Chicago, M, 25, 7878776, 10000
- 1001, George, #116 France, #116 France, M, 25, 434524, 20000
- 1002, Becker, #114 New York, #114 New York, M, 45, 431525, 15000.50

**5.** Add the Primary key constraint for CustomerId with the name CustId\_Prim.

**6.** Insert the row given below in the Customer table and see the message generated by the database server. (**To Do**)

- 1002, John, #114 Chicago, #114 Chicago, M, 45, 439525, 19000.60

**9.** Drop the constraint CustId\_Prim on CustomerId and insert the following Data. Alter Customer table, drop constraint Custid\_Prim.

- 1002, Becker, #114 New York, #114 New York , M, 45, 431525, 15000.50
- 1003, Nanapatekar, #115 India, #115 India , M, 45, 431525, 20000.50

**12.** Drop the GrossIncome column from Customer table. **(To Do)**

**13.** Add a new column AnnualIncome to Customer table. **(To Do)**

**14.** Create the Suppliers table based on the structure of the Customer table. Include only the CustomerId, CustomerName, Address1, Address2, and phoneno columns.

Name the columns in the new table as SupplID, SName, Addr1, Addr2, and Contactno respectively. **(To Do)**

**15.** Drop the Customer table referred above and recreate the following table with the name Customer. **(To Do)**

|              |                                                           |
|--------------|-----------------------------------------------------------|
| CustomerId   | INTEGER(5) Primary key(Name of constraint is CustId_Prim) |
| CustomerName | VARCHAR(30) Not Null                                      |
| Address1     | VARCHAR(30) Not Null                                      |
| Address2     | VARCHAR(30)                                               |
| Sex          | VARCHAR(1)                                                |
| Age          | INTEGER(3)                                                |
| PhoneNo      | INTEGER(10)                                               |
| Income       | INTEGER(10,2)                                             |

**16.** Create the Transactions table with the following Columns. **(To Do)**

|                 |                        |
|-----------------|------------------------|
| CustomerId      | INTEGER(5)             |
| CustomerName    | VARCHAR(10)            |
| Debit           | INTEGER(10,2)          |
| Credit          | INTEGER(10,2)          |
| Balance         | INTEGER(10,2) Not Null |
| TransactionDate | Date                   |

**17.** Relate Transactions table and Customer table through CustomerId column with the constraint name Cust\_tran.

```
Alter table Transaction
Add constraint Cust_Tran foreign key (CustomerId)
References Customer (CustomerId);
```

Example 7: Sample Code

**18.** Insert the following rows in Customer table (**To Do**):

- 1000, Allen, #115 Chicago, #115 Chicago, M, 25, 7878776, 10000
- 1001, George, #116 France, #116 France, M, 25, 434524, 20000
- 1002, Becker, #114 New York, #114 New York, M, 45, 431525, 15000.50

**19.** Insert the following rows to the Transactions table (**To Do**):

- 1000,Allen, 3000,2000,1000, 10 Oct 92
- 1001,George, 6000,2000,4000,08 Sep 92
- 1002, Becker, 10000,2000,8000,4 Nov 92
- 1003, 'Bill', 30000,2000, 280000,4 Nov 92

**20.** Modify the Transaction table with the Check constraint to ensure Credit should not be greater than Debit + Balance. (**To Do**)

**21.** Modify the Transaction table keeping a Check constraint with the name Balan\_Check for the Minimum Balance which should be greater than 500. (**To Do**)

**22.** Modify the Transaction table such that if Customer is deleted from Customer table then all his transactions should be deleted from Transaction table. (**To Do**)

**23.** Create Backup copy for the Transactions table with the name 'TransactionsPerDay'. (**To Do**)

**24.** Create a view 'Trans\_view' with columns CustomerId, CustomerName, Debit, Credit, and Balance from Table Transaction. In the view Transview, the columns names should be accountId, AccountHolderName, Debit, Credit, and Balance for the respective columns from Transaction table. (**To Do**)

**25.** Insert the following rows into Trans\_View (**To Do**):

- 5000,George,0,2000,2000,

**26.** Create a join view on EMP table and DEPT table with the name vw\_EMP\_DEPT containing columns such as ename, dname, deptno, sal, etc. (**To Do**)

**27.** Create a Unique index with the name No\_Name on DeptNo and Dname of Dept table. (**To Do**)

## Lab 7. Data Manipulation Language

|              |                                                                                                                            |
|--------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>• Use various DML commands to manipulate the underlying database objects</li> </ul> |
| <b>Time</b>  | 30 min                                                                                                                     |

1. Create Employee table with same structure as EMP table.

SQL>Create table employee as select \* from emp where 1=3

SQL>desc employee

| Name     | Null?    | Type         |
|----------|----------|--------------|
| EMPNO    | NOT NULL | INTEGER(4)   |
| ENAME    |          | VARCHAR(10)  |
| JOB      |          | VARCHAR(50)  |
| MGR      |          | INTEGER(4)   |
| HIREDATE |          | DATE         |
| SAL      |          | INTEGER(7,2) |
| COMM     |          | INTEGER(7,2) |
| DEPTNO   |          | INTEGER(2)   |

SQL>select \* from employee

no rows selected

2. Write a query to populate Employee table using EMP table's empno, ename, sal, deptno columns. **(To Do)**

SQL>select \* from employee

| EMPNO | ENAME  | JOB | MGR | HIREDATE | SAL  | COMM | DEPTNO |
|-------|--------|-----|-----|----------|------|------|--------|
| 7369  | SMITH  |     |     |          | 800  |      | 20     |
| 7499  | ALLEN  |     |     |          | 1600 |      | 30     |
| 7521  | WARD   |     |     |          | 1250 |      | 30     |
| 7566  | JONES  |     |     |          | 2975 |      | 20     |
| 7654  | MARTIN |     |     |          | 1250 |      | 30     |
| 7698  | BLAKE  |     |     |          | 2850 |      | 30     |
| 7782  | CLARK  |     |     |          | 2450 |      | 10     |
| 7788  | SCOTT  |     |     |          | 3000 |      | 20     |
| 7839  | KING   |     |     |          | 5000 |      | 10     |
| 7844  | TURNER |     |     |          | 1500 |      | 30     |
| 7876  | ADAMS  |     |     |          | 1100 |      | 20     |
| 7900  | JAMES  |     |     |          | 950  |      | 30     |
| 7902  | FORD   |     |     |          | 3000 |      | 20     |
| 7934  | MILLER |     |     |          | 1300 |      | 10     |

14 rows selected.

3. Write a query to change the job and deptno of employee whose empno is 7698 to the job and deptno of employee having empno 7788. **(To Do)**

4. Delete the details of department whose department name is 'SALES'. **(To Do)**

**5.** Write a query to change the deptno of employee with empno 7788 to that of employee having empno 7698. (**To Do**)

**6.** Insert the following rows to the Employee table

- 1000,Allen,Clerk,1001,12-jan-01,3000,2,10
- 1001,George,analyst,null,08 Sep 92,5000,0,10
- 1002,Becker,Manager,1000,4 Nov 92,2800,4,20
- 1003,'Bill',Clerk,1002,4 Nov 92,3000,0,20

## Lab 8. Using and execution of embedded SQL (on IBM Mainframes)

|       |                                                                                                             |
|-------|-------------------------------------------------------------------------------------------------------------|
| Goals | <ul style="list-style-type: none"><li>• Understand how to execute embedded SQL on IBM mainframes.</li></ul> |
| Time  | 90 minutes                                                                                                  |

- Create table definitions using DCLGENs
- Key in the program
- Modify & submit the Precompile JCL
- Modify & submit the Bindplan JCL
- Modify & submit the Complink JCL and Run JCL
- Check the output

### 8.1: Create table definitions using DCLGENs

Select 2 from the DB2I Primary Option menu.

Enter the following details on DCLGEN screen

**SOURCE TABLE NAME** (name of the DB2 table qualified by the owner name).

**DATA SET NAME** (name of the member of a PDS. It should be the same PDS where you have your source code)

After the member is generated a confirmation message is displayed. To view the contents of DEPTDCL, return to the ISPF/PDF Primary Option Menu.

The contents of DEPTDCL contain the options select in DCLGEN, the table definition...

Telnet - 192.168.113.3

Connect Edit Terminal Help

File Edit Settings Menu Utilities Compilers Test Help

```

EDIT DA0002T.ANNETTE.DB2SORCE(DEPTDCL) - 01.00 Columns 00001 00072
Command ==> ■
==MSG> -CAUTION- Profile is set to STATS ON. Statistics did not exist for
==MSG> this member, but will be generated if data is saved.
000001 ****
000002 * DCLGEN TABLE(DA0021T.DEPT)
000003 * LIBRARY(DA0002T.ANNETTE.DB2SORCE(DEPTDCL))
000004 * ACTION(REPLACE)
000005 * LANGUAGE(COBOL)
000006 * QUOTE
000007 * LABEL(YES)
000008 * ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS *
000009 ****
000010 EXEC SQL DECLARE DA0021T.DEPT TABLE
000011 (DEPTNO INTEGER NOT NULL,
000012 DNAME CHAR(10),
000013 LOCATION VARCHAR(10)
000014) END-EXEC.
000015 ****
000016 * COBOL DECLARATION FOR TABLE DA0021T.DEPT
PF 1=HELP 2=SPLIT 3=END 4=RETURN 5=RFIND 6=RCHANGE
SB# ■ 1p CRTL001 04-015 A

```

... and the COBOL variable definitions.

