DBMS IA-2 ANSWERS:

1. Give a syntax of the select query in sql.Give the 6 basic clauses of the select query?

Ans: Syntax. SELECT **column1, column2, columnN FROM table\_name**; Here, column1, column2... are the fields of a table whose values you want to fetch.

The six clauses in no particular order are SELECT, **FROM, WHERE, GROUP BY, HAVING, and ORDER BY.**

**SELECT Clause**

The SELECT clause specifies which values are to be returned. To display all the columns of a table, use the asterisk wildcard character (\*). For example, the following query displays all rows and columns from the employees table:

SELECT \* FROM employee;

**FROM Clause**

The FROM clause specifies the source tables and views from which data is to be read. The specified tables and views must exist at the time the query is issued.

One or more tables or views, specified using the following syntax:

[*schema*.]*table* [[AS] *corr\_name*]

where *table* is the name of a table, view, or synonym.

**WHERE Clause**

The WHERE clause specifies criteria that restrict the contents of the results table. You can test for simple relationships or, using subselects, for relationships between a column and a set of columns.

Using a simple WHERE clause, the contents of the results table can be restricted, as follows:

Comparisons:

SELECT ename FROM employee\_dim

    WHERE manager = 'Al Obidinski';

**GROUP BY Clause**

The GROUP BY clause groups the selected rows based on identical values in a column or expression. This clause is typically used with aggregate functions to generate a single result row for each set of unique values in a set of columns or expressions.

GROUP BY has the following format:

GROUP BY [ALL | DISTINCT] *grouping element* {,*grouping element*}

**HAVING Clause**

The HAVING clause filters the results of the GROUP BY clause by using an aggregate function. The HAVING clause uses the same restriction operators as the WHERE clause.

For example, to return parts that have sold more than ten times on a particular day during the past week:

SELECT odate, partno, count(\*) FROM orders  
GROUP BY odate, partno  
WHERE odate >= (CURRENT\_DATE – INTERVAL '7' day)  
HAVING count(\*) > 10;

**ORDER BY Clause**

The ORDER BY clause allows you to specify the columns on which the results table is to be sorted. For example

SELECT manager, emp\_name, dept\_no FROM employee\_dim  
ORDER BY manager, dept\_no, emp\_name

1. With suitable examples in SQL explain schema change statements?

Ans: The **schema evolution commands** available in SQL, which can be used to alter a schema by adding or dropping tables, attributes, constraints, and other schema elements. This can be done while the database is operational and does not require recompilation of the database schema.The 2 types are as follows

1. **The DROP Command**

The DROP command can be used to drop *named* schema elements, such as tables, domains, or constraints. One can also drop a schema. For example, if a whole schema is no longer needed, the DROP SCHEMA command can be used. There are two *drop behavior* options: CASCADE and RESTRICT. For example, to remove the COMPANY database schema and all its tables, domains, and other elements, the CASCADE option is used as follows:

DROP SCHEMA COMPANY CASCADE;

If the RESTRICT option is chosen in place of CASCADE, the schema is dropped only if it has *no elements* in it; otherwise, the DROP command will not be executed. To use the RESTRICT option, the user must first individually drop each element in the schema, then drop the schema itself.

1. The ALTER Command

The definition of a base table or of other named schema elements can be changed by using the ALTER command. For base tables, the possible **alter table actions** include adding or dropping a column (attribute), changing a column definition, and adding or dropping table constraints. For example, to add an attribute for keeping track of jobs of employees to the EMPLOYEE base relation in the COMPANY schema  we can use the command

ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);

the following command removes the attribute Address from the EMPLOYEE base table:

ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;

It is also possible to alter a column definition by dropping an existing default clause or by defining a new default clause. The following examples illustrate this clause:

 ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr\_ssn

 DROP DEFAULT;

 ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr\_ssn

 SET DEFAULT ‘333445555’;

1. how do you create a view in sql?give example,Can you update a view table discuss?

