Name: -Suyash Navanath Shinde Roll No: -D2223034

Course: - Database Management System Lab Batch: - B

Division: - TY CSF 2

Experiment No. -5

Aim - Write a Database Query for Joins, Nested queries, Sub-queries of Manufacturing industry / Hospital/ Company table**.**

Software Required - SQL Server 15.0 /16.0

Theory:-

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

1)SQL Joins:-

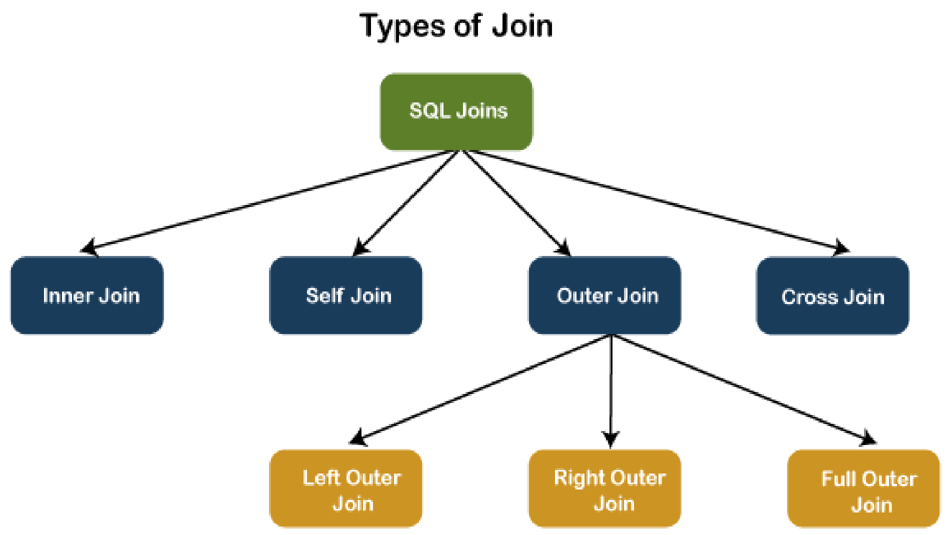
A SQL Join statement combines data or rows from two or more tables based on a common field between them. The join keyword merges two or more tables and creates a temporary image of the merged table. Then according to the conditions provided, it extracts the required data from the image table, and once data is fetched, the temporary image of the merged tables is dumped.

In a JOIN query, a condition indicates how two tables are related:

- Choose columns from each table that should be used in the join. A join condition indicates a foreign key from one table and its corresponding key in the other table.
- Specify the logical operator to compare values from the columns like =, <, or >.

Types of JOINS in SQL Server

SQL Server mainly supports **four types of JOINS**, and each join type defines how two tables are related in a query. The following are types of join supports in SQL Server:



a) **INNER JOIN**

This JOIN returns all records from multiple tables that satisfy the specified join condition. It is the simple and most popular form of join and assumes as a default join. If we omit the INNER keyword with the JOIN query, we will get the same output.

INNER JOIN Syntax

The following syntax illustrates the use of INNER JOIN in SQL Server:

