What Is a View in MySQL?

A **View** in **MySQL** is a **virtual table** based on the result of a SQL query.

- It does **not store data itself**
- It stores a **SQL SELECT statement**
- When you query the view, MySQL runs the stored query

Think of a view as a **saved query you can treat like a table**.

Why Use Views?

Views are useful for:

1. □ Simplifying complex queries
2. □ Hiding sensitive columns (security)
3. □ Reusing common queries
4. □ Presenting data in a specific format
5. □ Restricting user access to certain rows/columns

Step-by-Step Guide to Creating and Using Views

Step 1: Create Sample Tables

Let's create two example tables:

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    country VARCHAR(50)
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
```

```
    amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Insert sample data:

```
INSERT INTO customers VALUES
(1, 'John Doe', 'john@email.com', 'USA'),
(2, 'Alice Smith', 'alice@email.com', 'UK');

INSERT INTO orders VALUES
(101, 1, '2025-01-01', 500.00),
(102, 1, '2025-01-10', 200.00),
(103, 2, '2025-01-15', 300.00);
```

Step 2: Create a Basic View

Suppose we frequently need to see customer orders with customer names.

Create the View:

```
CREATE VIEW customer_orders AS
SELECT
    c.customer_id,
    c.name,
    o.order_id,
    o.order_date,
    o.amount
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id;
```

Now `customer_orders` behaves like a table.

Step 3: Use the View

You can query it like this:

```
SELECT * FROM customer_orders;
```

Output:

customer_id	name	order_id	order_date	amount
1	John Doe	101	2025-01-01	500.00

customer_id	name	order_id	order_date	amount
1	John Doe	102	2025-01-10	200.00
2	Alice Smith	103	2025-01-15	300.00

You can also filter:

```
SELECT * FROM customer_orders
WHERE amount > 250;
```

Step 4: Create a View with Aggregation

Let’s create a view showing total purchase per customer.

```
CREATE VIEW customer_total_spending AS
SELECT
    c.customer_id,
    c.name,
    SUM(o.amount) AS total_spent
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name;
```

Use it:

```
SELECT * FROM customer_total_spending;
```

Result:

customer_id	name	total_spent
1	John Doe	700.00
2	Alice Smith	300.00

Step 5: Update Data Through a View

 Not all views are updatable.

A view is usually updatable if:

- It selects from one table
- No GROUP BY
- No JOIN
- No aggregate functions

Example:

```
CREATE VIEW usa_customers AS
SELECT customer_id, name, email
FROM customers
WHERE country = 'USA';
```

Now update using the view:

```
UPDATE usa_customers
SET email = 'newjohn@email.com'
WHERE customer_id = 1;
```

This updates the original `customers` table.

Step 6: Replace an Existing View

```
CREATE OR REPLACE VIEW customer_orders AS
SELECT
    c.name,
    o.amount
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id;
```

Step 7: Drop a View

```
DROP VIEW customer_orders;
```

Step 8: See Existing Views

```
SHOW FULL TABLES WHERE TABLE_TYPE = 'VIEW';
```

Or:

```
SHOW CREATE VIEW customer_orders;
```

Important Concepts

1 Simple View vs Complex View

Simple View:

- Based on one table
- No aggregation
- Usually updatable

Complex View:

- JOIN
- GROUP BY
- Aggregates
- Usually NOT updatable

---

2 WITH CHECK OPTION

Ensures inserted/updated data satisfies the view condition.

Example:

```
CREATE VIEW usa_customers AS
SELECT * FROM customers
WHERE country = 'USA'
WITH CHECK OPTION;
```

Now this will fail:

```
INSERT INTO usa_customers VALUES (3, 'Mike', 'mike@email.com', 'Canada');
```

Because country ≠ USA.

Real-World Use Case Example

Scenario: Hide Customer Emails

```
CREATE VIEW public_customers AS
SELECT customer_id, name, country
FROM customers;
```

Now give users access to the view instead of the table:

```
GRANT SELECT ON public_customers TO 'report_user'@'localhost';
```

Users can see customer names but NOT emails.

Advantages of Views

✔ Security
✔ Simplicity
✔ Reusability
✔ Data abstraction
✔ Logical data independence