Database Management System

Part-4

11

VIEWS

Unit Structure

11.0 OBJECTIVES:

- Introduction to views
- Data Independence
- Security
- Updates on views
- Comparison between tables and views

1. INTRODUCTION

Definition:

- A view is a virtual table that consists of columns from one or more tables.
- A virtual table is like a table containing fields but it does not contain any data. In run time it contains the data and after that it gets free.
- But table stores the data in database occupy some space.
- Just like table, view contains Rows and Columns which is fully virtual based table.
- **Base Table** -The table on which view is defined is called as Base table.

CREATING A VIEW

This statement is used to create a view.

Syntax:

CREATE VIEW view_name

- The CREATE statement assigns a name to the view and also gives the query which defines the view.
- To create the view one should must have privileges to access all of the base tables on which view is defined.
- The create view can change name of column in view as per requirements.

HORIZONTAL VIEW

A Horizontal view will restrict the user's access to only a few rows of the table.

Example:

Define a view for Sue (employee number 1004) containing only orders placed by customers assigned to her.

```
CREATE VIEW SUEORDERS AS
SELECT *
FROM ORDERS
WHERE CUST IN
(SELECT CUST_NUM FROM CUSTOMERS WHERE
CUST_REP=1004)
```

VERTICAL VIEW

A vertical view restricts a user's access to only certain columns of a table.

Ex:

```
CREATE VIEW EMP_ADDRESS AS
SELECT EMPNO, NAME, ADDR1, ADDR2, CITY
FROM EMPLOYEE
```

ROW/COLUMN SUBSET VIEW.

- Views can be used to restrict a user to access only selected set of rows and columns of a table in a database.
- This view generally helps us to visualize how view can represent the base table.
- This type of view is combination of both horizontal and vertical views.

Ex:

```
CREATE VIEW STUDENTS_PASSED AS
SELECT ROLLNO, NAME, PERCENTAGE
FROM STUDENTS
WHERE RESULT ='PASS'
```

GROUPED VIEW

- A grouped view is one in which query includes GROUPBY CLAUSE.
- It is used to group related rows of data and produce only one result row for each group.

Ex:

Find summary information of Employee Salaries in sales Department.

Defining View

```
CREATE VIEW Summary_Empl_Sal
(
Total_Employees,
Minimum_salary,
Maximum_Salary,
Average_salary,
Total_salary
)
AS
SELECT COUNT(EmpID),
Min(Salary),
Max(Salary),
Avg(Salary),
SUM(Salary),
FROM Employee
GROUP BY Department
HAVING Department='Sales';
```

View Call

```
Selelct *
From Summary_Empl_Sal
```

The above Query will give,
Total No. Of Employees in sales Department, Minimum Salary in sales Department.

Maximum Salary in sales Department.
Average Salary in sales Department.
Total Salary of Employees in sales Department.

JOINED VIEWS

- A Query based on more than one base table is called as Joined View.
- It is also called as Complex View
- This gives a way to simplify multi table queries by joining two or more table query in the view definition that draws its data from multiple tables and presents the query results as a single view.
- The view once it is ready we can retrieve data from multiple tables without joining any table simply by accessing a view created.

Ex: for respective Company database find out all EMPLOYEES DEPARTMENTS.

Schema Definition:

EMPLOYEE-> EmpID, EmpName, Salary, DeptID
DEPARTMENT-> DeptID, DeptName

View Definition

```
CREATE VIEW Emp_Details
As
Select Employee,EmpID,
Department, DeptID,
Department, DeptName
From
Where Employee.DeptID=Department.DeptID;
```

View Call

```
Select * from Emp_Details
```

DROPPING VIEW

When a view is no longer needed, it can be removed by using DROP VIEW statement.

Syntax:

DROP VIEW <VIEW NAME> [CASCADE/RESTRICT]

CASCADE: It deletes the view with all dependent view on original view.

RESTRICT: It deletes the view only if they're in no other view depends on this view.

Example:
Consider that we have view VABC and VPQR .View VPQR depends on VABC.

Query:
DROP view VABC

If we drop VABC, then cascading affect takes place and view VPQR is also dropped.

Thus default option for dropping a view is CASCADE. The CASCADE option tells DBMS to delete not only the named view, but also query views that depend on its definition. But,

QUERY:
DROP view VABC RESTRICT

Here, the query will fail because of RESTRICT option tells DBMS to remove the view only if no other views depend on it. Since VPQR depends on VABC, will cause an error.

UPDATING VIEWS

- Records can be updated, inserted, and deleted though views.
- UPDATAEBLE VIEWS are those in which views are used against INSERT, DELETE and UPDATE statements.

