DB2 Day 4

Views
TRIGGERS

DB2 – Objects.

Data Objects.
System Objects.

VIEW

- View is like a window through which data and tables can be viewed or changed.
- Derived from another table or another view
- Stored as a SELECT statement only.
- It is a virtual table that does not physically exist in its own right, but appears to the user as if it does.
- Hence, views does not occupy extra memory.
- View is created for SELECT statements.

Types of VIEWS

1. UPDATABLE – If the attributes of the view are from a single table.

2. NON – UPDATABLE – If the attributes of the view are from more than one table.

VIEW

- A view has no data of its own, it manipulates data in the underlying base table
- Allows the same data to be seen by different users in different views.

Syntax:

```
CREATE VIEW  <view name>
AS
SELECT  col1, col2
        FROM   <table name>
        WHERE  <col name> =value;
```

VIEW

Actual table

```
  SELECT * FROM EMPLOYEE;                                                  00292165
--------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   FIRST_NAME  LAST_NAME  DEPTID  PHONENO  HIRTHDATE     SALARY  PROJID  PA
--------+---------+---------+---------+---------+---------+---------+---------+
1001    ARUN        KUMAR      D01     453459   2011-01-15   400000.  PJ001   12
1002    VARUN       CHAND      D01     456769   2008-07-05   200000.  PJ002   23
1004    TIMY        TOMMY      D03     109289   2005-01-29   800000.  PJ004   98
1005    SALY        SAMMY      D04     878769   2009-06-29   560000.  PJ003   23
1003    PERC        PRECY      D02     456512   2005-01-29   800000.  PJ003   98
DSNE610I NUMBER OF ROWS DISPLAYED IS 5
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
--------+---------+---------+---------+---------+---------+---------+---------+
```

VIEW

Creating a view myemployee

```
-- CREATING ANUPDATABLE VIEW FROM EMPLOYEE                          00350099
--*************************************************************     00360099
  CREATE VIEW MYEMPLOYEE AS SELECT FIRST_NAME,SALARY,PASSPORTNO      00370099
  FROM EMPLOYEE WHERE DEPTID ='D01';                                 00371099
--------+--------+--------+--------+--------+--------+--------+--------+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
--------+--------+--------+--------+--------+--------+--------+--------+
```

```
--*************************************************************     00372099
-- SELECTING FROM AN UPDATABLE VIEW FROM EMPLOYEE                   00373099
--*************************************************************     00374099
   SELECT * FROM MYEMPLOYEE;                                        00375099
--------+--------+--------+--------+--------+--------+--------+--------+
FIRST_NAME    SALARY  PASSPORTNO
--------+--------+--------+--------+--------+--------+--------+--------+
ARUN         400000.  12345
VARUN        200000.  23456
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
--------+--------+--------+--------+--------+--------+--------+--------+
```

VIEW

Updating view myemployee

```
--*************************************************************************      00376099
-- UPDATING VIEW MYEMPLOYEE                                                      00377099
--*************************************************************************      00378099
   UPDATE MYEMPLOYEE SET SALARY = 777777 WHERE FIRST_NAME = 'ARUN';              00379099
--------+---------+---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
--------+---------+---------+---------+---------+---------+---------+---------+
```

```
--*************************************************************************      00372099
-- SELECTING FROM MYEMPLOYEE AFTER UPDATION                                      00373099
--*************************************************************************      00374099
   SELECT * FROM MYEMPLOYEE;                                                     00375099
--------+---------+---------+---------+---------+---------+---------+---------+
FIRST_NAME    SALARY  PASSPORTNO
--------+---------+---------+---------+---------+---------+---------+---------+
ARUN         777777.  12345
VARUN        200000.  23456
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
--------+---------+---------+---------+---------+---------+---------+---------+
```

