**VIEW In details**

A view in SQL is a virtual table based on the result set of a SQL query. It does not store data itself but provides a way to look at the data from one or more tables in a specific, customized manner. Views can simplify complex queries, enhance security by restricting data access, and help in maintaining a consistent, reusable query structure.

### Creating a View

To create a view, you use the CREATE VIEW statement followed by the view name and the AS keyword, followed by the SQL query that defines the view.

#### Syntax

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

### Advantages of Using Views

1. **Simplicity**: Simplifies complex queries by encapsulating them into a view.
2. **Reusability**: Allows you to reuse SQL queries across different parts of your application.
3. **Security**: Restricts access to specific data by exposing only certain columns and rows.
4. **Maintenance**: Easier to maintain changes to the SQL logic in one place rather than updating multiple queries.

### Disadvantages

1. **Performance**: Views can sometimes impact performance, especially if they are complex and involve multiple joins or aggregations.
2. **Read-Only Nature**: Some views can be read-only, which means you can't perform INSERT, UPDATE, or DELETE operations directly on them.
3. **Complex Dependencies**: Changes to underlying tables might require updates to the views that depend on them.

### Use Cases

1. **Report Generation**: Simplifying the creation of complex reports by providing a single view that aggregates necessary data.
2. **Data Abstraction**: Providing an abstraction layer over the actual schema, making it easier for users to interact with the data.
3. **Security and Access Control**: Restricting access to sensitive data by exposing only a subset of columns or rows through the view.
4. **Encapsulation of Business Logic**: Encapsulating business logic within views to ensure consistency across different queries and applications.

As a data analyst, using SQL views can be incredibly beneficial for organizing, simplifying, and securing your data queries. Here’s how you can leverage SQL views in your role:

keyboard\_arrow\_down

### **Benefits of Views for Data Analysts**

1. **Simplification of Complex Queries**: Views encapsulate complex queries into a single, reusable entity. This makes your SQL scripts more readable and maintainable.
2. **Data Consistency**: By using views, you ensure that all users are accessing the same data and logic. This consistency is crucial for reporting and analysis.
3. **Security**: Views can restrict access to sensitive data by exposing only the necessary columns and rows, which is important for maintaining data privacy and compliance.
4. **Abstraction and Encapsulation**: Views abstract the underlying table structures, allowing you to change the database schema without affecting the analytical queries.

### **Example Scenarios**

Let's explore a few scenarios where views can be particularly useful for a data analyst:

#### Scenario 1: Simplifying Complex Queries

Suppose you regularly need to analyze sales data across multiple tables (e.g., customers, orders, order\_items, products). Instead of writing the complex joins and calculations every time, you can create a view.

##### **Creating the View**

CREATE VIEW sales\_summary AS  
SELECT   
    c.customer\_id,  
    c.first\_name,  
    c.last\_name,  
    o.order\_id,  
    o.order\_date,  
    p.product\_name,  
    oi.quantity,  
    p.price,  
    (oi.quantity \* p.price) AS total\_amount  
FROM   
    customers c  
JOIN   
    orders o ON c.customer\_id = o.customer\_id  
JOIN   
    order\_items oi ON o.order\_id = oi.order\_id  
JOIN   
    products p ON oi.product\_id = p.product\_id;

##### **Using the View**

You can now query the sales\_summary view to get insights without dealing with the complexity of joins and calculations.

SELECT   
    customer\_id,  
    first\_name,  
    last\_name,  
    SUM(total\_amount) AS total\_spent  
FROM   
    sales\_summary  
GROUP BY   
    customer\_id, first\_name, last\_name  
ORDER BY   
    total\_spent DESC;

#### Scenario 2: Enhancing Security

If you need to share data with different teams but want to restrict access to certain sensitive columns (e.g., salaries), you can create views that exclude these columns.

##### **Creating the View**

CREATE VIEW employee\_public\_view AS  
SELECT   
    employee\_id,  
    first\_name,  
    last\_name,  
    department  
FROM   
    employees;

##### **Using the View**

Users can query employee\_public\_view to get the necessary information without accessing sensitive data.

SELECT   
    first\_name,  
    last\_name,  
    department  
FROM   
    employee\_public\_view  
WHERE   
    department = 'Sales';

#### Scenario 3: Creating Analytical Reports

You can create views tailored for specific analytical reports. For instance, a monthly sales report.

##### **Creating the View**

CREATE VIEW monthly\_sales\_report AS  
SELECT   
    DATE\_TRUNC('month', o.order\_date) AS month,  
    p.product\_name,  
    SUM(oi.quantity) AS total\_quantity,  
    SUM(oi.quantity \* p.price) AS total\_sales  
FROM   
    orders o  
JOIN   
    order\_items oi ON o.order\_id = oi.order\_id  
JOIN   
    products p ON oi.product\_id = p.product\_id  
GROUP BY   
    DATE\_TRUNC('month', o.order\_date), p.product\_name;

##### **Using the View**

Generate the report by querying the view.

