MODULE 3

PART 1 : SQL DML
(DATA MANIPULATION LANGUAGE)

RETREIVAL QUERIES IN SQL

• SQL has one basic statement **for retrieving information from a database**: the **SELECT** statement.

• The SELECT statement is not the same as the SELECT operation of relational algebra

• There are many options to the SELECT statement in SQL

• In simple the SELECT statement is used to select data from a database and the data returned is stored in a result table, called the **result-set**.

SELECT-FROM-WHERE Structure of Basic SQL Queries

• The basic form of the SELECT statement, sometimes called a mapping or a select-from-where block, is formed of the three clauses SELECT, FROM, and WHERE and has the following form:

```
SELECT <attribute list> FROM <table list> WHERE <condition>;
```

Where,

- ■ <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- ■ <table list> is a list of the relation names required to process the query.
- ■ <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

• In SQL,the basic **logical comparison operators** for comparing attribute values with one another and with literal constants are **=, <, <=, >, >=, and <>.**

• These correspond to the relational algebra operators $=, <, \leq, >, \geq$, and $\neq$, respectively

• Eg :

Retrieve the birth date and address of the employee(s) whose name is ‘John B.Smith’.

Soln :

```
SELECT Bdate, Address FROM EMPLOYEE
WHERE Fname=‘John’ AND Minit=‘B’ AND Lname=‘Smith’;
```

• This query involves only the EMPLOYEE relation listed in the FROM clause.

• The query selects the individual EMPLOYEE tuples that satisfy the condition of the WHERE clause, then projects the result on the Bdate and Address attributes listed in the SELECT clause.

• The SELECT clause of SQL specifies the attributes whose values are to be retrieved, which are called the **projection attributes**, and the WHERE clause specifies the Boolean condition that must be true for any retrieved tuple, which is known as the **selection condition**.

Eg 2 :

Retrieve the name and address of all employees who work for the ‘Research’department.

Soln:

```
SELECT Fname, Lname, Address FROM EMPLOYEE, DEPARTMENT
WHERE Dname=‘Research’ AND Dnumber=Dno;
```

- In the WHERE clause of Q1, the condition Dname = ‘Research’is a selection condition that chooses the particular tuple of interest in the DEPARTMENT table,because Dname is an attribute of DEPARTMENT.
- The condition Dnumber = Dno is called a **join condition**, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to the value of Dno in EMPLOYEE.
- A query that involves only selection and join conditions plus projection attributes is known as a **select-project-join query**.

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- In SQL,the same name can be used for two (or more) attributes as long as the attributes are in different relations.
- If this is the case,and a multitable query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name **to prevent ambiguity**.
- This is **done by prefixing the relation name to the attribute name and separating the two by a period**
- Eg**:**

```
SELECT Fname, EMPLOYEE . Name, Address FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT . Name='Research' AND
DEPARTMENT . Dnumber  =  EMPLOYEE . Dnumber;
```

- The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice,as in the following example.
- Eg : For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

Soln :

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn=S.Ssn;
```

- In this case, we are required to declare alternative relation names E and S, **called aliases or tuple variables**, for the EMPLOYEE relation. **An alias can follow the keyword AS.**

• To **retrieve all the attribute values** of the selected tuples, we do not have to list the attribute names explicitly in SQL; we **just specify an asterisk (*),which stands for all the attributes**

• Syntax :

```
SELECT * FROM table_name;
```

• Apart from just using the SELECT keyword individually, you can use the following keywords with the SELECT statement:

DISTINCT

ORDER BY

GROUP BY

HAVING Clause

INTO

SELECT DISTINCT

• This statement is used **to return only different values**.

• Syntax :

```
SELECT DISTINCT Column1, Column2, ...ColumnN
FROM TableName;
```

• Example :

```
SELECT DISTINCT PhoneNumber FROM Employee_Info;
```

ORDER BY

- The ‘ORDER BY’ statement is used **to sort the required results in ascending or descending order.**
- The results are sorted in ascending order by default.
- To get the required results in descending order, you have to use the **DESC** keyword.
- Syntax :

```
SELECT Column1, Column2, ...ColumnN
FROM TableName
ORDER BY Column1, Column2, ... ASC | DESC;
```

- Example :

1. Select all employees from the 'Employee_Info' table sorted by EmergencyContactName:

```
SELECT * FROM Employee_Info
ORDER BY EmergencyContactName;
```

2. Select all employees from the 'Employee_Info' table sorted by EmergencyContactName in Descending order:

```
SELECT * FROM Employee_Info
ORDER BY EmergencyContactName DESC;
```

GROUP BY

- This ‘GROUP BY’ statement is used with the aggregate functions **to group the result-set by one or more columns**.
- Syntax :

```
SELECT Column1, Column2,..., ColumnN
FROM TableName
WHERE Condition
GROUP BY ColumnName(s)
ORDER BY ColumnName(s);
```

- Example :

1. To list the number of employees from each city.

```
SELECT COUNT(EmployeeID), City
FROM Employee_Info
GROUP BY City;
```

HAVING Clause

- The ‘HAVING’ clause is used in SQL because the WHERE keyword cannot be used everywhere.
- Syntax :

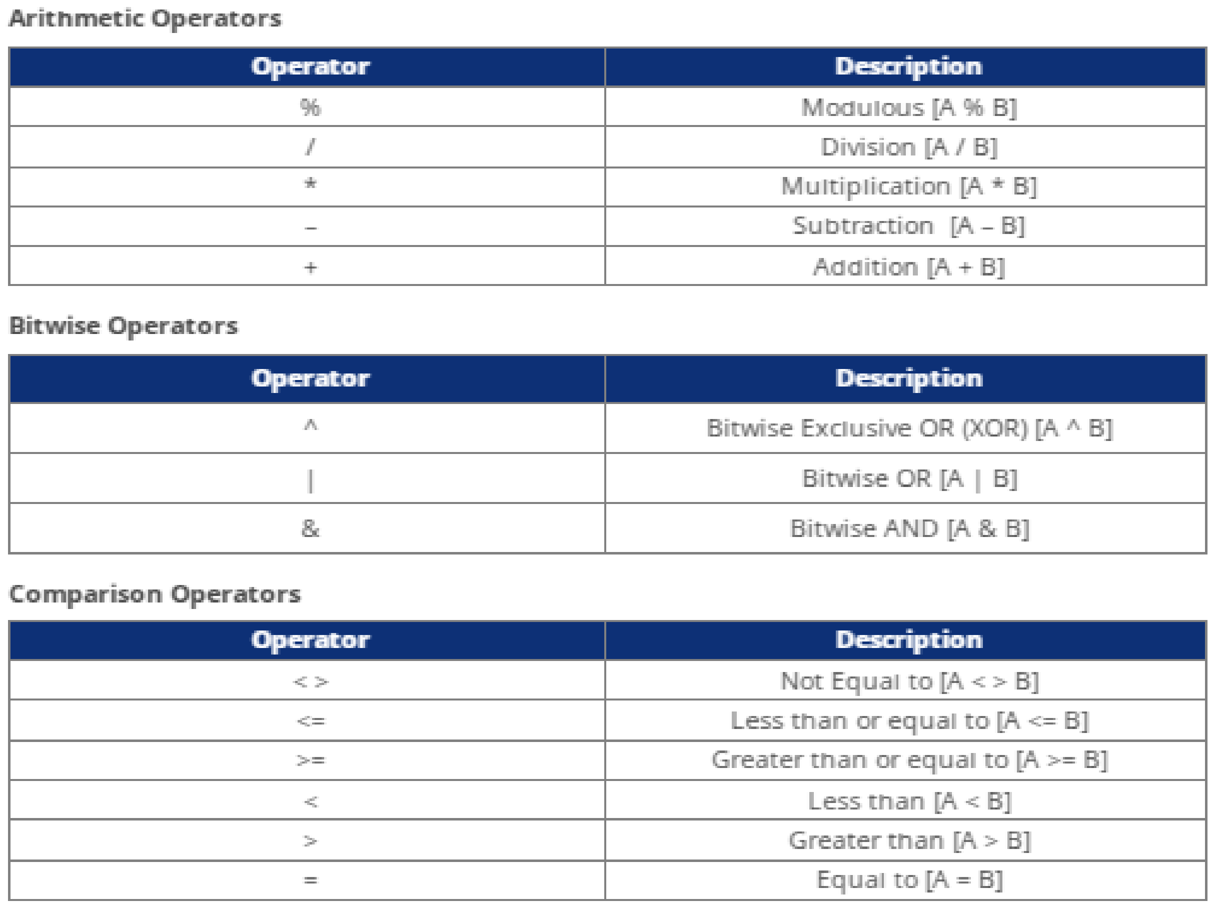
```
SELECT ColumnName(s) FROM TableName
WHERE Condition GROUP BY ColumnName(s)
HAVING Condition ORDER BY ColumnName(s);
```

- To list the number of employees in each city. The employees should be sorted high to low and only those cities must be included who have more than 5 employees:

