EXPERIMENT-3

Lab 3: To implement and use SQL Sub-Query

Objective: To understand the use of SQL Subquery.

Lab Performance Questions

Tables:

1. Student_Details
   - Columns: Student_RollNo, Stu_Name, Stu_Marks, Stu_City

2. Faculty_Details
   - Columns: Faculty_ID, Name, Dept_ID, Address

3. Department
   - Columns: Dept_ID, Faculty_ID, Dept_Name

4. Old_Employee
   - Columns: Emp_ID, Emp_Name, Emp_Salary, Address

5. New_Employee
   - Columns: Emp_ID, Emp_Name, Emp_Salary, Address

6. Employee_Details
   - Columns: Emp_ID, Emp_Name, Emp_Salary, Dept_ID

7. Student
   - Columns: Student_ID, Name, City

8. Student2
   - Columns: Student_ID, City

9. Orders
   - Columns: Order_ID, Cust_ID, Order_Date

10. Sales
   - Columns: Product_Category, Sales_Amount

11. Top_Students
   - Columns: Student_ID, Top_Marks

SQL Subquery Questions:

1. Basic **SELECT** Subquery
   - Retrieve the names and marks of students whose marks are greater than the average marks.

2. **IN Operator** Subquery
   - Fetch the names and addresses of faculties who belong to departments in either 'Noida' or 'Gurgaon.'

3. **INSERT** with Subquery
   - Insert the details of employees from the "Old_Employee" table into the "New_Employee" table, considering only those with a salary greater than 40000.

4. **UPDATE** with Subquery:
   - Increase the salary of employees by 10% for those whose department grade is 'A.'

5. **DELETE** with Subquery:
   - Delete the records of employees whose department grade is 'C.'

6. Subquery in **WHERE Clause - NOT IN**
- Retrieve the names of students who do not belong to the city 'Los Angeles' based on data from the "Student2" table.

7. Subquery in **FROM** Clause:
- Use a subquery in the FROM clause to create a derived table showing the maximum, minimum, and average number of items for each order.

8. Correlated **Subquery**
- Fetch the names, cities, and incomes of employees whose income is higher than the average income of employees in their respective cities.

9. **EXISTS** Operator
- Retrieve the name, occupation, and age of customers who have placed at least one order using the EXISTS operator.

10. **ROW** Subquery
- Retrieve all columns for customers whose (cust_id, occupation) matches any row of (order_id, order_date) from the "Orders" table.

11. Subquery with **ALL Operator**
- Find customers whose cust_id is greater than all cust_ids in the "Orders" table.

12. **Nested Subqueries**
- Write an SQL query using a subquery within another subquery to retrieve names and salaries of employees who earn more than the average salary of employees in the "IT" department.

13. Subquery in **HAVING Clause**
- Using the "Sales" table, find the total sales amount for each product category, displaying only those categories where the total sales amount is greater than the average total sales amount.

14. Subquery with **GROUP BY**
- Retrieve department names and the count of employees in each department, filtering out departments where the count is less than 5.

15. Multiple **Subqueries in a Single Query**
- Write an SQL query involving multiple subqueries to retrieve information about employees based on various conditions.

16. Subquery with **BETWEEN** Operators
- Find the names and marks of students who scored between 80 and 90.

17. Subquery in **INSERT Statement**
- Insert the details of employees from the "Old_Employee" table into the "New_Employee" table, considering only those with a salary greater than the average salary of all employees.

18. Subquery with **ANY Operator**
- Retrieve the names of students who scored more marks than ANY student from the "Top_Students" table.

19. Subquery in **UPDATE** Statement
- Update the salary of employees by 5% for those whose department is 'HR' and the salary is less than the average salary of HR department employees.

20. Subquery with **NOT EXISTS**
- Fetch the names and ages of customers who have NOT placed any orders based on the data from the "Orders" table.

MySQL Subquery

A subquery in MySQL is nested into another SQL query and embedded with SELECT, INSERT, UPDATE, or DELETE statements and the various operators. We can also nest the subquery with another subquery. A subquery is known as the **inner query**, and the query that contains a subquery is known as the **outer query**. The inner query executed first gives the result to the outer query, and then the main/outer query will be performed. MySQL allows us to use subquery anywhere but must be closed within parenthesis. All subquery forms and operations supported by the SQL standard will also be supported in MySQL.