The following conditions must be fulfilled for view updates:

- DISTINCT must not be specified; that is, duplicate rows must not be eliminated from the query results.
- The FROM clause must specify only one updateable table; that is, the view must have a Single source table for which the user has the required privileges. If the source table is itself a view, then that view must meet these criteria.
- Each select item must be a simple column reference; the select list cannot contain expressions, calculated columns, or column functions.
- The WHERE clause must not include a subquery; only simple row-by-row search conditions may appear.
- The query must not include a GROUP BY or a HAVING clause.

Data Independence

A major purpose of a database system is to provide the users with an abstract view of data.

To hide the complexity from users database apply different levels of abstraction. The following are different levels of abstraction.

i. Physical Level
ii. Logical Level
iii. View Level

Physical Level

- Physical Level is the lowest level of abstraction and it defines the storage structure.
- The physical level describes complex low level data structures in detail.
- The database system hides many of the lowest level storage details from the database programmers.
- Database Administrators may be aware of certain details of physical organization of data.

Logical Level

- This is the next higher level of abstraction which describe what data are stored in database, relation between data, types of data etc .
- Database programmers, DBA etc knows the logical structure of data

View Level

- This the highest level of abstraction.
- It provides different view to different users. At the view level users see a set of application programs that hide details of data types.
- The details such as data type etc are not available at this level.
- Only view or Access is given to a part of data according to the users access right

Physical Data Independence

The changes in Physical Level does not affect or visible at the logical level.
This is called physical data independence.

Logical Data Independence

The changes in the logical level do not affect the view level.
This is called logical data independence.

ADVANTAGES OF VIEWS

1.Security Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data.

2.Query simplicity A view can draw data from several different tab present it as a single table, turning multi table queries into single-tab queries against the view.

3.Structural simplicity Views can give a user a personalized view of the database structure, presenting the database as a set of virtual tables that make sense for that user.

4.Insulation from change A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed. Note, however, that the view definition must be updated whenever underlying tables or columns referenced by the view are renamed.

5.Data integrity If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets specified integrity constraints.

DISADVANTAGES OF VIEWS

While views provide substantial advantages, there are also three major disadvantages to using a view instead of a real table:

• **Performance**

Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables.

If the view is defined by a complex multitable query, then even a simple query against the view becomes a complicated join, and it may take a long time to complete.

However, the issue isn't because the query is in a view—any poorly constructed query can present performance problems—the hazard is that the complexity is hidden in the view, and thus users are not aware of how much work the query is performing.

• **Manageability**

Like all database objects, views must be managed. If developers and database users are allowed to freely create views without controls or standards, the DBA's job becomes that much more difficult.

This is especially true when views are created that reference other views, which in turn reference even more views.

The more layers between the base tables and the views, the more difficult it is to resolve problems attributed to the views.

• **Update restrictions**

When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying source tables.

This is possible for simple views, but more complex views cannot be updated; they are
read-only.

COMPARISON BETWEEN TABLES AND VIEWS

VIEWS

- View comprises of Query in view definition.
- Just like table, view contains Rows and columns which is fully virtual based table.
- The fields in a view are fields from one or more real tables in the database.
- When view is called, it does not contain any data. For that, it goes to memory and fetches data from base table and displays it.
- E-g: - An I.T. Faculty requires only I.T. related data of students so we can create view called as **Stud_IT_View** for Faculty as below which will only depicts I.T. data of students to I.T. faculty.
- A virtual table is like a table containing fields but it does not contain any data. In run time it contains the data and after that it gets free. But table stores the data in database occupy some space.

  Stud_IT_View (Student_Id,Student_Name, I.T.)

  We can also add functions like WHERE and JOIN statements to a view and present the data as if the data were coming from one single table.

TABLES

- Table stores the data and database occupies some space in database.
- Tables contain rows and columns, columns representing fields and rows containing data or records.

EX:

Consider a Employee containing following columns,

EMPLOYEE (Emp_ID, EmpName, Designation, Address, Salary)

❖❖❖❖

12

STRUCTURED QUERY LANGUAGE

Unit Structure

12.0 OBJECTIVES:

- Data Definition
- Aggregate Functions
- Null Values
- Nested Sub queries
- Joined relations
- Triggers

12.1 INTRODUCTION

- SQL stands for Structured Query Language
- It lets you access and manipulate databases.
- SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s.
- The first commercial relational database was released by Relational Software (Later called as Oracle).
- SQL is not a case sensitive as it is a keyword based language and each statement begins with a unique keyword.

FEATURES OF SQL

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert ,Update, Delete, records in a database
- SQL can create stored procedures in a database
- SQL can create views in a database

SQL COMMANDS:

- SQL commands are instructions used to communicate with the database to perform specific task that work with data.
- SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users.
- SQL commands are grouped into 2 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. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.

