MIT World Peace University

Data Base Management System

Assignment 4

Naman Soni Roll No. 10

Contents

1 Aim

Write suitable select command to get requested data from tables

2 Objective

To study Subqueries, Group, Joins and Views

3 Problem Statement

Create tables and solve given queries using, Group, Joins and Views

4 Theory

4.1 SQL Join types

SQL joins are used to combine two or more tables based on related columns between them. There are four types of SQL joins: INNER JOIN, LEFT JOIN (or LEFT OUTER JOIN), RIGHT JOIN (or RIGHT OUTER JOIN), and FULL OUTER JOIN.

Syntax:

```
    SELECT *
    FROM table1
    INNER JOIN table2
    ON table1.column = table2.column;
```

Description:
The INNER JOIN selects all rows from both tables where the join condition is true, which means only the matching rows between the tables will be returned. **EXAMPLE:**

```
    SELECT *
    FROM orders
    INNER JOIN customers
    ON orders.customer_id = customers.customer_id;
```

Table specifying the logical operators to be used:

Join Type	Logical Operator
Inner Join	=
Left Join	= or is NULL
Right Join	= or is NULL
Full Outer Join	=, is NULL or UNION ALL

Syntax:

```
    SELECT *
    FROM table1
    LEFT JOIN table2
    ON table1.column = table2.column;
```

Description:
The LEFT JOIN (or LEFT OUTER JOIN) returns all rows from the left table and matching rows from the right table based on the join condition. If there is no matching row in the right table, the result will contain NULL values for the right table columns. **EXAMPLE:**

```
SELECT *
FROM orders
LEFT JOIN customers
ON orders.customer_id = customers.customer_id;
```

Syntax:

```
SELECT *
FROM table1
RIGHT JOIN table2
ON table1.column = table2.column;
```

Description:
The RIGHT JOIN (or RIGHT OUTER JOIN) returns all rows from the right table and matching rows from the left table based on the join condition. If there is no matching row in the left table, the result will contain NULL values for the left table columns.
EXAMPLE:

```
SELECT *
FROM orders
RIGHT JOIN customers
ON orders.customer_id = customers.customer_id;
```

Syntax:

```
SELECT *
FROM table1
FULL OUTER JOIN table2
ON table1.column = table2.column;
```

Description:
The FULL OUTER JOIN returns all rows from both tables and NULL values for non-matching rows. This type of join is not supported by all SQL implementations, and it can also be emulated by combining a LEFT JOIN and a RIGHT JOIN with a UNION ALL operator.
EXAMPLE:

```
SELECT *
FROM orders
FULL OUTER JOIN customers
ON orders.customer_id = customers.customer_id;
```

5 Platform

- **Operating System:** Mac OS 64-bit
- **IDE or Text Editor:** Visual Studio Code

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA10)