Telnet - 192.168.113.3

Connect Edit Terminal Help

File Edit Settings Menu Utilities Compilers Test Help

```

EDIT DA0002T.ANNETTE.DB2SORCE(DEPTDCL) - 01.00 Columns 00001 00072
Command ==> ■
==MSG> -COBOL DECLARATION FOR TABLE DA0021T.DEPT
000016 * COBOL DECLARATION FOR TABLE DA0021T.DEPT
000017 ****
000018 01 DCLDEPT.
000019 *
000020 10 DEPTNO PIC S9(9) USAGE COMP.
000021 *
000022 10 DNAME PIC X(10).
000023 *
000024 10 LOCATION.
000025 49 LOCATION-LEN PIC S9(4) USAGE COMP.
000026 49 LOCATION-TEXT PIC X(10).
000027 ****
000028 * THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 3
000029 ****
***** ***** Bottom of Data *****

PF 1=HELP 2=SPLIT 3=END 4=RETURN 5=RFIND 6=RCHANGE
SB# ■ 1p CRTL001 04-015 A

```

## Key in the program

Key in the program using the ISPF editor.

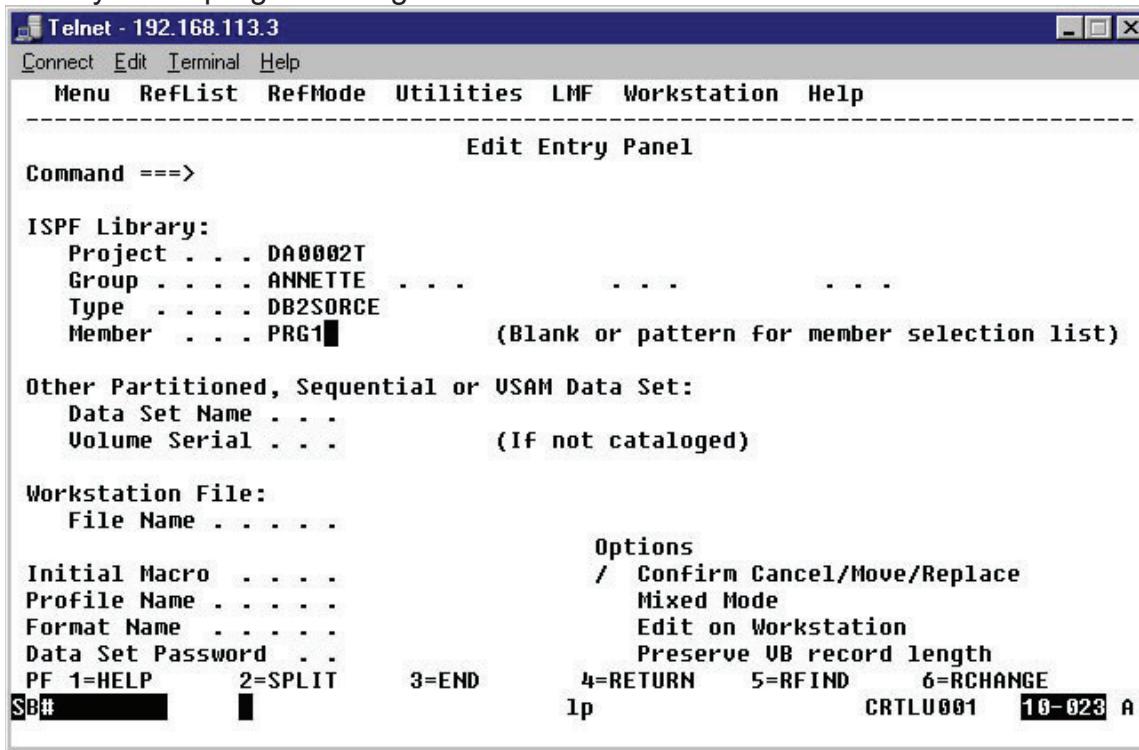


Figure 25

```

000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PRG1.
000300 AUTHOR. ANNETTE.
000302*****
000310* Display the name of the department, given the department no.*
000320*****
000400 ENVIRONMENT DIVISION.
000500 DATA DIVISION.
000600 WORKING-STORAGE SECTION.
000700 EXEC SQL
000710 INCLUDE DEPTDCL
000800 END-EXEC.
000900*
000910 EXEC SQL
000920 INCLUDE SQLCA
001000 END-EXEC.
001100 PROCEDURE DIVISION.
001200 0000-MAIN-PARA.
001210* Checking a row with salary as null.
001300 MOVE 10 TO DEPTNO.

```

```

001301 EXEC SQL
001820 SELECT DNAME
001830 INTO :DNAME
001840 FROM DA0021T.DEPARTMENT
001850 WHERE DEPTNO = :DEPTNO
001860 END-EXEC.
001861 DISPLAY 'DEPTNO : ' DEPTNO.
001700 STOP RUN.

```

- Modify & submit the Precompile JCL

```

//DA0002TP JOB (LA2719),'ANNETTE',CLASS=A,
// MSGCLASS=X,
// NOTIFY=DA0002T,
// TIME=(,01),
// REGION=4096K
//**ROUTE PRINT RMT196
//***
//** THIS JCL PRECOMPILES THE APPLICATION PGM
//** GENERATES PURE COBOL CODE AND A DBRM.
//***
//PRECOMP EXEC PGM=DSNHPC,
// PARM='HOST(COBOL),APOST',REGION=4096K
//*
//** IN SYSIN GIVEN BELOW SPECIFY PGM NAME TO BE PRECOMPILED
//** B
//SYSIN DD DSN=DA0002T.ANNETTE.DB2SORCE(ASS1),DISP=SHR
//*
//SYSLIB DD DSN=DA0002T.ANNETTE.DB2SORCE,DISP=SHR
//*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//STEPLIB DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//*
//** HERE YOU SPECIFY O/P MEMBER OF PRECOMPILE
//**
//SYSCIN DD DSN=DA0002T.ANNETTE.DB2SORCE(MASS1),
// DISP=(SHR),UNIT=SYSDA,
// SPACE=(CYL,(5,5,4))
//** HERE YOU SPECIFY MEMBER FOR DBRM
//DBRMLIB DD DSN=DA0002T.ANNETTE.DBRMLIB(DASS1),DISP=SHR
// DISP=(NEW,KEEP),UNIT=SYSDA,
// SPACE=(CYL,(5,5))
//*
//SYSPRINT DD SYSOUT=*
//SYSTEMR DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*

```

//  

- Modify & submit the Bindplan JCL

```

//DA0002TB JOB (LA2719),'ANNETTE',CLASS=A,
// MSGCLASS=X,
// NOTIFY=DA0002T,
// TIME=(,01),
// REGION=4096K
//ROUTE PRINT RMT196
//*****THIS JCL BIND THE DBRM & GENERATES APPLICATION PLAN
//*
//*****
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=512K
//STEPLIB DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//TEPLIB DD DSN=SYS1.DB2.SDSNLOAD,DISP=SHR
//YSPROC DD DSN=SYS1.DB2.SDSNCLST,DISP=SHR
//*
//* HERE YOU SPECIFY DBRM LIBRARY
//*
//DBRMLIB DD DSN=DA0002T.ANNETTE.DBRMLIB,DISP=SHR
//*
//DB2ERROR DD SYSOUT=*
//CSABEND DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//SYSDBABOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DST2) RETRY(5) TEST(0)
 BIND PLAN(DAPLAN1)
 MEMBER(DASS1)
 ACTION(REPLACE) RETAIN
 ISOLATION(CS)
 VALIDATE(RUN)
 ACQUIRE(USE)
 RELEASE(COMMIT)
 END
/*