## Ans: Syntax

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE  OR ALTER  VIEW  schema\_name.view\_name  WITH <view\_attribute>  AS select\_statement  [WITH CHECK OPTION]  Example:  CREATE TABLE Employees  (EmployeeID    INT NOT NULL,  FirstName     NVARCHAR(50) NOT NULL,  MiddleName    NVARCHAR(50) NULL,  LastName      NVARCHAR(75) NOT NULL,  Title         NVARCHAR(100) NULL,  HireDate      DATETIME NOT NULL,  VacationHours SMALLINT NOT NULL,  Salary        DECIMAL(19, 4) NOT NULL  );  GO  CREATE TABLE Products  (ProductID INT NOT NULL,  Name      NVARCHAR(255) NOT NULL,  Price     DECIMAL(19, 4) NOT NULL  );  GO  CREATE TABLE Sales  (SalesID    UNIQUEIDENTIFIER NOT NULL,  ProductID  INT NOT NULL,  EmployeeID INT NOT NULL,  Quantity   SMALLINT NOT NULL,  SaleDate   DATETIME NOT NULL  );  GO  **Yes we can insert,Update and delete**, if a view is not complex. In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. We can insert, update and delete a view. |

4) explain the concept of triggers with suitable examples?

Ans: **Triggers:**

A trigger is a block of code that is executed automatically from a database statement. Triggers is generally executed for DML statements such as INSERT, UPDATE or DELETE. It resides in a database code and is fired automatically when the database code requires to perform the INSERT ,UPDATE or DELETE statement. It is also stored in the database as stored procedures. It does not involve any COMMIT transaction rather it is a part of a transaction.

SYNTAX :

CREATE OR REPLACE TRIGGER <trigger-name>

[BEFORE/AFTER]

[INSERT/UPDATE/DELETE]

OF<column-name>

ON<table-name>

[REFERENCING OLD AS O NEW AS N]

[FOR EACH ROW]

WHEN <trigger-condition>

DECLARE

BEGIN

<sql-statement>

END;

**Example:**

CREATE TRIGGER Amount-CHECK BEFORE UPDATE ON account

FOR EACH ROW

BEGIN

IF (NEW.amount < 0) THEN

SET NEW.amount = 0;

ELSEIF (NEW.amount > 100) THEN

SET NEW.amount = 100;

END IF;

END;

5)Write short notes on EXISTS and UNION functions in sql?

Ans:

EXISTS-

The EXISTS condition 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 or False. It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

**Syntax:**

SELECT column\_name(s) FROM table\_name WHERE EXISTS (SELECT column\_name(s) FROM table\_name WHERE condition);

Example:

SELECT fname, lname FROM Customers WHERE EXISTS (SELECT \* FROM Orders WHERE Customers.customer\_id = Orders.c\_id);

UNION-

The SQL UNION operator is used to combine the result sets of 2 or more SELECT statements. It removes duplicate rows between the various SELECT statements.Each SELECT statement within the UNION must have the same number of fields in the result sets with similar data types.

The syntax for the UNION operator in SQL is:

SELECT expression1, expression2, ... expression\_n

FROM tables

[WHERE conditions]

UNION

SELECT expression1, expression2, ... expression\_n

FROM tables

[WHERE conditions];

For example:

SELECT supplier\_id

FROM suppliers

UNION

SELECT supplier\_id

FROM orders

ORDER BY supplier\_id;

7)What is JDBC?Explain its concepts?

Ans: **Java Database Connectivity** (**JDBC**) is an [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (API) for the programming language [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), which defines how a client may access a [database](https://en.wikipedia.org/wiki/Database). It is a Java-based data access technology used for Java database connectivity. It is part of the [Java Standard Edition](https://en.wikipedia.org/wiki/Java_Standard_Edition) platform, from [Oracle Corporation](https://en.wikipedia.org/wiki/Oracle_Corporation). It provides methods to query and update data in a database, and is oriented toward [relational databases](https://en.wikipedia.org/wiki/Relational_database). A JDBC-to-[ODBC](https://en.wikipedia.org/wiki/ODBC) bridge enables connections to any ODBC-accessible data source in the [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) host environment.

**Components of JDBC**  
There are generally four main components of JDBC through which it can interact with a database. They are as mentioned below:

1. **JDBC API:** It provides various methods and interfaces for easy communication with the database.It provides two packages as follows which contains the java SE and java EE platforms to exhibit WORA(write once run everywhere) capabilities.

java.sql.\*;

javax.sql.\*;

It also provides a standard to connect a database to a client application.