The following are the rules to use subqueries:

- Subqueries should always be in **parentheses.**
- If the main query does not have multiple columns for the subquery, then a subquery can have only one column in the SELECT command.
- We can use various comparison operators with the subquery, such as >, <, =, IN, ANY, SOME, and ALL. A multiple-row operator is very useful when the subquery returns more than one row.
- We cannot use the **ORDER BY** clause in a subquery, although it can be used inside the main query.
- If we use a subquery in a **set function**, it cannot be immediately enclosed in a set function.

The following are the advantages of using subqueries:

- The subqueries make the queries in a structured form that allows us to isolate each part of a statement.
- The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.
- The subqueries are more readable than complex join or union statements.

SQL Subquery

The Subquery or Inner query is an SQL query placed inside another SQL query. It is embedded in the HAVING or WHERE clause of the SQL statements.

Following are the important rules which the SQL Subquery must follow:

1. The SQL subqueries can be used with the following statements along with the SQL expression operators:

    - SELECT statement,
    - UPDATE statement,
    - INSERT statement, and
    - DELETE statement.

2. The subqueries in SQL are always enclosed in the parenthesis and placed on the right side of the SQL operators.

3. We cannot use the ORDER BY clause in the subquery. But, we can use the GROUP BY clause, which performs the same function as the ORDER BY clause.

4. If the subquery returns more than one record, we must use the multiple value operators before the subquery.

5. We can use the BETWEEN operator within the subquery but not with the subquery.

Subquery with SELECT statement

In SQL, the SELECT statement uses inner or nested queries most frequently. The syntax of the subquery with the SELECT statement is described in the following block:

```
SELECT Column_Name1, Column_Name2, ...., Column_NameN
FROM Table_Name WHERE Column_Name Comparison_Operator
( SELECT Column_Name1, Column_Name2, ...., Column_NameN
FROM Table_Name WHERE condition;
```

Examples of Subquery with the SELECT Statement

Example 1: This example uses the **Greater than comparison operator** with the subquery.

Let's take the table Student_Details, which contains Student_RollNo., Stu_Name, Stu_Marks, and Stu_City column.

Student_RollNo.	Stu_Name	Stu_Marks	Stu_City
1001	Akhil	85	Agra
1002	Balram	78	Delhi
1003	Bheem	87	Gurgaon
1004	Chetan	95	Noida
1005	Diksha	99	Agra
1006	Raman	90	Ghaziabad
1007	Sheetal	68	Delhi

The following SQL query returns the record of those students whose marks are greater than the average total marks:

SELECT * FROM Student_Details WHERE Stu_Marks> (SELECT AVG(Stu_Marks) FROM Student_ Details);

Output:

Student_RollNo.	Stu_Name	Stu_Marks	Stu_City
1003	Bheem	87	Gurgaon
1004	Chetan	95	Noida
1005	Diksha	99	Agra
1006	Raman	90	Ghaziabad

Example 2: This example uses the **IN operator** with the subquery.

Let's take the following two tables named **Faculty_Details** and **Department** tables. The **Faculty_Details** table contains the ID, Name, Dept_ID, and address of faculties. The Department table contains the Dept_ID, Faculty_ID, and Dept_Name.

Faculty_ID	Name	Dept_ID	Address
101	Bheem	1	Gurgaon
102	Chetan	2	Noida
103	Diksha	NULL	Agra
104	Raman	4	Ghaziabad
105	Yatin	3	Noida
106	Anuj	NULL	Agra
107	Rakes	5	Gurgaon

Dept_ID	Faculty_ID	Dept_Name
1	101	BCA
2	102	B.Tech
3	105	BBA
4	104	MBA
5	107	MCA

SELECT * FROM Department WHERE Faculty_ID IN (
SELECT Faculty_ID FROM Faculty WHERE City = 'Noida' OR City = 'Gurgaon') ;

Output:

Dept_ID	Faculty_ID	Dept_Name
1	101	BCA
2	102	B.Tech
3	105	BBA
5	107	MCA

Subquery with the INSERT statement

We can also use the subqueries and nested queries with the INSERT statement in Structured Query Language. We can insert the subquery results into the table of the outer query. The syntax of the subquery with the INSERT statement is described in the following block:

INSERT INTO Table_Name SELECT * FROM Table_Name WHERE Column_Name Operator (Subquery);

Examples of Subquery with the INSERT Statement

Example1: This example inserts the record of one table into another table using subquery with **WHERE clause**.

Let's take Old_Employee and New_Employee tables. The Old_Employee and New_Employee table contain the same number of columns. But, both the tables contain different records.