```

- Modify & submit the Complink JCL

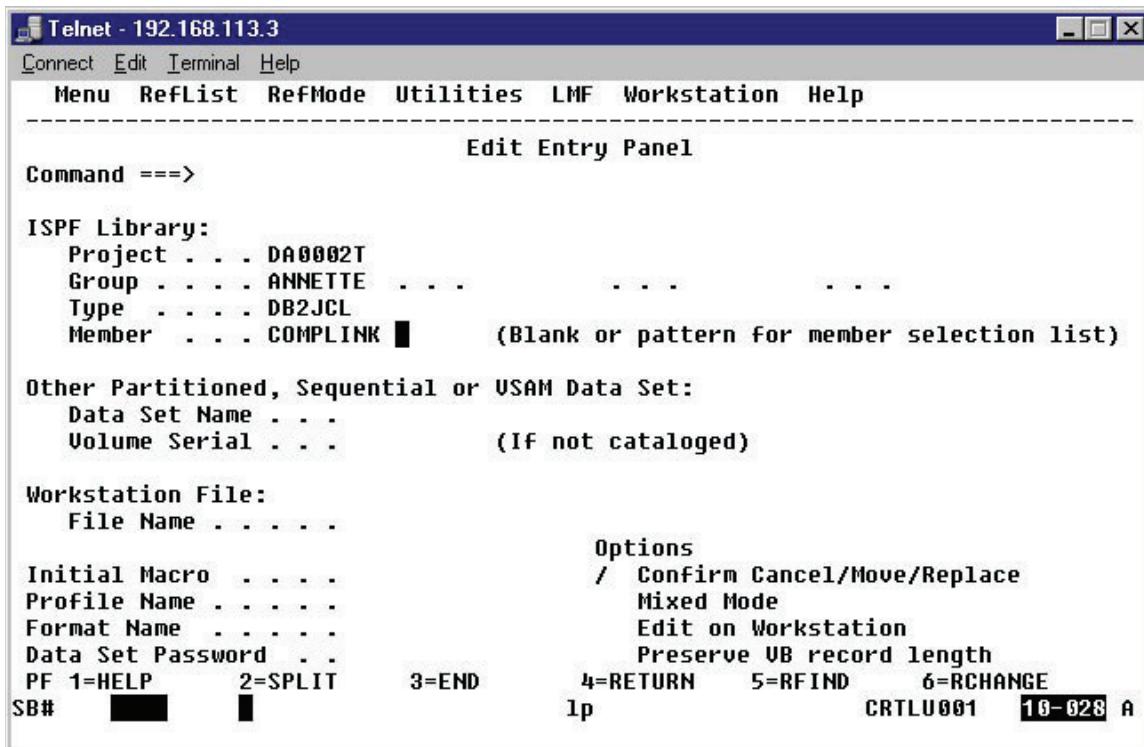


Figure 35

```

//DA0002TC JOB (LA2719),'ANNETTE',CLASS=A,
// MSGCLASS=X,
// NOTIFY=DA0002T,
// TIME=(,01),
// REGION=4096K
//*****
//** THIS JCL COMPILES AND LINK THE PRECOMPILED PGM
//*
//*****
//** THIS STEP COMPILES THE PRECOMPILED PGM
//*****
//COB EXEC PGM=IKFCBL00,REGION=1024K,
// PARM='NOTRUNC,NODYNAM,LIB,SIZE=4096K,BUF=116K,APOST,NORES'
// PARM='LOAD,SUPMAP,APOST'
//SYSLIB DD DSN=SYS1.COBCOMP,DISP=SHR
//SYSPRINT DD SYSOUT=*
// HERE TEMP DATASET TO RECEIVE O/P OF COMPILATION
//*
//SYSLIN DD DSN=&&TEMP,DISP=(NEW,PASS),
// UNIT=SYSALLDA,SPACE=(TRK,(40,40))
//*****

```

```

//SYSUT1 DD UNIT=SYSALLDA,SPACE=(TRK,(6,1))
//SYSUT2 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//SYSUT3 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//SYSUT4 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//*****

/* HERE YOU SPECIFY PRECOMPILED PGM MEMBER
/*

//SYSIN DD DSN=DA0002T.ANNETTE.DB2SORCE(MASS1),DISP=SHR
//*****B*****

/* STEP TO LINK THE COBOL PROGRAM
//*****

//LKED EXEC PGM=IEWL,PARM='LIST,XREF,LET,MAP',
// REGION=4096K,COND=(5,LT,COB)
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
//SYSLIB DD DSN=SYS1.COBLIB,DISP=SHR
// DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//SYSLMOD DD DSN=DA0002T.ANNETTE.LOADLIB(ASS1),DISP=SHR
/* UNIT=SYSALLDA
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//SYSPRINT DD SYSOUT=*

```

```

//DA0002TA JOB (LA2719),'ANNETTE',CLASS=A,
// MSGCLASS=X,
// NOTIFY=DA0002T,
// TIME=(0,1),
// REGION=4096K
//ROUTE PRINT RMT196
//*****

/* THIS JCL EXECUTES THE BATCH DB2 APPLICATION
/*

//*****

//RUNPGM EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=512K
//STEPLIB DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//TEPLIB DD DSN=SYS1.DB2.SDSNLOAD,DISP=SHR
//YSPROC DD DSN=SYS1.DB2.SDSNCLST,DISP=SHR
//BRMLIB DD DSN=DA0002T.SOURCE.DBRM,DISP=SHR
//

/* HERE YOU SPECIFY DBRM LIBRARY
//

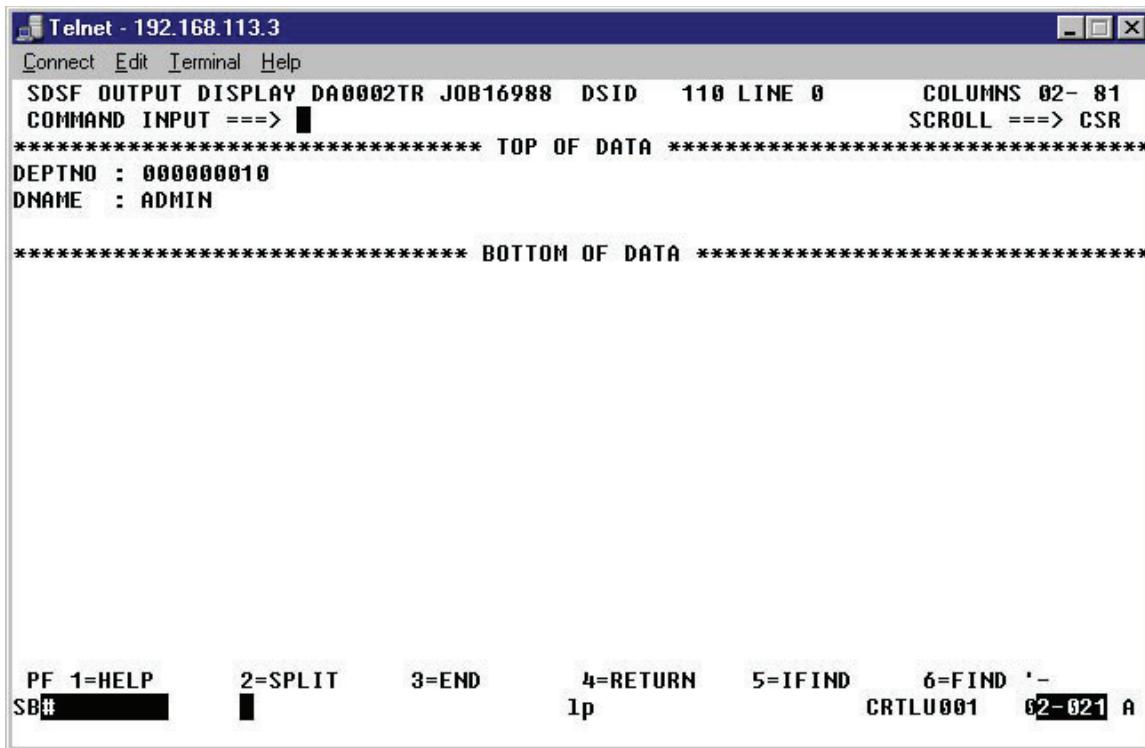
//DBRMLIB DD DSN=DA0002T.ANNETTE.DBRMLIB,DISP=SHR
//

//DB2ERROR DD SYSOUT=*
//CSABEND DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//SYSDBABOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSOUT DD SYSOUT=*

```

```
//SYSIN DD *
/*1
/*
//SYSTSIN DD *
 DSN SYSTEM(DST2) RETRY(5) TEST(0)
 RUN PROGRAM(ASS1) -
 PLAN(DAPLAN1) -
 LIB('DA0001T.ANNETTE.LOADLIB')
 END
/*
```

- Check the output



The screenshot shows a Telnet session window titled "Telnet - 192.168.113.3". The window displays the following text:

```
SDSF OUTPUT DISPLAY DA0002TR JOB16988 DSID 110 LINE 0 COLUMNS 02- 81
COMMAND INPUT ==> ■
***** TOP OF DATA *****
DEPTNO : 000000010
DNAME : ADMIN

***** BOTTOM OF DATA *****
```

At the bottom of the screen, there is a command line interface with function key definitions:

```
PF 1=HELP 2=SPLIT 3=END 4=RETURN 5=IFIND 6=FIND '-'
SB# [REDACTED] [REDACTED] 1p CRTLU001 02-021 A
```



## Lab 9. Using and execution of embedded SQL (on IBM Mainframes)

|              |                                                                                           |
|--------------|-------------------------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>Application Development using COBOL-DB2</li> </ul> |
| <b>Time</b>  | 120 minutes                                                                               |

Write a COBOL programs:

1. Display the details of the highest paid employee.
2. Display the name and salary of the employee whose empno is passed on as a host variable. If the search returns a NULL value, a NULL INDICATOR should be used to indicate the appropriate error message.
3. Refer to shared program '**DB2PGM**' for fetching a single record from DB2 table for particular employee.
  - Do modify this given program to fetch corresponding records for all employees who belongs to Department 10.
  - Do refer to appropriate DB2 table and accordingly update the code.
4. Write COBOL programs to fulfill the given requirements.
  - Increase the commission of employees, who have completed one year of service with following conditions (The % increase is w.r.t. the salary).
    - i. If salary > 10000 and salary <= 15000 grant him a commission of 20%
    - ii. If salary > 15000 and salary <=20000 grant him a commission of 15%
    - iii. If salary > 20000 grant him a commission of 10%
  - Compute and display the total salary of all the employees where the total salary is calculated as TSAL = SALARY + COMMISSION. An additional commission of 10% of salary is granted to those employees who work as salesmen in the firm. (Hint : Commission can be null)
  - Display the department name, location and the other employee details of all those employees who belong to department no. 10.

5. Refer to shared data file named 'PS.txt' for the data.

Task list includes:

- Transfer the above flat file to dataset.
- Create a DB2 table using SPUFI
  - One application program to transfer data from PS to newly created DB2 table

File Description is given here for your reference.

FD PSFILE

LABEL RECORDS STANDARD  
RECORDING MODE IS F  
BLOCK CONTAINS 0 RECORDS  
RECORD CONTAINS 99 CHARACTERS.