Updated salary 777777

VIEW

Verifying the actual table employee for the updation.

```
--*************************************************************************      00380099
-- SELECTING FROM EMPLOYEE AFTER UPDATING ITS VIEW MYEMPLOYEE                    00390099
--*************************************************************************      00400099
   SELECT FIRST_NAME,SALARY FROM EMPLOYEE;                                       00410099
---------+---------+---------+---------+---------+---------+---------+---------+
FIRST_NAME    SALARY
---------+---------+---------+---------+---------+---------+---------+---------+
ARUN         777777.
VARUN        200000.
TIMY         800000.
SALY         560000.
PERC         800000.
DSNE610I NUMBER OF ROWS DISPLAYED IS 5
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQL
---------+---------+---------+---------+---------+
```

Updated in the actual table EMPLOYEE

TRIGGERS

An SQL **trigger** is a named rule that is associated with a single base table.

- A trigger is a set of actions that will be executed when a defined event occurs.
- Triggers are defined for a specific table and once defined, a trigger is automatically active.
- A table can have multiple triggers defined on it and the order of activation is based on the trigger creation timestamp

TRIGGERS

- Trigger information is stored in the DB2 Catalog Tables called
  - **SYSIBM.SYSTRIGGERS**
  - **SYSIBM.SYSPACKAGE**

Purpose of Triggers

- To do data validation
- To maintain data integrity

TRIGGERS

COMPONENTS OF TRIGGERS

- **Trigger name**
- **Activation time**
- **Events**
- **On table name**
- **Transition Variables**
- **Granularity**
- **Mode**
- **Action**

TRIGGERS

NAME

The trigger can be named with any unique name of user's choice

TRIGGERS

ACTIVATION TIME

This specifies when the trigger should be activated

- BEFORE
- AFTER

TRIGGERS

ACTIVATION TIME - BEFORE

- Will activate THE TRIGGER BEFORE the trigger event occurs.
- The trigger body receives the new data values prior to its trigger event

TRIGGERS

ACTIVATION TIME - AFTER

- Will activate THE TRIGGER AFTER the trigger event occurs.

- The trigger body sees the table as being in a consistent state. (All transactions have been completed)

TRIGGERS

EVENTS

The triggering event can be the following SQL statements

- INSERT
- UPDATE
- DELETE

TRIGGERS

ON TABLE NAME

It specifies the name of the table for which the trigger is created.

Syntax: ON <TABLE NAME>

TRIGGERS

TRANSITION VARIABLES

– USING REFERENCING CLAUSE

It is used to refer to a row (s) that is not a member of a table.

- To refer to a row before inserting.
- To refer to a row after deleting
- To refer to the old values of a row after updating.

TRIGGERS

TRANSITION VARIABLES

– USING REFERENCING CLAUSE

There are two kinds of transition variables

1. REFERENCING OLD AS CORRELATION-NAME
2. REFERENCING NEW AS CORRELATION-NAME

CORRELATION-NAME:

- It can be any name of user's choice.
- It will have the structure of the table for which the trigger is created.
- Its scope is only within the trigger.

TRIGGERS

TRANSITION VARIABLES

1. REFERENCING OLD AS CORRELATION-NAME

Specifies a correlation name which captures the original state of the row, that is, before the triggered action (DELETE or UPDATE) is applied to the database.

TRIGGERS

TRANSITION VARIABLES

1. REFERENCING NEW AS CORRELATION-NAME

Specifies a correlation name which captures the value that is, or was, used to update the row in the database when the triggered action (INSERT or UPDATE) is applied to the database.

TRIGGERS

GRANULARITY

It specifies how many time the activated trigger must run

1. FOR EACH ROW

2. FOR EACH STATEMENT

TRIGGERS

GRANULARITY – FOR EACH ROW

* It runs as many times as the number of rows in the set of affected rows.

* If you need to refer to the specific rows affected by the triggered action, use FOR EACH ROW granularity.

TRIGGERS

GRANULARITY – FOR EACH STATEMENT

* It runs once for the entire trigger event.

If the set of affected rows is empty (that is, in the case of a searched UPDATE or DELETE in which the WHERE clause did not qualify any rows), a FOR EACH ROW trigger does not run. But a FOR EACH STATEMENT trigger still runs once.

TRIGGERS

MODE

It specifies the mode in which the trigger is declared.

