### ****Views in SQL****

A **view** is a virtual table based on a SQL query. It doesn't store data itself but retrieves data dynamically from the base tables when queried. It's mainly used for:

* Simplifying complex queries.
* Providing security by restricting access to specific columns or rows.

#### ****Creating a View****

sql

Copy code

CREATE VIEW employee\_view AS

SELECT emp\_id, emp\_name, department

FROM employees

WHERE department = 'IT';

#### ****Using a View****

sql

Copy code

SELECT \* FROM employee\_view;

This will display all employees in the IT department, dynamically fetching data from the employees table.

### ****Materialized Views in SQL****

A **materialized view** is a physical copy of a query's result stored in the database. Unlike regular views, it doesn’t fetch data dynamically but maintains a snapshot of the data. Materialized views are useful when:

* Query performance is critical.
* Data doesn’t change frequently.

#### ****Creating a Materialized View****

sql

Copy code

CREATE MATERIALIZED VIEW employee\_mview AS

SELECT emp\_id, emp\_name, department

FROM employees

WHERE department = 'IT';

#### ****Refreshing a Materialized View****

Materialized views need to be refreshed to stay up-to-date. They can be:

* **Manually refreshed:**

sql

Copy code

REFRESH MATERIALIZED VIEW employee\_mview;

* **Automatically refreshed (at intervals):** Some databases allow setting a schedule for refresh.

### ****Key Differences****

| Feature | View | Materialized View |
| --- | --- | --- |
| Storage | No storage, data fetched on demand | Physically stores query results |
| Performance | Slower for complex queries | Faster for complex queries |
| Data Freshness | Always up-to-date | Requires manual/automatic refresh |
| Use Case | Simplifies query logic | Improves performance for static data |

### ****Example Use Case****

1. **View Example**: If you want to provide a summary report that is updated with the latest data dynamically:

sql

Copy code

CREATE VIEW sales\_summary AS

SELECT region, SUM(sales) AS total\_sales

FROM sales

GROUP BY region;

1. **Materialized View Example**: If querying sales data from a massive table is slow, you can create a precomputed snapshot:

sql

Copy code

CREATE MATERIALIZED VIEW sales\_mview AS

SELECT region, SUM(sales) AS total\_sales

FROM sales

GROUP BY region;

Refresh the materialized view periodically to keep it relevant.

### Additional Examples for ****Views**** and ****Materialized Views****

#### ****1. View Example with Joins****

You can create a view to simplify a query that involves multiple tables:

sql

Copy code

CREATE VIEW employee\_project\_view AS

SELECT e.emp\_id, e.emp\_name, p.project\_name

FROM employees e

JOIN projects p ON e.project\_id = p.project\_id

WHERE e.department = 'Development';

Using the view:

sql

Copy code

SELECT \* FROM employee\_project\_view;

This will fetch all employees in the "Development" department along with their project names.

#### ****2. Materialized View Example with Aggregations****

For heavy aggregation queries that are frequently needed, a materialized view is a great choice:

sql

Copy code

CREATE MATERIALIZED VIEW monthly\_sales\_mview AS

SELECT region, SUM(sales) AS total\_sales, COUNT(order\_id) AS total\_orders

FROM sales

WHERE order\_date >= DATE\_TRUNC('month', CURRENT\_DATE)

GROUP BY region;

Fetching data:

sql

Copy code

SELECT \* FROM monthly\_sales\_mview;

Refreshing the materialized view manually:

sql

Copy code

REFRESH MATERIALIZED VIEW monthly\_sales\_mview;

#### ****3. Indexed Materialized View for Faster Performance****

In some databases (e.g., PostgreSQL), you can add an index to a materialized view for even faster access:

sql

Copy code

CREATE INDEX idx\_region\_sales ON monthly\_sales\_mview (region);

Now queries on monthly\_sales\_mview using the region column will be much faster.

#### ****4. View with Limited Access****

Views can be used to restrict access to sensitive data. For example, create a view to allow access only to non-sensitive columns:

sql

Copy code

CREATE VIEW employee\_public\_view AS

SELECT emp\_id, emp\_name, department

FROM employees;

Users querying employee\_public\_view won’t see sensitive columns like salary or address.

#### ****5. Materialized View for Pre-computed Joins****

For a scenario where frequent joins slow down performance:

sql

Copy code

CREATE MATERIALIZED VIEW sales\_customer\_mview AS

SELECT s.order\_id, s.sales, c.customer\_name

FROM sales s

JOIN customers c ON s.customer\_id = c.customer\_id;

This precomputes the join, making subsequent queries faster:

sql

Copy code

SELECT customer\_name, SUM(sales)

FROM sales\_customer\_mview

GROUP BY customer\_name;

#### ****6. Refresh Strategies for Materialized Views****

* **Fast Refresh** (if supported by your database): Refreshes only changes since the last update. Requires a log table for tracking changes.

sql

Copy code

REFRESH MATERIALIZED VIEW CONCURRENTLY sales\_customer\_mview;

* **Complete Refresh**: Recomputes the entire materialized view from scratch.

sql

Copy code

REFRESH MATERIALIZED VIEW sales\_customer\_mview;

#### ****7. Combining Views and Materialized Views****

You can use both views and materialized views together. For example:

* A **materialized view** to store heavy computations:

sql

Copy code

CREATE MATERIALIZED VIEW yearly\_sales\_mview AS

SELECT region, SUM(sales) AS total\_sales

FROM sales

WHERE order\_date >= DATE\_TRUNC('year', CURRENT\_DATE)

GROUP BY region;

* A **view** to filter the materialized view for specific needs:

sql

Copy code

CREATE VIEW top\_regions\_view AS

SELECT region, total\_sales

FROM yearly\_sales\_mview

WHERE total\_sales > 1000000;

### ****Conclusion****

* Use **views** for dynamic, lightweight operations or when you want to hide query complexity.
* Use **materialized views** when performance is critical, and the data doesn't need to be real-time.