01 PS-REC.

  05 PS-ID           PIC 9(6).  
  05 PS-NAME.  
    10 C-FIRST-NAME   PIC X(15).  
    10 C-LAST-NAME   PIC X(15).  
  05 PS-DOB           PIC X(10).  
  05 PS-HID           PIC X(10).  
  05 PS-DEPT          PIC X(1).  
  05 PS-GRADE        PIC X(02).  
  05 PS-DESG          PIC X(2).  
  05 PS-BASIC        PIC 9(6).  
  05 PS-GENDER        PIC X(1).  
  05 PS-MS            PIC X(1).  
  05 PS-ADDRESS       PIC X(20).  
  05 PS-PHN-NO       PIC 9(10).



## Lab 10. Using and execution of embedded SQL (on IBM Mainframes)

|       |                                                                                                         |
|-------|---------------------------------------------------------------------------------------------------------|
| Goals | <ul style="list-style-type: none"> <li>Application Development using COBOL-DB2- Case Studies</li> </ul> |
| Time  | 120 minutes                                                                                             |

### Problem Statement 1:

1. Create the following table definitions and write programs to INSERT data into tables
  - a. Create TABLE CUST\_MAST. (**CUST\_NO** to be the **PRIMARY KEY**)
  - b. Create TABLE CUST\_TRANS.
  - c. Create DCLGEN's for both the tables created using the DB2I and option 2.

| CUST_MAST           |      |                        |                  |
|---------------------|------|------------------------|------------------|
| COLUMNS             | SIZE | No of Decimal position | DATA TYPE        |
| CUST_NO             | 5    |                        | INTEGER NOT NULL |
| CUST_NAME           | 20   |                        | VARCHAR NOT NULL |
| DT_OF_LAST_PURCHASE | 8    |                        | DATE             |
| AMOUNT_OWED         | 9    | 2                      | DECIMAL          |

| CUST_TRANS      |      |                        |                  |
|-----------------|------|------------------------|------------------|
| COLUMNS         | SIZE | No of Decimal position | DATA TYPE        |
| CUST_NO         | 5    |                        | INTEGER NOT NULL |
| DT_OF_PURCHASE  | 8    |                        | DATE NOT NULL    |
| AMT_OF_PURCHASE | 7    | 2                      | DECIMAL NOT NULL |

2. Write a program to then update the CUST\_MAST table.
  - a. If a transaction exists in CUST\_TRANS table for which there is no corresponding row in CUST\_MAST table, display it as an error

- b. For all transactions in CUST\_TRANS table with corresponding row in the CUST\_MAST table (these rows are to be updated), add the AMT\_OF\_PURCHASE from the CUST\_TRANS to AMOUNT\_OWED in the CUST\_MAST table and UPDATE the column DT\_OF\_LAST\_PURCHASE.

#### Problem Statement 2:

ABC Company's monthly take-home salary is to be computed. The Company maintains the salary information of their employees in the EMPLOYEE table for computations.

Income Tax company has given the below data for computation of tax.

Create a new table **TAX\_DATA** with the column names and also INSERT the data as mentioned in the table below:

| TAXABLE_INCOME<br>(INTEGER NOT<br>NULL , Length 6) | FEDERAL_TAX_RATE<br>(DECIMAL NOT NULL,<br>Length V999) | STATE_TAX_RATE<br>(DECIMAL NOT<br>NULL, Length V999) |
|----------------------------------------------------|--------------------------------------------------------|------------------------------------------------------|
| 09800                                              | .040                                                   | .010                                                 |
| 12000                                              | .080                                                   | .020                                                 |
| 30000                                              | .150                                                   | .040                                                 |

**Note: The State Tax is 1% and the federal tax 4% for taxable income less than or equal to 9800. The taxable income between 9801 and 12000 (inclusive), the state tax is 2% and federal tax is 8%. The taxable income between 12000 and 30000 (inclusive), the state tax is 4% and federal tax is 15%.**

Calculate the Monthly statement based on the following conditions:

- Standard deduction = 10% of the first \$10,000 of annual salary
- Social Security and Medicare Taxes
  - Social Security Tax = 6.2% of the first \$62,700 of annual salary
  - Medicare Tax = 1.45% of each annual salary
- Taxable Income = Annual Salary – Standard Deduction – Dependent Deduction
- Use the Tax table and calculate the taxable income

- e. Annual Take-home pay = Annual salary – (state tax % \* taxable income) – (Federal tax % \* taxable income) - (Social Security Tax + Medicare Tax)
- f. Monthly take Home = Annual Take-home pay / 12

Write the Program to generate Flat file report for each employee monthly take home salary. For Taxable income write a subroutine to calculate the taxable income.

## Lab 11. Problem statements based on Analysis, Debugging and Enhancement tasks

|       |                                                                                                       |
|-------|-------------------------------------------------------------------------------------------------------|
| Goals | <ul style="list-style-type: none"><li>Application Development using COBOL-DB2- Case Studies</li></ul> |
| Time  | 180 minutes                                                                                           |

### Problem Statement 1: Migration from FMS (file management system) to RDBMS (DB2)

Refer to ‘Bank Transaction Updation and Reporting’ Lab Exercise developed in COBOL training.

- Migrate the complete project from file system to RDBMS data base.  
**Instead of using file system viz. Account Transaction file and the Account Master file, do use DB2 tables to maintain the data.**
- Analyze the program, do the changes to Cobol code, remove the error and successfully execute the program.
- Document the error in the excel sheet along with steps taken to resolve the error.

### Problem Statement 2: Migration from FMS (file management system) to RDBMS (DB2)

Refer to ‘Online Bus Booking’ Lab Exercise developed in COBOL training.  
[Online Bus Booking](#)

You have a program to read Booking Trans file, Check the availability of the seat in the Bus Detail file, Generate Ticket no., PNR no., Update the Booking Master file, Update Bus Detail file, Print Tickets & Print canceled report in case of cancellation.

- Migrate the complete project from file system to RDBMS data base.  
**Instead of using file system, do use DB2 tables to maintain the data.**
- Analyze the program, do the changes to Cobol code, remove the error and successfully execute the program.

- Document the error in the excel sheet along with steps taken to resolve the error.

### **Problem Statement 3: Analyzing the given code and do enhancement.**

Refer to **shared ‘Login Program’ and ‘Login Map’ for both COBOL-CICS and BMS coding.**

- Analyze the program
- Do create required tables along with data in database.
- Make the required changes to code and successfully execute the programs.
- Display the appropriate messages on the map based on successful login as well for failed to login due incorrect login credentials.
- Document the error in the excel sheet along with steps taken to resolve the error.

## Appendices

### Appendix A: JCL's

```

//DA0001TC JOB (LA2719),'FACULTY',CLASS=A,
// MSGCLASS=X,
// NOTIFY=DA0001T,
// TIME=(,01),
// REGION=4096K
//ROUTE PRINT RMT196
//***
// THIS JCL PRECOMPILES THE APPLICATION PGM
// GENERATES PURE COBOL CODE AND A DBRM.
//***
//PRECOMP EXEC PGM=DSNHPC,
// PARM='HOST(COBOL),APOST',REGION=4096K
//*
// IN SYSIN GIVEN BELOW SPECIFY PGM NAME TO BE PRECOMPILED
//* B
//SYSIN DD DSN=DA0001T.FACULTY.DB2SORCE(ASS1),DISP=SHR
//*
//SYSLIB DD DSN=DA0001T.FACULTY.DB2SORCE,DISP=SHR
//*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//STEPLIB DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//*
// HERE YOU SPECIFY O/P MEMBER OF PRECOMPILE
//*
//SYSCIN DD DSN=DA0001T.FACULTY.DB2SORCE(MASS1),
// DISP=(SHR),UNIT=SYSDA,
// SPACE=(CYL,(5,5,4))
// HERE YOU SPECIFY MEMBER FOR DBRM
//DBRMLIB DD DSN=DA0001T.FACULTY.DBRMLIB(DASS1),DISP=SHR
// DISP=(NEW,KEEP),UNIT=SYSDA,
// SPACE=(CYL,(5,5))
//*
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//

//DA0001TA JOB (LA2719),'FACULTY',CLASS=A,

```

```

// MSGCLASS=X,
// NOTIFY=DA0001T,
// TIME=(,01),
// REGION=4096K
/*ROUTE PRINT RMT196

/* THIS JCL BIND THE DBRM & GENERATES APPLICATION PLAN
/*

//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=512K
//STEPLIB DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//TEPLIB DD DSN=SYS1.DB2.SDSNLOAD,DISP=SHR
//YSPROC DD DSN=SYS1.DB2.SDSNCLST,DISP=SHR
/*
/* HERE YOU SPECIFY DBRM LIBRARY
/*
//DBRMLIB DD DSN=DA0001T.FACULTY.DBRMLIB,DISP=SHR
/*
//DB2ERROR DD SYSOUT=*
//CSABEND DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//SYSDBOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DST2) RETRY(5) TEST(0)
 BIND PLAN(DAPLAN1) -
 MEMBER(DASS1) -
 ACTION(REPLACE) RETAIN -
 ISOLATION(CS) -
 VALIDATE(RUN) -
 ACQUIRE(USE) -
 RELEASE(COMMIT)
 END
/*

