### ****What is a Materialized View?****

* A **materialized view** is a database object similar to a view but with a critical difference: it stores the **query** and the **data** generated by the query.
* Unlike a regular view, a materialized view does not re-execute the query each time it is accessed; it simply retrieves the stored data.

### ****Key Features of Materialized Views****

1. **Improves Performance:**
   * Speeds up queries by avoiding repeated execution of complex queries.
2. **Stored Data:**
   * Stores both the query and the resulting data at the time of creation.
3. **Requires Manual Refresh:**
   * Data in the materialized view does not update automatically when the underlying table changes.

### ****Difference Between View and Materialized View****

| Feature | View | Materialized View |
| --- | --- | --- |
| **Data Storage** | Does not store data. | Stores both query and data. |
| **Query Execution** | Executes the query every time | Fetches precomputed results. |
| **Performance** | No performance improvement. | Improves performance significantly. |
| **Updates** | Always shows latest data. | Requires manual refresh to update data. |

### ****How Materialized Views Improve Performance****

* The query and its results are precomputed and stored in the database.
* When accessed, the materialized view retrieves the precomputed data instead of executing the query again.
* This eliminates the need to repeatedly process large datasets or join multiple tables.

### ****Creating and Using Materialized Views****

#### ****1. Creating a Materialized View****

* Syntax:

sql

Copy code

CREATE MATERIALIZED VIEW mv\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* Example:

sql

Copy code

CREATE MATERIALIZED VIEW mv\_random\_table AS

SELECT id, AVG(value) AS avg\_value, COUNT(\*) AS record\_count

FROM random\_table

GROUP BY id;

#### ****2. Querying a Materialized View****

* Use the SELECT statement as you would with a regular table or view:

sql

Copy code

SELECT \* FROM mv\_random\_table;

#### ****3. Refreshing a Materialized View****

* To reflect changes in the base table, manually refresh the materialized view:

sql

Copy code

REFRESH MATERIALIZED VIEW mv\_random\_table;

### ****Example: Performance Comparison****

1. **Table Setup:**

sql

Copy code

CREATE TABLE random\_table (

id INT,

value DECIMAL

);

1. **Insert Data:**
   * Insert 20 million records using generate\_series (PostgreSQL):

sql

Copy code

INSERT INTO random\_table (id, value)

SELECT 1, generate\_series(1, 10000000);

INSERT INTO random\_table (id, value)

SELECT 2, generate\_series(1, 10000000);

1. **Query Without Materialized View:**

sql

Copy code

SELECT id, AVG(value) AS avg\_value, COUNT(\*) AS record\_count

FROM random\_table

GROUP BY id;

* + Execution Time: ~3 seconds.

1. **Create Materialized View:**

sql

Copy code

CREATE MATERIALIZED VIEW mv\_random\_table AS

SELECT id, AVG(value) AS avg\_value, COUNT(\*) AS record\_count

FROM random\_table

GROUP BY id;

1. **Query Materialized View:**

sql

Copy code

SELECT \* FROM mv\_random\_table;

* + Execution Time: ~60 milliseconds.

### ****Updating Materialized Views****

* Materialized views do not auto-update when the base table changes.
* Example:

sql

Copy code

DELETE FROM random\_table WHERE id = 1;

* Refresh the materialized view to reflect changes:

sql

Copy code

REFRESH MATERIALIZED VIEW mv\_random\_table;

### ****When to Use Materialized Views****

1. **When Data Updates Infrequently:**
   * Ideal for scenarios where base tables are updated periodically (e.g., monthly).
2. **Complex Queries:**
   * Use for queries involving heavy computations or joins.
3. **Reporting:**
   * Frequently used in dashboards and reporting systems where real-time data is not essential.

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

* **Scenario:** Generating business reports from a large table updated monthly.
* **Solution:** Create a materialized view for the report. Refresh it after monthly updates, enabling stakeholders to retrieve data instantly.

### ****Summary****

* **Materialized Views:**
  + Store both query and result data for faster query execution.
  + Require manual refresh to update data after changes in the base table.
* **Performance Improvement:**
  + Complex queries that take seconds or minutes to execute on large datasets can be reduced to milliseconds.
* **Best Use Cases:**
  + Reports, analytical dashboards, and scenarios where data updates are infrequent.

Materialized views are a powerful tool to optimize SQL performance, particularly for repetitive and computationally expensive queries.

4o