SELECT columns

FROM table1

INNER JOIN table2 ON condition1

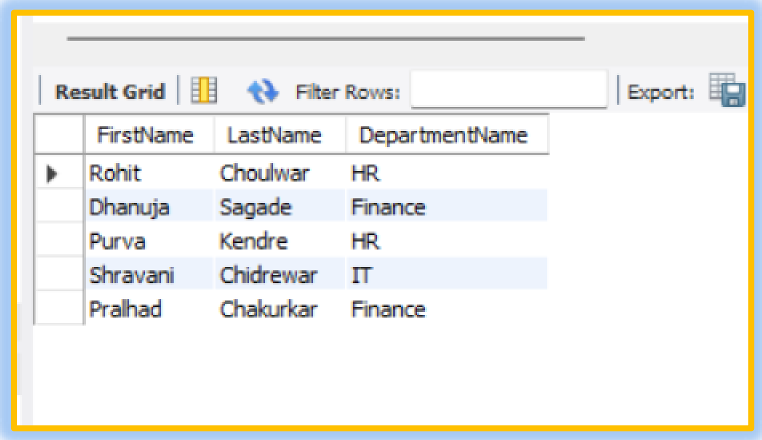
INNER JOIN table3 ON condition2

SQL QUERY: -

```
USE mydatabase;
-- Create an Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT
);

-- Insert dummy data into the Employees table
INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID)
VALUES
```

```
    (1, 'Rohit', 'Choulwar', 1),
    (2, 'Dhanuja', 'Sagade', 2),
    (3, 'Purva', 'Kendre', 1),
    (4, 'Shravani', 'Chidrewar', 3),
    (5, 'Pralhad', 'Chakurkar', 2);

-- Create a Departments table
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

-- Insert dummy data into the Departments table
INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
    (1, 'HR'),
    (2, 'Finance'),
    (3, 'IT');

-- Perform an INNER JOIN to retrieve employee names and their department names
SELECT Employees.FirstName, Employees.LastName, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

OUTPUT: -



FirstName	LastName	DepartmentName
Rohit	Choulwar	HR
Dhanuja	Sagade	Finance
Purva	Kendre	HR
Shravani	Chidrewar	IT
Pralhad	Chakurkar	Finance

b) SELF JOIN

A table is joined to itself using the SELF JOIN. It means that each table row is combined with itself and with every other table row. The SELF JOIN can be thought of as a JOIN of two copies of the same tables. We can do this with the help of table name aliases to assign a specific name to each table's instance. The table aliases enable us to use the table's temporary name that we are going to use in the query.

SELF JOIN Syntax

SELECT T1.col_name, T2.col_name...

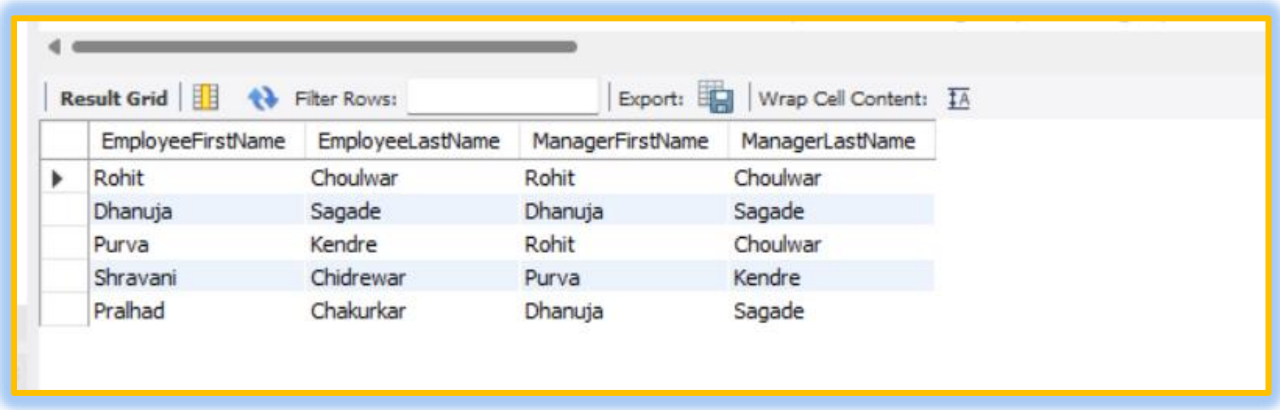
FROM table1 T1, table1 T2

WHERE join_condition;

SQL QUERY: -

```sql
USE mydatabase;

-- Create an Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    ManagerID INT
);

-- Insert updated dummy data into the Employees table
-- 1, 'Rohit', 'Choulwar', 1: Rohit is the top-level manager (no manager)
-- 2, 'Dhanuja', 'Sagade', 2: Dhanuja's manager is Rohit
-- 3, 'Purva', 'Kendre', 1: Purva's manager is Rohit
-- 4, 'Shravani', 'Chidrewar', 3: Shravani's manager is Purva
-- 5, 'Pralhad', 'Chakurkar', 2: Pralhad's manager is Dhanuja
INSERT INTO Employees (EmployeeID, FirstName, LastName, ManagerID)
VALUES
    (1, 'Rohit', 'Choulwar', 1),
    (2, 'Dhanuja', 'Sagade', 2),
    (3, 'Purva', 'Kendre', 1),
    (4, 'Shravani', 'Chidrewar', 3),
    (5, 'Pralhad', 'Chakurkar', 2);