7 Output

```
MariaDB [dbms_lab]> select * from Room;
+--------+---------+-------+-------+
| RoomNo | HotelNo | Type  | Price |
+--------+---------+-------+-------+
|   1    |    1    | Suite | 1646  |
|   2    |    2    | Suite | 1264  |
|   3    |    1    | 2 Bed | 773   |
|   4    |    4    | 2 Bed | 1949  |
|   5    |    1    | 3 Bed | 1959  |
    |   6   |     3    | 3 Bed | 674  |
    |   7   |     1    | 1 Bed | 1018 |
    |   8   |     3    | 1 Bed | 1314 |
    |   9   |     1    | Suite | 1308 |
    |   10  |     9    | 3 Bed | 1366 |
    |   11  |     10   | 1 Bed | 666  |
    |   12  |     7    | 2 Bed | 1498 |
    |   13  |     7    | Suite | 984  |
+--------+--------+------+-------+
13 rows in set (0.001 sec)
MariaDB [dbms_lab]> select * from Hotel;
+---------+------------------+----------------------+
| HotelNo | Name             | City |
+---------+------------------+----------------------+
|    1    | Hotel love       | Guernsey             |
|    2    | Hotel imagine    | Jordan               |
|    3    | Hotel rice       | Equatorial Guinea    |
|    4    | Hotel perhaps    | Bolivia              |
|    5    | Hotel show       | Reunion              |
|    6    | Hotel native     | Brunei               |
|    7    | Hotel pool       | Panama               |
|    8    | Hotel spin       | Guyana               |
|    9    | Hotel toward     | St. Barthelemy       |
|    10   | Hotel expression | St. Pierre & Miquelon |
|    11   | Hotel cheese     | Guinea -Bissau       |
|    12   | Hotel motion     | Latvia               |
|    13   | Hotel lay        | Fiji                 |
|    14   | Hotel stiff      | Brazil               |
|    15   | Hotel suddenly   | Lithuania            |
|    16   | Hotel stretch    | Montenegro           |
|    17   | Hotel current    | Isle of Man          |
|    18   | Hotel forest     | Haiti                |
+---------+------------------+----------------------+
18 rows in set (0.001 sec)
MariaDB [dbms_lab]> select * from booking;
+---------+---------+------------+------------+--------+
| HotelNo | GuestNo | DateFrom   | DateTo     | RoomNo |
+---------+---------+------------+------------+--------+
| 7       | 10      | 2096-04-21 | 2099-12-21 | 10     |
| 8       | 5       | 2077-09-29 | 2109-09-10 | 11     |
| 11      | 4       | 2123-01-05 | 2063-08-30 | 2      |
| 10      | 5       | 2027-02-05 | 2119-12-21 | 7      |
| 9       | 5       | 2081-07-11 | 2031-06-20 | 13     |
| 5       | 5       | 2059-11-19 | 2113-05-22 | 11     |
+---------+---------+------------+------------+--------+
6 rows in set (0.000 sec)
MariaDB [dbms_lab]> select * from Guest;
+---------+----------------+-----------------+
| GuestNo | GuestName      | GuessAddress    |
+---------+----------------+-----------------+
| 2       | Patrick Taylor | Lebanon         |
| 4       | Mattie Vargas  | St. Barthelemy  |
| 5       | Travis Frazier | Gambia          |
| 10      | Sarah Ramsey   | Jamaica         |
| 11      | Rachel Keller  | Kenya           |
| 15      | Nathan Higgins | Puerto Rico     |
| 16      | Maude Gonzales | St. Lucia       |
+---------+----------------+-----------------+
7 rows in set (0.000 sec)
MariaDB [dbms_lab]> MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 1. many hotels are there? MariaDB [dbms_lab]> select count(*) from
    Hotel;
+----------+
| count (*)|
+----------+
|    18    |
+----------+
1 row in set (0.000 sec)
```

```
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 2. the price and type of all rooms at the Grosvenor Hotel.

MariaDB [dbms_lab]> select price, type, Name from Room, Hotel where Room.HotelNo = Hotel.
HotelNo and Name = ‘Hotel love’;
+-------+-------+------------+
| price | type  | Name       |
+-------+-------+------------+
| 1646  | Suite | Hotel love |
| 773   | 2 Bed | Hotel love |
| 1959  | 3 Bed | Hotel love |
| 1018  | 1 Bed | Hotel love |
| 1308  | Suite | Hotel love |
+-------+-------+------------+
5 rows in set (0.001 sec)
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 3. the number of rooms in each hotel.
MariaDB [dbms_lab]> select Room.HotelNo, Hotel.NAME, count(*) from Room, Hotel where Room.
    HotelNo = Hotel.HotelNo group by HotelNo;
+---------+------------------+----------+
| HotelNo | NAME             | count(*)|
+---------+------------------+----------+
|    1    | Hotel love       | 5        |
|    2    | Hotel imagine    | 1        |
|    3    | Hotel rice       | 2        |
|    4    | Hotel perhaps    | 1        |
|    7    | Hotel pool       | 2        |
|    9    | Hotel toward     | 1        |
|    10   | Hotel expression | 1        |
+---------+------------------+----------+
7 rows in set (0.000 sec)
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 4. Update the price of all rooms by 5%.
MariaDB [dbms_lab]> select r.Price, r.Price + r.Price * 0.05 as Updated_price from Room r;
+-------+---------------+
| Price | Updated_price |
+-------+---------------+
| 1646  | 1728.30       |
| 1264  | 1327.20       |
| 773   | 811.65        |
| 1949  | 2046.45       |
| 1959  | 2056.95       |
| 674   | 707.70        |
| 1018  | 1068.90       |
| 1314  | 1379.70       |
| 1308  | 1373.40       |
| 1366  | 1434.30       |
| 666   | 699.30        |
| 1498  | 1572.90       |
| 984   | 1033.20       |
+-------+---------------+
13 rows in set (0.000 sec)
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 5. full details of all hotels in London.
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select * from Hotel where City = ‘Jordan’;
+---------+---------------+--------+
| HotelNo | Name          | City   |
+---------+---------------+--------+
| 2       | Hotel imagine | Jordan |
+---------+---------------+--------+
1 row in set (0.000 sec)
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 6. What is the average price of a room? MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select avg(Price) from Room;
+------------+
| avg(Price) |
+------------+
| 1263.0000  |
+------------+
1 row in set (0.000 sec)
MariaDB [dbms_lab]>
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 7. all guests currently staying at the Grosvenor Hotel.
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select Guest.* from Guest, booking, Hotel where Guest.GuestNo = booking
.GuestNo and booking.HotelNo = Hotel.HotelNo and Hotel.Name = ‘Hotel pool’;
+---------+--------------+--------------+
| GuestNo | GuestName    | GuessAddress |
+---------+--------------+--------------+
| 10      | Sarah Ramsey | Jamaica      |
+---------+--------------+--------------+
1 row in set (0.001 sec)
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> -- 8. the number of rooms in each hotel in London.
MariaDB [dbms_lab]>
MariaDB [dbms_lab]> select count(*) from Room, Hotel where Room.HotelNo = Hotel.HotelNo and
Hotel.City = ‘Jordan’;
+----------+
| count(*)|
+----------+
|    1     |
+----------+
1 row in set (0.000 sec)
MariaDB [dbms_lab]>
```