```
SELECT COUNT(EmployeeID), City
FROM Employee_Info
GROUP BY City
HAVING COUNT(EmployeeID) > 2
ORDER BY COUNT(EmployeeID) DESC;
```

Operators in SQL

• The different set of operators available in SQL are as follows :



Arithmetic Operators

Operator	Description
%	Modulous [A % B]
/	Division [A / B]
*	Multiplication [A * B]
–	Subtraction [A – B]
+	Addition [A + B]

Bitwise Operators

Operator	Description
^	Bitwise Exclusive OR (XOR) [A ^ B]
|	Bitwise OR [A | B]
&	Bitwise AND [A & B]

Comparison Operators

Operator	Description
< >	Not Equal to [A < > B]
<=	Less than or equal to [A <= B]
>=	Greater than or equal to [A >= B]
<	Less than [A < B]
>	Greater than [A > B]
=	Equal to [A = B]

SET Operations are:

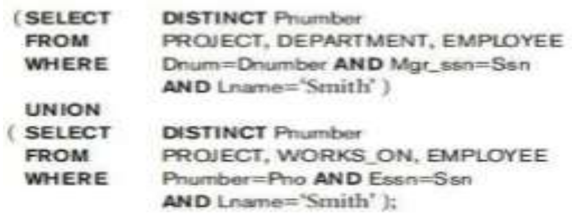
UNION,UNION ALL, INTERSECT, MINUS (EXCEPT)

1. UNION

- It is used to combine result sets of two or more SELECT statements and removes duplicate rows from final result.
- These set operations are apply only to union compatible relations, so we must make sure that the two relations on which we apply the operation have the same attributes and the attributes appear in the same order in both relations
- Syntax :

```
SELECT <column1,…column n > FROM < table1>
UNION
SELECT <column1,…column n > FROM < table2> ;
```

- Example :



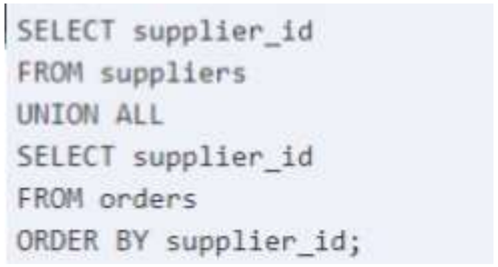
```
( SELECT   DISTINCT Pnumber
  FROM     PROJECT, DEPARTMENT, EMPLOYEE
  WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn
           AND Lname='Smith' )
  UNION
( SELECT   DISTINCT Pnumber
  FROM     PROJECT, WORKS_ON, EMPLOYEE
  WHERE    Pnumber=Pno AND Essn=Ssn
           AND Lname='Smith' );
```

The first SELECT query retrieves the projects that involve a ‘Smith’ as manager of the department that controls the project, and the second retrieves the projects that involve a ‘Smith’ as a worker on the project. Notice that if several employees have the last name ‘Smith’, the project names involving any of them will be retrieved. Applying the UNION operation to the two SELECT queries gives the desired result.

2. UNION ALL

• The UNION ALL command combines the result set of two or more SELECT statements (allowsduplicate values).

• The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the"Suppliers" table:



```
SELECT supplier_id
FROM suppliers
UNION ALL
SELECT supplier_id
FROM orders
ORDER BY supplier_id;
```

This SQL UNION ALL example would return the supplier_id multiple times in the result set if that same value appeared in both the suppliers and orders table. The SQL UNIONALL operator does not remove duplicates. If you wish to remove duplicates, try using the UNION operator.

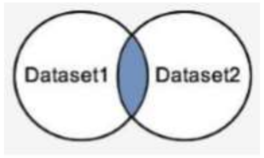
3. INTERSECT

• INTERSECT operator is used to return that are in common between two SELECT statements or data sets.

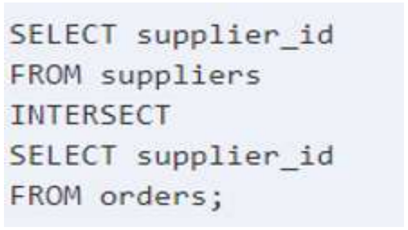
• If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.

• It is the intersection of the two SELECT statements.

• In this operation, the number of datatype and columns must be same



Example :



```
SELECT supplier_id
FROM suppliers
INTERSECT
SELECT supplier_id
FROM orders;
```

In this SQL INTERSECT example, if a supplier_id appeared in both the suppliers and order stable, it would appear in your result set

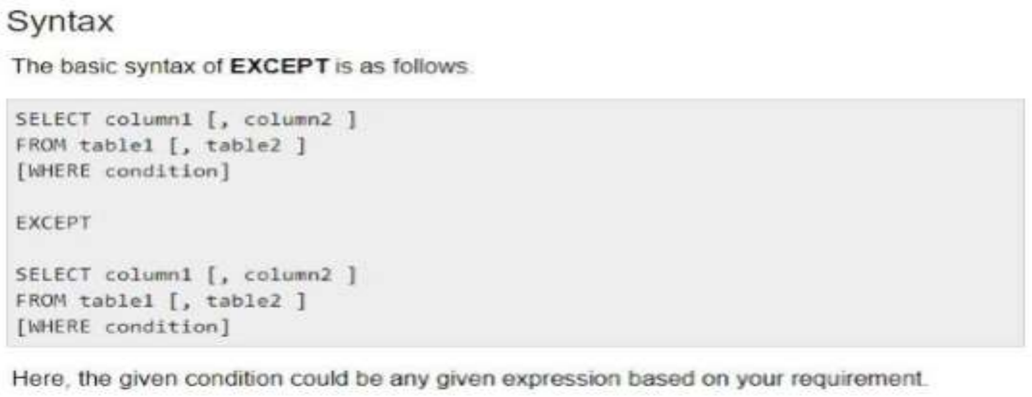
4. EXCEPT (MINUS)

• The SQL EXCEPT clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

• This means EXCEPT returns only rows, which are not available in the second SELECT statement.

• Just as with the UNION operator, the same rules apply when using the EXCEPT operator.

• It has no duplicates and datas are arranged in ascending order by default.



Syntax

The basic syntax of **EXCEPT** is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

EXCEPT

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Logical Operators

• The Logical operators present in SQL are as follows:

1. <u>AND Operator</u>

• This operator is used to filter records that rely on more than one condition.

• This operator displays the records, which satisfy all the conditions separated by AND, and give the output TRUE.

• **Syntax**

```
SELECT < Column1, Column2, ..., Column N > FROM  <TableName >
WHERE  < Condition1 AND Condition2 AND Condition3 ... > ;
```

• **Example**

```
SELECT * FROM Employee_Info
WHERE City='Mumbai' AND City='Hyderabad';
```

2. <u>OR Operator</u>

• This operator displays all those records which satisfy any of the conditions separated by OR and give the output TRUE.

• **Syntax**

```
SELECT < Column1, Column2, ..., Column N > FROM  < TableName >
WHERE  < Condition1 OR Condition2 OR Condition3 ... > ;
```

- **Example**

```
SELECT * FROM Employee_Info
WHERE City='Mumbai' OR City='Hyderabad';
```

3. **NOT Operator**

- The NOT operator is used, when you want to display the records which do not satisfy a condition.
- **Syntax**