Table: Old_Employee

Emp_ID	Emp_Name	Emp_Salary	Address
1001	Akhil	50000	Agra
1002	Balram	25000	Delhi
1003	Bheem	45000	Gurgaon
1004	Chetan	60000	Noida
1005	Diksha	30000	Agra
1006	Raman	50000	Ghaziabad
1007	Sheetal	35000	Delhi

Table: New_Employee

Emp_ID	Emp_Name	Emp_Salary	Address
1008	Sumit	50000	Agra
1009	Akash	55000	Delhi
1010	Devansh	65000	Gurgaon

The New_Employee contains the details of new employees. If you want to move the details of those employees whose salary is greater than 40000 from the Old_Employee table to the New_Employee table. Then for this issue, you have to type the following query in SQL:

INSERT INTO New_Employee SELECT * FROM Old_Employee WHERE Emp_Salary > 40000;

Now, you can check the details of the updated New_Employee table by using the following SELECT query:

SELECT * FROM New_Employee;

Output:

Table: New_Employee

Emp_ID	Emp_Name	Emp_Salary	Address
1008	Sumit	50000	Agra
1009	Akash	55000	Delhi
1010	Devansh	65000	Gurgaon
1001	Akhil	50000	Agra
1003	Bheem	45000	Gurgaon
1004	Chetan	60000	Noida
1006	Raman	50000	Ghaziabad

Example 2: This example describes how to use **ANY operator** with subquery in the INSERT Statement.

Here we have taken the New_Employee, old_Employee, and Department table. The data of the New_Employee table is shown in the following table:

Table: New_Employee

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1008	Sumit	50000	401

The data of the old_Employee table is shown in the below table:

Table: Old_Employee

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL

The data of Department table is shown in the below table:

Dept_ID	Dept_Name	Emp_ID
401	Administration	1008
402	HR	1004
403	Testing	1002
404	Coding	1001
405	Sales	1003
406	Marketing	NULL
407	Accounting	1005

```
INSERT INTO New_Employee
SELECT * FROM Old_Employee
WHERE Emp_ID = ANY( SELECT Emp_ID FROM Department WHERE Dept_ID = 407 OR Dept_ID =
406 );
```

Now, check the details of the **New_Employee table by using the following SELECT statement:**

Output:

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1008	Sumit	50000	401
1005	Ram	65000	407

Subquery with the UPDATE statement

The subqueries and nested queries can be used with the UPDATE statement in Structured Query Language to update the existing table's columns. We can easily update one or more columns using a subquery with the UPDATE statement.

Syntax of Subquery with the UPDATE statement

```
UPDATE Table_Name SET Column_Name = New_value WHERE Value OPERATOR (SELECT COLU
MN_NAME FROM TABLE_NAME WHERE Condition) ;
```

Example of Subquery with the UPDATE statement

This example updates the record of one table using the IN operator with subquery in the UPDATE statement. Let's take an Employee_Details and Department table. The data of the Employee_Details table is shown in the following table:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID

1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL

The data of Department table is shown in the below table:

Table: Department

Dept_ID	Dept_Name	Emp_ID	Dept_Grade
401	Administration	1008	B
402	HR	1004	A
403	Testing	1002	A
404	Coding	1001	B
405	Sales	1003	A
406	Marketing	NULL	C
407	Accounting	1005	A

The following updates the salary of those employees whose Department Grade is A:

```
UPDATE Employee_Details
SET Emp_Salary = Emp_Salary + 5000
WHERE Emp_ID IN (SELECT Emp_ID FROM Department WHERE Dept_Grade = 'A');
```

The following query will show the updated data of the Employee_Details table in the output:

```
SELECT * FROM Employee_Details ;
```

Output:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	30000	403
1003	Bheem	50000	405
1004	Chetan	65000	402
1005	Ram	70000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL

Subquery with the DELETE statement

We can easily delete one or more records from the SQL table using subquery with the DELETE statement in Structured Query Language.

