PostgreSQL

Views and Materialized Views

Surendra Panpaliya

Agenda

- Create and use views
- Create and refresh materialized views
- Comparison with Oracle views

Agenda

Hands-On:

Create customer-product view and materialized view

Assignment:

Create a sales summary materialized view and manually refresh it

Views and Materialized Views

Surendra Panpaliya

Views

VIRTUAL TABLE

REPRESENTING

RESULT OF

STORED QUERY

Views

Does not store data physically

Data is fetched live from base tables.

Acts like a saved query.

Views

Useful for

Simplifying complex queries,

Enforcing security

Creating a level of abstraction

Over raw data tables

Creating a View

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Advantages of Views

Simplifies complex queries.

Enhances security by restricting access

to specific rows or columns.

Provides a consistent, reusable interface.

Materialized Views

Physical copy

of the result

of a query

Materialized View

Stores data on disk

Can be **refreshed manually or periodically.**

Faster for read-heavy operations,

especially with large joins or aggregations.

Materialized View

```
CREATE MATERIALIZED VIEW mat_view_name AS
SELECT column1, column2
FROM base_table
WHERE condition
WITH DATA;
```

Materialized Views

Unlike regular views,

Materialized views

store data

Must be refreshed

when underlying

data changes.

Materialized View

```
-- Refresh manually
REFRESH MATERIALIZED VIEW mat_view_name;
```

Why Use Views or Materialized Views?

Feature	View	Materialized View
Storage	No	Yes (stored on disk)
Freshness	Always up-to-date	May become stale
Performance	Slower for large joins/queries	Faster for read-heavy workloads
Use Case	Simple abstraction/query reuse	Performance optimization
Can be indexed	No	Yes (materialized views can be indexed)

Summary: When to Use What?

Requirement	Use
Real-time data, always up-to-date	View
Performance boost for large aggregations	Materialized View
Simplify access for analysts	View
Precompute dashboard metrics	Materialized View
Reusable logic without data duplication	View
Indexable precomputed dataset	Materialized View

Creating a Materialized View

```
CREATE MATERIALIZED VIEW mat_view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition
WITH [ NO ] DATA;
```

Refreshing a Materialized View

Materialized views

must be refreshed

to get updated data

Advantages of Materialized Views

Improves performance

by storing query results.

Useful for read-heavy workloads

where data does not change frequently.

Differences Between Views and Materialized Views

STORAGE:

VIEWS DO NOT STORE DATA;

MATERIALIZED VIEWS STORE THE QUERY RESULT.

Differences Between Views and Materialized Views

Performance:

Views execute the underlying query

each time they are accessed;

Materialized views provide

faster access

to precomputed results.

Differences Between Views and Materialized Views

REFRESH:

VIEWS ALWAYS SHOW

UP-TO-DATE DATA;

MATERIALIZED VIEWS

NEED TO BE EXPLICITLY REFRESHED.

Creating and using views

Surendra Panpaliya

Creating and using views

Views can be used

to simplify complex queries

improve security

by restricting data access

Creating and using views

Provide a consistent Reusable interface for frequently accessed queries

Benefits of materialized views

Surendra Panpaliya

1. Performance Improvement

Faster Query Response

Materialized views

Store the result of a query physically.

Reduces the time needed

To fetch results

2. Reduced Load on Source Tables

Offloading Queries

Complex and resource-intensive queries

are offloaded from source tables,

reducing the load and

contention on those tables.

Improved Performance for Concurrent Users

IN ENVIRONMENTS WITH MANY CONCURRENT USERS,

MATERIALIZED VIEWS HELP IN

REDUCING THE PERFORMANCE IMPACT

ON THE UNDERLYING TABLES.

Consistent Results

MATERIALIZED VIEWS PROVIDE

CONSISTENT AND REPEATABLE RESULTS

FOR COMPLEX QUERIES,

ENSURING DATA INTEGRITY AND ACCURACY.

4. Historical Data and Snapshots

Data Archiving

Materialized views can be used

to create snapshots of data

at specific points in time.

Useful for

Historical analysis and reporting

4. Historical Data and Snapshots

Trend Analysis

Historical snapshots help in analyzing trends over time, such as patient admission rates or medication usage patterns.

5. Improved Data Security and Access Control

Restricting Access Sensitive data can be filtered and stored in a materialized view, restricting direct access to the underlying tables

5. Improved Data Security and Access Control

Role-Based Access Control

Materialized views can be used

to provide access to aggregated or

anonymized data,

ensuring compliance with

data protection regulations.

6. Reduced Network Traffic

Local Storage of Data

Materialized views store data locally,

reducing the need to fetch data

from remote databases repeatedly.

6. Reduced Network Traffic

Efficient Data Sharing

Data can be shared efficiently

between different departments or

applications without impacting

the performance of the source tables.