1. [JDBC Driver manager](https://www.geeksforgeeks.org/jdbc-drivers/)**:**It loads database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
2. **JDBC Test suite:** It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.
3. **JDBC-ODBC Bridge Drivers**: It connects database drivers to the database.This bridge translates JDBC method call to the ODBC function call.It makes the use of

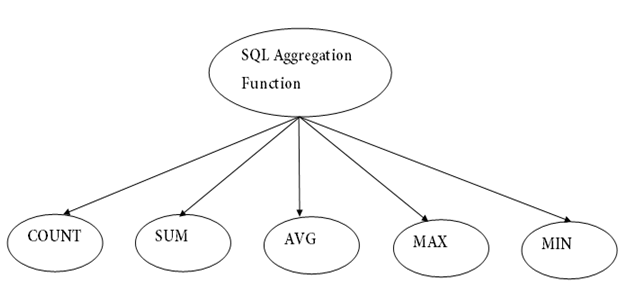
sun.jdbc.odbc package that includes native library to access ODBC characteristics.

8) Discuss all the aggregate functions with suitable relational algebra examples?

# Ans: **SQL Aggregate Functions**

* SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
* It is also used to summarize the data.

## Types of SQL Aggregation Function



### **1. COUNT FUNCTION**

* COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
* COUNT function uses the COUNT(\*) that returns the count of all the rows in a specified table. COUNT(\*) considers duplicate and Null.

**Syntax**

COUNT(\*)  or  COUNT( [ALL|DISTINCT] expression )

**Example: COUNT()**

SELECT COUNT(\*)

FROM PRODUCT\_MAST;

### **2. SUM Function**

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

**Syntax**

SUM()  or  SUM( [ALL|DISTINCT] expression )

**Example: SUM()**

SELECT SUM(COST)  FROM PRODUCT\_MAST;

### **3. AVG function**

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

**Syntax**

AVG()  or  AVG( [ALL|DISTINCT] expression )

**Example:**

SELECT AVG(COST)  FROM PRODUCT\_MAST;

### **4. MAX Function**

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

**Syntax**

MAX()  or  MAX( [ALL|DISTINCT] expression )

**Example:**

SELECT MAX(RATE)  FROM PRODUCT\_MAST;

### **5. MIN Function**

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

**Syntax**

MIN()  or  MIN( [ALL|DISTINCT] expression )

**Example:**

SELECT MIN(RATE)  FROM PRODUCT\_MAST;

9)How are the outer join operations different from inner join operations ?how is the outer union operations different from union?

Ans: difference between **INNER JOIN** and **OUTER JOIN**:

|  |  |  |
| --- | --- | --- |
| S.No | Inner Join | Outer Join |
| 1. | It returns the combined tuple between two or more tables. | It returns the combined tuple from a specified table even join condition will fail. |
| 2. | Used clause INNER JOIN and JOIN. | Used clause LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, etc. |
| 3. | When any attributes are not common then it will return nothing. | It does not depend upon the common attributes. If the attribute is blank then here already placed NULL. |
| 4. | If tuples are more. Then INNER JOIN works faster than OUTER JOIN. | Generally, The OUTER JOIN is slower than INNER JOIN. But except for some special cases. |
| 5. | It is used when we want detailed information about any specific attribute. | It is used when we want to complete information. |
| 6. | JOIN and INNER JOIN both clauses work the same. | FULL OUTER JOIN and FULL JOIN both clauses work the same. |
|  |

**Outer Union operation**- If there are any values in one table which do not have corresponding value(s) within the other, in an equi-join that will not be chosen. Such rows could be forcefully selected through using the outer join. The corresponding columns for in which row will have NULLs. There are actually three forms of the outer-join operation: left outer join ( X), right outer join (X ) and full outer join ( X ).

Union - The UNION operator is used to combine the result-set of two or more SELECT statements.

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

### **UNION Syntax**

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

10 ans:) **Creating table sailors**

CREATE TABLE sailors

(

sid integer,

sname varchar(20),

rating integer,

age integer

);

**Creating table boats**

CREATE TABLE boats

(

bid integer,

bname varchar(20),

color varchar(20)

);

**Creating table reserves**

CREATE TABLE reserves

(

sid integer,

bid integer,

day1 date

);

Insert records into the **sailors** table:

insert into sailors values(22,'dustin',7,45);

insert into sailors values(29,'brutus',1,33);

insert into sailors values(31,'lubber',79,55);

insert into sailors values(32,'andy',8,25);

insert into sailors values(58,'rusty',10,35);

insert into sailors values(58,'buplb',10,35);

insert into sailors values(58,'buplerb',10,35);

insert into sailors values(22,'bb',10,35);

Insert records into the **boats** table:

insert into boats values(101,'interlake','blue');

insert into boats values(102,'interlake','red');

insert into boats values(103,'clipper','green');

insert into boats values(104,'marine','red');

Insert records into the **reserves** table:

insert into reserves values(22,101,'2004-01-01');

insert into reserves values(22,102,'2004-01-01');

insert into reserves values(22,103,'2004-02-01');

insert into reserves values(22,105,'2004-02-01');

insert into reserves values(31,103,'2005-05-05');

insert into reserves values(32,104,'2005-04-07');

**Problem#1:**

Find the names of sailors who have reserved boat 103.

