In SQL, a view is a virtual table generated by a query. It presents a subset of data from one or more tables, and it can be used like a regular table for querying purposes. Views offer several benefits in database management, including simplifying complex queries, enhancing security, and providing a layer of abstraction.

What is a View?

A view is a part of a database created from a query and stored as a permanent object. Essentially, a view is an SQL query saved as an object.

Although the view's definition remains constant, the data it contains can change based on when the view is accessed. Views can represent a subset of the data in a table and can join and simplify multiple tables into one virtual table.

Advantages:

- **Minimal Storage:** Views take up very little storage space because the database only stores the view definition, not the data itself.
- **Dynamic Calculations:** They can provide results for different calculations (like sums and averages) along with the stored data.
- **Data Security:** Views can limit the exposure of tables to the outer world, enhancing data security.

Purpose of Views

1. **Data Abstraction**:
    - Views hide complex SQL queries by presenting a simplified interface.
    - Users can query a view without needing to know the underlying table structure or join conditions.

2. **Security**:

   - Views can restrict access to specific columns or rows of a table.
   - Users may have access only to views that show specific data subsets, enhancing data security.

3. **Simplifying Queries**:

   - Views can encapsulate joins and aggregations, making queries simpler and more readable.
   - They reduce the amount of repetitive SQL code in applications.

Types of Views

1. **Simple Views**:

   - Derived from a single base table.
   - Can include all or some columns of the base table.

```
CREATE VIEW SimpleView AS
SELECT column1, column2
FROM table_name;
```

2. **Complex Views**:

   - Derived from multiple base tables using joins or subqueries.
   - Allow complex data transformations and aggregations.

```
CREATE VIEW ComplexView AS
SELECT t1.column1, t2.column2
FROM table1 t1
JOIN table2 t2 ON t1.id = t2.id;
```

3. **Indexed Views**:

   - Stored physically in the database as a result set.

- Improve performance for frequently used queries by precomputing aggregations or joins.

```
CREATE INDEXED VIEW IndexedView AS
SELECT column1, COUNT(*)
FROM table_name
GROUP BY column1;
```

Benefits of Using Views

1. **Simplification of Complex Queries**:
   - Views encapsulate complex SQL logic, making queries more readable and maintainable.
2. **Security Control**:
   - Views restrict access to sensitive columns or rows, providing security at the database level.
3. **Performance Optimization**:
   - Indexed views can improve query performance by storing precomputed results.
4. **Data Independence**:
   - Views can shield applications from changes in the underlying table schema.

Limitations of Views

1. **Performance Overhead**:
   - Complex views or views on large datasets can impact performance due to query optimization overhead.
2. **Updatability**:

- Not all views are updatable, especially those involving multiple tables or certain SQL constructs like aggregate functions.

3. **Complexity Management**:

   Managing and maintaining a large number of views can lead to complexity in database administration.

Example

Let's create a simple view that aggregates data from an Orders table:

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(10, 2)
);

INSERT INTO Orders (OrderID, CustomerID, OrderDate,
TotalAmount) VALUES
(1, 101, '2023-01-15', 150.50),
(2, 102, '2023-01-16', 200.25),
(3, 101, '2023-01-16', 75.00);

-- Create a view to show total sales per customer
CREATE VIEW CustomerSales AS
SELECT CustomerID, SUM(TotalAmount) AS TotalSales
FROM Orders
GROUP BY CustomerID;
```

In this example, CustomerSales view aggregates the total sales (TotalAmount) per customer (CustomerID) from the Orders table. Users can query CustomerSales without needing to understand the details of how the aggregation is performed.

Views are powerful tools in SQL that enhance data management, security, and query simplicity. They provide a flexible way to present and manipulate data while maintaining data integrity and security policies within the database.

Step 1: Create a Table

First, you need to create a table. For example, let's create a table called Employees.

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Position VARCHAR(50),
   Salary DECIMAL(10, 2),
   HireDate DATE
);
```

Step 2: Insert Records

Next, let's insert some records into the Employees table.

```
INSERT INTO Employees7 (FirstName, LastName, Position, Salary,
HireDate) VALUES
('Rahul', 'Sharma', 'Manager', 60000, '2020-01-15'),
('Anjali', 'Verma', 'Developer', 50000, '2019-02-10'),
('Raj', 'Kumar', 'Designer', 45000, '2021-03-22'),
('Priya', 'Patel', 'Tester', 40000, '2018-07-30');
```

Step 3: Create a View

Now, let's create a view to simplify querying the data. A view can be used to present data in a specific way without altering the original table. For instance, let's create a view to show employees who earn more than Rs. 45,000.

```
CREATE VIEW HighEarningEmployees AS
SELECT EmployeeID, FirstName, LastName, Position, Salary
```

```
FROM Employees
WHERE Salary > 45000;
```

Step 4: Query the View

Finally, you can query the view just like you would query a table.

```
SELECT * FROM HighEarningEmployees;
```

Complete SQL Script

Here is the complete SQL script combining all the steps:

```
-- Create the Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Position VARCHAR(50),
    Salary DECIMAL(10, 2),
    HireDate DATE
);

INSERT INTO Employees7 (FirstName, LastName, Position, Salary,
HireDate) VALUES
('Rahul', 'Sharma', 'Manager', 60000, '2020-01-15'),
('Anjali', 'Verma', 'Developer', 50000, '2019-02-10'),
('Raj', 'Kumar', 'Designer', 45000, '2021-03-22'),
('Priya', 'Patel', 'Tester', 40000, '2018-07-30');

-- Create a view for high earning employees
CREATE VIEW HighEarningEmployees AS
SELECT EmployeeID, FirstName, LastName, Salary
FROM Employees
WHERE Salary > 45000;

SELECT * FROM HighEarningEmployees;
```

```
UPDATE HighEarningEmployees
SET Salary = 60000
WHERE EmployeeID = 2;

SELECT * FROM HighEarningEmployees;


DELETE FROM HighEarningEmployees
WHERE EmployeeID = 1;

SELECT * FROM HighEarningEmployees;


DROP VIEW IF EXISTS HighEarningEmployees;

-- Query the view
SELECT * FROM HighEarningEmployees;
```

This script will create the Employees table, insert records into it, create a view for employees with a salary greater than Rs. 45,000, and then query that view.