```

```

//DA0001TC JOB (LA2719),'FACULTY',CLASS=A,
// MSGCLASS=X,
// NOTIFY=DA0001T,
// TIME=(,01),
// REGION=4096K

```

```

//*****
// THIS JCL COMPILES AND LINK THE PRECOMPIED PGM
//*
//*****
// THIS STEP COMPILES THE PRECOMPILED PGM
//*****
//COB EXEC PGM=IKFCBL00,REGION=1024K,
// PARM='NOTRUNC,NODYNAM,LIB,SIZE=4096K,BUF=116K,APOST,NORES'
//* PARM='LOAD,SUPMAP,APOST'
//SYSLIB DD DSN=SYS1.COBCOMP,DISP=SHR
//SYSPRINT DD SYSOUT=*
//* HERE TEMP DATASET TO RECEIVE O/P OF COMPILATION
//*
//SYSLIN DD DSN=&&TEMP,DISP=(NEW,PASS),
// UNIT=SYSALLDA,SPACE=(TRK,(40,40))
//*****
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(TRK,(6,1))
//SYSUT2 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//SYSUT3 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//SYSUT4 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//*****
//* HERE YOU SPECIFY PRECOMPILED PGM MEMBER
//*
//SYSIN DD DSN=DA0001T.FACULTY.DB2SORCE(MASS1),DISP=SHR
//*****
//** STEP TO LINK THE COBOL PROGRAM
//*****
//LKED EXEC PGM=IEWL,PARM='LIST,XREF,LET,MAP',
// REGION=4096K,COND=(5,LT,COB)
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
//SYSLIB DD DSN=SYS1.COBLIB,DISP=SHR
// DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//SYSLMOD DD DSN=DA0001T.FACULTY.LOADLIB(ASS1),DISP=SHR
//* UNIT=SYSALLDA
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(CYL,(6,1))
//SYSPRINT DD SYSOUT=*

//DA0001TA JOB (LA2719),'FACULTY',CLASS=A,
// MSGCLASS=X,
// NOTIFY=DA0001T,
// TIME=(0,1),
// REGION=4096K
//ROUTE PRINT RMT196
//*****
//* THIS JCL EXECUTES THE BATCH DB2 APPLICATION

```

```
/*
//***** *****
//RUNPGM EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=512K
//STEPLIB DD DSN=SYS0.DB2.TEST.DSNLOAD,DISP=SHR
//TEPLIB DD DSN=SYS1.DB2.SDSNLOAD,DISP=SHR
//YSPROC DD DSN=SYS1.DB2.SDSNCLST,DISP=SHR
//BRMLIB DD DSN=DA0001T.SOURCE.DBRM,DISP=SHR
/*
// HERE YOU SPECIFY DBRM LIBRARY
/*
//DBRMLIB DD DSN=DA0001T.FACULTY.DBRMLIB,DISP=SHR
/*
//DB2ERROR DD SYSOUT=*
//CSABEND DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//SYSDBABOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
/*1
/*
//SYSTSIN DD *
DSN SYSTEM(DST2) RETRY(5) TEST(0)
RUN PROGRAM(ASS1)
 -
 PLAN(DAPLAN1)
 -
 LIB('DA0001T.FACULTY.LOADLIB')
END
/*
```

## Appendix B: DB2 datatypes and the COBOL equivalent Picture clauses

| DB2 datatype                        | COBOL picture clause                                            |
|-------------------------------------|-----------------------------------------------------------------|
| SMALLINT                            | PIC S9(4) COMP.                                                 |
| INTEGER                             | PIC S9(9) COMP.                                                 |
| DECIMAL (3,2)                       | PIC S9V99 COMP-3.                                               |
| DECIMAL (15,2)                      | PIC S9(13)V99 COMP-3.                                           |
| DECIMAL (9,4)                       | PIC S9(5)V9(4) COMP-3.                                          |
| CHAR (10)                           | PIC X(10).                                                      |
| VARCHAR(20)<br>E.G. JOB VARCHAR(20) | 01 JOB.<br>49 JOB-LEN PIC S9(4) COMP.<br>49 JOB-TEXT PIC X(20). |
| DATE                                | PIC X(10)                                                       |
| INDICATOR VARIABLES                 | PIC S9(4) COMP.                                                 |

| DB2 Data Type | Cobol                                                                          | Maximum Value/Format                                                                                    |
|---------------|--------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| smallint      | PIC S9(4) COMP                                                                 | range of values -32,768 to +32,767                                                                      |
| integer       | PIC S9(9) COMP                                                                 | range of values -2,147,483,648 to +2,147,483,647                                                        |
| decimal(p,s)  | PIC S9(p)V9(s) COMP-3                                                          | p is the total number of digits, s is the number of digits to the right of the decimal                  |
| char(n)       | PIC X(n)                                                                       | n cannot exceed 254                                                                                     |
| varchar(n)    | 01 IDENTIFIER<br>49 IDENTIFIER-LEN<br>PIC S9(4)<br>49 IDENTIFIER-TEXT PIC X(n) | n is maximum length and cannot exceed 254                                                               |
| date          | PIC X(10)                                                                      | format: yyyy-mm-dd                                                                                      |
| time          | PIC X(6) OR PIC X(8)                                                           | format: hh.mm.ss                                                                                        |
| timestamp     | PIC X(26)                                                                      | format: yyyy-mm-dd-hh.mm.ss.nnnnnn                                                                      |
| Graphic       | PIC G(n)                                                                       | n must be between 1 and 127 and refers to the number of double-byte characters, not the number of bytes |



## Appendix C: SQLCA

### Fields of SQL Communication Area

```
01 SQLCA.
 05 SQLCAID PIC X(8).
 05 SQLCABC PIC S9(9) COMP-4.
 05 SQLCODE PIC S9(9) COMP-4.
 05 SQLERRM.
 49 SQLERRML PIC S9(4) COMP-4.
 49 SQLERRMC PIC X(70).
 05 SQLERRP PIC X(8).
 05 SQLERRP OCCURS 6 TIMES PIC S9(9) COMP-4.
 05 SQLWARN.
 10 SQLWARN0 PIC X.
 10 SQLWARN1 PIC X.
 10 SQLWARN2 PIC X.
 10 SQLWARN3 PIC X.
 10 SQLWARN4 PIC X.
 10 SQLWARN5 PIC X.
 10 SQLWARN6 PIC X.
 10 SQLWARN7 PIC X.
 05 SQLEXT PIC X(8).
```

## Appendix D :List of SQL codes

| SQLCODE | Keyword   | Meaning                                                                                                                                                                                                                                    |
|---------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| +100    | SELECT    | Row not found during FETCH, SELECT, UPDATE or DELETE                                                                                                                                                                                       |
| +304    | Program   | Value and host variables are incompatible                                                                                                                                                                                                  |
| -305    | Variables | Null values occurred, and no indicator variable was defined                                                                                                                                                                                |
| -501    | Cursor    | Cursor named in FETCH or CLOSE is not open                                                                                                                                                                                                 |
| -551    | Authority | You lack the authority to access the named object, possibly because the name is not spelled correctly                                                                                                                                      |
| -803    | Updating  | Duplicate keys not allowed                                                                                                                                                                                                                 |
| -805    | Plan      | DBRM not bound into this plan                                                                                                                                                                                                              |
| -811    | SELECT    | Embedded SELECT or sub select returned more than one row                                                                                                                                                                                   |
| -818    | Plan      | Timestamps in load module and plan do not agree; program was probably re-precompiled without being rebound                                                                                                                                 |
| -901    | System    | Mysterious system error, permits running more SQL statements                                                                                                                                                                               |
| -904    | System    | Unavailable resource; if resource name is a table view, et. This was probably caused by contention and re-trying the operation may work; if resource name is a 44 character VSAM file name, the file has probably been archived or deleted |
| -911    | System    | Deadlock or timeout, updates rolled back                                                                                                                                                                                                   |
| -913    | System    | Deadlock or timeout, updates not rolled back                                                                                                                                                                                               |

## Lab 12. Using and execution of embedded SQL (CA-Realia Workbench )

|              |                                                                                                                  |
|--------------|------------------------------------------------------------------------------------------------------------------|
| <b>Goals</b> | <ul style="list-style-type: none"><li>• Understand how to execute embedded SQL on CA-Realia Workbench.</li></ul> |
| <b>Time</b>  | 60 minutes                                                                                                       |

### Steps

- 1) Create a user-defined DSN
- 2) Setup CA-Realia Workbench
- 3) Create table definitions
- 4) Key in the program
- 5) Compile the program
- 6) Link the program
- 7) Debug

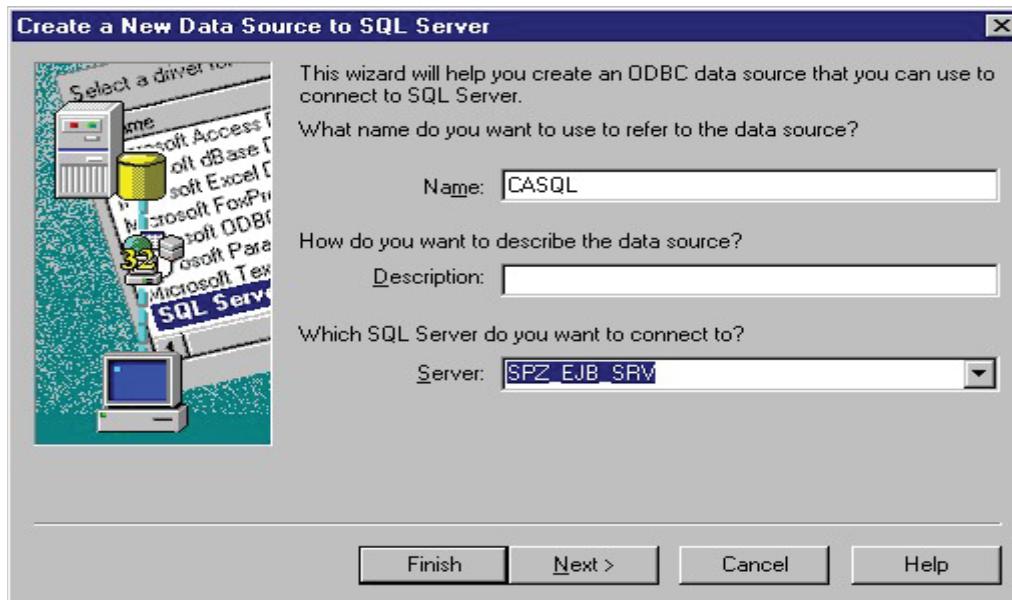
#### 1. Create a user-defined DSN

Note : We will be creating a DSN to connect to a SQL Server database on the SPZ\_EJB\_SRV server.