**Solution:**

**select** s.sname

**from** sailors s,reserves r

**where** s.sid=r.sid and r.bid=103;

**Problem#2:**

Find the names of sailors who have reserved a red boat.

**Solution:**

SELECT s.sname

FROM sailors s

JOIN reserves r

ON r.sid=s.sid join boats b

ON r.bid=b.bid where b.color='red';

**Problem#3:**

Find the names of sailors who have reserved a red and a

green boat.

**Solution:**

SELECT s.sname, b.color, s.sid

FROM sailors s

JOIN reserves r ON r.sid=s.sid

JOIN boats b ON r.bid=b.bid

AND b.color='red'

WHERE r.sid IN(

SELECT s.sid

FROM sailors s

JOIN reserves r ON r.sid=s.sid

JOIN boats b ON r.bid=b.bid

WHERE b.color='green'

);

**Problem#4:**

Find the names of sailors who have reserved all the boats.

**Solution:**

**select** s.sid,s.sname

**from** sailors s

**where** not exists (**select** b.bid

**from** boats b

**where** not exists (**select** r.bid

**from** reserves r

**where** r.bid=b.bid and r.sid=s.sid));

11 ans:)write the create and insert queries on ur own

a]Retrieve the name of each employee who works on all the projects controlled by department number 5

SELECT FNAME, LNAME FROM EMPLOYEE WHERE ( (SELECT PNO FROM WORKS\_ON WHERE SSN = ESSN) CONTAINS (SELECT PNUMBER FROM PROJECT WHERE DNUM = 5) );

B]Retrieve the names of employees who have no dependents

SELECT FNAME, LNAME FROM EMPLOYEE WHERE NOT EXISTS (SELECT \* FROM DEPENDENT WHERE SSN = ESSN);

c]Retrieve the names of all employees who have two or more dependents

SELECT LNAME, FNAME FROM EMPLOYEE WHERE (SELECT COUNT (\*) FROM DEPENDENT WHERE SSN = ESSN) >= 2;

D] For every project located in ‘Boston’, list the project number, the controlling department number and the department manager’s last name, address and birthdate

SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE DNUM = DNUMBER AND MGRSSN = SSN AND PLOCATION = ‘Boston’;

E] List the names of all employees who have a dependant with the same first names as themselves

Not found

12) explain the concept of stored procedure ?

Ans: A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

### **Stored Procedure Syntax**

CREATE PROCEDURE procedure\_name  
AS  
sql\_statement  
GO;

### **To Execute a Stored Procedure**

EXEC procedure\_name;

### **Example**

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

Then we type

EXEC SelectAllCustomers;

13) Write short notes on null and the three valued logic?

Ans:NULL-

**Null** or **NULL** is a special marker used in [Structured Query Language](https://en.wikipedia.org/wiki/SQL) to indicate that a data value does not exist in the [database](https://en.wikipedia.org/wiki/Database). Introduced by the creator of the [relational](https://en.wikipedia.org/wiki/Relational_model) database model, [E. F. Codd](https://en.wikipedia.org/wiki/Edgar_F._Codd), SQL Null serves to fulfil the requirement that all *true relational database management systems (*[*RDMS*](https://en.wikipedia.org/wiki/Relational_database#RDBMS)*)* support a representation of "missing information and inapplicable information". Codd also introduced the use of the lowercase Greek [omega](https://en.wikipedia.org/wiki/Omega) (ω) symbol to represent Null in [database theory](https://en.wikipedia.org/wiki/Database_theory). In SQL, NULL is a [reserved word](https://en.wikipedia.org/wiki/Reserved_word) used to identify this marker.

