**Complex SQL retrieval queries:**

SQL deals with NULL, NULL is used to represent a missing value.

NULL can of 3 types:

1)Value is present, but unknow(Ex: DOB-Not known because of confusion between day and month)

2)Not available(Ex: No educational fields if not studied at all)

3)Purposely withheld(Ex: home number not needed when you have personal contact)

SQL uses 3 valued logic unlike other programming langauges.

They are true, false and unknown.

To see and , or , unknown operations result:

**AND**

True && Unknown = Unknown

False && Unknown = False

Unknown && Unknown = Unknown

**OR**

True || Unknown = Unknown

False || Unknown = False

Unknown || Unknown = Unknown

**NOT**

Unknown ! = Unknown

**Comparison operators:**

**Is and Is not:**

There are used to compare an attribute value to null.

Query that can used as per Question given:

Select \* from employee where supervisors is NULL:

**SQL Join operators:**

**There are different types of joins:**

For performing joins we need to have atleast 2 or more tables.

**Inner join:**

This type of join returns those records which have matching values in both tables. So, if you perform an INNER JOIN operation between the Employee table and the Projects table, all the tuples which have matching values in both the tables will be given as output.

Syntax:

SELECT Table1.Column1, Table1.Column2, Table2.Column1,…

FROM Table 1

INNER JOIN Table2

ON Table1. MatchingColumnName = Table2.MatchingcolumnName;

**Self Join**

SELF JOIN in is a join of a table to itself. This implies that each row in a table is joined

with itself.

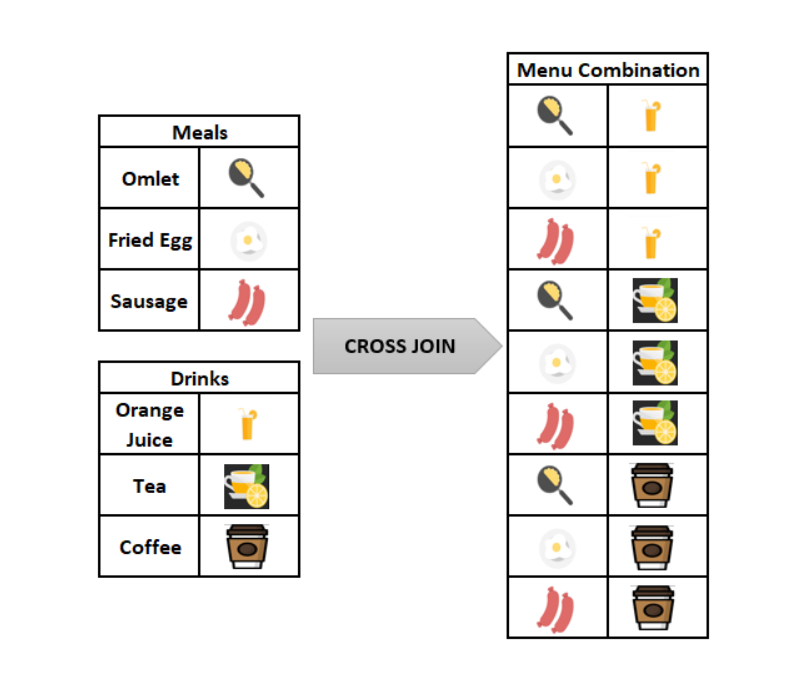
**Cross Join:**

The CROSS JOIN is a type of join in which a join clause is applied to each row of a table to

every row of the other table. Also, when the WHERE condition is used, this type of JOIN

behaves as an INNER JOIN, and when the WHERE condition is not present, it behaves like a

CARTESIAN product.



|  |  |
| --- | --- |
|  | SELECT ColumnName\_1,  ColumnName\_2,  ColumnName\_N  FROM [Table\_1]  CROSS JOIN [Table\_2] |

Or

SELECT ColumnName\_1,

       ColumnName\_2,

       ColumnName\_N

FROM [Table\_1],[Table\_2]

# Outer Join:

# It returns not only the matching rows but also the unmatched rows.

# There are three types of outer joins available which are Left, Right and full outer joins.

# While describing the FROM clause in the outer join, the first table which mentioned in the FROM clause is the left side and the second table mentioned in the FROM clause is the right side.

Syntax:

SELECT column\_list FROM table1 LEFT OUTER JOIN table2 on Join\_Condition

SELECT column\_list FROM table1 OUTER JOIN table2 on Join\_Condition

**Q4A:**

**SELECT DISTINCT** Pnumber

**FROM** PROJECT

**WHERE** Pnumber **IN**

( **SELECT** Pnumber

**FROM** PROJECT, DEPARTMENT, EMPLOYEE

**WHERE** Dnum=Dnumber **AND**

Mgr\_ssn=Ssn **AND** Lname=‘Smith’ )

**OR**

Pnumber **IN**

( **SELECT** Pno

**FROM** WORKS\_ON, EMPLOYEE

**WHERE** Essn=Ssn **AND** Lname=‘Smith’ );

The above query first evaluates the inner query and when a value is returned from the inner query it is used by the external query till the outer query is evaluated.

Any or Some operator

The In operator basically searches for the value that is in the parenthesis while the any or some operator searches for something matching like the given query.

**IN Operator:**

The IN Operator is basically a comparison and selection of v values from a set of V values.

It is to be noted that the Inner queries always run first and the final output is produced as part of the outer queries.

Q16

* NESTED QUERY – query within another query embedded within WHERE clause
* create tuple variables (aliases) for all the tables referenced in an SQL query
* Tuple: Single row of a table
* **Correlated Nested Queries:** a condition in the WHERE clause of a nested query references some attribute of the outer query, the two queries are said to be **correlated**.

