Lecture 9. VIEWS

What is a View?

> **A view = a named, stored SELECT query**

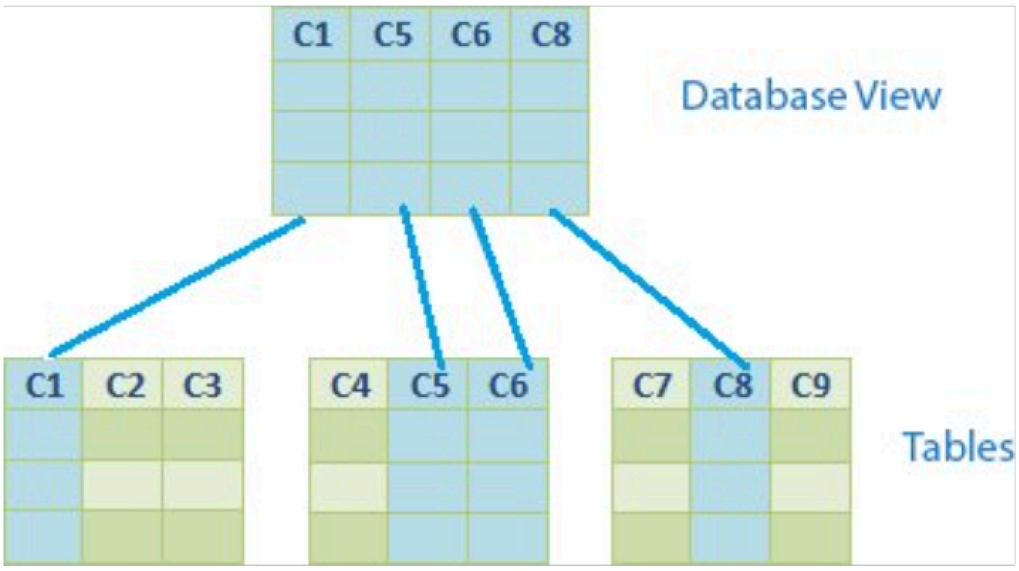
It behaves like a **virtual table:**

- You can `SELECT` from it
- It doesn't store its own data (except materialized view)

Data always comes from **underlying/base tables**; if base data changes, the view result changes too.

Informal idea:

You save a complex query once, give it a name, and then use that name like a table.



Why use Views?

1. **Simplicity**
    a. Hide complex joins, filters, calculations

    b. Application only sees something like `SELECT * FROM my_view;`

2. **Securiy**
    a. Expose only some columns/rows of a table
    b. Give users privileges on the view, not on the base table
3. **Logical Data Independence**
    a. You can change base table structure, but keep the same view interface: applications keep working if they use views

---

Creating a Basic View

General syntax (simplified):

```
CREATE VIEW view_name AS
SELECT ...
FROM ...
WHERE ...;
```

Example - only active customers:

```
CREATE VIEW active_customers AS
SELECT customer_id, first_name, last_name
FROM customers
WHERE active = TRUE;
```

Now you can use:

```
SELECT * FROM active_customers;
```

Using a View

A view can be quiried like a table:

```
SELECT first_name, last_name
FROM active_customers
WHERE last_name LIKE 'A%';
```

You can also join a view with tables or other views:

```
SELECT ac.first_name, ac.last_name, o.order_id
FROM active_customers ac
JOIN orders o ON ac.customer_id = o.customer_id;
```

Creating or Replacing View

If you want to **change** definition of a view:

```
CREATE OR REPLACE VIEW active_customers AS
SELECT customer_id, first_name, last_name, email
FROM customers
WHERE active = TRUE AND country = 'USA';
```

`CREATE OR REPLACE VIEW` keeps the same name but replaces query.

Temporary View (TEMP Views)

You can create **temporary views** that live only in the current session:

```
CREATE TEMPORARY VIEW current_session_sales AS
SELECT *
FROM sales
WHERE sale_date = CURRENT_DATE;
```

- Dropped automatically when the session ends.
- Good for one-off reporting or complex intermediate steps.

Changing View Metadata (ALTER VIEW)

You cannot change the query with `ALTER VIEW` (that's done with `CREATE OR REPLACE`), but you can change **metadata** like the name:

```
ALTER VIEW customer_master RENAME TO customer_info;
```

Dropping Views

To delete a view:

```
DROP VIEW view_name;
```

Options:

```
DROP VIEW IF EXISTS view_name
   [CASCADE | RESTRICT]
```

- `CASCADE`: also drops objects depending on this view.
- `RESTRICT`: error if something depends on it (default).

Multiple view at once:

```
DROP VIEW IF EXISTS v1, v2, v3;
```

Simple vs Complex Views

- **Simple view**:
  - Based on a **single table**.
  - No aggregates, `GROUP BY`, etc.
  - Often automatically **updatable**.

- **Complex view**:
  - Joins, aggregates, subqueries, etc.
  - Usually **read-only** (not updatable automatically).

Updatable Views (High-Level)

PostgreSQL supports **automatically updatable views** if the view is “simple enough”:

Rough conditions (simplified):

- `FROM` clause has **exactly one** base table or updatable view.
- No top-level `DISTINCT`, `GROUP BY`, `HAVING`, `UNION`, `INTERSECT`, `EXCEPT`, `LIMIT`, `OFFSET`, `WITH`.
- No aggregates, window functions, or set-returning functions in the select list.

Then you can:

```
INSERT INTO active_customers (customer_id, first_name, last_name)
VALUES (1001, 'Alice', 'Lee');

UPDATE active_customers
SET last_name = 'Smith'
WHERE customer_id = 1001;

DELETE FROM active_customers
WHERE customer_id = 1001;
```

PostgreSQL will map these operations to the underlying base table.

`WITH CHECK OPTION`

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

Meaning:

- Any `INSERT` or `UPDATE` via `usa_customers` must still satisfy `country = 'USA'`.
- Prevents "escaping" the view condition by changing data so it no longer matches the WHERE clause.

Variants:

- `WITH LOCAL CHECK OPTION` – only the current view's condition is enforced.
- `WITH CASCADED CHECK OPTION` – this view + all underlying base views are checked (default if you just write `WITH CHECK OPTION`).

Materialized View

> Materialized views extend usual views **by storing data physically**

```
CREATE MATERIALIZED VIEW sales_summary AS
SELECT store_id, SUM(amount) AS total_sales
FROM sales
GROUP BY store_id
WITH DATA;
```

- `WITH DATA` – fill it immediately.
- `WITH NO DATA` – create empty structure; must load later.

Refresh when data changes:

```
REFRESH MATERIALIZED VIEW sales_summary;
```

Drop:

```
DROP MATERIALIZED VIEW sales_summary;
```

Use cases:

- Heavy, complex queries that need to be fast (anlytics, BI , data warehouse)