A null should not be confused with a value of 0. A null value indicates a lack of a value, which is not the same thing as a value of zero. For example, consider the question "How many books does Adam own?" The answer may be "zero" (we *know* that he owns *none*) or "null" (we *do not know* how many he owns). In a database table, the [column](https://en.wikipedia.org/wiki/Column_(database)) reporting this answer would start out with no value (marked by Null), and it would not be updated with the value "zero" until we have ascertained that Adam owns no books.

SQL null is a state, not a value. This usage is quite different from most programming languages, where [null value](https://en.wikipedia.org/wiki/Null_pointer) of a reference means it is not pointing to any [object](https://en.wikipedia.org/wiki/Object_(computer_science)).

# **The Three-Valued Logic of SQL**

SQL uses a [*three-valued logic*](https://en.wikipedia.org/wiki/Three-valued_logic#Application_in_SQL): besides true and false, the result of logical expressions can also be unknown. SQL’s three valued logic is a consequence of supporting [**null** to mark absent data](https://modern-sql.com/concept/null). If a **null** value affects the result of a logical expression, the result is neither true nor false but unknown.

The three-valued logic is an integral part of [Core SQL](https://modern-sql.com/standard/levels) and it is followed by pretty much every SQL database.

The SQL **null** value basically means “could be anything”. It is therefore impossible to tell whether a comparison to **null** is true or false. That’s where the third logical value, unknown, comes in. Unknown means “true or false, depending on the **null** values”.

The result of each of the following comparisons is therefore unknown:[0](https://modern-sql.com/concept/three-valued-logic#footnote-0)

NULL = 1

NULL <> 1

NULL > 1

NULL = NULL

Nothing equals **null**. Not even **null** equals **null** because each **null** could be different.

14) Give the syntax of the select query in sql.Write short notes on partiton product with examples?

Ans: Syntax of select query:

SELECT **column1, column2, columnN FROM table\_name**; Here, column1, column2... are the fields of a table whose values you want to fetch.

## database table partitioning

Partitioning is the **database process where very large tables are divided into multiple smaller parts**. By splitting a large table into smaller, individual tables, queries that access only a fraction of the data can run faster because there is less data to scan.

## Vertical Partitioning on SQL Server tables

Vertical table partitioning is mostly used to increase SQL Server performance especially in cases where a query retrieves all columns from a table that contains a number of very wide text or BLOB columns. In this case to reduce access times the BLOB columns can be split to its own table.

## Horizontal Partitioning on SQL Server tables

Horizontal partitioning divides a table into multiple tables that contain the same number of columns, but fewer rows. For example, if a table contains a large number of rows that represent monthly reports it could be partitioned horizontally into tables by years, with each table representing all monthly reports for a specific year. This way queries requiring data for a specific year will only reference the appropriate table. Tables should be partitioned in a way that queries reference as few tables as possible.

15)Discuss various types of inner join operations.Why is theta join required?

## Ans: Types of Join

There are mainly two types of joins in DBMS:

1. Inner Joins: Theta, Natural, EQUI
2. Outer Join: Left, Right, Full

Let’s see them in detail:

**Inner Join**

**Inner Join** is used to return rows from both tables which satisfy the given condition. It is the most widely used join operation and can be considered as a default join-type

An Inner join or equijoin is a comparator-based join which uses equality comparisons in the join-predicate. However, if you use other comparison operators like “>” it can’t be called equijoin.

Inner Join further divided into three subtypes:

* Theta join
* Natural join
* EQUI join

**Theta Join**

**Theta Join** allows you to merge two tables based on the condition represented by theta. Theta joins work for all comparison operators. It is denoted by symbol **θ**. The general case of JOIN operation is called a Theta join.

**Syntax:** A ⋈θ B

**For example:**

A ⋈ A.column 2 > B.column 2 (B)

## EQUI Join

**EQUI Join** is done when a Theta join uses only the equivalence condition. EQUI join is the most difficult operation to implement efficiently in an RDBMS, and one reason why RDBMS have essential performance problems.

**For example:**

A ⋈ A.column 2 = B.column 2 (B)

## Natural Join (⋈)

**Natural Join** does not utilize any of the comparison operators. In this type of join, the attributes should have the same name and domain. In Natural Join, there should be at least one common attribute between two relations.

It performs selection forming equality on those attributes which appear in both relations and eliminates the duplicate attributes.

**Example:**

C ⋈ D

Now a theta join is required because:

A *theta join* allows for arbitrary comparison relationships (such as ≥).  
An *equijoin* is a theta join using the equality operator.  
A *natural join* is an equijoin on attributes that have the same name in each relationship.