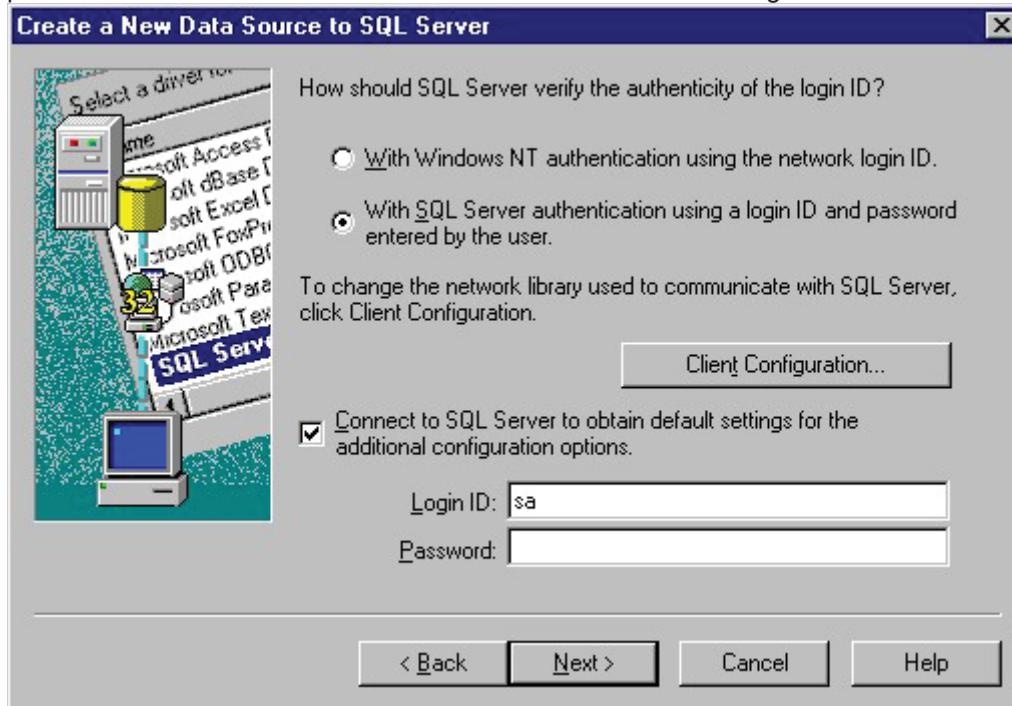
From the Windows Start Menu select Settings-> Control Panel -> Data Sources (ODBC). The following window is displayed. Click on the Add button.

In the Create New Data Source window select SQL Server and click on the Finish button.

On this screen you will have to provide a name for this data source. We will have to specify this name in CA-Realia Workbench. Select the server on which the SQL Server database resides. Click on the Next> button.

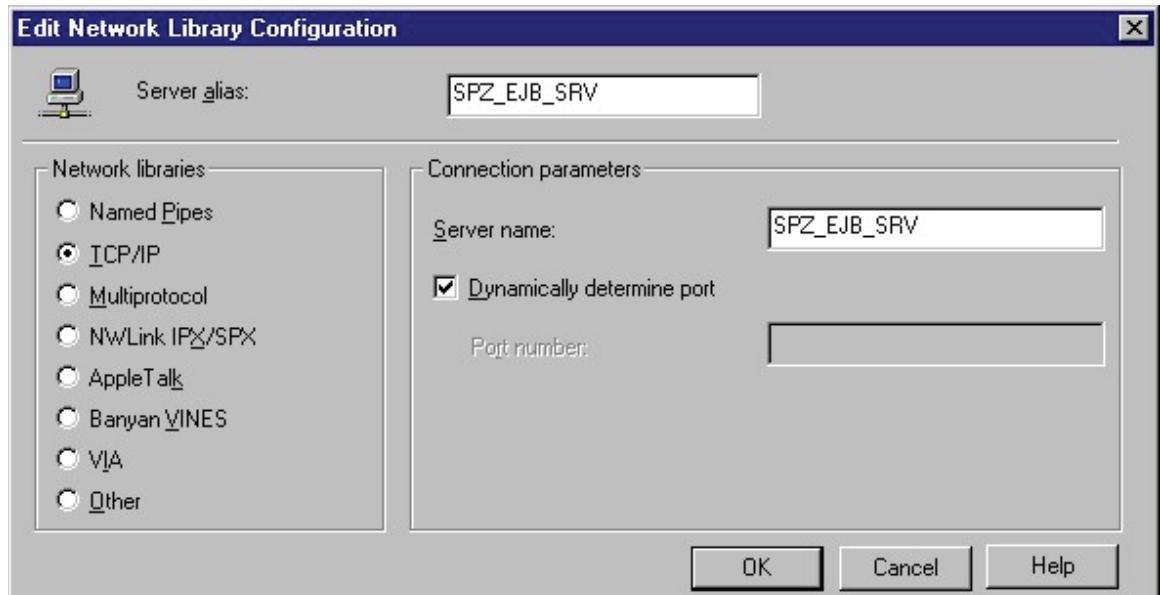


Select the type of authentication. Here we will select With SQL Server authentication using a login id and password entered by the user. Select Connect to SQL Server to obtain default settings for the additional configuration options. Specify your login id and password for the SQL Server database. Click on the Client Configuration button.

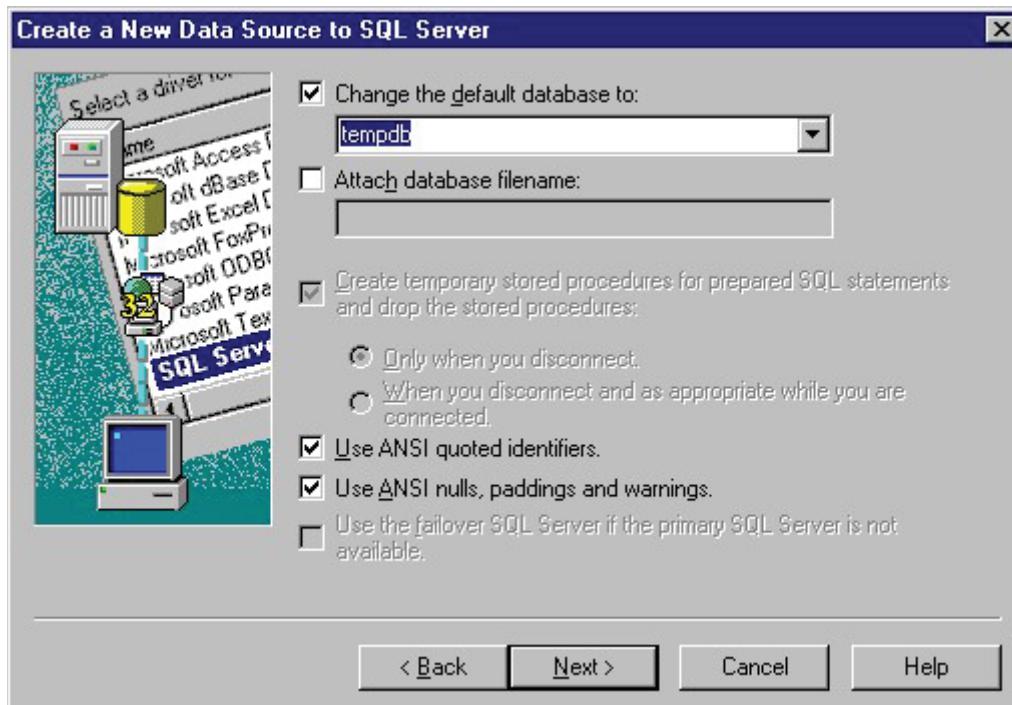


**Figure 49**

Check the Network Library Configuration settings. Click on the OK button. Click on the Next> button on the Create a New Data Source to SQL Server dialog window.