Syntax of Subquery with DELETE statement

```
DELETE FROM Table_Name WHERE Value OPERATOR (SELECT COLUMN_NAME FROM TABL
E_NAME WHERE Condition) ;
```

Example of Subquery with DELETE statement

This example deletes the records from the table using the IN operator with subquery in the DELETE statement. Let's take an Employee_Details and Department table. The data of the Employee_Details table is shown in the following table:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL
1008	Ankit	48000	401

The data of Department table is shown in the below table:

Dept_ID	Dept_Name	Emp_ID	Dept_Grade
401	Administration	1008	C
402	HR	1004	A
403	Testing	1002	C
404	Coding	1001	B
405	Sales	1003	A
406	Marketing	NULL	C
407	Accounting	1005	C

The following query deletes the record of those employees from the Employee_Details whose Department Grade is C:

DELETE FROM Employee_Details WHERE Emp_ID IN (SELECT Emp_ID FROM Department WHE RE Dept_Grade = 'C') ;

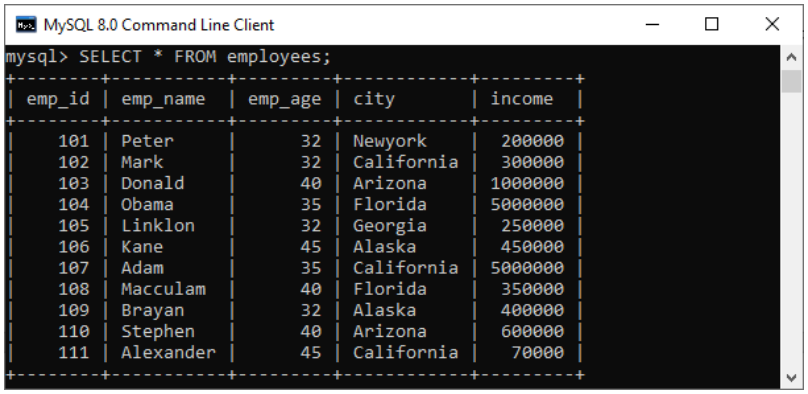
The following query will show the updated data of the Employee_Details table in the output:

SELECT * FROM Employee_Details ;

Output:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1003	Bheem	45000	405
1004	Chetan	60000	402
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL



MySQL Subquery with Comparison Operator

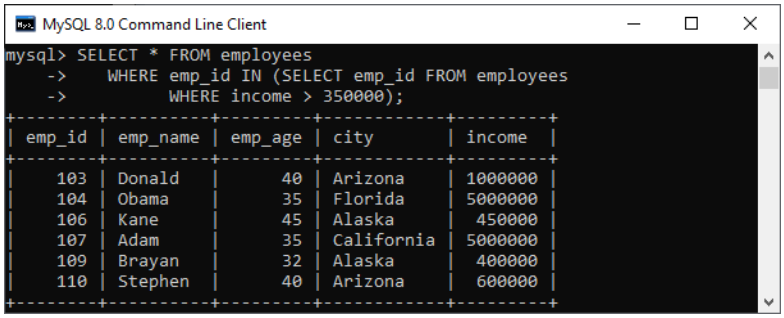
A comparison operator is an operator used to compare values and returns the result, either true or false. The following comparison operators are used in MySQL <, >, =, <>, <=>, etc. We can use the subquery before or after the comparison operators that return a single value. The returned value can be the arithmetic expression or a

column function. After that, SQL compares the subquery results with the value on the other side of the comparison operator. The example below explains it more clearly:

Following is a simple statement that returns the **employee detail whose income is more than 350000** with the help of a subquery:

```
SELECT * FROM employees
  WHERE emp_id IN (SELECT emp_id FROM employees
      WHERE income > 350000);
```

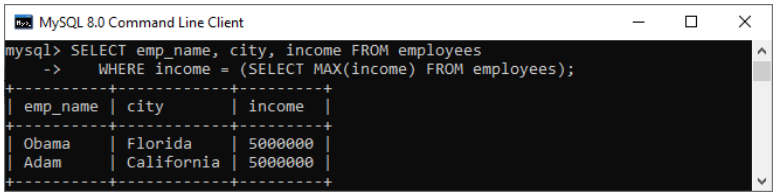
This query first executes the subquery that returns the **employee id whose income > 350000**. Second, the main query will return the employees all details whose employee id are in the result set returned by the subquery.
After executing the statement, we will get the below output, where we can see the employee detail whose income>350000.



Let us see an example of another comparison operator, such as equality (=) to find employee details with **maximum income** using a subquery.

```
SELECT emp_name, city, income FROM employees
  WHERE income = (SELECT MAX(income) FROM employees);
```

It will give the output where we can see two employees detail who have maximum income.