```
SELECT < Column1, Column2, ..., Column N > FROM  <Table Name>
WHERE  NOT < Condition > ;
```

- **Example**

```
SELECT * FROM Employee_Info
WHERE  NOT City='Mumbai';
```

4. <u>BETWEEN Operator</u>

- The BETWEEN operator is used, when you want to select values within a given range.
- Since this is an inclusive operator, **both the starting and ending values are considered**.
- **Syntax**

```
SELECT < Column Name (s) > FROM < Table Name >
WHERE < Column Name >  BETWEEN  < Value1 AND Value2 > ;
```

- **Example**

```
SELECT * FROM Employee_Salary
WHERE Salary BETWEEN 40000 AND 50000;
```

5. <u>IN Operator</u>

- This operator is used for multiple OR conditions.
- This allows you to specify multiple values in a WHERE clause.
- **Syntax**

```
SELECT  < Column Name (s) > FROM  < Table Name >
WHERE  < Column Name > IN (Value1,Value2...) ;
```

- **Example**

```
SELECT * FROM Employee_Info
WHERE City IN ('Mumbai', 'Bangalore', 'Hyderabad');
```

6. Substring Pattern Matching

<u>LIKE Operator</u>

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column of a table.
- ie, This can be used for string pattern matching
- There are mainly two wildcards that are used in conjunction with the LIKE operator (or Partial strings are specified using two reserved characters)

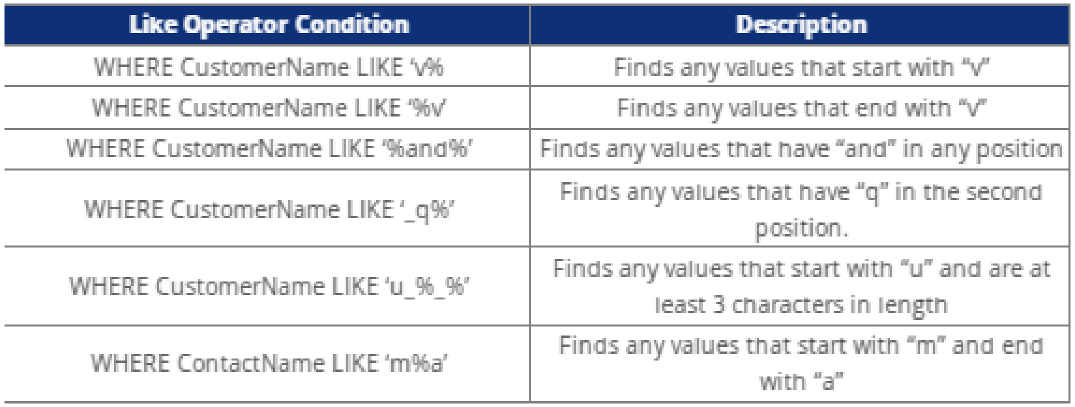
  % – It matches 0 or more character.

  _ – It matches exactly one character.

- **Syntax**

```
SELECT  < Column Name(s) > FROM <Table Name>
WHERE < Column Name>  LIKE  pattern ;
```

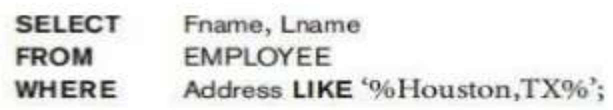
Refer to the following table for the various patterns that you can mention with the LIKE operator.



Like Operator Condition	Description
WHERE CustomerName LIKE 'v%	Finds any values that start with "v"
WHERE CustomerName LIKE '%v'	Finds any values that end with "v"
WHERE CustomerName LIKE '%and%'	Finds any values that have "and" in any position
WHERE CustomerName LIKE '_q%'	Finds any values that have "q" in the second position.
WHERE CustomerName LIKE 'u_%_%'	Finds any values that start with "u" and are at least 3 characters in length
WHERE ContactName LIKE 'm%a'	Finds any values that start with "m" and end with "a"

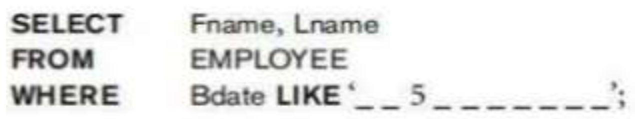
- Examples :

1. Retrieve all employees whose address is in Houston, Texas.



```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     Address LIKE '%Houston,TX%';
```

2. Find all employees who were born during the 1950s



```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';
```

• To retrieve all employees who were born during the 1950s,

• Here, ‘5’ must be the third character of the string (according to our format for date),

• So we use the value ‘_ _ 5 _ _ _ _ _ _ _’, with each under score serving as a placeholder for an arbitrary character.

• If an underscore or % is needed as a literal character in the string, the character should be preceded by an escape character, which is specified after the string using the keyword ESCAPE.

• For example, ‘AB_CD\%EF’ ESCAPE ‘\’ represents the literal

• String ‘AB_CD%EF’ because \ is specified as the escape character.

• Any character not used in the string can be chosen as the escape character.

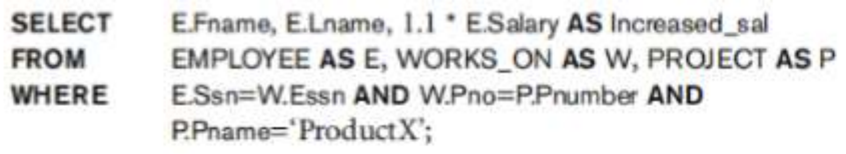
• Also, we need a rule to specify apostrophes or single quotation marks (‘ ’) if they are to be included in a string because they are used to begin and end strings.

• If an apostrophe (’) is needed, it is represented as two consecutive apostrophes (”) so that it will not be interpreted as ending the string.

• The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains

• Example :

1. Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.



```
SELECT   E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM     EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE    E.Ssn=W.Essn AND W.Pno=P.Pnumber AND
         P.Pname='ProductX';
```

• For string data types, the concatenate operator || can be used in a query to append two string values.

• For date, time, time stamp, and interval data types , operators include incrementing (+) or decrementing (–) a date, time, or time stamp by an interval.

• In addition, an interval value is the result of the difference between two date, time ,or timestamp values.

• Another comparison operator ,which can be used for convenience , is BETWEEN

Aggregate Functions in SQL

- Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary.
- Grouping is used to create sub-groups of tuples before summarization.
- A number of built-in aggregate functions exist:

1. **<u>MIN() Function</u>**

- The MIN function returns the smallest value of the selected column in a table.
- **Syntax :**

```
SELECT  MIN < (Column Name) > FROM   < Table Name > WHERE  < Condition > ;
```

- **Example :**

```
SELECT MIN ( EmployeeID ) AS SmallestID  FROM Employee_Info;
```

2. <u>MAX() Function</u>

- The MAX function returns the largest value of the selected column in a table.
- **Syntax :**

```
SELECT  MAX  < (Column Name) >  FROM   < Table Name > WHERE  < Condition > ;
```

- **Example :**

```
SELECT  MAX(Salary)  AS  LargestFees  FROM Employee_Salary;
```

3. COUNT() Function

- The COUNT function returns the number of rows which match the specified criteria.
- **Syntax**

```
SELECT COUNT < (Column Name) > FROM  < TableName > WHERE  < Condition > ;
```

- **Example**

```
SELECT  COUNT (EmployeeID) FROM Employee_Info;
```

4. SUM() Function

- The SUM function returns the total sum of a numeric column that you choose.
- **Syntax**

```
SELECT SUM < (Column Name) > FROM  < Table Name > WHERE  < Condition > ;
```

- **Example**

```
SELECT SUM(Salary) FROM Employee_Salary;
```

5 . AVG() Function

- The AVG function returns the average value of a numeric column that you choose.
- **Syntax**

```
SELECT  AVG < ( Column Name) >  FROM  < Table Name > WHERE < Condition > ;
```

- **Example**

```
SELECT  AVG (Salary) FROM Employee_Salary;
```

- COUNT function returns the number of tuples or values as specified in a query.
- The functions SUM, MAX, MIN,and AVG can be applied to a set or multi set of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values
- These functions can be used in the SELECT clause or in a HAVING clause

<u>NULL Functions</u>

- The NULL functions are those functions which let you return an alternative value if an expression is NULL.
- In the SQL Server, the function is **IS NULL()**.
- **Example**