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

DATA DEFINITION LANGUAGE (DDL)

- DDL statements are used to build and modify the objects and structure of tables in database.
- The DDL part of SQL permits database tables to be created or deleted.
- It also defines indexes (keys), specifies links between tables, and imposes constraints between tables.
- The most important DDL statements in SQL are:

- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

a. CREATE COMMAND

This statement used to create Database.

Syntax:

```
CREATE TABLE tablename
(
        column_name data_type attributes…,
        column_name data_type attributes…,
        …
)
```

- Table and column names can't have spaces or be "reserved words" like TABLE, CREATE, etc.

Example:

```
CREATE TABLE Employee
(
        EmpId varchar2(10),
        FirstName char(20),
        LastName char(20),
        Designation char(20),
        City char(20)
);
```

OUTPUT:

Emp_Id	FirstName	LastName	Designation	City

b. ALTER COMMAND:

- This statement is used to make modifications to the table structure.
- This statement is also used to add, delete, or modify columns in an existing table

Syntax:

```
ALTER TABLE table_name
ADD column_name datatype

OR

ALTER TABLE table_name
DROP COLUMN column_name

OR

ALTER TABLE table_name
MODIFY COLUMN column_name
```

Example:

```
ALTER TABLE Employee
ADD DateOfBirth date
```

OUTPUT:

EMP_Id	FirstName	LastName	Designation	City	DateOfBirth
1	Raj	Malhotra	Manager	Mumbai	
2	Henna	Setpal	Executive	Delhi	

DROP COMMAND:

This statement is used to delete a table.

Syntax:

DROP TABLE table_name

Example:

DROP TABLE Employee

DATA MANIPULATION LANGUAGE (DML)

DML is set of commands used to,

- Insert data into table
- Delete data from table
- Update data of table.

EMP_Id	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

a. INSERT

The INSERT statement is used to insert a new row in a table.

Syntax:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

Example:

INSERT INTO Employee VALUES (4,'Nihar)
INSERT INTO Employee VALUES (5,'savita)
INSERT INTO Employee VALUES (6,'Diana)

OUTPUT:

Emp_Id	FirstName
4	Nihar
5	Savita
6	Diana

b. DELETE

The DELETE statement is used to delete records in a table.

Syntax:

DELETE FROM table_name
WHERE some_column=some_value

Example:

EMP_Id	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

```
DELETE FROM Employee
WHERE LastName='Malhotra' AND FirstName='Raj'
```

OUTPUT:

EMP_Id	FirstName	LastName	Designation	City
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

c. UPDATE

The UPDATE statement is used to update records in a table.

Syntax:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Example:

EMP_Id	FirstName	LastName	Designation	City
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore
4	Nihar	Sarkar		

```
UPDATE Employee
SET Designation='CEO, City='Mumbai'
WHERE LastName='Sarkar' AND FirstName='Nihar'
```

OUTPUT:

EMP_Id	FirstName	LastName	Designation	City
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore
4	Nihar	Sarkar	CEO	Mumbai

SQL BASIC QUERIES

a. SELECT CLAUSE

This statement is used for various attributes or columns of a table. SELECT can have 2 options as SELECT ALL OR SELECT DISTINCT, where SELECT ALL is default select all rows from table and SELECT DISTINCT searches for distinct rows of outputs.

Syntax:
SELECT * **FROM** table_name

b. FROM CLAUSE

This clause is used to select a Relation/Table Name in a database.

c. WHERE CLAUSE

This clause is used to put a condition on a query result.

Example:

Ex1: SELECT * FROM Employee

EmpID	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

Ex 2: To select only the distinct values from the column named "City" from the table above.

```
SELECT DISTINCT City
FROM Employee
WHERE City='Mumbai'
```

Output:

EmpID	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai

Aliases

- SQL Aliases are defined for columns and tables.
- Basically aliases are created to make the column selected more readable.

Example:
To select the first name of all the students, the query would be like:

Aliases for columns:

SELECT FirstName AS Name FROM Employee;
or
SELECT FirstName Name FROM Employee;

In the above query, the column FirstName is given a alias as 'name'.
So when the result is displayed the column name appears as 'Name' instead of 'FirstName'.

Output:

Name
Raj
Henna
Aishwarya
Nihar

Aliases for tables:

SELECT e.FirstName FROM Employee e;

In the above query, alias 'e' is defined for the table Employee and the column FirstName is selected from the table.

- Aliases is more useful when
- There are more than one tables involved in a query,
- Functions are used in the query,
- The column names are big or not readable,
- More than one columns are combined together

SQL ORDER BY

The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order. Oracle sorts query results in ascending order by default.