-- Perform a SELF JOIN to retrieve employee information and their managers
SELECT e1.FirstName AS EmployeeFirstName, e1.LastName AS EmployeeLastName,
       e2.FirstName AS ManagerFirstName, e2.LastName AS ManagerLastName
FROM Employees e1
LEFT JOIN Employees e2 ON e1.ManagerID = e2.EmployeeID;
```

OUTPUT: -



EmployeeFirstName	EmployeeLastName	ManagerFirstName	ManagerLastName
Rohit	Choulwar	Rohit	Choulwar
Dhanuja	Sagade	Dhanuja	Sagade
Purva	Kendre	Rohit	Choulwar
Shravani	Chidrewar	Purva	Kendre
Pralhad	Chakurkar	Dhanuja	Sagade

c) CROSS JOIN

CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as CARTESIAN JOIN because it produces the Cartesian product of all linked tables.

CROSS JOIN Syntax

SELECT column_lists

FROM table1

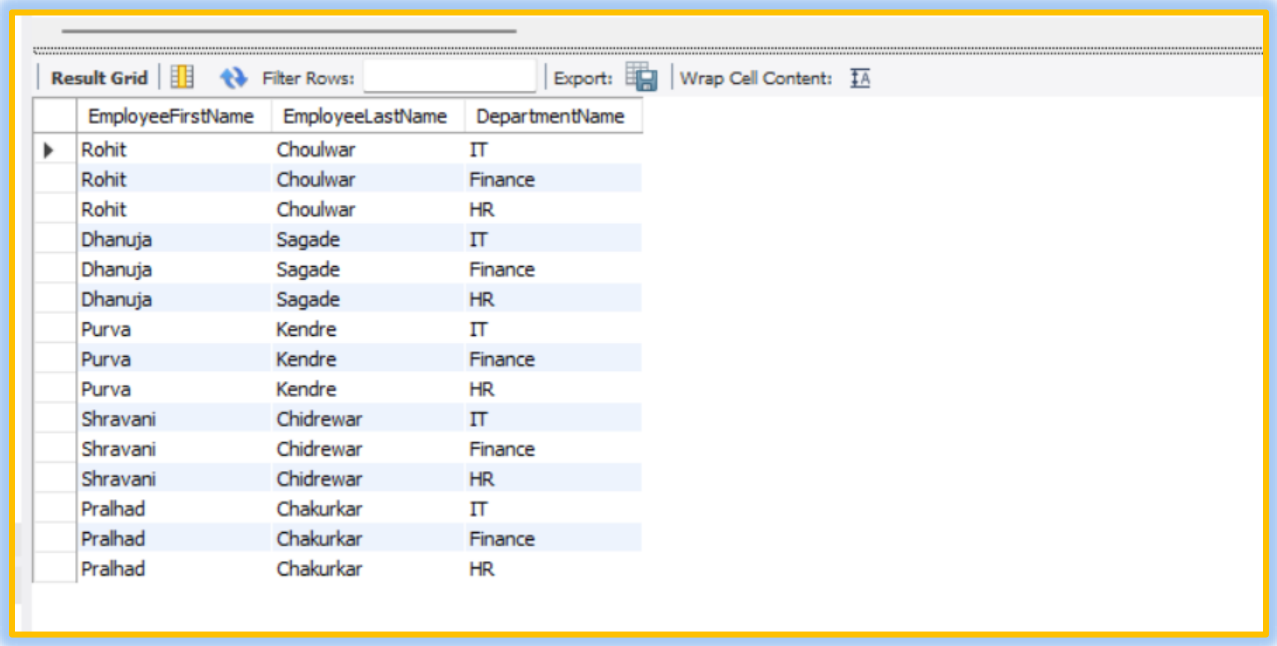
CROSS JOIN table2;

SQL QUERY: -

```
USE mydatabase;

-- Create an Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    ManagerID INT
);

-- Insert updated dummy data into the Employees table
INSERT INTO Employees (EmployeeID, FirstName, LastName, ManagerID)
VALUES
    (1, 'Rohit', 'Choulwar', 1),
    (2, 'Dhanuja', 'Sagade', 2),
    (3, 'Purva', 'Kendre', 1),
```

```sql
    (4, 'Shravani', 'Chidrewar', 3),
    (5, 'Pralhad', 'Chakurkar', 2);

-- Create a Departments table
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

-- Insert dummy data into the Departments table
INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
    (1, 'HR'),
    (2, 'Finance'),
    (3, 'IT');

-- Perform a CROSS JOIN to generate all combinations of employee names and
departments
SELECT Employees.FirstName AS EmployeeFirstName, Employees.LastName AS
EmployeeLastName,
       Departments.DepartmentName
FROM Employees
CROSS JOIN Departments;
```

OUTPUT: -



EmployeeFirstName	EmployeeLastName	DepartmentName
Rohit	Choulwar	IT
Rohit	Choulwar	Finance
Rohit	Choulwar	HR
Dhanuja	Sagade	IT
Dhanuja	Sagade	Finance
Dhanuja	Sagade	HR
Purva	Kendre	IT
Purva	Kendre	Finance
Purva	Kendre	HR
Shravani	Chidrewar	IT
Shravani	Chidrewar	Finance
Shravani	Chidrewar	HR
Pralhad	Chakurkar	IT
Pralhad	Chakurkar	Finance
Pralhad	Chakurkar	HR

d) OUTER JOIN

OUTER JOIN in SQL Server returns all records from both tables that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.

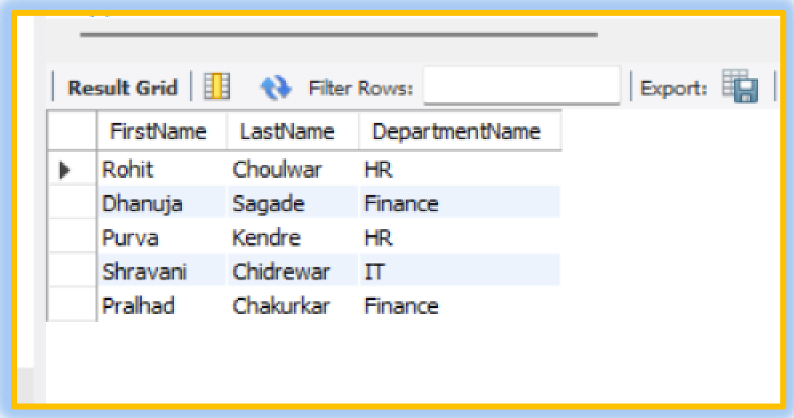
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

SQL QUERY: -

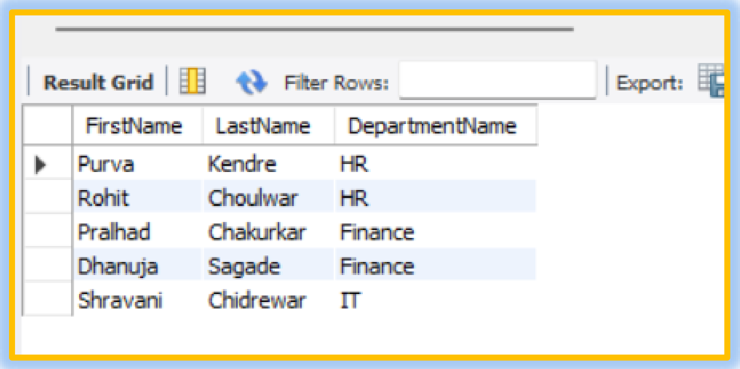
```
USE mydatabase;