MODE DB2SQL

TRIGGERS

ACTION

The activation of a trigger results in the running of its associated triggered action.

Every trigger has exactly one triggered action which, in turn, has two components:

1. An optional **triggered action condition** or WHEN clause
2. A set of **triggered SQL statement(s)**.

TRIGGERS

ACTION

1. TRIGGERED ACTION CONDITION

The triggered action condition defines whether or Not the set of triggered statements are performed for The row or for the statement for which the triggered action is executing.

2. TRIGGERED SQL STATEMENT(S).

The set of triggered statements define the set of actions performed by the trigger in the database as a consequence of its event having occurred.

TRIGGERS

Example

- Now let us define trigger for the following scenario.
- To validate that whenever a new employee joins the company, his/her salary should be > 100000
- It has to be done before the insertion actually happens in the table
- So we need to write an before event trigger

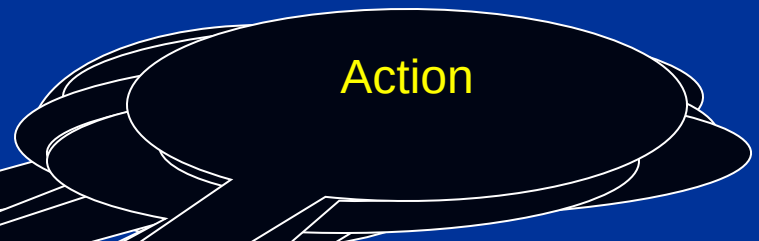
TRIGGERS

Records from EMPLOYEE and DEPT

```
  SELECT * FROM EMPLOYEE;                                                00292165
--------+---------+---------+---------+---------+---------+---------+---------+
EMPNO    FIRST_NAME  LAST_NAME  DEPTID  PHONENO  HIRTHDATE     SALARY  PROJID  PA
--------+---------+---------+---------+---------+---------+---------+---------+
1001     ARUN        KUMAR      D01     453459   2011-01-15   400000.  PJ001   12
1002     VARUN       CHAND      D01     456769   2008-07-05   200000.  PJ002   23
1004     TIMY        TOMMY      D03     109289   2005-01-29   800000.  PJ004   98
1005     SALY        SAMMY      D04     878769   2009-06-29   560000.  PJ003   23
1003     PERC        PRECY      D02     456512   2005-01-29   800000.  PJ003   98
  SELECT * FROM DEPT;                                                    00292069
--------+---------+---------+---------+---------+---------+---------+---------+
DEPTID   DEPTNAME    NO_OF_EMPLOYS  HOD
--------+---------+---------+---------+---------+---------+---------+---------+
D01      TRAINEE              20.   SAM
D02      DEVE                 30.   TOM
D03      MRKT                 10.   TIM
D04      HR                    5.   TAM
DSNE610I NUMBER OF ROWS DISPLAYED IS 4
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
--------+---------+---------+---------+---------+---------+---------+---------+
```

TRIGGERS

Before Trigger Example




```
CREATE TRIGGER SALTRIG
   NO CASCADE BEFORE INSERT
   ON EMPLOYEE
      REFERENCING NEW AS N
      FOR EACH ROW MODE DB2SQL
      WHEN (N.SALARY<100000)
      SIGNAL SQLSTATE '75001'
        ('SALARY SHOULD BE GREATER THAN 10000');
```

TRIGGERS

example

```
--*************************************************************************
-- TRIGGER
--*************************************************************************
  CREATE TRIGGER SALTRIG
        NO CASCADE BEFORE INSERT
        ON EMPLOYEE
           REFERENCING NEW AS N
           FOR EACH ROW
           MODE DB2SQL
           WHEN(N.SALARY<100000)
                SIGNAL SQLSTATE '75001'
                ('SALRY SHOULD BE MORE THAN 100000')
--------+---------+---------+---------+---------+---------+---------+---------+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
--------+---------+---------+---------+---------+---------+---------+---------+

--*************************************************************************
-- INSERTING EMPLOYEE THAT VIOLATES THE SALARY CONSTRAINT IN TRIGGEER
--*************************************************************************
--*************************************************************************
  INSERT INTO EMPLOYEE VALUES('100009','ALIN','SHAH','D04','423459',
  '2011-05-15',40000,'PJ001','00345');
--------+---------+---------+---------+---------+---------+---------+---------+
DSNT408I SQLCODE = -438, ERROR:  APPLICATION RAISED ERROR WITH DIAGNOSTIC TEXT:
         SALRY SHOULD BE MORE THAN 100000
```