16)Distinguish prime attributes with non-prime attributes?

Ans: Prime attribute:

Prime attribute is the attribute which is the part of the candidate key.

Non-prime attribute:

Non -prime attribute is the attribute which is not the part of the candidate key.

**Example:**

Table have four attribute : A,B,C,D

candidate key : AB

then,

prime attribute is: A and B

non-prime attribute is: C and D

17)Write short notes on Union,Intersection,and Minus operations with examples?

#### Ans: UNION

**The Union is a binary set operator in DBMS. It is used to combine the result set of two select queries.**Thus, It combines two result sets into one. In other words, the result set obtained after union operation is the collection of the result set of both the tables.

The syntax for the union operation is as follows:

SELECT (coloumn\_names) from table1 [WHERE condition] UNION **SELECT** (coloumn\_names) from table2 [WHERE condition];

Example:

SELECT color\_name FROM colors\_a UNION SELECT color\_name FROM colors\_b;

#### MINUS

**Minus is a binary set operator in DBMS. The minus operation between two selections returns the rows that are present in the first selection but not in the second selection.**The Minus operator returns only the distinct rows from the first table.

The syntax for the minus operation is as follows:

SELECT (coloumn\_names) from table1 [WHERE condition] MINUS **SELECT** (coloumn\_names) from table2 [WHERE condition];

Example: SELECT color\_name FROM colors\_a WHERE color\_name NOT **IN**(SELECT color\_name FROM colors\_b);

#### INTERSECT

**Intersect is a binary set operator in DBMS. The intersection operation between two selections returns only the common data sets or rows between them.**It should be noted that the intersection operation always returns the distinct rows. The duplicate rows will not be returned by the intersect operator.

The syntax for the intersection operation is as follows:

SELECT (coloumn\_names) from table1[WHERE condition] INTERSECT **SELECT** (coloumn\_names) from table2 [WHERE condition];

Example: SELECT color\_name FROM colors\_a WHERE color\_name **IN**(SELECT color\_name FROM colors\_b);

18)Define functional dependency using suitable examples?

Ans: **Functional Dependency (FD)** is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS). Functional Dependency helps to maintain the quality of data in the database. It plays a vital role to find the difference between good and bad database design.

A functional dependency is denoted by an arrow “→”. The functional dependency of X on Y is represented by X → Y. Let’s understand Functional Dependency in DBMS with example.

**Example:**

| **Employee number** | **Employee Name** | **Salary** | **City** |
| --- | --- | --- | --- |
| 1 | Dana | 50000 | San Francisco |
| 2 | Francis | 38000 | London |
| 3 | Andrew | 25000 | Tokyo |

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

**Rules of Functional Dependencies**

Below are the Three most important rules for Functional Dependency in Database:

* Reflexive rule –. If X is a set of attributes and Y is\_subset\_of X, then X holds a value of Y.
* Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
* Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

19)Write short notes on join and division operations with examples?

Ans: **The JOIN Operation**

The JOIN operation, denoted by  , is used to combine *related tuples* from two rela-tions into single “longer” tuples. This operation is very important for any relational database with more than a single relation because it allows us to process relation-ships among relations. To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department. To get the manager’s name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr\_ssn value in the department tuple. We do this by using the JOIN operation and then projecting the result over the necessary attributes, as follows:

DEPT\_MGR ← DEPARTMENT Mgr\_ssn=SsnEMPLOYEE

RESULT ← πDname, Lname, Fname(DEPT\_MGR)

A general join condition is of the form

 <condition> AND <condition> AND...AND <condition>

**The DIVISION Operation**

The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications. An example is *Retrieve the names of* *employees who work on****all****the projects that ‘John Smith’ works on*. To express thisquery using the DIVISION operation, proceed as follows. First, retrieve the list of project numbers that ‘John Smith’ works on in the intermediate relation

SMITH\_PNOS:

SMITH ← σFname=‘John’ AND Lname=‘Smith’(EMPLOYEE)

SMITH\_PNOS ← πPno(WORKS\_ONEssn=SsnSMITH)

The DIVISION operation can be expressed as a sequence of π, ×, and – operations as follows:

*T*1← π*Y*(*R*)

*T*2← π*Y*((*S*×*T*1) –*R*)

*T*←*T*1 –*T*2