Syntax

SELECT column-list
FROM table_name [WHERE condition]
[ORDER BY column1 [, column2, .. columnN] [DESC]];

Database table "Employee";

EmpID	Name	LastName	Designation	Salary	City
1	Raj	Malhotra	Manager	56000	Mumbai
2	Henna	Setpal	Executive	25000	Delhi
3	Aishwarya	Rai	Trainee	20000	Indore

Example:

To select the entire Employee from the table above, however, we want to sort the empoyee by their last name.

```
SELECT * FROM Employee
ORDER BY LastName
```

Output:

EmpID	Name	LastName	Designation	Sa
1	Raj	Malhotra	Manager	56
2	Henna	Setpal	Executive	25
3	Aishwarya	Rai	Trainee	20

ORDER BY DESC Clause

Using ORDER BY clause of a SELECT statement.

Example:
To select all the Employee from the table above, however, we want to sort the empoyee descending by their last name.

```
SELECT * FROM Employee
ORDER BY LastName DESC
```

OUTPUT:

EmpID	Name	LastName	Designation	Salary	City
3	Aishwarya	Rai	Trainee	20000	Indore
2	Henna	Setpal	Executive	25000	Delhi
1	Raj	Malhotra	Manager	56000	Mumbai

AGGREGATE FUNCTIONS

SQL aggregate functions return a single value, calculated from values in a column.

Aggregate functions in SQL are as follows:

- **AVG()** – This functions returns the average value
- **COUNT() -** This functions returns the number of rows
- **MAX() -** This functions returns the largest value
- **MIN() -** This functions returns the smallest value
- **SUM()** - This functions returns the sum

Example

StudID	Name	Marks
1	Rahul	90
2	Savita	90
3	Diana	80
4	Heena	99
5	Jyotika	89
6	Rubi	88

AVG() Function

The AVG() function returns the average value of a numeric column.

This function first calculates sum of column and then divide by total number of rows.

Syntax:

SELECT AVG(column_name) FROM table_name

Example:

Find average Marks of Students from above table.

SELECT AVG(Marks) AS AvgMarks FROM Employees

The result-set will look like this:

AvgMarks
89.3

COUNT() Function

The COUNT() function returns the number of rows that matches a specified criteria.

Syntax:

SELECT COUNT(column_name) FROM table_name

Example

SELECT COUNT(StudID) AS Count FROM Students

Count
6

SUM() Function

The SUM() function returns the total sum of a numeric column.

Syntax

SELECT SUM(column_name) FROM table_name

Example

Find total of marks scored by students.

Select SUM (Marks) as Sum from Students

OutPut:

SUM
536

MIN() Function

The MIN() function returns the smallest value of the selected column.

Syntax

SELECT MIN(column_name) FROM table_name

Example

Find minimum scored by students

Select MIN(Marks) as Min from Students

Min
80

MAX() Function

The MAX() function returns the largest value of the selected column.

Syntax
SELECT MAX(column_name) FROM table_name

Example

Find maximum scored by students

Select MAX(Marks) as Max from Students

Max
90

NESTED SUB-QUERIES

- A query within a query is called Sub-Query.
- Subquery or Inner query or Nested query is a query in a query.
- Sub query in **WHERE Clause (<>,⇔, =, <>)**: It is used to select some rows from main query.
- Sub query in **HAVING Clause (IN/ANY/ALL)**: It is used to select some groups from main querySubqueries can be used with the following sql statements along with the comparison operators like =, <, >, >=, <= etc.

SYNTAX:

SELECT select_Item
FROM table_name
WHERE expr_Operator**(SELECT** select_item
FROM Table_name**)**

Expression operator can be of 2 types:

1. **Single Row Operator**
2. **Multiple-row Operator**

Single Row Operator

A single-row subquery is one where the subquery returns only one value. In such a subquery you must use a single-row operator such as:

Operator	Description
=	Equal To
<>	Not Equal To
>	Greater Than
>=	Greater Than Equal To
<	Less Than
<=	Less Than Equal To

The single-row operators are used to write single-row subqueries. The table below demonstrates the use of the single-row operators in writing single-row subqueries.

Operator	Query	Example
=	Retreive the details of employees who get the same salary as the employee whose ID is 101.	SELECT * FROM EMPLOYEES WHERE SALARY=(SELECT SALARY FROM EMPLOYEES WHERE EMPLOYEE_ID=101);
<>	Retreive the details of departments that are not located in the same location ID as department 10.	SELECT * ROM DEPARTMENTS WHERE LOCATION_ID <>(SELECT LOCATION_ID FROM DEPARTMENTS WHERE EPARTMENT_ID=10);
>	Retrieve the details of employees whose salary is greater than the minimum salary.	SELECT *FROM EMPLOYEES WHERE SALARY > (SELECT MIN(SALARY) FROM EMPLOYEES);
>=	Retrieve the details of employees who were hired on or after the same date that employee 201 was hired.	SELECT * FROM EMPLOYEES WHERE HIRE_DATE >= (SELECT HIRE_DATE FROM EMPLOYEES WHERE EMPLOYEE_ID=201);
<	Retrieve the details of employees whose salary is less than the maximum salary of employees in department 20.	SELECT * FROM EMPLOYEES WHERE SALARY < (SELECT MAX(SALARY) FROM EMPLOYEES WHERE DEPARTMENT_ID=20);
<=	Retrieve the details of employees who were hired on or before the same date that employee 201 were hired.	SELECT * FROM EMPLOYEES WHERE HIRE_DATE <=(SELECT HIRE_DATE FROM EMPLOYEES WHERE EMPLOYEE_ID=201);