```
SELECT EmployeeID * (Month_Year_of_Salary  +  IS NULL(Salary, 0))
FROM Employee_Salary ;
```

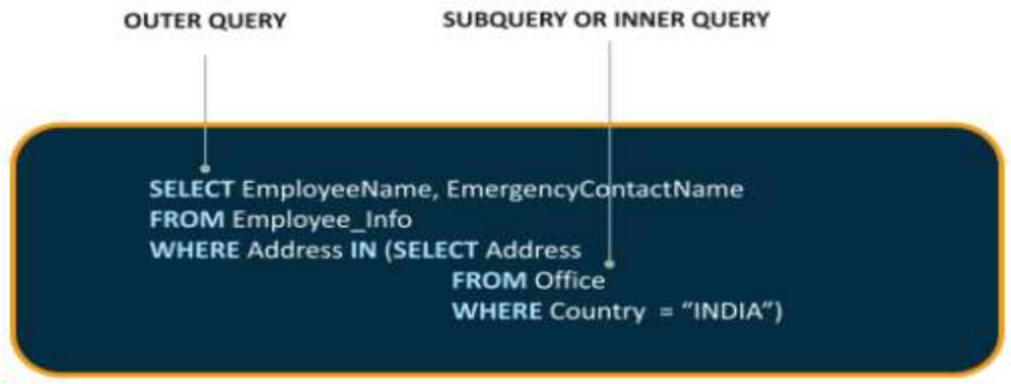
NESTING OF QUERIES

• A complete SELECT query, called a nested query , can be specified within the WHERE-clause of another query, called the outer query.

• **ie, Nested queries** are those queries which have an outer query and inner subquery.

• So, basically, the sub-query is a query which is nested within another query such as SELECT, INSERT,DELETE,UPDATE.

• Refer to the image below:



```
SELECT EmployeeName, EmergencyContactName
FROM Employee_Info
WHERE Address IN (SELECT Address
                  FROM Office
                  WHERE Country  = "INDIA")
```

• Nested queries are useful for breaking down complex problems into smaller and making it easier to retreive specific results from large database

• In nested queries, inner query is executed first and returns set of values that are then used by outer query.

Example :

Retreive the name and address of all employees who works for the “Research” department

```
SELECT   Fname, lname, Address  FROM  Employee  WHERE
Dno IN ( SELECT  Dnumber  FROM  Department  WHERE  Dname=’Research’);
```

• The nested query selects the number of the 'Research' department

• The outer query select an EMPLOYEE tuple if its DNO value is in th result of either nested query

• The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V

• In general, we can have several levels of nested queries

• A reference to an unqualified attribute refers to the relation declared in the inner most nested query

• In this example, the nested query is not correlated with the outer query

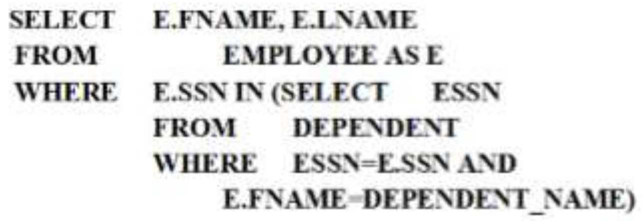
CORRELATED NESTED QUERIES

• If a condition in the WHERE-clause of a nested query references an attribute of a relation declared in the outer query, the two queries are said to be correlated

• The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) the outer query

• Example

Retrieve the name of each employee who has a dependent with the same first name as the employee



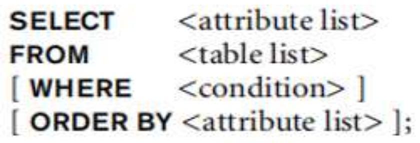
```
SELECT    E.FNAME, E.LNAME
FROM            EMPLOYEE AS E
WHERE     E.SSN IN (SELECT     ESSN
          FROM       DEPENDENT
          WHERE      ESSN=E.SSN AND
                E.FNAME=DEPENDENT_NAME)
```

• Here the nested query has a different result for each tuple in the outer query.

• A query written with nested SELECT... FROM... WHERE...blocks and using the = or IN comparison operators can always be expressed as a single block query.

A Simple Retrieval Query in SQL

- A simple retrieval query in SQL can consist of up to four clauses,
    - ▫ but only the first two SELECT and FROM are mandatory
    - ▫ the clauses between square brackets[...] being optional:



```
SELECT      <attribute list>
FROM        <table list>
[ WHERE     <condition> ]
[ ORDER BY <attribute list> ];
```

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations(or aliases)needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause
- There are three SQL commands to modify the database; INSERT,DELETE, and UPDATE

The EXISTS and UNIQUE Functions in SQL

• EXISTS function in SQL is used to check whether the result of a correlated nested query is empty(contains no tuples) or not

• The result of EXISTS is a Boolean value

▫ TRUE if the nested query result contains at least one tuple, or

▫ FALSE if the nested query result contains no tuples.

• Example :

```
SELECT E.Fname, E.Lname FROM Employee AS E WHERE EXISTS
( SELECT * FROM Dependent  AS D WHERE  E.Ssn =D. EssnAND E.Sex=D.Sex );
```

• **EXISTS(Q)** returns TRUE if there is at least one tuple in the result of the nested query Q, and it returns FALSE otherwise.

• On the other hand, **NOT EXISTS(Q)** returns TRUE if there are no tuples in the result of nested query Q, and it returns FALSE otherwise

SQL JOIN

- Joins are used to get data from multiple tables
- In simple, SQL JOIN combines records from two or more tables, based on a related column between those tables
- A JOIN locates related column values in the two tables.
- A query can contain zero, one, or multiple JOIN operations.
- The following are the types of joins:
- **INNER JOIN :** This join returns those records which have matching values in both the tables.

  ie, Select records that have matching values in both tables

  INNER JOIN is the same as JOIN; the keyword INNER is optional

- **FULL JOIN :** This join returns all those records which either have a match in the left or the right table.

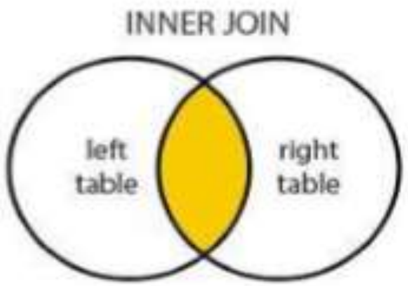
  ie, Selects all records that match either left or right table records

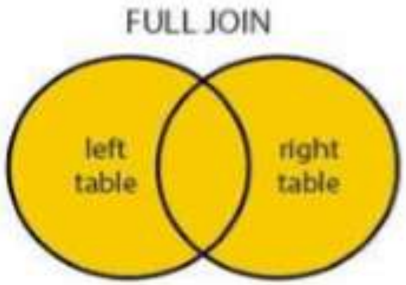
- **LEFT JOIN :** This join returns records from the left table, and also those records which satisfy the condition from the right table.

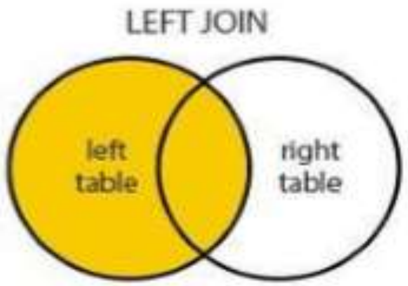
  ie, Select records from the first (left-most) table with matching right table records

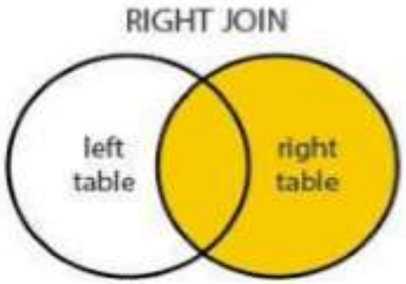
- **RIGHT JOIN :** This join returns records from the right table, and also those records which satisfy the condition from the left table.

  ie, Select records from the second (right-most) table with matching left table records


INNER JOIN
left table
right table


FULL JOIN
left table
right table


LEFT JOIN
left table
right table


RIGHT JOIN
left table
right table

1. SQL INNER JOIN

• The general **syntax** is