-- Create an Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT
);

-- Insert updated dummy data into the Employees table
INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID)
VALUES
    (1, 'Rohit', 'Choulwar', 1),
    (2, 'Dhanuja', 'Sagade', 2),
    (3, 'Purva', 'Kendre', 1),
    (4, 'Shravani', 'Chidrewar', 3),
    (5, 'Pralhad', 'Chakurkar', 2);

-- Create a Departments table
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

-- Insert dummy data into the Departments table
INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
    (1, 'HR'),
    (2, 'Finance'),
    (3, 'IT');

-- Perform a LEFT OUTER JOIN to retrieve all employees and their corresponding
departments (including unmatched employees)
SELECT Employees.FirstName, Employees.LastName, Departments.DepartmentName
FROM Employees
```

```
LEFT OUTER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

-- Perform a RIGHT OUTER JOIN to retrieve all departments and their corresponding
employees (including unmatched departments)
SELECT Employees.FirstName, Employees.LastName, Departments.DepartmentName
FROM Employees
RIGHT OUTER JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;

-- Perform a FULL OUTER JOIN to retrieve all employees and departments, including
unmatched rows from both tables
SELECT Employees.FirstName, Employees.LastName, Departments.DepartmentName
FROM Employees
FULL OUTER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

OUTPUT: -



FirstName	LastName	DepartmentName
Rohit	Choulwar	HR
Dhanuja	Sagade	Finance
Purva	Kendre	HR
Shravani	Chidrewar	IT
Pralhad	Chakurkar	Finance



FirstName	LastName	DepartmentName
Purva	Kendre	HR
Rohit	Choulwar	HR
Pralhad	Chakurkar	Finance
Dhanuja	Sagade	Finance
Shravani	Chidrewar	IT

3)Nested Queries :-

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use STUDENT, COURSE, STUDENT_COURSE tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1

S3	C2
S4	C2
S4	C3

In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.
IN: If we want to find out **S_ID** who are enrolled in **C_NAME** 'DSA' or 'DBMS', we can write it with the help of independent nested query and IN operator. From **COURSE** table, we can find out **C_ID** for **C_NAME** 'DSA' or DBMS' and we can use these **C_ID**s for finding **S_ID**s from **STUDENT_COURSE** TABLE.

STEP 1: Finding **C_ID** for **C_NAME** ='DSA' or 'DBMS'
Select **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME** = 'DBMS'

STEP 2: Using **C_ID** of step 1 for finding **S_ID**
Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN
(SELECT **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME**='DBMS');

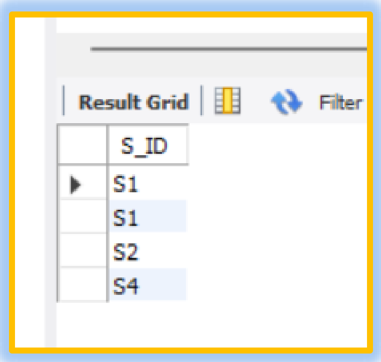
Output :- The inner query will return a set with members C1 and C3 and outer query will return those S_IDs for which C_ID is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

SQL QUERY: -

```sql
USE mydatabase;
-- Create a COURSE table
CREATE TABLE COURSE (
    C_ID INT PRIMARY KEY,
    C_NAME VARCHAR(50)
);

-- Insert dummy data into the COURSE table
INSERT INTO COURSE (C_ID, C_NAME)
VALUES
    (1, 'DSA'),
    (2, 'Programming'),
    (3, 'DBMS');

-- Create a STUDENT_COURSE table
```

```sql
CREATE TABLE STUDENT_COURSE (
    S_ID VARCHAR(2),
    C_ID INT
);

-- Insert dummy data into the STUDENT_COURSE table
INSERT INTO STUDENT_COURSE (S_ID, C_ID)
VALUES
    ('S1', 1),
    ('S1', 3),
    ('S2', 1),
    ('S3', 2),
    ('S4', 2),
    ('S4', 3);

-- Find S_IDs of students enrolled in 'DSA' or 'DBMS' using an independent nested
query
SELECT S_ID
FROM STUDENT_COURSE
WHERE C_ID IN (
    SELECT C_ID
    FROM COURSE
    WHERE C_NAME = 'DSA' OR C_NAME = 'DBMS'
);
```

OUTPUT: -



S_ID
S1
S1
S2
S4

4)SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause.