Displays what we have asked to display in the trigger

TRIGGERS

Example for After Trigger

- Once the insertion is successful the <NoofEmp> in the DEPT table should be increased depending on the <Deptno> for which the employee is added
- We write an AFTER Trigger

TRIGGERS

AFTER Trigger Example


```
CREATE TRIGGER UPDATE_DEPT
   AFTER INSERT
   ON EMPLOYEE
        REFERENCING NEW AS P
        FOR EACH ROW MODE DB2SQL
         UPDATE DEPT
         SET NO_OF_EMPLOYS = NO_OF_EMPLOYS +1
         WHERE DEPT.DEPTID=P.DEPTID;
```

TRIGGERS

EXAMPLE

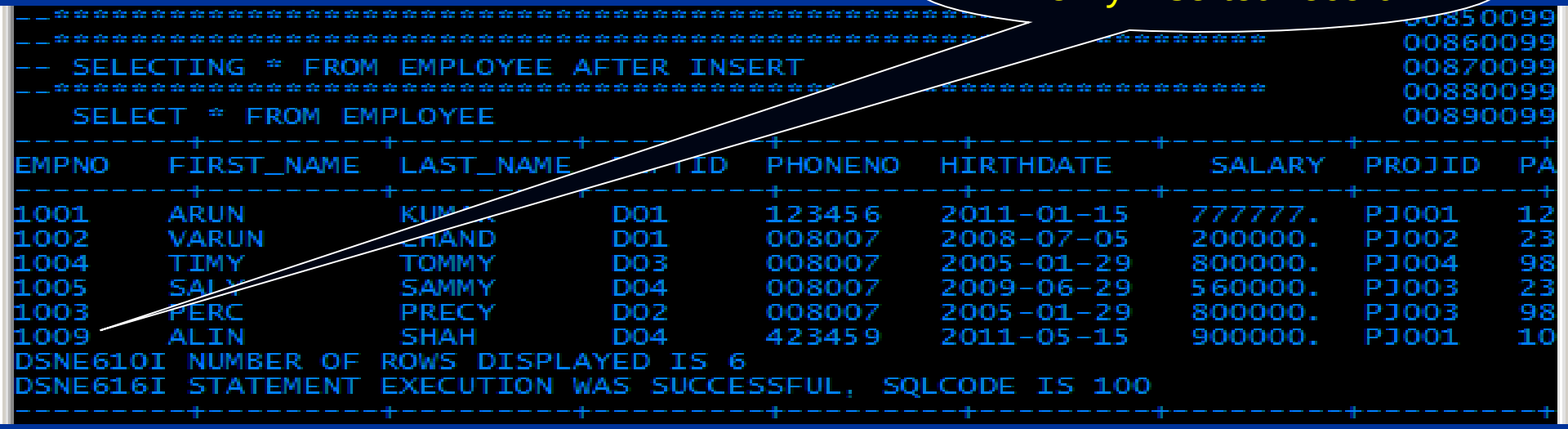
```
--*************************************************************************      00550099
-- TRIGGER - AFTER INSERT                                                       00560099
--*************************************************************************      00570099
   CREATE TRIGGER UPDTRIG                                                       00580099
         AFTER INSERT                                                           00590099
         ON EMPLOYEE                                                            00600099
            REFERENCING NEW AS N                                                00610099
            FOR EACH ROW MODE DB2SQL                                            00620099
          UPDATE DEPT SET NO_OF_EMPLOYS = NO_OF_EMPLOYS + 1                     00631199
            WHERE DEPT.DEPTID = N.DEPTID;                                       00640099
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
--*************************************************************************      00670099
```

```
--*************************************************************************      00670099
-- INSERTING EMPLOYEE THAT IS ACCEPTABLE                                        00680099
--*************************************************************************      00690099
--*************************************************************************      00700099
  INSERT INTO EMPLOYEE VALUES('1009','ALIN','SHAH','D04','423459',              00710099
  '2011-05-15',900000,'PJ001','10345');                                         00720099
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
```

