Excellent — let’s learn the **complete concept of Views in SQL** — from **basics to advanced level**, with examples, syntax, and interview Q&A.

**🧩 1. What is a View in SQL?**

A **View** is a **virtual table** that shows data from one or more tables using a **SELECT query**.  
It **doesn’t store data** itself — it stores only the **query definition**.

**✅ Example:**

CREATE VIEW EmployeeDetails AS

SELECT EmpID, Name, DeptName

FROM Employee

JOIN Department ON Employee.DeptID = Department.DeptID;

Now you can query the view just like a table:

SELECT \* FROM EmployeeDetails;

**🧠 2. Why Use Views?**

| **Purpose** | **Description** |
| --- | --- |
| **Simplify complex queries** | Write once, reuse easily |
| **Data Security** | Hide sensitive columns |
| **Data Abstraction** | User sees only necessary data |
| **Consistency** | Keep common logic in one place |
| **Virtual Table** | No physical storage (unless materialized) |

**🧾 3. Syntax of a View**

**➤ Create View**

CREATE VIEW view\_name AS

SELECT columns

FROM table\_name

WHERE condition;

**➤ Query a View**

SELECT \* FROM view\_name;

**➤ Modify View**

CREATE OR REPLACE VIEW view\_name AS

SELECT ...

**➤ Drop View**

DROP VIEW view\_name;

**🧱 4. Example Step-by-Step**

Suppose you have:

CREATE TABLE Employee (

EmpID INT PRIMARY KEY,

Name VARCHAR(50),

Salary DECIMAL(10,2),

DeptID INT

);

CREATE TABLE Department (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(50)

);

**👉 Create a Simple View:**

CREATE VIEW EmpView AS

SELECT EmpID, Name, Salary

FROM Employee;

**👉 Create a Join View:**

CREATE VIEW EmpDeptView AS

SELECT E.EmpID, E.Name, D.DeptName

FROM Employee E

JOIN Department D ON E.DeptID = D.DeptID;

**👉 Create a Conditional View:**

CREATE VIEW HighSalaryEmp AS

SELECT Name, Salary

FROM Employee

WHERE Salary > 50000;

**👉 Query the View:**

SELECT \* FROM HighSalaryEmp;

**⚙️ 5. Types of Views**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **1. Simple View** | Based on one table | CREATE VIEW EmpView AS SELECT \* FROM Employee; |
| **2. Complex View** | Based on multiple tables / joins / group by | CREATE VIEW EmpDeptView AS SELECT ... JOIN ...; |
| **3. Materialized View** | Physically stores query result for faster access | Supported in Oracle, PostgreSQL |
| **4. Inline View** | A subquery used in FROM clause | SELECT \* FROM (SELECT \* FROM Employee WHERE Salary>50000) AS T; |

**⚡ 6. Updating Data Through a View**

Views are **read-only by default**, but some can be **updatable** if they meet certain conditions.

**✅ You can UPDATE through a view if:**

* View is based on **one table**
* No DISTINCT, GROUP BY, HAVING, or aggregate functions
* No JOIN, UNION, or TOP
* All **NOT NULL** columns are included

**Example:**

CREATE VIEW EmpSalary AS

SELECT EmpID, Name, Salary FROM Employee;

UPDATE EmpSalary

SET Salary = Salary + 5000

WHERE EmpID = 101;

This will update the base table Employee.

**🚫 7. When Views Are NOT Updatable**

| **Reason** | **Example** |
| --- | --- |
| Contains aggregate function | SUM(), COUNT() |
| Uses GROUP BY / HAVING | GROUP BY DeptID |
| Joins multiple tables | JOIN Department |
| Contains DISTINCT or UNION | SELECT DISTINCT ... |

**🔐 8. Security with Views**

You can **restrict access** to sensitive data by granting access to a view only.

**Example:**

CREATE VIEW PublicEmp AS

SELECT Name, DeptID FROM Employee;

Then:

GRANT SELECT ON PublicEmp TO trainee;

The trainee user can see only Name and DeptID, not Salary or other private data.

**🧮 9. Performance and Optimization**

* Views themselves **don’t store data** → no performance gain.
* But **Materialized Views** (in Oracle, PostgreSQL) **store results** and improve performance for large aggregations.
* Use views to **standardize logic**, not to optimize speed (unless materialized).

**🧰 10. Managing Views**

| **Command** | **Purpose** |
| --- | --- |
| SHOW FULL TABLES WHERE TABLE\_TYPE = 'VIEW'; | List all views (MySQL) |
| sp\_helptext 'view\_name'; | Show definition (SQL Server) |
| DROP VIEW view\_name; | Delete view |
| ALTER VIEW view\_name AS ... | Modify existing view |

**💡 11. Real-Life Example**

**Scenario:**

You have confidential employee data but want to share a report of each department’s total salary to managers.

CREATE VIEW DeptWiseSalary AS

SELECT D.DeptName, SUM(E.Salary) AS TotalSalary

FROM Employee E

JOIN Department D ON E.DeptID = D.DeptID

GROUP BY D.DeptName;

Manager query:

SELECT \* FROM DeptWiseSalary;

Result:

| **DeptName** | **TotalSalary** |
| --- | --- |
| IT | 250000 |
| HR | 180000 |

**🧠 12. Interview Questions on Views**

| **#** | **Question** | **Answer** |
| --- | --- | --- |
| 1 | What is a view? | A virtual table that stores a SELECT query definition. |
| 2 | Does a view store data? | No, it stores only the query (except materialized views). |
| 3 | Difference between table and view? | Table stores data, view shows data dynamically. |
| 4 | Can we update data through a view? | Yes, if it’s simple and meets updatable conditions. |
| 5 | What are materialized views? | Views that store data physically for faster access. |
| 6 | Can a view have joins? | Yes (then it’s called a complex view). |
| 7 | Can a view call another view? | Yes, nested views are allowed. |
| 8 | What happens if the base table is dropped? | The view becomes invalid (error on query). |
| 9 | Can we apply indexes on views? | Not directly, but possible on **materialized** views. |
| 10 | What are advantages of using views? | Security, simplicity, abstraction, consistency. |

**🧭 13. Summary Mind Map**

Views

│

├── Definition → Virtual table from SELECT query

├── Syntax → CREATE VIEW ... AS SELECT ...

├── Types

│ ├─ Simple

│ ├─ Complex

│ ├─ Materialized

│ └─ Inline

├── Features

│ ├─ Read-only / Updatable

│ ├─ Security layer

│ └─ Simplifies queries

├── Limitations

│ ├─ No indexes

│ ├─ Some not updatable

│ └─ Performance depends on base tables

└── Key Commands

├─ CREATE VIEW

├─ DROP VIEW

├─ ALTER VIEW

└─ SHOW VIEW