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

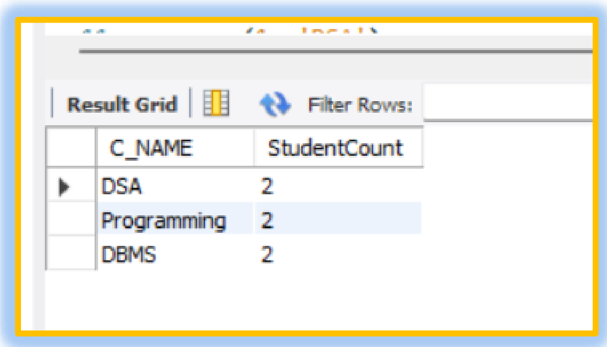
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

a). Subqueries with the Select Statement

- SQL subqueries are most frequently used with the Select statement.
- Syntax:-
  SELECT column_name
  FROM table_name
  WHERE column_name expression operator
  (SELECT column_name from table_name WHERE ...);

<u>SQL QUERY: -</u>

```sql
USE studentinfo;
-- Create a COURSE table
CREATE TABLE COURSE (
    C_ID INT PRIMARY KEY,
    C_NAME VARCHAR(50)
);

-- Insert dummy data into the COURSE table
INSERT INTO COURSE (C_ID, C_NAME)
VALUES
    (1, 'DSA'),
    (2, 'Programming'),
    (3, 'DBMS');

-- Create a STUDENT_COURSE table
CREATE TABLE STUDENT_COURSE (
    S_ID VARCHAR(2),
    C_ID INT
);

-- Insert dummy data into the STUDENT_COURSE table
INSERT INTO STUDENT_COURSE (S_ID, C_ID)
VALUES
    ('S1', 1),
    ('S1', 3),
    ('S2', 1),
    ('S3', 2),
    ('S4', 2),
    ('S4', 3);

-- Use a subquery within the SELECT statement to count the number of students in
each course
SELECT C_NAME,
       (SELECT COUNT(DISTINCT S_ID) FROM STUDENT_COURSE WHERE C_ID = COURSE.C_ID)
AS StudentCount
FROM COURSE;
```



```
C_NAME        StudentCount
DSA           2
Programming   2
DBMS          2
```

<u>OUTPUT: -</u>

b) Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

- Syntax:
  INSERT INTO table_name (column1, column2, column3....)
  SELECT *
  FROM table_name WHERE VALUE OPERATOR

<u>SQL QUERY: -</u>

```sql
USE studentinfo;

-- Create a COURSE table
CREATE TABLE COURSE (
    C_ID INT PRIMARY KEY,
    C_NAME VARCHAR(50)
);

-- Insert dummy data into the COURSE table
INSERT INTO COURSE (C_ID, C_NAME)
VALUES
    (1, 'DSA'),
    (2, 'Programming'),
    (3, 'DBMS');

-- Create a STUDENT_COURSE table
CREATE TABLE STUDENT_COURSE (
    S_ID VARCHAR(2),
    C_ID INT
);

-- Insert dummy data into the STUDENT_COURSE table
INSERT INTO STUDENT_COURSE (S_ID, C_ID)
VALUES
    ('S1', 1),
    ('S1', 3),
    ('S2', 1),
    ('S3', 2),
    ('S4', 2),
    ('S4', 3);

-- Create a new table to insert data
CREATE TABLE COURSE_ENROLLMENT (
    S_ID VARCHAR(2),
    C_ID INT
);

-- Use a subquery within the INSERT statement to select data from STUDENT_COURSE
-- and insert it into COURSE_ENROLLMENT
INSERT INTO COURSE_ENROLLMENT (S_ID, C_ID)
SELECT S_ID, C_ID
FROM STUDENT_COURSE
WHERE C_ID = (SELECT C_ID FROM COURSE WHERE C_NAME = 'Programming');
```

c) Subqueries with the UPDATE Statement
The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