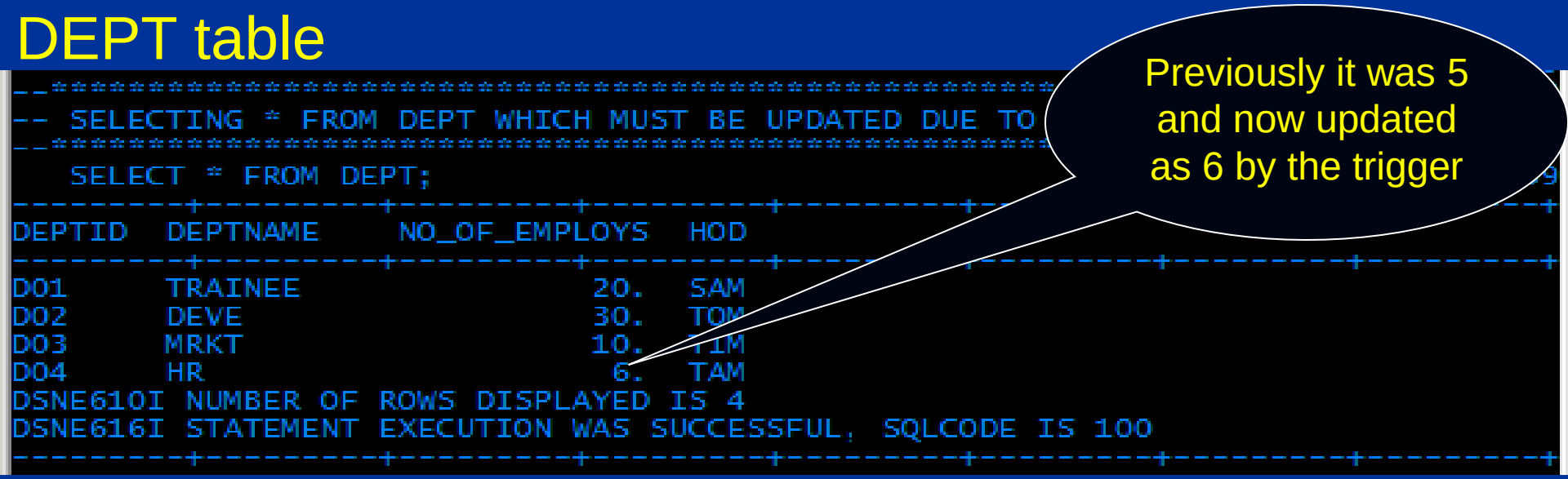
TRIGGERS

Now let us see the content of both the tables

EMPLOYEE table



DEPT table



TRIGGERS

DROP Trigger

- When you no longer want a trigger we can drop the trigger

DROP TRIGGER TRGAIORD

EMBEDDED SQL

Static SQL

Program Preparation.

DB2 objects.
There are 2 objects.

1. System objects - Controlled and used by DB2

2. Data objects - Created, accessed and manipulated by the users, but maintained by DB2

DB2 Objects

1. System Objects

- DB2 Catalog
- DB2 Directory
- Boot Strap Data Set
- Active and Archive Logs
- Buffer Pools
- Locks

2. Data Objects

- Storage Groups
- Databases
- Table spaces
- Tables
- Indexes
- Views

1. SYSTEM OBJECTS
DB2 catalog

- Set of tables containing information about everything defined to the system.
- Used by DB2 to determine access path and manage resources
- Table structure for catalog differs by platform.