```
SELECT < column-names >  FROM  < table-name1 > INNER JOIN < table-name2 >
 ON < column-name1 = column-name2 > WHERE  < condition > ;
```

• The INNER keyword is optional: it is the default as well as the most commonly used JOIN operation.

• **Example** :

```
SELECT  Technologies.TechID,  Employee_Info.EmployeeName
FROM Technologies
INNER JOIN Employee_Info   ON  Technologies .EmpID = Employee_Info .EmpID;
```

2. SQL LEFT JOIN

• A LEFT JOIN performs a join starting with the first (leftmost) table.

• Then, any matched records from the second table (rightmost) will be included.

• LEFT JOIN and LEFT OUTER JOIN are the same.

• The general **LEFT JOIN syntax** is :

```
SELECT < column-names > FROM < table-name1> LEFT JOIN < tablename2 > ON
< column-name1 = column-name2 > WHERE < condition > ;
```

(OR)

```
SELECT < column-names > FROM  < table-name1 > LEFT OUTER JOIN  < tablename2 >
 ON < column-name1 = column-name2  >WHERE < condition >;
```

• **Example**

```
SELECT Employee_Info.EmployeeName, Technologies.TechID
FROM Employee_Info
LEFT JOIN Technologies ON Employee_Info.EmployeeID = Technologies.EmpIDID
ORDER BY Employee_Info.EmployeeName;
```

3. <u>SQL RIGHT JOIN</u>

• A RIGHT JOIN performs a join starting with the second (right-most) table and then any matching first(left-most) table records.

• RIGHT JOIN and RIGHT OUTER JOIN are the same.

• The general **syntax** is

```
SELECT < column-names  > FROM < table-name1 > RIGHT JOIN < tablename2 >
 ON  < column-name1 = column-name2  > WHERE  < condition >;
                                    ( OR )
SELECT  < column-names >  FROM  < table-name1 >  RIGHT OUTER JOIN  < tablename2 >
 ON  < column-name1 = column-name2 >  WHERE  < condition > ;
```

• **Example**

```
SELECT   Technologies.TechID   FROM  Technologies   RIGHT JOIN  Employee_Info
ON  Technologies. EmpID = Employee_ Info.EmployeeID  ORDER BY  Technologies.TechID;
```

4. SQL FULL JOIN

- FULL JOIN returns all matching records from both tables whether the other table matches or not.
- Be aware that a FULL JOIN can potentially return very larged at a sets.
- These two: FULL JOIN and FULL OUTER JOIN are the same.
- The general **syntax** is:

```
SELECT < column-names > FROM  < table-name1 > FULL JOIN  < tablename2  >
ON  < column-name1 = column-name2 >  WHERE  < condition > ;
                                   (OR)
SELECT < column-names > FROM  < table-name1>  FULL OUTER JOIN  < tablename2 >
 ON  < column-name1 = column-name2  > WHERE  < condition > ;
```

- **Example**

```
SELECT Employee_Info.EmployeeName, Technologies.TechID
FROM Employee_Info
FULL OUTER JOIN Orders ON Employee_Info.EmpID=Employee_Salary.EmpID
ORDER BY Employee_Info.EmployeeName;
```

VIEWS (Virtual Tables) in SQL

- A view in SQL is a saved SQL query that acts as a virtual table.
- A view in SQL terminology is a single table that is derived from other tables
- These other tables can be base tables or previously defined views
- A view does not necessarily exist in physical form;
    - it is considered to be a virtual table, in contrast to base tables
    - whose tuples are always physically stored in the database.
- This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view.
- It can fetch data from one or more tables and present it in a customized format, allowing developers to:

    - **Simplify Complex Queries:** Encapsulate complex joins and conditions into a single object.
    - **Enhance Security:** Restrict access to specific columns or rows.
    - **Present Data Flexibly:** Provide tailored data views for different users.

- We can think of a view as a way of specifying a table that we need to reference frequently, even though it may not exist physically.
- Views, which are a type of virtual tables allow users to do the following
    - Structure data in a way that users or classes of users find natural or intuitive.
    - Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
    - Summarize data from various tables which can be used to generate reports.

Specification of Views in SQL

- In SQL, the command to specify a view is **CREATE VIEW.**
- The view is given a (virtual) table name (or view name), a list of attribute names, and a query to specify the contents of the view.
- If none of the view attributes results from applying functions or arithmetic operations,
    - we do not have to specify new attribute names for the view
    - since they would be the same as the names of the attributes of the defining tables in the default case

- A view is supposed to bealways up-to-date;
    - if we modify the tuples in the base tables on which the view is defined, the view must automatically reflect these changes.

- Hence, the view is not realized or materialized at the time of view definition but rather at the time when we specify a query on the view.
- It is the responsibility of the DBMS and not the user to make sure that the view is kept up-to-date

CREATE VIEWS in SQL

- We can create a view using **CREATE VIEW** statement.
- A View can be created from a single table or multiple tables.
- **Syntax:**

```
CREATE VIEW < view_name >  AS
SELECT  < column1, column2…..>
FROM  < table_name >
WHERE < condition > ;
```

Parameters:

view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows

Delete View in SQL

- If we do not need a view any more, we can use the **DROP VIEW** command to dispose of it.
- **Syntax:**

DROP VIEW < view_name > ;

UPDATING VIEWS

- There are certain conditions needed to be satisfied to update a view :

1. The view is defined based on one and only one table.
2. The view must include the PRIMARYKEY of the table based upon which the view has been created.
3. The view should not have any field made out of aggregate functions.
4. The view must not have any DISTINCT clause in its definition.
5. The view must not have any GROUPBY or HAVING clause in its definition.
6. The view must not have any SUBQUERIES in its definitions.
7. If the view you want to update is based up on another view, the later should be updatable.
8. Any of the selected output fields (of the view) must not use constants, strings or value expressions.

Uses of a View

- **Restricting data access**
  - ▫ Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
- **Hiding data complexity**
  - ▫ A view can hide the complexity that exists in a multiple table join.
- **Simplify commands for the user**
  - ▫ Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join
- **Store complex queries**
  - ▫ Views can be used to store complex queries.
- **Rename Columns**
  - ▫ Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement.

    Thus, renaming helps to to hide the names of the columns of the base tables.
- **Multiple view facility**
  - ▫ Different views can be created on the same table for different users.

ASSERTIONS

• When a constraint involves two (or) more tables, the table constraint mechanism is sometimes hard and results may not come as expected.

• To cover such a situation SQL supports the creation of assertions that are constraints not associated with only one table.

• An assertion statement should ensure a certain condition will always exist in the database.

• ie, **Assertions are conditions that the database must always satisfy**

• These are used to implement complex rules that cannot enforced using simple constraints like CHECK, PRIMARY KEY or FOREIGN KEY

• Domain constraints and referential-integrity constraints are specific forms of assertions

  • CHECK – verify the assertion on one-table,one-attribute

  • ASSERTION – verify one or more tables, one or more attributes

• Some constraints cannot be expressed by using only domain constraints or referential-integrity constraints

• **Syntax –**

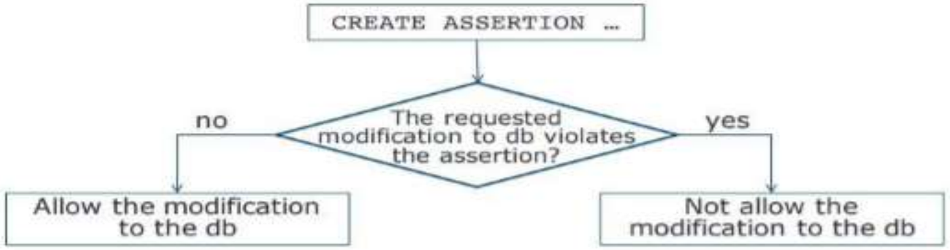
```
CREATE ASSERTION [ assertion_name ] CHECK ( [ condition ] );
```

• For example,

"Every department must have at least five courses offered every semester" – must be expressed as an assertion