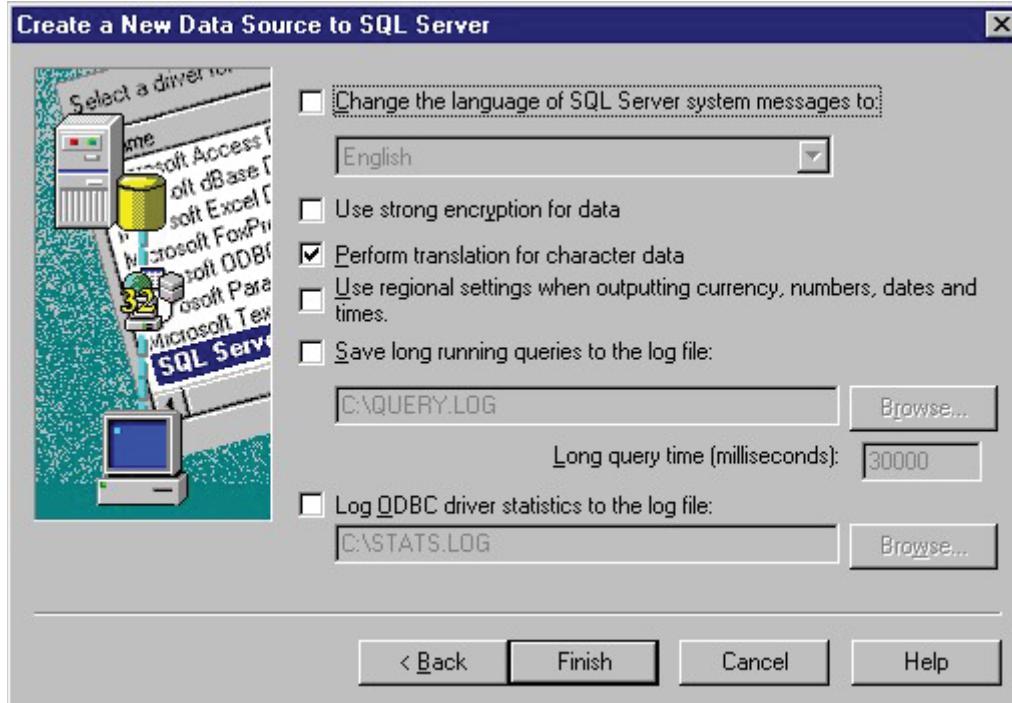


Select the Change the default database to checkbox. We have our sample tables created on the tempdb database, hence we have selected tempdb from the dropdown list. Click on the Next> button.



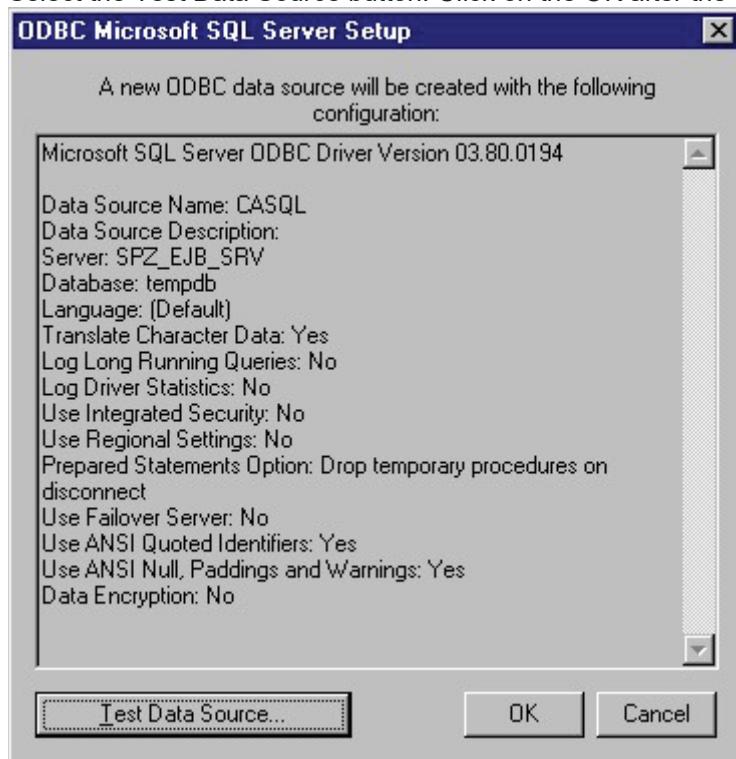
**Figure 51**

Select Perform translation for character data and click on the Finish button.



**Figure 52**

Select the Test Data Source button. Click on the OK after the Testing is successful.

**Figure 53**

## 2. Setup CA-Realia Workbench

Start CA-Realia Workbench

Select Build -> Options -> System DEFAULT

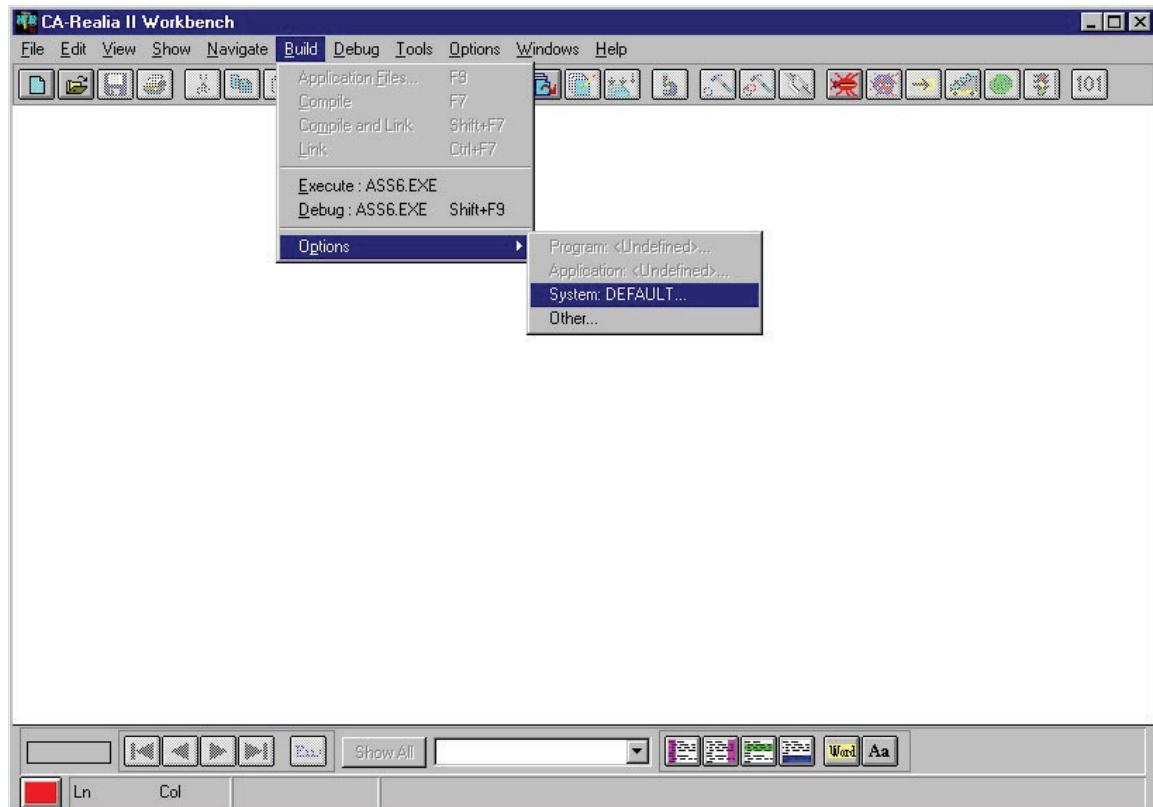
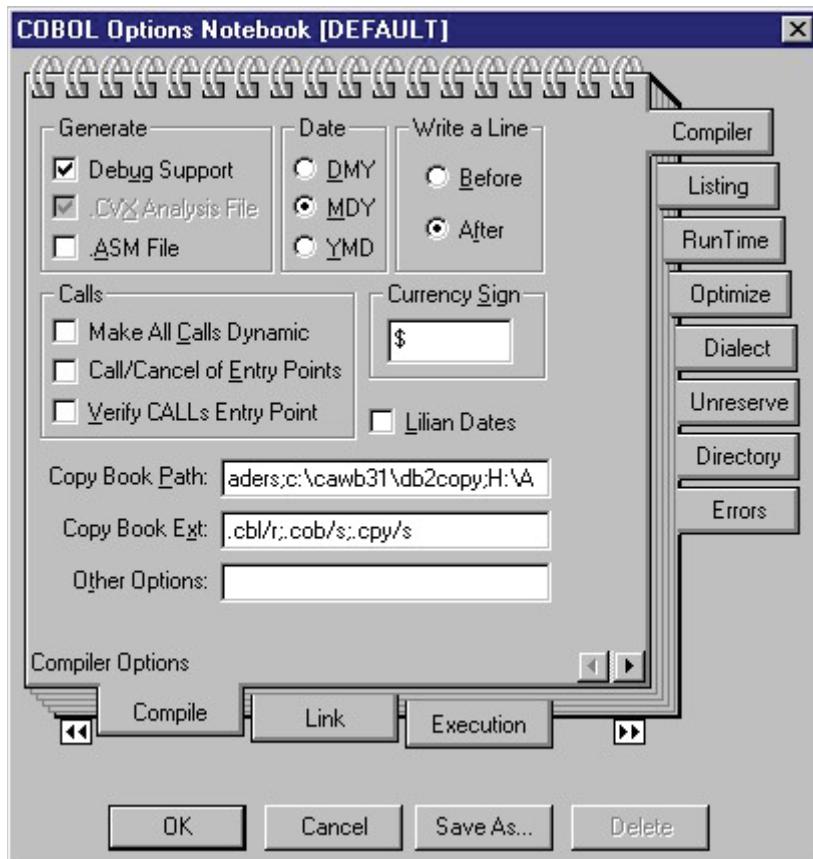


Figure 54

We will have to add a reference to the DB2COPY folder in the Workbench folder, as this directory contains the copy books e.g. SQLCA. We will also give a reference to our directory where we store our copy books (for table definitions).