SELECT   
    month,  
    product\_name,  
    total\_quantity,  
    total\_sales  
FROM   
    monthly\_sales\_report  
ORDER BY   
    month, total\_sales DESC;

### **Best Practices for Using Views**

1. **Name Views Clearly**: Use descriptive names that reflect the purpose of the view.
2. **Keep Views Simple**: Avoid overly complex views that can degrade performance. Break down complex logic into multiple views if necessary.
3. **Document Views**: Ensure that the purpose and logic of each view are well-documented for future reference.
4. **Monitor Performance**: Regularly check the performance of views, especially if the underlying data changes frequently.

### **Conclusion**

By effectively using SQL views, you can streamline your data analysis tasks, improve data security, and maintain a consistent and clean query environment. Whether simplifying complex queries, restricting data access, or creating tailored reports, views are a powerful tool in a data analyst’s toolkit.

**PRACTICE Set**

### Creating Tables and Inserting Sample Data

#### Creating Tables

Let's look at some real-world use cases for SQL views in a business context, focusing on typical questions a data analyst might need to answer. These examples will illustrate how views can simplify complex queries, enhance data security, and provide a foundation for analytical reporting.

**create the datasets** for the provided use cases using SQL. This will involve creating tables and populating them with sample data for customers, orders, order items, products, employees, and departments.

### **Use Case 1: Sales Analysis**

#### 1.1 Creating Tables

-- Create customers table  
CREATE TABLE customers (  
    customer\_id INT PRIMARY KEY,  
    first\_name VARCHAR(50),  
    last\_name VARCHAR(50)  
);  
  
-- Create orders table  
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)  
);  
  
-- Create products table  
CREATE TABLE products (  
    product\_id INT PRIMARY KEY,  
    product\_name VARCHAR(100),  
    price DECIMAL(10, 2)  
);  
  
-- Create order\_items table  
CREATE TABLE order\_items (  
    order\_item\_id INT PRIMARY KEY,  
    order\_id INT,  
    product\_id INT,  
    quantity INT,  
    FOREIGN KEY (order\_id) REFERENCES orders(order\_id),  
    FOREIGN KEY (product\_id) REFERENCES products(product\_id)  
);

#### 1.2 Inserting Sample Data

-- Insert sample data into customers  
INSERT INTO customers (customer\_id, first\_name, last\_name) VALUES  
(1, 'John', 'Doe'),  
(2, 'Jane', 'Smith'),  
(3, 'Michael', 'Brown');  
  
-- Insert sample data into orders  
INSERT INTO orders (order\_id, customer\_id, order\_date, amount) VALUES  
(1, 1, '2023-01-01', 150.00),  
(2, 1, '2023-01-15', 200.00),  
(3, 2, '2023-02-01', 250.00),  
(4, 3, '2023-02-15', 300.00);  
  
-- Insert sample data into products  
INSERT INTO products (product\_id, product\_name, price) VALUES  
(1, 'Laptop', 1000.00),  
(2, 'Mouse', 50.00),  
(3, 'Keyboard', 70.00);  
  
-- Insert sample data into order\_items  
INSERT INTO order\_items (order\_item\_id, order\_id, product\_id, quantity) VALUES  
(1, 1, 1, 1),  
(2, 1, 2, 2),  
(3, 2, 1, 1),  
(4, 3, 3, 3),  
(5, 4, 1, 2);

### **Use Case 2: Employee Performance and Department Analysis**

#### 2.1 Creating Tables

-- Create departments table  
CREATE TABLE departments (  
    department\_id INT PRIMARY KEY,  
    department\_name VARCHAR(100)  
);  
  
-- Create employees table  
CREATE TABLE employees (  
    employee\_id INT PRIMARY KEY,  
    first\_name VARCHAR(50),  
    last\_name VARCHAR(50),  
    department\_id INT,  
    salary DECIMAL(10, 2),  
    performance\_rating DECIMAL(3, 2),  
    FOREIGN KEY (department\_id) REFERENCES departments(department\_id)  
);

#### 2.2 Inserting Sample Data

-- Insert sample data into departments  
INSERT INTO departments (department\_id, department\_name) VALUES  
(1, 'Sales'),  
(2, 'Engineering'),  
(3, 'HR');  
  
-- Insert sample data into employees  
INSERT INTO employees (employee\_id, first\_name, last\_name, department\_id, salary, performance\_rating) VALUES  
(1, 'Alice', 'Johnson', 1, 60000, 4.7),  
(2, 'Bob', 'Lee', 2, 80000, 4.5),  
(3, 'Charlie', 'Kim', 3, 50000, 4.8),  
(4, 'David', 'Brown', 1, 70000, 4.6),  
(5, 'Eve', 'Davis', 2, 75000, 4.9);

### **Use Case 3: Customer Behavior Analysis**

#### 3.1 Reusing the Customers and Orders Tables from Use Case 1

We already have the customers and orders tables created and populated with sample data from Use Case 1. No need to create these tables again.