A multiple row subquery is one where the subquery may return more than one value. In such type of subquery, it is necessary to use a multiple-row operator

The table below describes the multiple-row operators that can be used when writing multiple-row subqueries:

Operator	Meaning
IN	Equal to any value returned by the subquery
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

The multiple-row operators are used to write multiple-row subqueries. The table below demonstrates the use of the multiple-row operators in writing multiple-row subqueries.

Operator	Query	Example
IN	Retreive the department ID, department name and location ID of departments that are located in the same location ID as a location in the UK.	SELECT DEPARTMENT_ID, DEPARTMENT_NAME, LOCATION_ID FROM DEPARTMENTS WHERE LOCATION_ID IN (SELECT LOCATION_ID FROM LOCATIONS WHERE COUNTRY_ID='UK')
>ALL (Greater than the maximum returned by the subquery)	Retrieve the first name of employees whose salary is greater than the all the salaries of employees belonging to department 20.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=20)
<ALL (Less than the least value returned by the subquery)	Retrieve the first name of employees whose salary is less than all the salaries of employees belonging to department 20.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY < ALL (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=20)
>ANY (Greater than the minimum value returned by the subquery)	Retrieve the first name of employees whose salary is greater than the minimum salary of employees in department 60.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY > ANY (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=60)

<ANY (Less than the maximum value returned by the subquery)	Retrieve the first name of employees whose salary is less than the maximum salary employees department 60.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY < ANY of (SELECT SALARY in FROM EMPLOYEES WHERE DEPARTMENT_ID=10)

EXISTS CLAUSE

- Exist Clause specifies a sub query to test for the existence of rows.
- Their results type is in BOOLEAN format.
- It Returns TRUE if a sub query contains any rows

Example:

```
SELECT *
FROM suppliers
WHERE EXISTS
  (select *
    from orders
    where suppliers.supplier_id = orders.supplier_id);
```

This select statement will return all records from the suppliers table where there is at least one record in the orders table with the same supplier_id.

NOT EXISTS CLAUSE

- The EXISTS condition can also be combined with the NOT operator.

Example:

```
SELECT *
FROM suppliers
WHERE not exists (select * from orders Where
suppliers.supplier_id = orders.supplier_id);
```

This will return all records from the suppliers table where there are **no** records in the *orders* table for the given supplier_id.

NULL VALUES

- NULL values represent missing unknown data.
- By default, a table column can hold NULL values.

- If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.
- NULL values are treated differently from other values.
- NULL is used as a placeholder for unknown or inapplicable values.

"Employee" table:

EmpId	FirstName	LastName	Address	City
1	Hussain	Lakdhwala		Santacruz
2	Elie	Sen	Juhu Road	Santacruz
3	Ranbir	Kapoor		Bhayander

Suppose that the "Address" column in the "Employee" table is optional.

This means that if we insert a record with no value for the "Address" column, the "Address" column will be saved with a NULL value.

IS NULL VALUES

How do we select only the records with NULL values in the "Address" column?
We will have to use the IS NULL operator:

SELECT FirstName, LastName, Address **FROM** Employee **WHERE** Address **IS NULL**

Output:

FirstName	LastName	Address
Hussain	Lakdhwala	
Ranbir	Kapoor	

IS NOT NULL VALUES

How do we select only the records with no NULL values in the "Address" column?
We will have to use the **IS NOT NULL** operator:

SELECT LastName,FirstName,Address **FROM** Empoyee **WHERE** Address **IS NOT NULL**

Output:

FirstName	LastName	Address
Elie	Sen	Juhu Road

JOINS

- Joins are used to relate information in different tables.
- A Join condition is a part of the sql query that retrieves rows from two or more tables.
- A SQL Join condition is used in the SQL WHERE Clause of select, update, delete statements.

Syntax for joining two tables is:

```
SELECT col1, col2, col3...
FROM table_name1, table_name2
WHERE table_name1.col2 = table_name2.col1;
```

If a sql join condition is omitted or if it is invalid the join operation will result in a Cartesian product. The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined.

Example:
If the first table has 20 rows and the second table has 10 rows, the result will be 20 * 10, or 200 rows.
This query takes a long time to execute.

Let us use the below two tables to explain the sql join conditions.