```
CREATE ASSERTION <assertion-name>
CHECK <predicate>;
```

```
DROP ASSERTION <assertion-name>
```


CREATE ASSERTION …
The requested modification to db violates the assertion?
no
yes
Allow the modification to the db
Not allow the modification to the db

TRIGGER

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.
- A trigger is a database object that is associated with the table, it will be activated when a defined action is executed for the table.
- The trigger can be executed when we run the following statements: INSERT, UPDATE, DELETE
- And it can be invoked before or after the event.
- For example,
    - a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
    - it may be useful to specify a condition that, if violated, causes some user to be informed of the violation

Benefits of Triggers

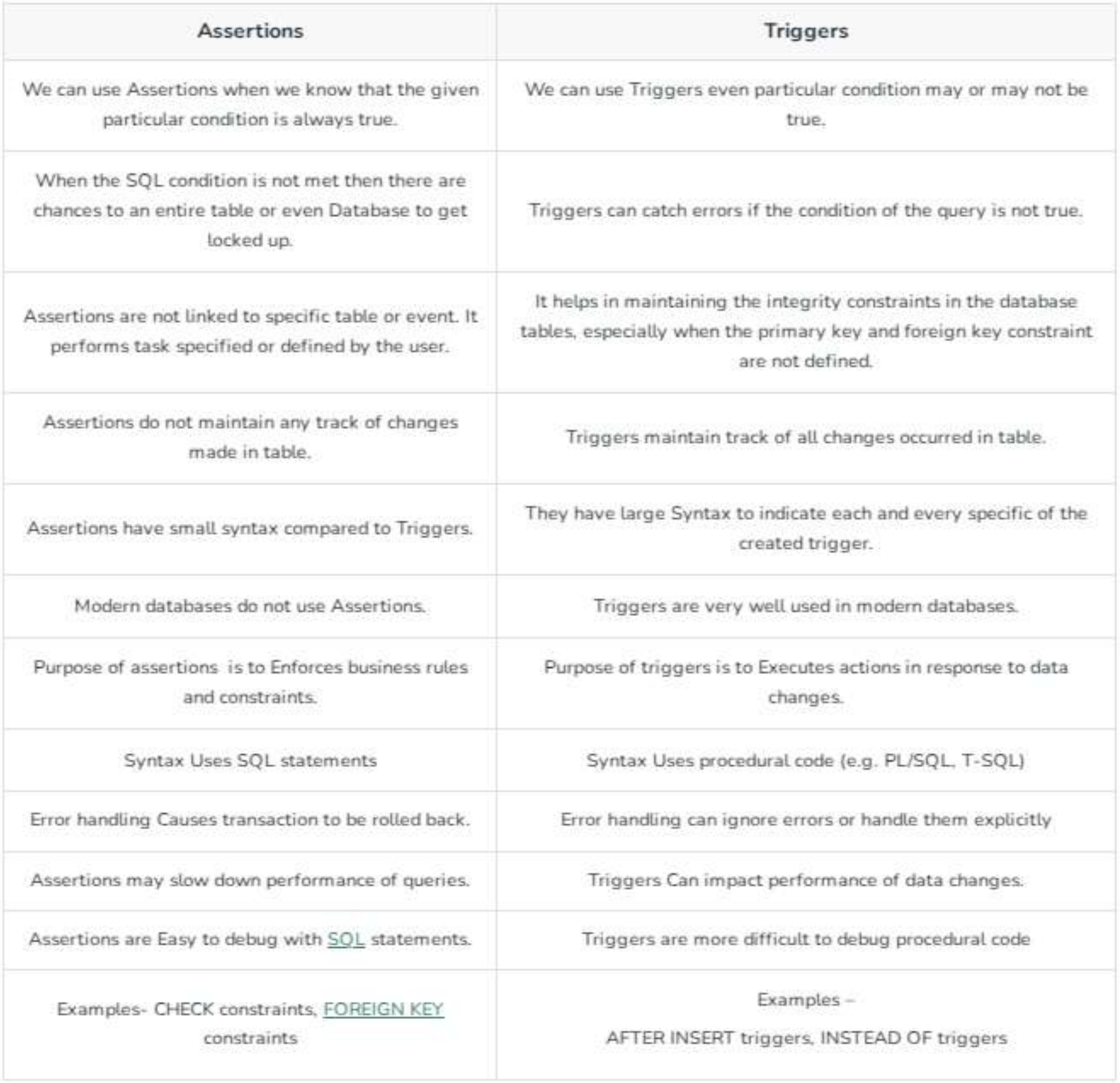
- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Syntax –