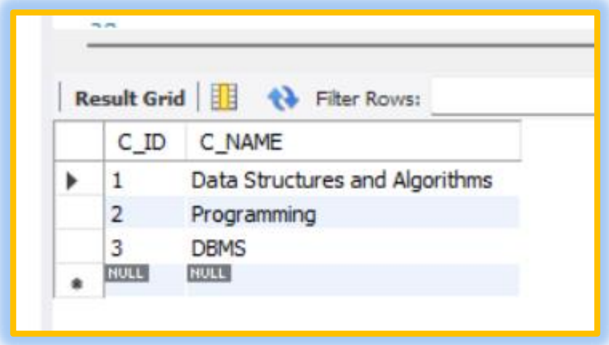
Syntax

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

SQL QUERY: -

```sql
USE studentinfo;

-- Create a COURSE table
CREATE TABLE COURSE (
    C_ID INT PRIMARY KEY,
    C_NAME VARCHAR(50)
);

-- Insert dummy data into the COURSE table
INSERT INTO COURSE (C_ID, C_NAME)
VALUES
    (1, 'DSA'),
    (2, 'Programming'),
    (3, 'DBMS');

-- Create a STUDENT_COURSE table
CREATE TABLE STUDENT_COURSE (
    S_ID VARCHAR(2),
    C_ID INT
);

-- Insert dummy data into the STUDENT_COURSE table
INSERT INTO STUDENT_COURSE (S_ID, C_ID)
VALUES
    ('S1', 1),
    ('S1', 3),
    ('S2', 1),
    ('S3', 2),
    ('S4', 2),
    ('S4', 3);

-- Use a subquery within the UPDATE statement to update the 'C_NAME' column in the 'COURSE' table
```

```
UPDATE COURSE
SET C_NAME = 'Data Structures and Algorithms'
WHERE C_ID = (SELECT C_ID FROM STUDENT_COURSE WHERE S_ID = 'S1' LIMIT 1);

select * from course;
```



```
C_ID  C_NAME
1     Data Structures and Algorithms
2     Programming
3     DBMS
NULL  NULL
```

OUTPUT: -

d) Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

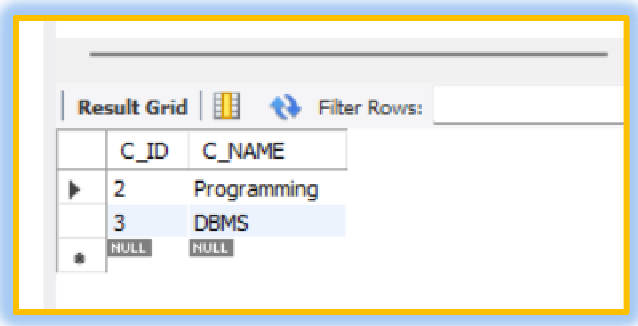
Syntax

```
DELETE FROM TABLE_NAME
WHERE VALUE OPERATOR
   (SELECT COLUMN_NAME
   FROM TABLE_NAME
   WHERE condition);
```

SQL QUERY: -

```
USE studentinfo;

-- Create a COURSE table
CREATE TABLE COURSE (
    C_ID INT PRIMARY KEY,
    C_NAME VARCHAR(50)
);

-- Insert dummy data into the COURSE table
INSERT INTO COURSE (C_ID, C_NAME)
```

```sql
VALUES
    (1, 'DSA'),
    (2, 'Programming'),
    (3, 'DBMS');

-- Create a STUDENT_COURSE table
CREATE TABLE STUDENT_COURSE (
    S_ID VARCHAR(2),
    C_ID INT
);

-- Insert dummy data into the STUDENT_COURSE table
INSERT INTO STUDENT_COURSE (S_ID, C_ID)
VALUES
    ('S1', 1),
    ('S1', 3),
    ('S2', 1),
    ('S3', 2),
    ('S4', 2),
    ('S4', 3);

-- Use a subquery within the DELETE statement to delete records from the 'COURSE'
table
DELETE FROM COURSE
WHERE C_ID = (SELECT C_ID FROM STUDENT_COURSE WHERE S_ID = 'S1' LIMIT 1);

select *from course;
```



C_ID	C_NAME
2	Programming
3	DBMS
NULL	NULL

OUTPUT: -

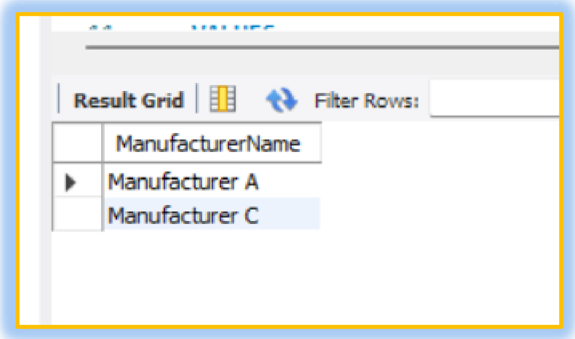
Exercise (Perform anyone in addition to assignment title)

1) **Write SQL nested, sub queries for Manufacturing Industry.**

SQL QUERY: -

```sql
CREATE DATABASE db1;
USE db1;
-- Create a Manufacturers table
CREATE TABLE Manufacturers (
    ManufacturerID INT PRIMARY KEY,
    ManufacturerName VARCHAR(50)
);

-- Insert dummy data into the Manufacturers table
INSERT INTO Manufacturers (ManufacturerID, ManufacturerName)
VALUES
    (1, 'Manufacturer A'),
    (2, 'Manufacturer B'),
    (3, 'Manufacturer C');

-- Create a Products table
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(50),
    Price DECIMAL(10, 2),
    ManufacturerID INT
);

-- Insert dummy data into the Products table
INSERT INTO Products (ProductID, ProductName, Price, ManufacturerID)
VALUES
    (1, 'Product 1', 100.00, 1),
    (2, 'Product 2', 150.00, 1),
    (3, 'Product 3', 80.00, 2),
    (4, 'Product 4', 200.00, 3);

-- Use nested subqueries to find manufacturers producing products with prices
higher than the average price
SELECT ManufacturerName
FROM Manufacturers
WHERE ManufacturerID IN (
    SELECT DISTINCT ManufacturerID
    FROM Products
    WHERE Price > (
        SELECT AVG(Price)
        FROM Products
    )
);
```