Database table "product";

Product_id	Product_name	Supplier_name	Unit_price
100	Camera	Nikon	300
101	Television	LG	100
102	Refrigerator	Videocon	150
103	IPod	Apple	75
104	Mobile	Nokia	50

Database table "order_items";

order_id	product_id	total_units	customer
5100	104	30	Infosys
5101	102	5	Satyam
5102	103	25	Wipro
5103	101	10	TCS

Joins can be classified into Equi join and Non Equi join.

1) **SQL Equi joins**
2) **SQL Non equi joins**

1) SQL Equi joins

- It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.

Example:

We can get Information about a customer who purchased a product and the quantity of product.

An equi-join is classified into two categories:
a) SQL Inner Join
b) SQL Outer Join

a) SQL Inner Join:
All the rows returned by the sql query satisfy the sql join condition specified.

Example:
To display the product information for each order the query will be as given below.

Since retrieving the data from two tables, you need to identify the common column between these two tables, which is the product_id.

QUERY:

```
SELECT order_id, product_name, unit_price, supplier_name,
total_units
FROM product, order_items
WHERE order_items.product_id = product.product_id;
```

The columns must be referenced by the table name in the join condition, because product_id is a column in both the tables and needs a way to be identified.

b) SQL Outer Join:

- Outer join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables.
- The syntax differs for different RDBMS implementation.
- Few of them represent the join conditions as" **LEFT OUTER JOIN"** and **"RIGHT OUTER JOIN".**

Example

Display all the product data along with order items data, with null values displayed for order items if a product has no order item.

QUERY

```
SELECT p.product_id, p.product_name, o.order_id,
o.total_units
FROM order_items o, product p
WHERE o.product_id (+) = p.product_id;
```

Output:

Product_id	product_name	order_id	total_units
100	Camera		
101	Television	5103	10
102	Refrigerator	5101	5
103	IPod	5102	25

SQL Self Join:

A Self Join is a type of sql join which is used to join a table to it, particularly when the table has a FOREIGN KEY that references its own PRIMARY KEY.

It is necessary to ensure that the join statement defines an alias for both copies of the table to avoid column ambiguity.

Example

```
SELECT a.sales_person_id, a.name, a.manager_id,
b.sales_person_id, b.name
FROM sales_person a, sales_person b
WHERE a.manager_id = b.sales_person_id;
```

2) SQL Non Equi Join:

A Non Equi Join is a SQL Join whose condition is established using all comparison operators except the equal (=) operator.
Like >=, <=, <, >

Example:

Find the names of students who are not studying either Economics, the sql query would be like, (lets use Employee table defined earlier.)

QUERY:

```
SELECT first_name, last_name, subject
FROM Employee
WHERE subject != 'Economics'
```

Output:

first_name	last_name	subject
Anajali	Bhagwat	Maths
Shekar	Gowda	Maths
Rahul	Sharma	Science
Stephen	Fleming	Science

TRIGGERS

A trigger is an operation that is executed when some kind of event occurs to the database. It can be a data or object change.

Creation of Triggers

- Triggers are created with the CREATE TRIGGER statement.
- This statement specifies that the on which table trigger is defined and on which events trigger will be invoked.
- To drop Trigger one can use DROP TRIGGER statement.

Syntax:

```
CREATE TRIGGER [owner.]trigger_name
ON[owner.] table_name
FOR[INSERT/UPDATE/DELETE] AS
IF UPDATE(column_name)
[{AND/OR} UPDATE(COLUMN_NAME)...]
{ sql_statements }
```

Triggers Types:

a. **Row level Triggers**
b. **Statement Level Triggers**

a. Row Level triggers-

A row level trigger is fired each time the table is affected by the triggering statement.

Example:

- If an **UPDATE** statement updates multiple rows of a table, a row trigger s fired once for each row affected by the update statement.
- A row trigger will not run, if a triggering statement affects no rows.
- If FOR EACH ROW clause is written that means trigger is row level trigger.

b. Statement Level Triggers

A statement level trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even If no rows are affected.

Example:

- If a DELETE statement deletes several rows from a table, a statement level DELETE trigger is fired only once.
- Default when FOR EACH ROW clause is not written in trigger that means trigger is statement level trigger

Rules of Triggers

- Triggers cannot create or modify Database objects using triggers
  - For example, cannot perform “CREATE TABLE… or ALTER TABLE” sql statements under the triggers
- It cannot perform any administrative tasks
  - For example, cannot perform “BACKUP DATABASE…” task under the triggers
- It cannot pass any kind of parameters
- It cannot directly call triggers
- WRITETEXT statements do not allow a trigger

Advantages of Triggers:-

Triggers are useful for auditing data changes or auditing database as well as managing business rules.

Below are some examples:

- Triggers can be used to enforce referential integrity (For example you may not be able to apply foreign keys)
- Can access both new values and old values in the database when going to do any insert, update or delete

Disadvantages of Triggers

- Triggers hide database operations.
- For example when debugging a stored procedure, it's possible to not be aware that a trigger is on a table being checked for data changes
- Executing triggers can affect the performance of a bulk import operation .