```
CREATE TRIGGER [trigger_name] [BEFORE | AFTER]
{INSERT | UPDATE | DELETE} ON [table_name]
[for each row] [trigger_body];
```

- CREATE [OR REPLACE] TRIGGER trigger_name
  - Creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF}
  - This specifies when the trigger will be executed.
  - The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}
  - This specifies the DML operation.
- [OF col_name]
  - This specifies the column name that will be updated.
- [ON table_name]
  - This specifies the name of the table associated with the trigger.

Difference Between Assertions and Triggers



Assertions	Triggers
We can use Assertions when we know that the given particular condition is always true.	We can use Triggers even particular condition may or may not be true.
When the SQL condition is not met then there are chances to an entire table or even Database to get locked up.	Triggers can catch errors if the condition of the query is not true.
Assertions are not linked to specific table or event. It performs task specified or defined by the user.	It helps in maintaining the integrity constraints in the database tables, especially when the primary key and foreign key constraint are not defined.
Assertions do not maintain any track of changes made in table.	Triggers maintain track of all changes occurred in table.
Assertions have small syntax compared to Triggers.	They have large Syntax to indicate each and every specific of the created trigger.
Modern databases do not use Assertions.	Triggers are very well used in modern databases.
Purpose of assertions is to Enforces business rules and constraints.	Purpose of triggers is to Executes actions in response to data changes.
Syntax Uses SQL statements	Syntax Uses procedural code (e.g. PL/SQL, T-SQL)
Error handling Causes transaction to be rolled back.	Error handling can ignore errors or handle them explicitly
Assertions may slow down performance of queries.	Triggers Can impact performance of data changes.
Assertions are Easy to debug with SQL statements.	Triggers are more difficult to debug procedural code
Examples- CHECK constraints, FOREIGN KEY constraints	Examples – AFTER INSERT triggers, INSTEAD OF triggers

PHYSICAL DATA ORGANIZATION

INTRODUCTION

•There are two types of storage devices used with computers:

- a primary storage device
- a secondary storage device

• **Primary storage devices**: Generally smaller in size, these are designed to hold data temporarily and are internal to the computer. They have the fastest data access speed, and include RAM and cache memory.

• **Secondary storage devices**: These usually have large storage capacity, and they store data permanently. They can be either internal or external to the computer, and they include the hard disk, optical disk drive and USB storage device.

• Relative data and information is stored collectively in file formats.

• The File is a collection of records

• A file can contain:

● Fixed-length records - all the records are exactly the same length

● Variable-length records - the length of each record varies

• Using variable-length records might enable you to save disk space.

• When you use Fixed-length records, you need to make the record length equal to the length of the longest record

PHYSICAL FILES

• Physical files contain the actual data that is stored on the system, and a description of how data is to be presented to or received from a program.

• They contain only one record format, and one or more members.

• Records in database files can be externally or program described.

• A physical file can have a keyed sequence access path. This means that data is presented to a program in a sequence based on one or more key fields in the file.

LOGICAL FILES

• Logical files do not contain data.

• They contain a description of records found in one or more physical files.

• A logical file is a view or representation of one or more physical files.

• Logical files that contain more than one format are referred to as multi-format logical files.

Physical v/s Logical File

Physical File	Logical File
•It occupies the portion of memory •It contains the original data	•It does not occupy memory space • It does not contain data
• A physical file contains one record format	• It can contain upto 32 record formats
• It can exist without logical file	• It cannot exist without physical file
• If there is a logical file for physical file, the physical file cannot be deleted until and unless we delete the logical file	• If there is a logical file for a physical file, the logical file can be deleted without deleting the physical file

File organizations

• File organization refers to the organization of the data of a file into records, blocks, and access structures; this includes the way records and blocks are placed on the storage medium and interlinked.

• An access method, on the other hand, provides a group of operations that can be applied to a file.

• In general, it is possible to apply several access methods to a file organization Primary file organizations

• **Primary file organizations** determines how the file records are physically placed on the disk, and hence how the records can be accessed.

1. Heap file (or unordered file) places the records on disk in no particular order by appending new records at the end of the file

2. Sorted file (or sequential file) keeps the records ordered by the value of a particular field (called the sort key).

3. Hashed file uses a hash function applied to a particular field (called the hash key) to determine a record’s placement on disk.

4. Other primary file organizations, such as **B-trees,** use tree structures

• **Secondary organization :**

• A secondary organization or auxiliary access structure allows efficient access to file records based on alternate fields than those that have been used for the primary file organization

RECORDS AND RECORD TYPES

• Data is usually stored in the form of records.

• Each record consists of a collection of related data values or items, where each value is formed of one or more bytes and corresponds to a particular field of the record.

• Records usually describe entities and their attributes. •

A collection of field names and their corresponding data types constitutes a record type or record format definition.

• A data type, associated with each field, specifies the types of values a field can take

Files, Fixed-Length Records, and Variable Length Records

• A file is a sequence of records.

• In many cases, all records in a file are of the same record type.

• If every record in the file has exactly the same size(in bytes), the file is said to be made up of fixed-length records.

• If different records in the file have different sizes, the file is said to be made up of variable-lengthr ecords

SPANNED AND UN SPANNED ORGANIZATION

• A block is the unit of data transfer between disk and memory

• When the block size is larger than the record size, each block will contain numerous records, although some files may have unusually large records that cannot fit in one block.

• Part of the record can be stored on one block and the rest on another.

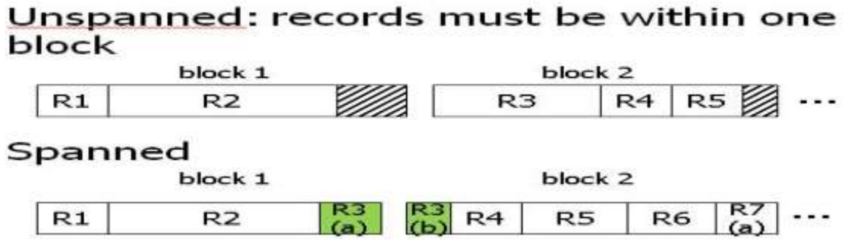
• A pointer at the end of the first block points to the block containing the remainder of the record. This organization is called spanned because records can span more than one block.

• Whenever a record is larger than a block, we must use a spanned organization.

• If records are not allowed to cross block boundaries, the organization is called un-spanned.

A part of record will get stored on one block and remaining on some other disk block. In such a case there will be a pointer at the end of the disk block that will point to the next block. Such a organization is called Spanned.

If Records are not allowed to cross block boundaries we call them un-spanned.



BLOCKING FACTOR FOR THE FILE

• The records of a file must be allocated to disk blocks because a block is the unit of data transfer between disk and memory.

• When the block size is larger than the record size, each block will contain numerous records, although some files may have unusually large records that cannot fit in one block.

• Suppose that the block size is B bytes.

• For a **file of fixed-length records** of size R bytes, with B ≥ R,

we can fit **bfr = $\lfloor B / R \rfloor$ records per block.**

• The value bfr is called the blocking factor for the file.

• A file is a sequence of records stored in binary format

• A disk drive is formatted into several blocks that can store records.

• File records are mapped onto those disk blocks.

• **Blocking factor is the number of records in a block at a particular time.**

• **Blocking factor = floored((Block size)/(Record size))**

• Block size is larger than the record size.

- In general, R may not divide B exactly, so we have some unused space in each block equal to

$$\mathbf{B - (bfr * R)} \text{ bytes}$$

- To utilize this unused space, we can store part of a record on one block and the rest on another.
- A pointer at the end of the first block points to the block containing the remainder of the record incase it is not the next consecutive block on disk.
- This organization is called spanned because records can span more than one block.
- When ever a record is larger than a block, we must use a spanned organization.
- If records are not allowed to cross block boundaries, the organization is called un-spanned
- If the average record is large, it is advantageous to use spanning to reduce the lost space in each block
- For **variable-length records using spanned organization, each block may store a different number of records.**
- **In this case, the blocking factor ,bfr represents the average number of records per block for the file.**
- We can use **bfr to calculate** the number of blocks b needed for a file of r records:

$$\mathbf{b = F(r/bfr)\rceil \text{ blocks}}$$

where , the $F(x)\rceil$ (ceiling function) rounds the value x up to the next integer.

FILE ALLOCATION METHODS

• The allocation methods define how the files are stored in the disk blocks.

• There are different main disk space or file allocation methods.

● **Contiguous Allocation**: File occupies a contiguous set of blocks on the disk.
This method suffers from both internal and external fragmentation.

● **Linked Allocation** : Each file is a linked list of disk blocks which need not be contiguous.
The disk blocks can be scattered anywhere on the disk.
Each block contains a pointer to the next block occupied by the file.

● **Indexed Allocation** : A special block known as the Index block contains the pointers to all the blocks occupied by a file.
Each file has its own index block.

• **Clusters Allocation** : A combination of the two allocates clusters of consecutive disk blocks, and the clusters are linked.
Clusters are sometimes called file segments or extents

1. **Files of Unordered Records (Heap Files)**

• In this simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file.

• Such an organization is called a heap or pile file.

• There is no sorting or ordering of the records.

• Once the data block is full, the next record is stored in the new block.

• This new block need not be the very next block.

• This method can select any block in the memory to store the new records.

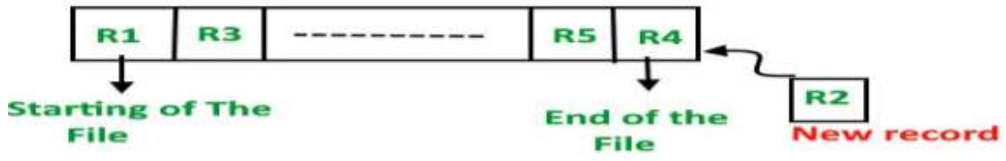
• When a record has to be retrieved from the database, in this method, we need to traverse from the beginning of the file till we get the requested record (linear search).

• To delete a record, a program must first find its block, copy the block into a buffer, delete the record from the buffer, and finally rewrite the block back to the disk.

• This leaves unused space in the disk block.