OUTPUT: -



Result Grid | Filter Rows:

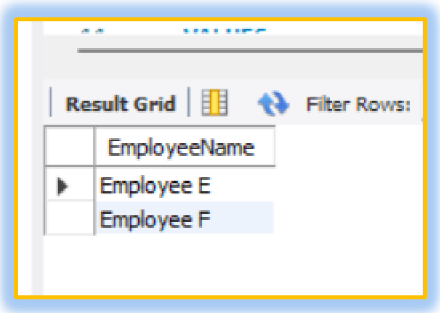
```
ManufacturerName
Manufacturer A
Manufacturer C
```

2) **Write SQL nested, sub queries for Company databases.**

SQL QUERY: -

```
CREATE DATABASE db2;
USE db2;
-- Create a Departments table
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

-- Insert dummy data into the Departments table
INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
    (1, 'HR'),
    (2, 'Finance'),
    (3, 'Engineering');

-- Create an Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
```

```sql
    EmployeeName VARCHAR(50),
    Salary DECIMAL(10, 2),
    DepartmentID INT
);

-- Insert dummy data into the Employees table
INSERT INTO Employees (EmployeeID, EmployeeName, Salary, DepartmentID)
VALUES
    (1, 'Employee A', 50000.00, 1),
    (2, 'Employee B', 60000.00, 1),
    (3, 'Employee C', 75000.00, 2),
    (4, 'Employee D', 80000.00, 2),
    (5, 'Employee E', 70000.00, 3),
    (6, 'Employee F', 85000.00, 3);

-- Use nested subqueries to find employees in the department with the highest
average salary
SELECT EmployeeName
FROM Employees
WHERE DepartmentID = (
    SELECT DepartmentID
    FROM (
        SELECT DepartmentID, AVG(Salary) AS AvgSalary
        FROM Employees
        GROUP BY DepartmentID
        ORDER BY AvgSalary DESC
        LIMIT 1
    ) AS HighestAvgSalaryDept
);
```



```
EmployeeName
Employee E
Employee F
```

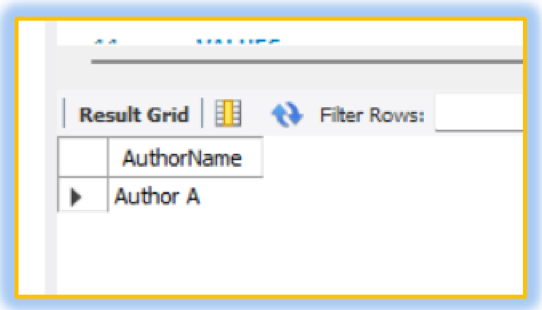
OUTPUT: -

3) Write SQL nested, sub queries for Library management system.

SQL QUERY: -

```sql
CREATE DATABASE db3;
```

```sql
USE db3;
-- Create an Authors table
CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY,
    AuthorName VARCHAR(50)
);

-- Insert dummy data into the Authors table
INSERT INTO Authors (AuthorID, AuthorName)
VALUES
    (1, 'Author A'),
    (2, 'Author B'),
    (3, 'Author C');

-- Create a Books table
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    BookTitle VARCHAR(100),
    Genre VARCHAR(50)
);

-- Insert dummy data into the Books table
INSERT INTO Books (BookID, BookTitle, Genre)
VALUES
    (1, 'Book 1', 'Mystery'),
    (2, 'Book 2', 'Science Fiction'),
    (3, 'Book 3', 'Mystery'),
    (4, 'Book 4', 'Romance');

-- Create a BookAuthors table to associate books with authors
CREATE TABLE BookAuthors (
    BookID INT,
    AuthorID INT
);

-- Insert dummy data into the BookAuthors table
INSERT INTO BookAuthors (BookID, AuthorID)
VALUES
    (1, 1),
    (2, 2),
    (3, 1),
    (4, 3);

-- Use nested subqueries to find authors of books with the 'Mystery' genre
SELECT AuthorName
FROM Authors
WHERE AuthorID IN (
    SELECT AuthorID
    FROM BookAuthors
    WHERE BookID IN (
        SELECT BookID
        FROM Books
        WHERE Genre = 'Mystery'
    )
);
```



```
AuthorName
Author A
```

OUTPUT: -

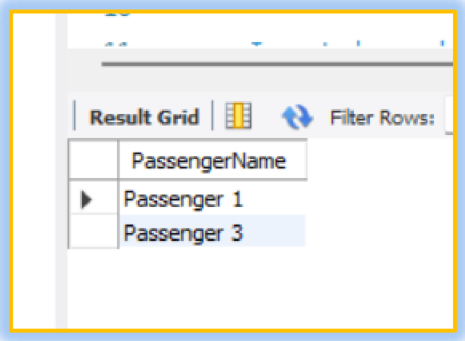
4) Write SQL nested, sub queries for Online Railway reservation system.