8 FAQ’s

8.1 *When to use self join? How does it differ from other joins?*

Ans. A self join is a type of join where a table is joined with itself. This is useful when the table contains hierarchical or recursive data, where each row has a relationship with another row within the same table. In such cases, a self join can be used to retrieve related data from the same table.

For example, consider a table that represents an organization’s employee hierarchy. Each row in the table contains an employee ID, a manager ID that identifies the employee’s manager, and other employee details such as name and department. To retrieve the names of all employees and their managers, we can use a self join:

```
    SELECT e.name as employee_name, m.name as manager_name
    FROM employees e
    INNER JOIN employees m ON e.manager_id = m.employee_id
```

In this example, we join the ‘employees’ table with itself using the ‘manager-id’ and ‘employee-id’ columns. The result set includes the name of each employee and the name of their respective manager.

Compared to other types of joins, self joins do not differ in terms of syntax or behavior. They follow the same rules for join conditions, filtering, and grouping as other types of joins. The only difference is that a self join involves joining a table with itself instead of with another table.

It’s worth noting that self joins can potentially be computationally expensive, especially when dealing with large tables or deep recursion. In such cases, it may be necessary to optimize the join conditions or use other techniques such as recursive common table expressions (CTEs) to retrieve hierarchical data.

8.2 *Compare Cross Join with Natural Join. Share your comments.*

Ans. A cross join and a natural join are both types of joins in SQL, but they differ in their behavior and the types of join conditions they use.

A cross join, also known as a Cartesian join, is a type of join where every row of one table is combined with every row of another table. This results in a Cartesian product of the two tables, where the number of rows in the result set is equal to the product of the number of rows in each table. A cross join does not require a join condition, as it simply combines every possible combination of rows.

For example, consider two tables 'A' and 'B':

```
A
+---+
| 1 |
| 2 |
| 3 |
+---+
B
+---+
| x |
| y |
+---+
```

A cross join between these tables would result in the following:

```
SELECT * FROM A CROSS JOIN B

1 x
1 y
2 x
2 y
3 x
3 y
```

A natural join, on the other hand, is a type of join where the join condition is implicitly determined based on columns with the same name and data type in both tables. A natural join returns only the rows from each table where the join condition is true.

For example, consider two tables 'A' and 'B':

```
A
---
1
2
3

B
---
2
3
4
```

A natural join between these tables on the 'id' column would result in the following:

```
SELECT * FROM A NATURAL JOIN B

2
3
```

In this example, only the rows with matching 'id' values in both tables are returned.

In terms of when to use each type of join, it depends on the specific requirements of the query. A cross join is useful when you need to generate all possible combinations of rows from two tables, whereas a natural join is useful when you need to combine rows from two tables based on columns with the same name and data type. It's important to note that a natural join may not always be the most appropriate or efficient way to join tables, as it relies on implicit join conditions and may not always return the desired result set. In general, it's recommended to use explicit join conditions and to carefully consider the behavior of each type of join before using them in a query.

8.3 *What is the importance of SQL joins in database management? Explain its types.*

Ans.SQL joins are an essential component of database management, as they allow users to combine data from multiple tables based on specific conditions. The ability to join tables is crucial for organizing and analyzing data in a relational database, where information is often spread across multiple tables.

SQL supports several types of joins, including:

- Inner join: Returns only the matching rows from both tables based on the join condition.
- Left outer join: Returns all the rows from the left table and matching rows from the right table based on the join condition.
- Right outer join: Returns all the rows from the right table and matching rows from the left table based on the join condition.
- Full outer join: Returns all the rows from both tables and matches them where possible based on the join condition.
- Cross join: Combines every row from the left table with every row from the right table, resulting in a Cartesian product.
- Self join: Joins a table to itself based on a relationship within the table.