Solution for Best Programming Practice

- Do not use triggers unnecessarily, if using triggers use them to resolve a specific situation
- Where possible, replace a trigger operation with a stored procedure or another kind of operation
- Do not write lengthy triggers as they can increase transaction duration; and also reduce the performance of data insert, update and delete operations as the trigger is fired every time the operation occurs.

13

TRANSACTION MANAGEMENT

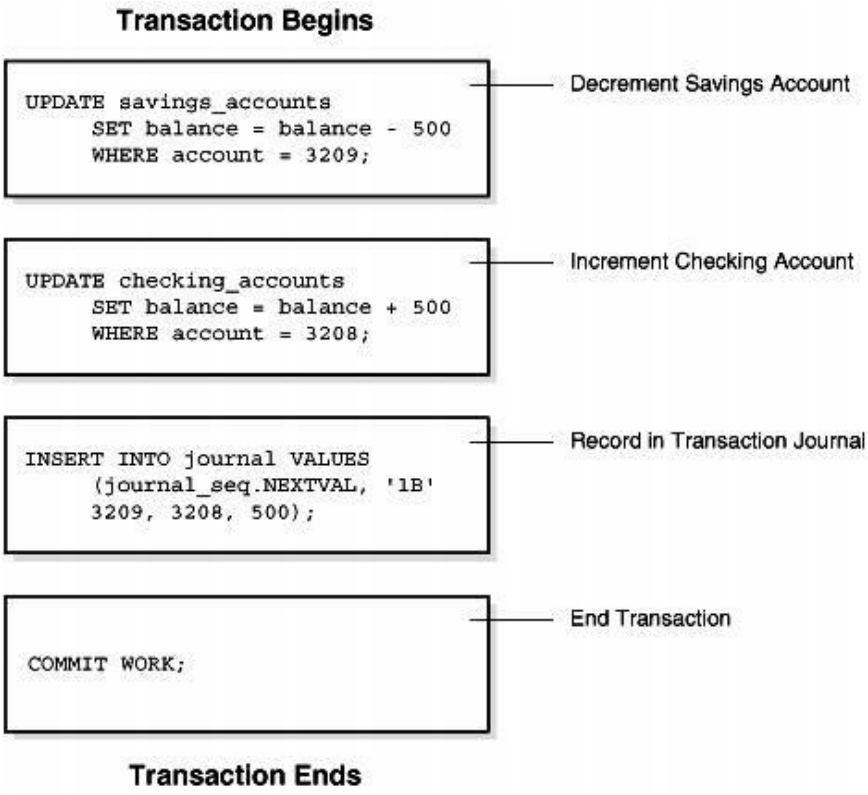
Unit Structure

TRANSACTION

- A **transaction** is a logical unit of work that contains one or more SQL statements. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all **committed** (applied to the database) or all **rolled back** (undone from the database).
- A transaction begins with the first executable SQL statement.
- A transaction ends when it is committed or rolled back, either explicitly with a **COMMIT** or **ROLLBACK** statement or implicitly when a DDL statement is issued.
- To illustrate the concept of a transaction, consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operation:

  i. Decrement the savings account
  ii. Increment the checking account
  iii. Record the transaction in the transaction journal

EXAMPLE:
To illustrate Banking transaction:



PROPERTIES OF TRANSACTION:

Four properties of Transaction: **(ACID PROPERTIES)**

1. Atomicity= all changes are made (commit), or none (rollback).
2. Consistency = transaction won't violate declared system integrity constraints
3. Isolation= results independent of concurrent transactions.
4. Durability= committed changes survive various classes of hardware failure

ATOMICITY

- All-or-nothing, no partial results.
- An event either happens and is committed or fails and is rolled back.
- **EXAMPLE**: In a money transfer, debit one account, credit the other. Either both debiting and crediting.
- If a transaction ends, we say its **commits**, otherwise it **aborts**

- Transactions can be incomplete for three reasons:
  1. It can be **aborted** by the DBMS,
  2. A system crash.
  3. The transaction aborts itself.
- When a transaction does not commit, its partial effects should be undone
- Users can then forget about dealing with incomplete transactions
- But if it is committed it should be durable
- The DBMS uses a **log** to ensure that incomplete transactions can be undone, if necessary.

CONSISTENCY

- If the database is in a consistent state before the execution of the transaction, the database remains consistent after the execution of the transaction.

Example:

Transaction T1 transfers $100 from Account A to Account B. Both Account A and Account B contains $500 each before the transaction.

