Day 2: Advanced Filtering, Functions, and Operators

05 September 2023 03:05 PM

Schema (MySQL v8.0)

```
    CREATE TABLE customers (
        customer_id INT PRIMARY KEY,
        customer_name VARCHAR(50),
        email VARCHAR(100),
        service_type VARCHAR(50),
        last_payment_date DATE
    );

    CREATE TABLE payments (
        payment_id INT PRIMARY KEY,
        customer_id INT,
        payment_amount DECIMAL(10, 2),
        payment_date DATE,
        FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
    );

    INSERT INTO customers (customer_id, customer_name, email, service_type,
last_payment_date)
    VALUES
        (1, 'John Doe', 'johndoe@example.com', 'Internet', '2023-01-10'),
        (2, 'Jane Smith', 'janesmith@example.com', 'Mobile', '2023-01-15'),
        (3, 'Mike Johnson', 'mikejohnson@example.com', 'Internet', '2023-01-05'),
        (4, 'Emily Brown', 'emilybrown@example.com', 'Landline', '2023-01-20'),
        (5, 'David Wilson', 'davidwilson@example.com', 'Internet', '2023-01-12');

    INSERT INTO payments (payment_id, customer_id, payment_amount, payment_date)
    VALUES
        (1, 1, 50.00, '2023-01-05'),
        (2, 2, 35.00, '2023-01-10'),
        (3, 3, 75.00, '2023-01-15'),
        (4, 4, 20.00, '2023-01-18'),
        (5, 5, 60.00, '2023-01-12'),
        (6, 1, 40.00, '2023-01-20'),
        (7, 2, 30.00, '2023-01-25'),
        (8, 3, 80.00, '2023-01-28'),
        (9, 4, 25.00, '2023-01-22'),
        (10, 5, 65.00, '2023-01-30');
```

Query #1

```
    SELECT UPPER(email)
    FROM customers;
```

```
| UPPER(email)            |
| ----------------------- |
| JOHNDOE@EXAMPLE.COM     |
| JANESMITH@EXAMPLE.COM   |
| MIKEJOHNSON@EXAMPLE.COM |
```

| EMILYBROWN@EXAMPLE.COM |
| DAVIDWILSON@EXAMPLE.COM |

Query #2

```
SELECT customer_id, SUM(payment_amount)
FROM payments
GROUP BY customer_id;
```

customer_id	SUM(payment_amount)
1	90.00
2	65.00
3	155.00
4	45.00
5	125.00

Query #3

```
SELECT c.customer_id,
      c.customer_name,
      SUM(payment_amount) AS total_payment_amount
FROM customers c
JOIN payments p
ON c.customer_id = p.customer_id
GROUP BY c.customer_id
ORDER BY total_payment_amount DESC;
```

customer_id	customer_name	total_payment_amount
3	Mike Johnson	155.00
5	David Wilson	125.00
1	John Doe	90.00
2	Jane Smith	65.00
4	Emily Brown	45.00

Query #4

```
SELECT * FROM customers
WHERE customer_id IN
(SELECT customer_id FROM payments
WHERE payment_amount >
(SELECT AVG(payment_amount) from payments));
```

customer_id	customer_name	email	service_type	last_payment_date
1	John Doe	johndoe@example.com	Internet	2023-01-10
3	Mike Johnson	mikejohnson@example.com	Internet	2023-01-05
5	David Wilson	davidwilson@example.com	Internet	2023-01-12

Query #5

```
SELECT * FROM customers
```

```
   WHERE service_type = 'Internet';

| customer_id | customer_name | email                  | service_type | last_payment_date |
| ----------- | ------------- | ---------------------- | ------------ | ----------------- |
| 1           | John Doe      | johndoe@example.com     | Internet     | 2023-01-10        |
| 3           | Mike Johnson  | mikejohnson@example.com | Internet     | 2023-01-05        |
| 5           | David Wilson  | davidwilson@example.com | Internet     | 2023-01-12        |
```

Query #6

```
   SELECT
     c.customer_id,
     c.customer_name,
     SUM(p.payment_amount) AS total_payment_amount,
     CASE
       WHEN SUM(p.payment_amount) > 100 THEN 'Good'
       WHEN SUM(p.payment_amount) > 50 THEN 'Average'
       ELSE 'Poor'
     END AS payment_history
   FROM payments p
   JOIN customers c
   ON p.customer_id = c.customer_id
   GROUP BY c.customer_name, c.customer_id;

| customer_id | customer_name | total_payment_amount | payment_history |
| ----------- | ------------- | -------------------- | --------------- |
| 1           | John Doe      | 90.00                | Average         |
| 2           | Jane Smith    | 65.00                | Average         |
| 3           | Mike Johnson  | 155.00               | Good            |
| 4           | Emily Brown   | 45.00                | Poor            |
| 5           | David Wilson  | 125.00               | Good            |
```

[View on DB Fiddle](https://www.db-fiddle.com/f/t6abyQXdU8tXJZLADXkf4u/0)

```
-- 1. Retrieve the email addresses of customers in uppercase.

SELECT UPPER(email)
FROM customers;
```

/*
This SQL query retrieve the email addresses of customers and convert them to uppercase. Here's an explanation of each part of the query:

SELECT UPPER(email): This part of the query is responsible for selecting data from the "email" column of the "customers" table and converting it to uppercase using the UPPER function. The UPPER function is a SQL function that transforms text to uppercase.

FROM customers: This specifies the table from which we want to retrieve data, which is the "customers" table in this case. It tells the database where to find the "email" column.