The importance of SQL joins in database management can be summarized as follows:

- Data integration: SQL joins allow users to integrate data from multiple tables, providing a complete view of the information. This is especially important in large databases where information is spread across multiple tables.
- Data analysis: SQL joins enable users to analyze data in more detail, allowing them to identify patterns and trends across multiple tables.
- Data normalization: SQL joins are a key part of the normalization process, which aims to minimize data redundancy and improve database performance.
- Flexibility: SQL joins provide users with flexibility in querying databases, allowing them to create customized views of data based on their specific requirements.

8.4 *What are the different types of Joins in SQL?*

Ans. There are several types of joins in SQL, each with its own purpose and behavior. The most common types of joins include:

- Inner Join: An inner join, also known as a simple join, returns only the matching rows from both tables based on the join condition. It compares each row from the first table with every row from the second table to find the matching rows.
- Left Join or Left Outer Join: A left join returns all the rows from the left table and matching rows from the right table based on the join condition. If there is no match in the right table, the result set will contain NULL values for the columns from the right table.
- Right Join or Right Outer Join: A right join returns all the rows from the right table and matching rows from the left table based on the join condition. If there is no match in the left table, the result set will contain NULL values for the columns from the left table.
- Full Join or Full Outer Join: A full join returns all the rows from both tables and matches them where possible based on the join condition. If there is no match in either table, the result set will contain NULL values for the columns from the non-matching table.

- Cross Join or Cartesian Product: A cross join combines every row from the left table with every row from the right table, resulting in a Cartesian product. It does not require a join condition, as it simply combines every possible combination of rows.
- Self Join: A self join is a join in which a table is joined with itself based on a relationship within the table. It is useful when you need to compare rows within the same table.

8.5 *State the difference between inner join and left join.*

Ans. The main difference between inner join and left join in SQL is the way they handle non-matching rows from one of the tables being joined.

In an inner join, only the matching rows from both tables based on the join condition are returned in the result set. Any rows that do not have a matching row in the other table are excluded from the result set. This means that only the intersection of the two tables is returned.

In a left join, all the rows from the left table are returned in the result set, along with any matching rows from the right table based on the join condition. If there is no matching row in the right table, the result set will contain NULL values for the columns from the right table. This means that the result set includes all the rows from the left table, and only the matching rows from the right table.

To illustrate this difference, let's say we have two tables: Customers and Orders. The Customers table has columns for customer ID, name, and address, while the Orders table has columns for order ID, customer ID, and order date.

If we want to find all the customers who have placed an order, we can use an inner join between the Customers and Orders tables on the customer ID column. This will return only the rows from both tables where there is a matching customer ID in both tables.

On the other hand, if we want to find all the customers, including those who have not placed an order, we can use a left join between the Customers and Orders tables on the customer ID column. This will return all the rows from the Customers table, along with any matching rows from the Orders table. If a customer has not placed an order, the result set will contain NULL values for the columns from the Orders table.

8.6 *State difference between left join and right join.*

Ans. The main difference between left join and right join in SQL is the way they handle non-matching rows from one of the tables being joined.

In a left join, all the rows from the left table are returned in the result set, along with any matching rows from the right table based on the join condition. If there is no matching row in the right table, the result set will contain NULL values for the columns from the right table. This means that the result set includes all the rows from the left table, and only the matching rows from the right table.

On the other hand, in a right join, all the rows from the right table are returned in the result set, along with any matching rows from the left table based on the join condition. If there is no matching row in the left table, the result set will contain NULL values for the columns from the left table. This means that the result set includes all the rows from the right table, and only the matching rows from the left table.

To illustrate this difference, let's say we have two tables: Customers and Orders. The Customers table has columns for customer ID, name, and address, while the Orders table has columns for order ID, customer ID, and order date.

If we want to find all the customers, including those who have not placed an order, we can use a left join between the Customers and Orders tables on the customer ID column. This will return all the rows from the Customers table, along with any matching rows from the Orders table. If a customer has not placed an order, the result set will contain NULL values for the columns from the Orders table.

On the other hand, if we want to find all the orders, including those that have not been placed by a customer, we can use a right join between the Customers and Orders tables on the customer ID column. This will return all the rows from the Orders table, along with any matching rows from the Customers table. If an order has not been placed by a customer, the result set will contain NULL values for the columns from the Customers table.