### **Creating Views for Business Questions**

**Use Case 1: Sales Analysis**

#### Business Questions:

1. What is the total sales revenue generated each month?
2. Which products are the top sellers?
3. Who are the top customers by sales revenue?

#### Tables and Sample Data:

* **customers**: customer\_id, first\_name, last\_name
* **orders**: order\_id, customer\_id, order\_date, amount
* **order\_items**: order\_item\_id, order\_id, product\_id, quantity
* **products**: product\_id, product\_name, price

1. **Monthly Sales Revenue View**

CREATE VIEW monthly\_sales\_revenue AS  
SELECT   
    DATE\_TRUNC('month', o.order\_date) AS month,  
    SUM(o.amount) AS total\_revenue  
FROM   
    orders o  
GROUP BY   
    DATE\_TRUNC('month', o.order\_date);

1. **Top Selling Products View**

CREATE VIEW top\_selling\_products AS  
SELECT   
    p.product\_name,  
    SUM(oi.quantity) AS total\_quantity\_sold,  
    SUM(oi.quantity \* p.price) AS total\_sales  
FROM   
    order\_items oi  
JOIN   
    products p ON oi.product\_id = p.product\_id  
GROUP BY   
    p.product\_name  
ORDER BY   
    total\_sales DESC;

1. **Top Customers View**

CREATE VIEW top\_customers AS  
SELECT   
    c.customer\_id,  
    c.first\_name,  
    c.last\_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.first\_name, c.last\_name  
ORDER BY   
    total\_spent DESC;

### **Use Case 2: Employee Performance and Department Analysis**

#### Business Questions:

1. What is the average salary by department?
2. Which employees have the highest performance ratings?
3. How many employees are there in each department?

#### Tables and Sample Data:

* **employees**: employee\_id, first\_name, last\_name, department\_id, salary, performance\_rating
* **departments**: department\_id, department\_name

#### Use Case 2: Employee Performance and Department Analysis

1. **Average Salary by Department View**

CREATE VIEW avg\_salary\_by\_department AS  
SELECT   
    d.department\_name,  
    AVG(e.salary) AS average\_salary  
FROM   
    employees e  
JOIN   
    departments d ON e.department\_id = d.department\_id  
GROUP BY   
    d.department\_name;

1. **Top Performing Employees View**

CREATE VIEW top\_performing\_employees AS  
SELECT   
    e.employee\_id,  
    e.first\_name,  
    e.last\_name,  
    e.performance\_rating  
FROM   
    employees e  
WHERE   
    e.performance\_rating >= 4.5  
ORDER BY   
    e.performance\_rating DESC;

1. **Employee Count by Department View**

CREATE VIEW employee\_count\_by\_department AS  
SELECT   
    d.department\_name,  
    COUNT(e.employee\_id) AS employee\_count  
FROM   
    employees e  
JOIN   
    departments d ON e.department\_id = d.department\_id  
GROUP BY   
    d.department\_name;

### **Use Case 3: Customer Behavior Analysis**

#### Business Questions:

1. What is the average order value for each customer?
2. How many orders does each customer place on average?
3. Which customers have not placed an order in the last 6 months?

#### Tables and Sample Data:

* **customers**: customer\_id, first\_name, last\_name
* **orders**: order\_id, customer\_id, order\_date, amount

#### View create

1. **Average Order Value by Customer View**

CREATE VIEW avg\_order\_value\_by\_customer AS  
SELECT   
    c.customer\_id,  
    c.first\_name,  
    c.last\_name,  
    AVG(o.amount) AS average\_order\_value  
FROM   
    customers c  
JOIN   
    orders o ON c.customer\_id = o.customer\_id  
GROUP BY   
    c.customer\_id, c.first\_name, c.last\_name;

1. **Order Count by Customer View**

CREATE VIEW order\_count\_by\_customer AS  
SELECT   
    c.customer\_id,  
    c.first\_name,  
    c.last\_name,  
    COUNT(o.order\_id) AS order\_count  
FROM   
    customers c  
JOIN   
    orders o ON c.customer\_id = o.customer\_id  
GROUP BY   
    c.customer\_id, c.first\_name, c.last\_name;

1. **Customers with No Orders in the Last 6 Months View**

CREATE VIEW inactive\_customers AS  
SELECT   
    c.customer\_id,  
    c.first\_name,  
    c.last\_name  
FROM   
    customers c  
LEFT JOIN   
    orders o ON c.customer\_id = o.customer\_id AND o.order\_date >= DATEADD(month, -6, GETDATE())  
WHERE   
    o.order\_id IS NULL;

### **Summary**

By creating these datasets and views, data analysts can easily and efficiently answer a variety of business questions related to sales analysis, employee performance, and customer behavior. The use of views simplifies complex queries, ensures data consistency, and enhances data security.