• Deleting a large number of records in this way results in wasted storage space.

• The deletion techniques require periodic reorganization of the file to reclaim the unused space of deleted records.



Insertion

- Inserting a new record is very efficient.
- The last disk block of the file is copied into a buffer, the new record is added, and the block is then rewritten back to disk.
- The address of the last file block is kept in the file header.

Searching

- Searching a record using any search condition involves a linear search through the file block by block an expensive procedure.
- If only one record satisfies the search condition, then, on the average, a program will read into memory and search half the file blocks before it finds the record.
- For a file of b blocks, this requires searching (b/2) blocks, on average.
- If no records or several records satisfy the search condition, the program must read and search all b blocks in the file

Deletion

- A program must first find its block, copy the block into a buffer, delete the record from the buffer, and finally rewrite the block back to the disk.
- This leaves unused space in the disk block.
  Deleting a large number of records in this way results in wasted storage space.

2. Files of Ordered Records (Sorted Files)

- We can physically order the records of a file on disk based on the values of one of their fields called the ordering field.
- This leads to an ordered or sequential file.
- If the ordering field is also a key field of the file a field guaranteed to have a unique value in each record then the field is called the ordering key for the file
- **Sorting** is based on a key field.
- Insertion and deletion operation is expensive.
- To **insert and delete** we have to find the exact block of records.
- For **selection**, both linear search and binary search are used.

Ordered records advantages over unordered files

- First, reading the records in order of the ordering key values becomes extremely efficient because no sorting is required.
- Second, finding the next record from the current one in order of the ordering key usually requires no additional block accesses because the next record is in the same block as the current one
- Third, using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used, which constitutes an improvement over linear searches, although it is not often used for disk files.
- Ordered files are blocked and stored on contiguous cylinders to minimize the seek time

3. Hashed Files

• Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure.

• Data is stored at the data blocks whose address is generated by using hash function.

• Hashing for disk files is called **External Hashing**

• The file blocks are divided into M equal-sized buckets, numbered bucket 0, bucket 1, ..., bucket M-1

• Typically, a bucket corresponds to one(or a fixed number of) disk block.

• One of the file fields is designated to be the **hash key** of the file.

• The record with hash key value K is stored in bucket i, where i = h(K), and h is the hashing function.

• Search is very efficient on the hash key.

INDEX STRUCTURES

• An index is a data structure which is used to quickly locate and access the data in a database table.

• Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.

• An index on a database table provides a convenient mechanism for locating a row (data record) without scanning the entire table and thus greatly reduces the time it takes to process a query.

• The index is usually specified on one field of the file.

• One form of an index is a file of entries < field value, pointer to record >, which is ordered by field value.

• The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller.

Advantages :

▫ Stores and organizes data into computer files.

▫ Makes it easier to find and access data at any given time.

▫ It is a data structure that is added to a file to provide faster access to the data.

▫ It reduces the number of blocks that the DBMS has to check.

Disadvantages :

▫ Index needs to be updated periodically for insertion or deletion of records in the main table.

Structure of index

- An index is a small table having only two columns.
- The first column contains a copy of the primary or candidate key of a table
- The second column contains a set of pointers holding the address of the disk block where that particular key value can be found.
- If the indexes are sorted, then it is called as ordered indices.
- There are different types of indexes, these include:
    - Primary Indexes
    - Clustering index
    - Secondary index

- **Primary Index** : Primary index is defined on an ordered data file.
    - The data file is ordered on a key field.
    - The key field is generally the primary key of the relation.
    - Primary index is created automatically when the table is created in the DB.
    - Primary index is defined on an ordered data file.
    - The data file is ordered on a key field.
    - The key field is generally the primary key of the relation.

- A primary index is a non dense (sparse) index, since it includes an entry for eachdisk block of the data file and the keys of its anchor record rather than for every search value
- There are two types of primary index,
  - Dense index
  - Sparse Index

Dense Index

- In dense index, there is an index record for every search key value in the database.
- This makes searching faster but requires more space to store index records itself.
- Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

- Index records are created only for some of the search key.
- To locate a record, we find the index record with search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along the pointers in the file (that is, sequentially) until we find the desired record.

Secondary Index : • A secondary index provides a secondary means of accessing a file for which some primary access already exists.

• The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

• The index is an ordered file with two fields.

• The first field is of the same data type as some non-ordering field of the data file that is an indexing field.

• The second field is either a block pointer or a record pointer.

• There can be many secondary indexes (and hence, indexing fields)for the same file.

• Includes one entry for each record in the data file; hence, it is a dense index.

Clustering Index :

• If file records are physically ordered on a non key field which does not have a distinct value for each record that field is called the clustering field and the data file is called a clustered file.

• Clustering index speeds up retrieval of all the records that have the same value for the clustering field.

• This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record.

Single level and Multi-level indexing

Single level index :

- A single-level index is an ordered file, we can create a primary index to the index itself
- In this case, the original index file is called the first level index and the index to the index is called the second-level index.
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the top level fit in one disk block.

Multi level Index :

- If the primary index does not fit in the memory access become expensive.
- To reduce the number of disk accesses to index records, treat primary index kept on the disk as sequential file and construct a sparse index on it.
    - Outer index – A sparse index of primary index.
    - Inner index – The primary index file.
- If even outer index is too large to fit in main memory, yet another level of index can be created.
- Indices from all levels must be updated on insertion or deletion from the file.
- Insertion and deletion are complicated because all index files at different levels are ordered files.

- One solution to this problem is to keep some space reserved in each block for new entries.
- The outer index is the first level sparse index.
- The inner index is the second level sparse index.
- In order to search for an index, one can first apply binary search on the second level index to find the largest value which is less than or equal to the search key.
- The pointer corresponding to this value points to the block of the first level index that contains an entry for the search record.
- This block level index is searched to locate the largest value which is less than or equal to the search key.
- The pointer corresponding to this block directs us to the block that contains the required content.
- If the second level index is too small to fit in main memory at once, fewer blocks of index file needs to be accessed from the disk to locate a particular record.
- However if the second level is too large to fit in the main memory at once, another level of index can be created.
- This process of creating index on index can be repeated until the size of the index become small enough to fit in the main memory.
- This type of index with multiple levels of index is called Multilevel Index.

B-TREES

• The B-tree has additional constraints that ensure that the tree is always balanced.

• B Tree is a ***specialized m-way tree that can be widely used for disk access***.

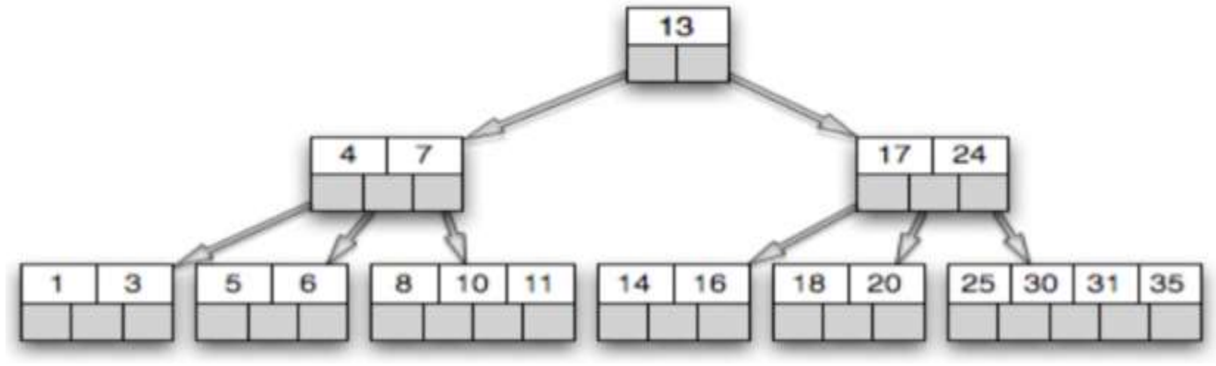
• A B-Tree of order m can have ***at most m-1 keys and m children***.

• One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

• B-tree of order m is a tree which satisfies the following **properties**

1. Every node has at most m children.
2. Every internal node has at least $\lceil m/2 \rceil$ children.
3. Every non-leaf node has at least two children.
4. All leaves appear on the same level and carry no information.
5. A non-leaf node with k children contains $k-1$ keys.

- Internal nodes (also known as inner nodes) are all nodes except for leaf nodes and the root node.
- The root node's number of children has the same upper limit as internal nodes, but has no lower limit



B + -TREES

- B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
- In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records can only be stored on the leaf nodes while internal nodes can only store the key values.
- The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.
- B+ Tree are used to store the large amount of data which can not be stored in the main memory.
- The internal nodes of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.
- The internal nodes of B+ tree are often called index nodes

Properties :

- All leaf nodes must be at same level.
- All nodes except root must have at least [m / 2] -1keys and maximum of m-1 keys in the case of order.
- All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.
- If the root node is a non leaf node, then it must have atleast 2 children.
- A non leaf node with n-1 keys must have n number of children.
- All the key values in a node must be in Ascending Order.

B Tree V/s B + Tree

- In B + trees, search keys can be repeated but this is not the case for B-trees
- B + trees allow satellite data to be stored in leaf nodes only, whereas B-trees store data in both leaf and internal nodes
- In B + trees, data stored on the leaf node makes the search more efficient since we can store more keys in internal nodes , this means we need to access fewer nodes
- Deleting data from a B + tree is easier and less time consuming because we only need to remove data from leaf nodes
- Leaf nodes in a B + tree are linked together making range search operations efficient and quick
- Finally, although B – trees are useful ,B + trees are more popular.
- In fact, 99% of database management systems use B + trees for indexing.
- This is because the B + tree holds no data in the internal nodes.
- This maximizes the number of keys stored in a node thereby minimizing the number of levels needed in a tree.