SYSTEM OBJECTS
DB2 catalog

- All catalog consists of a group of tables containing information about its objects.
- All catalog tables have a high-level qualifier of **SYSIBM.**
  - SYSCOLUMNS – Contains one row for every column of each table and view.
  - SYSINDEXES – Contains one row for every index.
  - SYSTABLES – Contains one row for each table, view and alias.
  - SYSTABAUTH – Contains the privileges held by user on tables and views
  - SYSRELS – Contains one row for every referential constraint.

SYSTEM OBJECTS
DB2 Directory(DSNDB01)

The DB2 directory contains information used byDB2 during normal operation, this data cannot be accessed using SQL.

It contains information required to start DB2, and their activities and utilities in the DB2 environment.

SYSTEM OBJECTS
BOOT STRAP data sets

Contains inventory of all active and archive log datasets.

During installation of DB2, two BSDS datasets are created, kept in different volumes.

Contains Archive and Active logs

SYSTEM OBJECTS
BOOT STRAP data sets

Active and Archive Logs

- DB2 records all data changes and significant events in active logs as they occur. In case of failure, it uses this data to recover the lost information.
- When the active log is full, DB2 copies the contents of the active log to a DASD or magnetic tape data set called archive logs.

SYSTEM OBJECTS
BUFFER POOLS

Buffer Pools are areas of virtual storage which DB2 uses during execution of an application program or an interface SQL, to temporarily store pages of tablespace

SYSTEM OBJECTS
LOCKS

DB2 guarantees data integrity by using several locking mechanisms. These strategies permits multiple users from multiple environments to access and modify data concurrently without lose of data integrity.

- DB2 supports locking at three levels. Table Space Level, Table Level and Page Level.
- DB2 uses locks to control concurrency and prevent inconsistent data.
- IRLMPROC will take care of the locking services.

SYSTEM OBJECTS
LOCKS

- Why Locking ?

  'Locking is used to provide multiple user access to the same system'.

- How does DB2 manage locking ?

  DB2 uses locking services provided by an MVS subsystem called the IMS Resource Lock Manager(IRLM).

  The above is based on Transaction Processing - the systemcomponent that provides this is 'TRANSACTION MANAGER'

  COMMIT & ROLLBACK are key methods of implementing this.

SYSTEM OBJECTS
LOCKS - explicit

Explicit locking can be achieved on a table using four methods.

1. The SQL statement **LOCK TABLE.**

2. The **ISOLATION** parameter on the BIND PACKAGE command - the two possible values are
   RR('Repeatable Read')
   CS('Cursor Stability').
   CS is the value specified if the application program is used in an online environment.

SYSTEM OBJECTS
LOCKS - explicit

3. The tablespace **LOCKSIZE** parameter - physically DB2 locks data in terms of pages or tables or tablespaces. This parameter is specified in ‘CREATE or ALTER Tablespace’ option ‘LOCKSIZE’. The options are ‘Tablespace’, ‘Table’, ‘Page’ or ‘Any’.

4. The **ACQUIRE/RELEASE** parameters on the BIND PLAN command specifies when table locks(which are implicitly acquired by DB2) are to be acquired and released.

Types :

- ACQUIRE
  - Use
  - Allocate
- RELEASE
  - Commit
  - Deallocate

SYSTEM OBJECTS
LOCKS - explicit

Attributes of LOCKS

- There 3 main attributes of Lock:
  1. MODE.
  2. SIZE.
  3. DURATION.

SYSTEM OBJECTS
LOCKS - explicit

Attributes 1. MODE

It Specifies what access to the locked object is permitted to the lock owner and to any concurrent application process.

3 MODES

a. EXCLUSIVE(X)
b. UPDATE(U)
c. SHARE(S)

SYSTEM OBJECTS
LOCKS - explicit

Attributes 1. MODE

(a) EXCLUSIVE (X)

- The lock owner can read or change the locked page.

- Concurrent Processes cannot acquire ANY lock on the page nor they access the locked page.

SYSTEM OBJECTS
LOCKS - explicit

Attributes 1. MODE

(b) UPDATE(U)

- The lock owner can read but cannot change the locked page. However the owner can promote the lock to ‘X’ and update.