7. Data Consistency and Availability

Ensuring Data Consistency

Materialized views provide

a consistent state of data,

which is especially useful

in environments where

data consistency is critical.

7. Data Consistency and Availability

HIGH AVAILABILITY

MATERIALIZED VIEWS CAN BE REFRESHED

AT REGULAR INTERVALS,

ENSURING THAT DATA IS UP-TO-DATE AND

AVAILABLE FOR REPORTING AND ANALYSIS.

Conclusion

Materialized views provide

numerous benefits in the healthcare domain

by improving query performance,

reducing load on source tables,

simplifying complex queries,

enhancing data security.

Conclusion

Are a valuable tool

for efficient data management

analysis in healthcare databases.

Refreshing materialized views

Surendra Panpaliya

Refreshing materialized views

Materialized views can be refreshed

Manually or Automatically

at specified intervals,

depending on the requirements.

Types of Refresh Methods

Complete Refresh

This rebuilds

the entire materialized view

from scratch.

Types of Refresh Methods

Incremental Refresh

Known as "fast refresh,"

Updates only the changes since

the last refresh,

which is more efficient

than a complete refresh.

Automatic Refresh

Set up Automatic refresh

using pg_cron extension or

Similar job scheduler

to refresh the materialized view

at regular intervals.

Automatic Refresh

Example: Using pg_cron

First, install the pg_cron extension

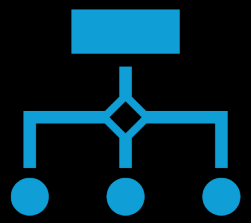
(if not already installed)

CREATE EXTENSION pg_cron;

Summary

Depending on the requirements,

materialized views can be refreshed



manually or automatically.

Summary

Automating the refresh process

ensures timely updates

without manual intervention

PostgreSQL vs Oracle Views and Materialized Views

Surendra Panpaliya

BASIC VIEW CREATION

PostgreSQL

```
CREATE VIEW customer_basic_info AS
SELECT customer_id, customer_name
FROM telecom.customers;
```

BASIC VIEW CREATION

Oracle

```
CREATE VIEW customer_basic_info AS
SELECT customer_id, customer_name
FROM telecom.customers;
```

✅ **No difference in basic syntax.**

Both databases allow views based on SELECT queries.

UPDATABLE VIEWS

Feature	PostgreSQL	Oracle
Updatable View	✅ Yes (if based on one table and no joins/aggregates)	✅ Yes
With Check Option Support	✅ Supported	✅ Supported

UPDATABLE VIEWS

PostgreSQL

```
CREATE VIEW updatable_view AS
SELECT customer_id, customer_name
FROM telecom.customers
WHERE circle = 'Delhi'
WITH CHECK OPTION;
```

UPDATABLE VIEWS

Oracle

```
CREATE VIEW updatable_view AS
SELECT customer_id, customer_name
FROM telecom.customers
WHERE circle = 'Delhi'
WITH CHECK OPTION;
```

MATERIALIZED VIEWS – CREATION

PostgreSQL

```
CREATE MATERIALIZED VIEW recharge_summary AS
SELECT circle, COUNT(*) AS total
FROM telecom.recharges
GROUP BY circle
WITH DATA;
```

MATERIALIZED VIEWS – CREATION

Oracle

```
CREATE MATERIALIZED VIEW recharge_summary
BUILD IMMEDIATE
REFRESH ON DEMAND  AS
SELECT circle, COUNT(*) AS total
FROM telecom.recharges
GROUP BY circle;
```

MATERIALIZED VIEWS – CREATION

Feature	PostgreSQL	Oracle
Refresh Options	Manual only (REFRESH MATERIALIZED VIEW)	✅ Manual / ✅ Automatic (ON COMMIT / ON DEMAND / Scheduled)
Incremental Refresh	❌ Not Supported	✅ Supported with Materialized View Logs
Index Support	✅ Yes	✅ Yes

MATERIALIZED VIEW REFRESH

- **PostgreSQL**
- REFRESH MATERIALIZED VIEW recharge_summary;
- **Oracle (Manual Refresh)**
- EXEC DBMS_MVIEW.REFRESH('RECHARGE_SUMMARY');

Oracle (Auto Refresh)

```
CREATE MATERIALIZED VIEW recharge_summary
BUILD IMMEDIATE
REFRESH FAST
START WITH SYSDATE
NEXT SYSDATE + 1
AS
SELECT circle, COUNT(*) AS total
FROM telecom.recharges
GROUP BY circle;
```

PERFORMANCE AND USE CASES

Use Case	PostgreSQL	Oracle
Real-time dashboards	Use Views	Use Views
Heavy aggregation	Use Materialized Views	Use Materialized Views
Incremental aggregation updates	Not supported natively	Use Materialized View Logs + FAST REFRESH
Replicated remote snapshots	❌ Not supported	✅ Supported via database links
Auto refresh without external scheduler	❌ Use pg_cron / pgAgent	✅ Use NEXT clause