MySQL Subquery with IN or NOT-IN Operator

If the subquery produces more than one value, we need to use the IN or NOT IN Operator with the WHERE clause. Suppose we have a table named **"Student"** and **"Student2"** that contains the following data:

Table: Student

Stud_ID	Name	Email	City
1	Peter	peter@javatpoint.com	Texas
2	Suzi	suzi@javatpoint.com	California
3	Joseph	joseph@javatpoint.com	Alaska
4	Andrew	andrew@javatpoint.com	Los Angeles
5	Brayan	brayan@javatpoint.com	New York

Table: Student2

Stud_ID	Name	Email	City
1	Stephen	stephen@javatpoint.com	Texas
2	Joseph	joseph@javatpoint.com	Los Angeles
3	Peter	peter@javatpoint.com	California
4	David	david@javatpoint.com	New York
5	Maddy	maddy@javatpoint.com	Los Angeles

The following subquery with NOT IN Operator returns the **student detail who does not belong to Los Angeles City** from both tables as follows:

```
SELECT Name, City FROM student
WHERE City NOT IN (
SELECT City FROM student2 WHERE City='Los Angeles');
```

After execution, we can see that the result contains the student details that do not belong to Los Angeles City.

```
MySQL 8.0 Command Line Client
mysql> SELECT Name, City FROM student
    -> WHERE City NOT IN (
    -> SELECT City FROM student2 WHERE City='Los Angeles');
+---------+------------+
| Name    | City       |
+---------+------------+
| Stephen | Texas      |
| Peter   | California |
| David   | New York   |
+---------+------------+
```

MySQL Subquery in the FROM Clause

If we use a subquery in the FROM clause, MySQL will return the output from a subquery is used as a temporary table. We called this table as a derived table, inline views, or materialized subquery. The following subquery returns the maximum, minimum, and average number of items in the order table:

```
SELECT Max(items), MIN(items), FLOOR(AVG(items))
FROM
   (SELECT order_id, COUNT(order_id) AS items FROM orders
   GROUP BY order_date) AS Student_order_detail;
```

It will give the output as follows:

```
MySQL 8.0 Command Line Client
mysql> SELECT Max(items), MIN(items), FLOOR(AVG(items))
    -> FROM
    ->     (SELECT order_id, COUNT(order_id) AS items FROM orders
    ->     GROUP BY order_date) AS Student_order_detail;
+------------+------------+-------------------+
| Max(items) | MIN(items) | FLOOR(AVG(items)) |
+------------+------------+-------------------+
|          1 |          1 |                 1 |
+------------+------------+-------------------+
```

MySQL Correlated Subqueries

A correlated subquery in MySQL is a subquery that depends on the outer query. It uses the data from the outer query or contains a reference to a parent query that also appears in the outer query. MySQL evaluates it once from each row in the outer query.

```
SELECT emp_name, city, income
FROM employees emp WHERE income > (
SELECT AVG(income) FROM employees WHERE city = emp.city);
```

In the above query, we select an **employee name and city** whose income is higher than the average income of all employees in each city.

```
MySQL 8.0 Command Line Client
mysql> SELECT emp_name, city, income
    -> FROM employees emp WHERE income > (
    -> SELECT AVG(income) FROM employees WHERE city = emp.city);
+----------+------------+---------+
| emp_name | city       | income  |
+----------+------------+---------+
| Donald   | Arizona    | 1000000 |
| Obama    | Florida    | 5000000 |
| Kane     | Alaska     |  450000 |
| Adam     | California | 5000000 |
+----------+------------+---------+
```

The subquery executes for every city of the specified table because it will change for every row. Therefore, the average income will also be changed. Then, the main query filters employee details whose income is higher than the average income from the subquery.

MySQL Subqueries with EXISTS or NOT EXISTS

The EXISTS operator is a Boolean operator that returns a true or false result. It is used with a subquery and checks the existence of data in a subquery. If a subquery returns any record at all, this Operator returns true. Otherwise, it will return false. The NOT EXISTS Operator used for negation that gives true value when the subquery does not return any row. Otherwise, it returns false. Both EXISTS and NOT EXISTS used with correlated subqueries. The following example illustrates it more clearly. Suppose we have a table **customer and order** that contains the data as follows:

```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM customer;
+---------+---------+------------+------+
| cust_id | name    | occupation | age  |
+---------+---------+------------+------+
|     101 | Peter   | Engineer   |   32 |
|     102 | Joseph  | Developer  |   30 |
|     103 | John    | Leader     |   28 |
|     104 | Stephen | Scientist  |   45 |
|     105 | Suzi    | Carpenter  |   26 |
|     106 | Bob     | Actor      |   25 |
|     107 | NULL    | NULL       | NULL |
+---------+---------+------------+------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM Orders;
+----------+---------+-----------+------------+
| order_id | cust_id | prod_name | order_date |
+----------+---------+-----------+------------+
|        1 |     101 | Laptop    | 2020-01-10 |
|        2 |     103 | Desktop   | 2020-02-12 |
|        3 |     106 | Iphone    | 2020-02-15 |
|        4 |     104 | Mobile    | 2020-03-05 |
|        5 |     102 | TV        | 2020-03-20 |
+----------+---------+-----------+------------+
```