SYSTEM OBJECTS
LOCKS - explicit

Attributes 1. MODE

(c) SHARED(S)

- The lock owner and any concurrent processes can read but not change the locked page.

SYSTEM OBJECTS LOCKS - explicit

Attributes 2. SIZE

Db2 support locking at three levels:

- Table Space level locking
- Table level locking
- Page level locking

- Locks cannot be taken at a lower level, without compatible higher-level lock is also being taken.

SYSTEM OBJECTS
LOCKS - explicit

Attributes 3.DURATION

- It is the length of time the lock is held.

- It varies according to when the lock is acquired and released.

SYSTEM OBJECTS
LOCKS ISSUES

- Lock Suspension :

  - IRLM suspends an application process if it requests a lock on an object that is already owned by another application process and cannot be shared.

- Deadlock :

  - When 2 or more transactions are in simultaneous wait stage, each waiting for one of others to release a lock before it can proceed, DEADLOCK occurs.

2.DATA OBJECTS
Storage Group

- It's a set of volume on single DASD.
- The volumes hold the datasets in which table spaces and indexes are actually stored.

```
Create  stogroup  MAPLESG
                volumes  (SMS001,SMS002)
                vcat DSN610;
```

DATA OBJECTS
Database

- A collection of logically related objects - like Tablespaces, Index spaces, Tables etc.
- Not a physical kind of object - may occupy more than one disk space
- A STOGROUP & BUFFERPOOL (is buffer area used to maintain recently accessed table and index pages) must be defined for each database.
- Stogroup and user-defined VSAM are the two storage allocations for a DB2 dataset definition.

DATA OBJECTS
Database

- In a given database, all the spaces need not have the same stogroup
- More than one volume can be defined in a stogroup.
- DB2 keeps track of which volume was defined first & uses that volume.

```
Create database  MAPLEDB
        stogroup  MAPLESG
        buffer pool  BP01;
```

DATA OBJECTS
Tablespace

- Logical address space on secondary storage to hold one or more tables
- A ‘SPACE’ is basically an extendable collection of pages with each **page** of size 4K or 32K bytes. (K=1024)
- It is the storage unit for recovery and reorganizing purpose
- Three Type of Tablespaces - **Simple, Partitioned & Segmented**

DATA OBJECTS
Tablespace - Simple

- Can contain more than one table
- Usually only one is preferred.
- Depending on application, storing more than one Table might enable faster retrieval for joins using these tables.
- Can consider for highly related tables
- Table Rows Inserted and deleted together
- Table Rows accessed together

DATA OBJECTS
Tablespace - Segmented

- The table space is divided into equal segments.(4 to 64 KB)
- Each Segment can have only one table.
- Minimize the table space scan time.
- Good for small table.
- A ‘Segment’ consists of a logically contiguous set of ‘n’ pages. (4 – 64).
- **Segsize** parameter decides the allocation size for the table space.

DATA OBJECTS
Tablespace - Segmented

- No segment is allowed to contain records for more than one table.
- Sequential access to a particular table is more efficient.
- Mass Delete is much more efficient than in any other Tablespace.
- Reorganizing the tablespace will restore every table to its clustered order.
- Lock Table on table locks only the table, not the entire tablespace.

DATA OBJECTS
Tablespace - Partitioned

- The table space is divided into equal partitions.
- The entire table space can have only one table.
- Utilities can be run on one partition.
- Primarily used for Very large tables.
- Only one table in a partitioned TS; 2 to 64 partitions/TS .
- **Numpart** parameter specifies the no. of partitions.
- Individual partitions can be independently recovered and reorganized.
- Different partitions can be stored on different storage groups for efficient access.

DATA OBJECTS
TABLES, INDEX AND VIEWS

- Table is a logical structure of storing records.
- Index is an ordered set of points to the data in a DB2 table.
- An INDEX is a DB2 tool, used to locate the row or rows that contain a given value.
- A view is an alternate way of representing the data.