Add C:\CAWB31\DB2COPY;H:\ANNETTE\CA in the Copy Book Path.

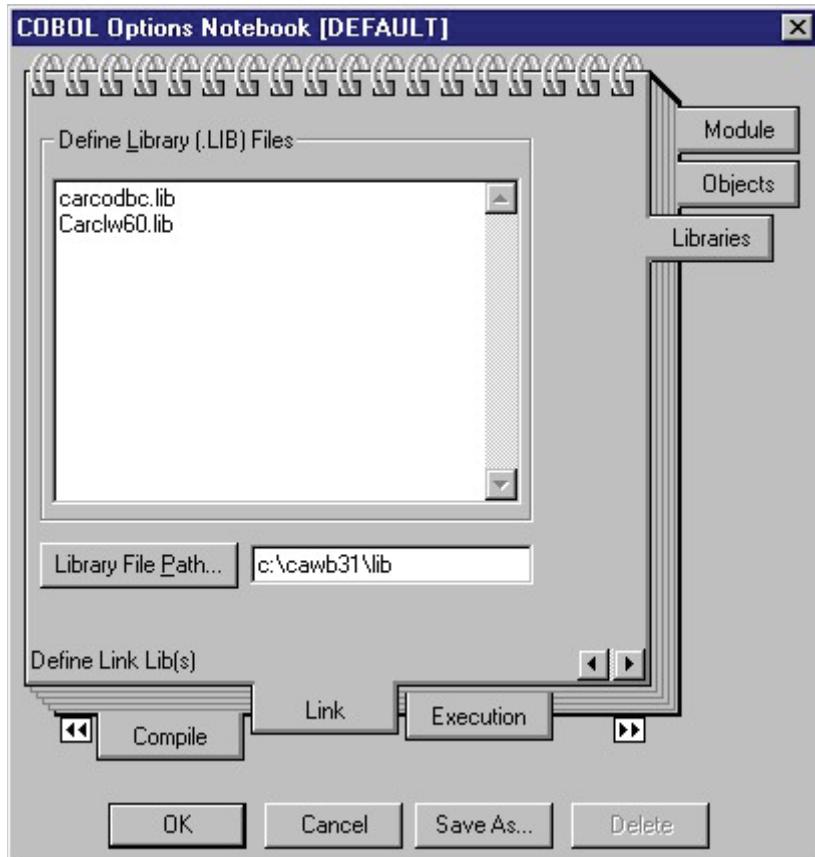


**Figure 55**

In the Link tab in COBOL Options Notebook mention the following library files :

Carcodbc.lib  
Carclw60.lib

Specify the Library File Path as C:\CAWB31\LIB.

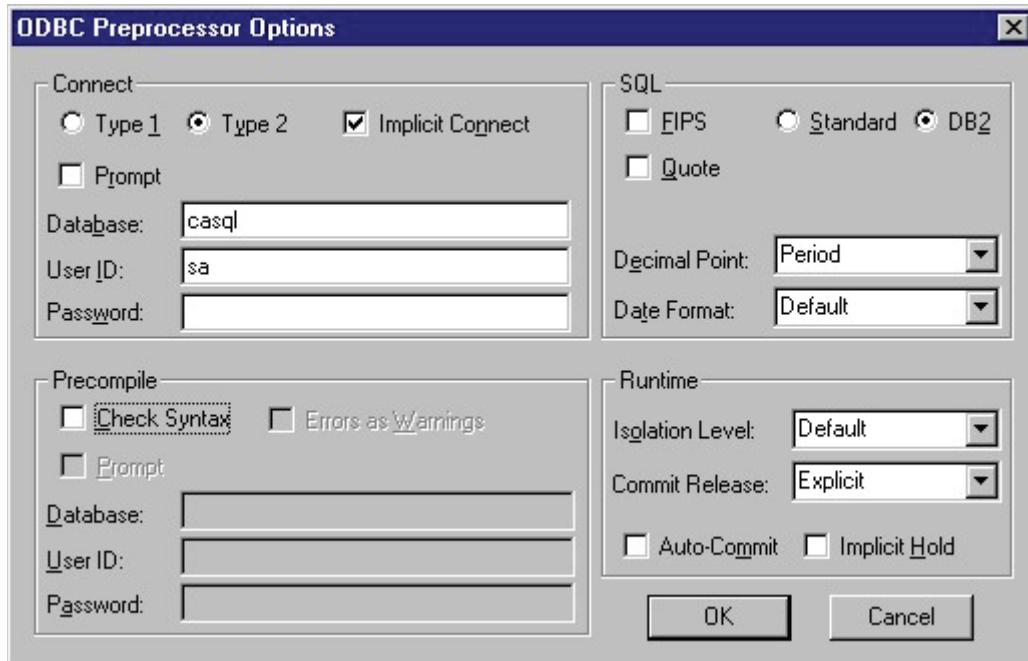


**Figure 56**

In the Preprocessor tab, select the Preprocessor Group as ODBC. ODBC appears in the list of Associated steps. Click on the ODBC step in the associated steps group. The Options and View Step... button will be enabled. Click on the Options button.

**Figure 57**

In the Connect group select the connection type as Type 2, Implicit Connect, against Database enter the name of the user-defined DSN which we created earlier.  
In the SQL group select DB2.  
Click on the OK button.



**Figure 58**

### 3. Create table definitions

For the following table definition :

```
(DEPTNO INTEGER NOT NULL,
 DNAME CHAR(10),
 LOCATION VARCHAR(10)
)
```

The COBOL definition would be as follows :

```
01 DCLDEPT.
 10 DEPTNO PIC S9(9) USAGE COMP.
 10 DNAME PIC X(10).
 10 LOCATION.
 49 LOCATION-LEN PIC S9(4) USAGE COMP.
 49 LOCATION-TEXT PIC X(10).
```

Save the COBOL definition as DCLDEPT.cpy in the folder H:\ANNETTE\CA (The path for the copy books).

### 4. Key in the program

Key in the following program and save it as Prg1.cbl in the H:\Annette\CA folder.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRG1.
AUTHOR. ANNETTE.

* Display the name of the department, given the department no.*

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
 EXEC SQL
 INCLUDE DEPTDCL
 END-EXEC.
*
 EXEC SQL
 INCLUDE SQLCA
 END-EXEC.
PROCEDURE DIVISION.
0000-MAIN-PARA.
* Checking a row with salary as null.
 MOVE 10 TO DEPTNO.
 EXEC SQL
 SELECT DNAME
 INTO :DNAME
 FROM DBO.DEPARTMENT
 WHERE DEPTNO = :DEPTNO
 END-EXEC.
 DISPLAY 'DEPTNO : ' DEPTNO.
 DISPLAY 'DNAME: ' DNAME.

STOP RUN.
```

#### 5. Compile the program

To compile the program, select Build -> Compile.

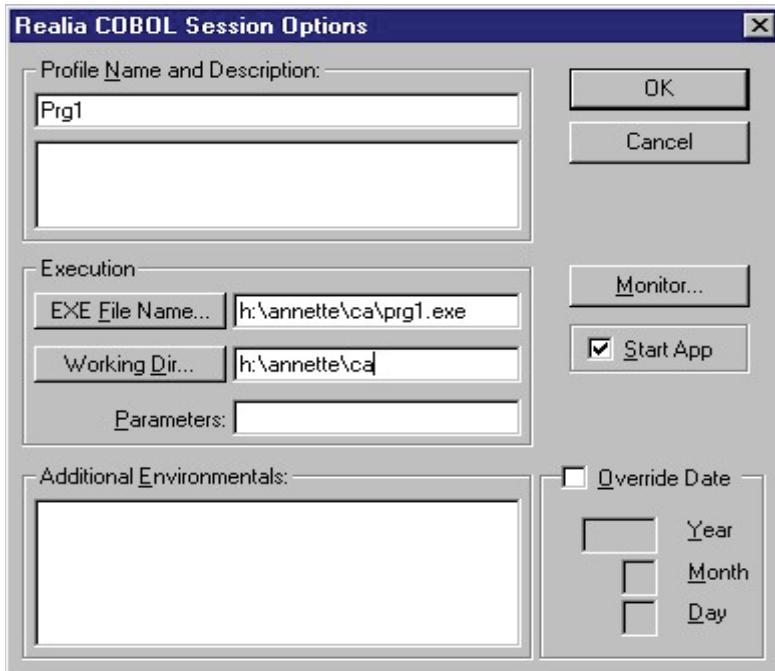
#### 6. Link the program

After successful compilation, select Build -> Link .

Create a new profile for executing the program. Select Build -> Options -> System: DEFAULT ... In the Execution tab,

In the Execution tab of the COBOL Options Notebook, create a new Execution Profile by clicking on the New button.

Enter the profile name. In the Execution group, click on the EXE File Name button and select the name of the executable file. This file name to the appropriate executable file to be run. Select the working directory, the directory which contains the executable files.



#### 7. Debug the program

To check the output of the program, click on the Debug icon. Step through the program, the output of the program is displayed.



A screenshot of a Windows command-line window titled "MS-DOS h:\annette\ca\final\prg1.exe". The window displays the following text:  
DEPTNO : 000000010  
DNAME : ADMIN  
-