Transaction T1

Read (A)
A=A-100
Write (A)
Read (B)
B=B+10

Consistency Constraint

Before Transaction execution Sum = A + B
Sum = 500 + 500
Sum = 1000

After Transaction execution Sum = A + B
Sum = 400 + 600
Sum = 1000

Before the execution of transaction and after the execution of transaction SUM must be equal.

ISOLATION

- **Isolation** requires that multiple transactions occurring at the same time not impact each other's execution.
- **Example**, if Joe issues a transaction against a database at the same time that Mary issues a different transaction; both transactions should operate on the database in an isolated manner.
- The database should either perform Joe's entire transaction before executing Mary's or vice-versa.
- This prevents Joe's transaction from reading intermediate data produced as a side effect of part of Mary's transaction that will not eventually be committed to the database.
- Note that the isolation property does not ensure which transaction will execute first, merely that they will not interfere with each other.

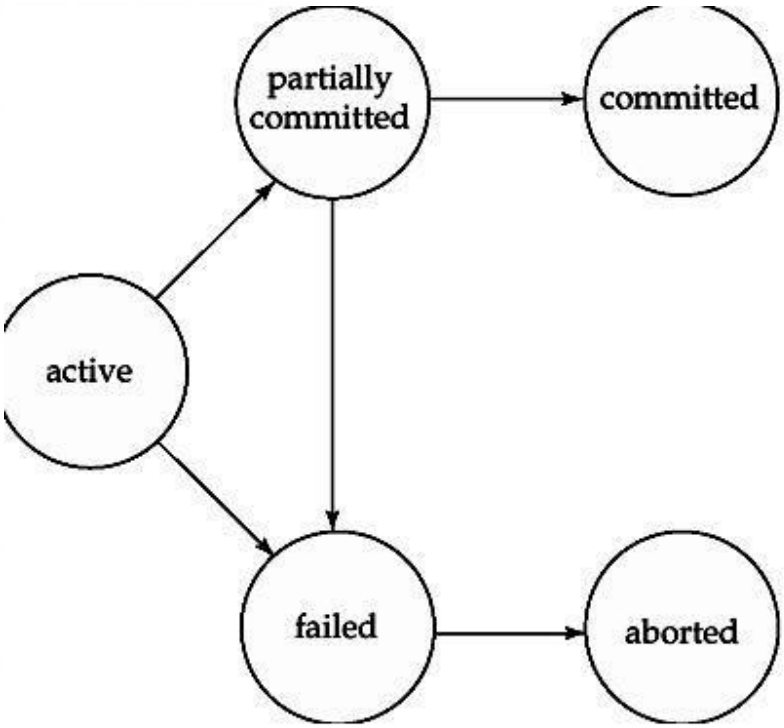
DURABILITY

- **Durability** ensures that any transaction committed to the database will not be lost.
- Durability is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures.

TRANSACTION STATE DIAGRAM

The following are the different states in transaction processing in a Database System.

1. Active
2. Partially Committed
3. Failed
4. Aborted
5. Committed



1. **Active**
   This is the initial state. The transaction stay in this state while it is executing.

2. **Partially Committed**
   This is the state after the final statement of the transaction is executed.

3. **Failed**
   After the discovery that normal execution can no longer proceed.

4. **Aborted**
   The state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

5. **Committed**
   The state after successful completion of the transaction. We cannot abort or rollback a committed transaction.

TRANSACTION SCHEDULE

When multiple transactions are executing concurrently, then the order of execution of operations from the various transactions is known as schedule.

Serial Schedule

Non-Serial Schedule

Serial Schedule

Transactions are executed one by one without any interleaved operations from other transactions.

Non-Serial Schedule

A schedule where the operations from a set of concurrent transactions are interleaved.

SERIALIZABILITY

What is Serializability?

A given non serial schedule of n transactions is serializable if it is equivalent to some serial schedule.

i.e. this non serial schedule produce the same result as of the serial schedule. Then the given non serial schedule is said to be serializable.

A schedule that is not serializable is called a non-serializable.

Non-Serial Schedule Classification

Serializable
Not Serializable
Recoverable
Non Recoverable

Serializable Schedule Classification

Conflict Serializable
View Serializable

Conflict Serializable Schedule

If a schedule S can be transformed into a schedule S' by a series of swaps of non conflicting instruction then we say that S and S' are conflict equivalent.

A schedule S is called conflict serializable if it is conflict equivalent to a serial schedule.

View Serializable Schedule

All conflict serializable schedule are view serializable.

But there are view serializable schedule that are not conflict serializable.

A schedule S is a view serializable if it is view equivalent to a serial schedule.

Recoverable Schedule Classification

Cascade
Cascadeless

To recover from the failure of a transaction Ti, we may have to rollback several transactions.

This phenomenon in which a single transaction failure leads to a series of transaction roll back is called cascading roll back.

Avoid cascading roll back by not allowing reading uncommitted data.

But this lead to a serial schedule.

❖❖❖❖

Thank You