The below SQL statements uses EXISTS operator to find the name, occupation, and age of the customer who has placed at least one order.

```
SELECT name, occupation, age FROM customer C
WHERE EXISTS (SELECT * FROM Orders O
WHERE C.cust_id = O.cust_id);
```

This statement uses NOT EXISTS operator that returns the customer details who have not placed an order.

```
SELECT name, occupation, age FROM customer C
WHERE NOT EXISTS (SELECT * FROM Orders O
WHERE C.cust_id = O.cust_id);
```

We can see the below output to understand the above queries result.

```
MySQL 8.0 Command Line Client
mysql> SELECT name, occupation, age FROM customer C
    -> WHERE EXISTS (SELECT * FROM Orders O
    -> WHERE C.cust_id = O.cust_id);
+---------+------------+------+
| name    | occupation | age  |
+---------+------------+------+
| Peter   | Engineer   |   32 |
| Joseph  | Developer  |   30 |
| John    | Leader     |   28 |
| Stephen | Scientist  |   45 |
| Bob     | Actor      |   25 |
+---------+------------+------+
5 rows in set (0.10 sec)

mysql> SELECT name, occupation, age FROM customer C
    -> WHERE NOT EXISTS (SELECT * FROM Orders O
    -> WHERE C.cust_id = O.cust_id);
+------+------------+------+
| name | occupation | age  |
+------+------------+------+
| Suzi | Carpenter  |   26 |
| NULL | NULL       | NULL |
+------+------------+------+
```

MySQL ROW Subqueries

It is a subquery that returns a single row where we can get more than one column values. We can use the following operators for comparing row subqueries =, >, <, >=, <=, <>, !=, <=>. Let us see the following example:

```
SELECT * FROM customer C WHERE ROW(cust_id, occupation) = (
SELECT order_id, order_date FROM Orders O WHERE C.cust_id = O.cust_id);
```

If given row has cust_id, occupation values equal to the order_id, order_date values of any rows in the first table, the WHERE expression is TRUE, and each query returns those first table rows. Otherwise, the expression is FALSE, and the query produces an empty set, which can be shown in the below image:

```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM customer C WHERE ROW(cust_id, occupation) = (
    -> SELECT order_id, order_date FROM Orders O WHERE C.cust_id = O.cust_id);
Empty set (0.00 sec)
```

MySQL Subqueries with ALL, ANY, and SOME

We can use a subquery which is followed by the keyword ALL, ANY, or SOME after a comparison operator. The following are the syntax to use subqueries with ALL, ANY, or SOME:

```
operand comparison_operator ANY (subquery)
operand comparison_operator ALL (subquery)
operand comparison_operator SOME (subquery)
```

The ALL keyword compares values with the value returned by a subquery. Therefore, it returns TRUE if the comparison is TRUE for ALL of the values returned by a subquery. The ANY keyword returns TRUE if the comparison is TRUE for ANY of the values returned by a subquery. The ANY and SOME keywords are the same because they are the alias of each other. The following example explains it more clearly:

```
SELECT cust_id, name FROM customer WHERE
cust_id > ANY (SELECT cust_id FROM Orders);
```

```
MySQL 8.0 Command Line Client
mysql> SELECT cust_id, name FROM customer WHERE
    -> cust_id > ANY (SELECT cust_id FROM Orders);
+---------+---------+
| cust_id | name    |
+---------+---------+
|     102 | Joseph  |
|     103 | John    |
|     104 | Stephen |
|     105 | Suzi    |
|     106 | Bob     |
|     107 | NULL    |
+---------+---------+
```

If we use ALL in place of ANY, it will return TRUE when the comparison is TRUE for ALL values in the column returned by a subquery. For example:

```
SELECT cust_id, name FROM customer WHERE
cust_id > ALL (SELECT cust_id FROM Orders);
```

We can see the output as below:

```
MySQL 8.0 Command Line Client
mysql> SELECT cust_id, name FROM customer WHERE
    -> cust_id > ALL (SELECT cust_id FROM Orders);
+---------+------+
| cust_id | name |
+---------+------+
|     107 | NULL |
+---------+------+
```