# 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

# The EXISTS and UNIQUE Functions in SQL

# EXISTS:

# used to check whether the result of a correlated nested query is *empty*

# Results : True/False

# True : nested query result contains at least one tuple

# False: nested query result contains no tuples

* EXISTS and NOT EXISTS are typically used in conjunction with a correlated nested query.
* NOT EXISTS(Q) returns **TRUE** if there are *no tuples* in the result of nested query Q, and it returns **FALSE** otherwise.

# Query7: List the names of managers who have at least one dependent.

# 🡪by using EXISTS

**Q 17. Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.**

SELECT DISTINCT Essn FROM WORKS\_ON WHERE Pno IN (1, 2, 3);

* The SELECT DISTINCT statement is used to return only different values.
* SYNTAX

SELECT DISTINCT column1, column2, ...  
FROM table\_name;

**Q 19. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.**

SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary) FROM EMPLOYEE;

* **Various Aggregate Functions in SQL**

1) Count() **-**counts how many rows are there in particular column

2) Sum() – adds together all the values in a particular column

3) Avg() – calculates the average of a group of selected values

4) Min() – return the lowest value in a particular column

5) Max() - return the highest value in a particular column

**Q23: Count the number of distinct salary values in the database.**

**SELECT COUNT** (**DISTINCT** Salary) **FROM** EMPLOYEE;

* [COUNT()](https://www.w3resource.com/sql/aggregate-functions/COUNT-function.php) function with DISTINCT clause **🡪** eliminates the repetitive appearance of the same data. The DISTINCT can come only once in a given select statement.
* NULL values are discarded when aggregate functions are applied.

**Q 24 For each department, retrieve the department number, the number of employees in the department, and their average salary**.

* **SELECT** Dno, **COUNT** (\*), **AVG** (Salary) **FROM** EMPLOYEE **GROUP BY** Dno;

**GROUP BY** 🡪 groups rows that have the same values into summary rows, like "find the number of customers in each country".

* It is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

.

**HAVING CLAUSE -** Places the condition in the groups defined by the GROUP BY clause in the SELECT statement.

**Syntax**

**SELECT** column\_Name1, column\_Name2, ....., column\_NameN aggregate\_function\_name(column\_Name) **FROM** table\_name **GROUP** **BY** column\_Name1 **HAVING** condition;

**ASSERTIONS -** It is a statement in SQL that ensures a certain condition is always exists in database

**CREATE ASSERTION** SALARY\_CONSTRAINT **CHECK** ( NOT EXISTS ( **SELECT** \* **FROM** EMPLOYEE E, EMPLOYEE M, DEPARTMENT D **WHERE** E.Salary>M.Salary **AND** E.Dno=D.Dnumber **AND** D.Mgr\_ssn=M.Ssn ) );

* **NOT EXISTS** – Check the subquery for row existence

Return **TRUE** – If there are no rows

Return **FALSE -** If there are rows

* **CHECK**

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

**TRIGGERS**

It’s a procedure that gets invoked automatically when certain specified event occurs.

BEFORE and AFTER keyword is used to specify weather action to be taken before the event or after the event.

Trigger structure

Event – which events should invoke the trigger?

Condition – condition to be satisfied to perform the action

Action – action to be taken

Example:

**CREATE TRIGGER** SALARY\_VIOLATION

**BEFORE INSERT OR UPDATE OF** SALARY, //(event) When to trigger?

SUPERVISOR\_SSN **ON** EMPLOYEE

**FOR EACH ROW //**(condition)Condition to be checked

**WHEN** ( **NEW**.SALARY >

( **SELECT** SALARY **FROM** EMPLOYEE

**WHERE** SSN = **NEW**.SUPERVISOR\_SSN ) )

INFORM\_SUPERVISOR(**NEW**.Supervisor\_ssn,

**NEW**.Ssn ); //(action) Calling a function which is the action to be taken here

**VIEWS**

Instead of doing multiple join operations to retrieve data from different table at once,

We can alternatively create a virtual table holding values from other tables

Views are for viewing only.

Update operation won’t work here.

View table gets automatically updated when base table gets updated.

We can derive view from base tables as well as other views.

View table exist till we drop it.

Sample:

**CREATE VIEW** WORKS\_ON1 //view name

**AS SELECT** Fname, Lname, Pname, Hours

**FROM** EMPLOYEE, PROJECT, WORKS\_ON //selection of rows from base table

**WHERE** Ssn=Essn **AND** Pno=Pnumber; //condition

**SQL PROGRAM**

use sys;

CREATE TABLE Student(

studentID INT NOT NULL AUTO\_INCREMENT,

FName VARCHAR(20),

LName VARCHAR(20),

Address VARCHAR(30),

City VARCHAR(15),

Marks INT,

PRIMARY KEY(studentID)

);

create TABLE teacher(

teacherID INT NOT NULL AUTO\_INCREMENT,

FName VARCHAR(20),

LName VARCHAR(20),

Address VARCHAR(30),

PRIMARY KEY(teacherID)

);

CREATE TRIGGER calculate

before INSERT

ON student

FOR EACH ROW

SET new.marks = new.marks+100;

insert into Student (studentID, FName, LName, Address, City, Marks) values(4, "vimal" , "jose" , "kottayam", "kottayam", 5);

insert into teacher (teacherID, fName, lName, Address) values(4, "james" , "mathew" , "kottayam");

select \* from Student;

select \* from teacher;

CREATE VIEW newrecord AS

SELECT Student.Fname, teacher.LName

FROM Student, teacher;

select\* from newrecord;