LIMITATIONS IN POSTGRESQL VS ORACLE

Feature	PostgreSQL	Oracle
Auto refresh on schedule	❌ External only	✅ Built-in
FAST refresh	❌ Not supported	✅ Supported
Refresh on COMMIT	❌ Not supported	✅ Supported
Materialized view logs	❌ Not available	✅ Required for FAST
Partitioned materialized views	❌ Not supported	✅ Supported
Dependency tracking (cascading)	✅ Supported	✅ Supported

Summary Table

Feature	PostgreSQL	Oracle
View Syntax	Standard SQL	Standard SQL
Updatable View with Check Option	✅ Yes	✅ Yes
Materialized Views	✅ Yes (Manual refresh only)	✅ Yes (Manual + Auto Refresh)
Incremental/FAST Refresh	❌ Not supported	✅ Supported with logs
ON COMMIT Refresh	❌ Not supported	✅ Yes
Schedulers	External (pg_cron, pgAgent)	Built-in with NEXT clause
Index Support on Materialized Views	✅ Yes	✅ Yes

Conclusion

Use **PostgreSQL materialized views**

for caching heavy queries and

refreshing them with

REFRESH MATERIALIZED VIEW.

Conclusion

Use **Oracle materialized views**

when you need **incremental refresh**,

ON COMMIT, or

automatic scheduling

without third-party tools.

Conclusion

PostgreSQL views are ideal for abstraction;

Oracle has more **enterprise-grade features**

built-in for **distributed systems**,

snapshots, and **automated refresh logic**.

View vs Materialized View

Feature	View	Materialized View
Definition	Saved SQL query	Saved SQL query with physical storage
Data Storage	No (virtual)	Yes (stored snapshot)
Real-time Data	Always up-to-date	Requires manual refresh
Use Case	Simplify complex queries	Performance optimization, reporting

Benefits of Views

Benefit	Description
Simplifies SQL	Encapsulates logic
Reusability	Use in multiple queries
Real-time data	Always reflects latest changes

Create and Use Materialized Views

Stores **precomputed results**

Faster for large data sets

Needs manual refresh

Summary Table

Feature	View	Materialized View
Storage	Virtual	Physical
Performance	Executes query each time	Fast after refresh
Use Case	Simplify query logic	Reporting, heavy aggregation
Real-time	Yes	No (needs REFRESH)

Basic Views

Feature	Oracle	PostgreSQL
Create View	CREATE OR REPLACE VIEW	CREATE OR REPLACE VIEW
Real-time data	✅	✅
Updatable View	✅ (if no joins, group by, etc.)	✅ (similar rules)
WITH CHECK OPTION	Supported	Supported

Materialized Views

Feature	Oracle	PostgreSQL
Create Materialized View	CREATE MATERIALIZED VIEW	CREATE MATERIALIZED VIEW
Refresh Modes	FAST, COMPLETE, FORCE, ON COMMIT	REFRESH MATERIALIZED VIEW (manual only)
Automatic Refresh	✅ ON COMMIT or ON DEMAND	❌ Manual refresh only
Incremental Refresh (FAST)	✅ Supported with materialized view logs	❌ Not natively supported (needs triggers or custom logic)
Query Rewrite	✅ QUERY REWRITE ENABLED	❌ Not supported

Key Differences: PostgreSQL vs Oracle

Feature	Oracle	PostgreSQL
View Behavior	Same	Same
Materialized View Refresh	Auto & Manual	Manual Only
Incremental Refresh	✅ (FAST with logs)	❌ (Full refresh only)
Query Rewrite	✅	❌
Storage Location	Tablespace options available	Default tablespace

Performance Considerations

Use Case	Oracle	PostgreSQL
Real-time OLTP views	Good	Good
Heavy aggregations	Use Materialized View with FAST refresh	Use Materialized View (Manual refresh)
Reporting systems	QUERY REWRITE ENABLED	Materialized View + Manual Refresh

Summary Table

Feature	Oracle Views	PostgreSQL Views
Standard View	✅	✅
Materialized View	✅	✅
Automatic Refresh	✅	❌
Incremental Refresh	✅	❌
Updatable Views	Partially	Partially
Query Rewrite	✅	❌

Conclusion

If you need	Recommendation
Real-time virtual tables	Use Views (Both Oracle & PostgreSQL)
Performance caching	Use Materialized Views
Incremental refresh / fast refresh	Oracle (better support)
Full refresh reporting	PostgreSQL (use REFRESH MATERIALIZED VIEW)

Surendra Panpaliya
Founder and CEO
GKTCS Innovations
https://www.gktcs.com