So, when you run this query, the database will return a result set with all the email addresses

from the "customers" table, but they will all be in uppercase.
*/

-- 2. Calculate the total payment amount for each customer.

```sql
SELECT customer_id, SUM(payment_amount)
FROM payments
GROUP BY customer_id;
```

/*
This SQL query will calculate the total payment amount for each customer. Here's a breakdown of each part of the query:

SELECT customer_id, SUM(payment_amount): This part of the query specifies the columns that will be included in the result set. It selects the "customer_id" column, which identifies each customer, and it calculates the sum of the "payment_amount" column using the SUM function. The result set will include these two columns: "customer_id" and the total payment amount for each customer.

FROM payments: This part of the query indicates the source table from which the data will be retrieved. In this case, it's the "payments" table, which likely contains records of payments made by customers.

GROUP BY customer_id: This is a crucial part of the query. It specifies that the data should be grouped based on the "customer_id" column. The SUM function will then be applied within each group, calculating the total payment amount separately for each unique customer.

So, when you run this query, the database will aggregate payment data for each customer. It will sum up all the payment amounts associated with each customer (identified by their unique "customer_id"), and the result will display a list of customer IDs along with the total payment amount for each customer.

This type of query is useful for generating reports or analyses that require summarizing payment information on a per-customer basis, allowing businesses to understand the payment behavior of their customers.

*/

```sql
SELECT c.customer_id,
    c.customer_name,
    SUM(payment_amount) AS total_payment_amount
FROM customers c
JOIN payments p
ON c.customer_id = p.customer_id
GROUP BY c.customer_id
ORDER BY total_payment_amount DESC;
```

-- 3. Retrieve the customers who have made payments greater than the average payment amount.

```sql
SELECT * FROM customers
WHERE customer_id IN
(SELECT customer_id FROM payments
WHERE payment_amount >
(SELECT AVG(payment_amount) from payments));
```

/*
This SQL retrieve customers who have made payments greater than the average payment amount. Here's an explanation of how this query works:

Innermost Subquery (SELECT AVG(payment_amount) from payments)): This subquery calculates the average payment amount from the "payments" table. It computes the total sum of all payment amounts and divides it by the number of payments to find the average.

Middle Subquery (SELECT customer_id FROM payments WHERE payment_amount > ...): This subquery selects customer IDs from the "payments" table where the payment amount is greater than the average payment amount calculated in the innermost subquery. In other words, it identifies customers who have made payments above the average.

Outer Query (SELECT * FROM customers WHERE customer_id IN ...): The outermost query retrieves all columns from the "customers" table for customers whose IDs are found in the result of the middle subquery. In other words, it retrieves customer information for those who have made payments greater than the average.

When you run this query, it will return a list of customer records (all columns from the "customers" table) for customers who have made payments exceeding the average payment amount. This allows you to identify and analyze customers who are making above-average payments in your database.

*/

-- 4. Retrieve the customers who have used the 'Internet' service.

```
SELECT * FROM customers
WHERE service_type = 'Internet';
```

/*
This SQL query retrieve customers who have used the 'Internet' service. Here's an explanation of the query:

SELECT * FROM customers: This part of the query specifies that you want to retrieve all columns (represented by *) from the "customers" table. It indicates that you want to obtain all customer information.

WHERE service_type = 'Internet': This is the filtering condition. It tells the database to select only those rows where the value in the "service_type" column is equal to 'Internet'. In other words, it identifies customers who have used the 'Internet' service.

When you run this query, it will return a list of customer records, including all columns from the "customers" table, but only for customers who have used the 'Internet' service. This query is helpful for segmenting and analyzing customers based on the specific services they have used.
*/

-- 5. Segment customers based on their payment history: 'Good', 'Average', or 'Poor'.

```
SELECT
  c.customer_id,
  c.customer_name,
  SUM(p.payment_amount) AS total_payment_amount,
  CASE
    WHEN SUM(p.payment_amount) > 100 THEN 'Good'
    WHEN SUM(p.payment_amount) > 50 THEN 'Average'
    ELSE 'Poor'
  END AS payment_history
FROM payments p
JOIN customers c
ON p.customer_id = c.customer_id
GROUP BY c.customer_name, c.customer_id;

/*
This SQL query will segment customers based on their payment history into 'Good', 'Average', or
'Poor' categories. Here's an explanation of each part of the query:

SELECT c.customer_id, c.customer_name, SUM(p.payment_amount) AS total_payment_amount:
This part of the query selects specific columns from both the "customers" and "payments"
tables. It selects the customer's ID and name from the "customers" table and calculates the total
payment amount for each customer by summing the "payment_amount" column from the
"payments" table. The calculated sum is aliased as "total_payment_amount" for readability.

CASE WHEN SUM(p.payment_amount) > 100 THEN 'Good' WHEN SUM(p.payment_amount) > 50
THEN 'Average' ELSE 'Poor' END AS payment_history: In this section, the CASE statement is used
to categorize customers' payment history based on the calculated total payment amount.
Customers are categorized as follows:

If their total payment amount is greater than 100, they are categorized as 'Good.'
If their total payment amount is greater than 50 but not more than 100, they are categorized as
'Average.'
If their total payment amount is 50 or less, they are categorized as 'Poor.'
FROM payments p JOIN customers c ON p.customer_id = c.customer_id: This part of the query
specifies the tables involved in the query and the relationship between them. It joins the
"payments" table (aliased as "p") with the "customers" table (aliased as "c") using the
"customer_id" column as the common link.

GROUP BY c.customer_name, c.customer_id: The GROUP BY clause groups the results by
customer name and customer ID. This is necessary because the SUM function is used to
calculate the total payment amount for each customer, and grouping ensures that each
customer is treated as a separate group.

When you run this query, it will return a result set with customer information, the total payment
amount for each customer, and their payment history category ('Good,' 'Average,' or 'Poor').
This query helps segment customers based on their payment behavior, making it easier to
analyze and target different customer groups.

*/
```