SQL QUERY: -

```
CREATE DATABASE db4;
USE db4;

-- Create a Trains table
CREATE TABLE Trains (
    TrainID INT PRIMARY KEY,
    TrainName VARCHAR(50),
    Destination VARCHAR(50)
);

-- Insert dummy data into the Trains table
INSERT INTO Trains (TrainID, TrainName, Destination)
VALUES
    (1, 'Train A', 'City X'),
    (2, 'Train B', 'City Y'),
    (3, 'Train C', 'City Z');

-- Create a Passengers table
CREATE TABLE Passengers (
    PassengerID INT PRIMARY KEY,
    PassengerName VARCHAR(50)
);
```

```sql
-- Insert dummy data into the Passengers table
INSERT INTO Passengers (PassengerID, PassengerName)
VALUES
    (1, 'Passenger 1'),
    (2, 'Passenger 2'),
    (3, 'Passenger 3');

-- Create a Reservations table to associate passengers with trains
CREATE TABLE Reservations (
    ReservationID INT PRIMARY KEY,
    TrainID INT,
    PassengerID INT
);

-- Insert dummy data into the Reservations table
INSERT INTO Reservations (ReservationID, TrainID, PassengerID)
VALUES
    (1, 1, 1),
    (2, 2, 2),
    (3, 1, 3),
    (4, 3, 2);

-- Use nested subqueries to find passengers who have reserved a seat on a train
with the destination 'City X'
SELECT PassengerName
FROM Passengers
WHERE PassengerID IN (
    SELECT PassengerID
    FROM Reservations
    WHERE TrainID IN (
        SELECT TrainID
        FROM Trains
        WHERE Destination = 'City X'
    )
);
```

OUTPUT: -


Result Grid

PassengerName
Passenger 1
Passenger 3

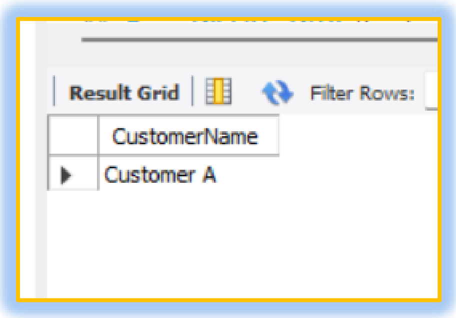
5) **Write SQL nested sub queries for any E-commerce website.**

SQL QUERY: -

```
CREATE DATABASE db5;
USE db5;
-- Create a Products table
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Category VARCHAR(50)
);

-- Insert dummy data into the Products table
INSERT INTO Products (ProductID, ProductName, Category)
VALUES
    (1, 'Product 1', 'Electronics'),
    (2, 'Product 2', 'Clothing'),
    (3, 'Product 3', 'Electronics'),
    (4, 'Product 4', 'Books');

-- Create a Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50)
);

-- Insert dummy data into the Customers table
INSERT INTO Customers (CustomerID, CustomerName)
VALUES
    (1, 'Customer A'),
    (2, 'Customer B'),
    (3, 'Customer C');

-- Create an Orders table to associate customers with products
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
```

```sql
    ProductID INT
);

-- Insert dummy data into the Orders table
INSERT INTO Orders (OrderID, CustomerID, ProductID)
VALUES
    (1, 1, 1),
    (2, 2, 2),
    (3, 1, 3),
    (4, 3, 2);

-- Use nested subqueries to find customers who have placed an order for a product
with the category 'Electronics'
SELECT CustomerName
FROM Customers
WHERE CustomerID IN (
    SELECT CustomerID
    FROM Orders
    WHERE ProductID IN (
        SELECT ProductID
        FROM Products
        WHERE Category = 'Electronics'
    )
);
```

OUTPUT: -



```
CustomerName
Customer A
```

FAQ

1) Which of the following is true about sub-queries?

A. They execute after the main query executes
B. They execute in parallel to the main query
C. The user can execute the main query and then, if wanted, execute the sub-query
D. They execute before the main query executes.

Ans- They Executes before the main query executes

2) **Which of the following is true about sub-queries?**

A. They execute after the main query executes
B. They execute in parallel to the main query
C. The user can execute the main query and then, if wanted, execute the sub-query
D. They execute before the main query executes.

Ans- They Execute before the main query executes

3) **Which of the following clause is mandatorily used in a sub-query?**

A. SELECT
B. WHERE
C. ORDER BY
D. GROUP BY

Ans- WHERE

4) **Which of the following is a method for writing a sub-query in a main query?**

A. By using JOINS
B. By using WHERE clause
C. By using the GROUP BY clause
D. By writing a SELECT statement embedded in the clause of another SELECT statement

Ans- By Writing a SELECT statement embedded in the clause of another SELECT statement

5) **Which of the following multi-row operators can be used with a sub-query?**

A. IN
B. ANY
C. ALL
D. All of the above

Ans- All of the above

.