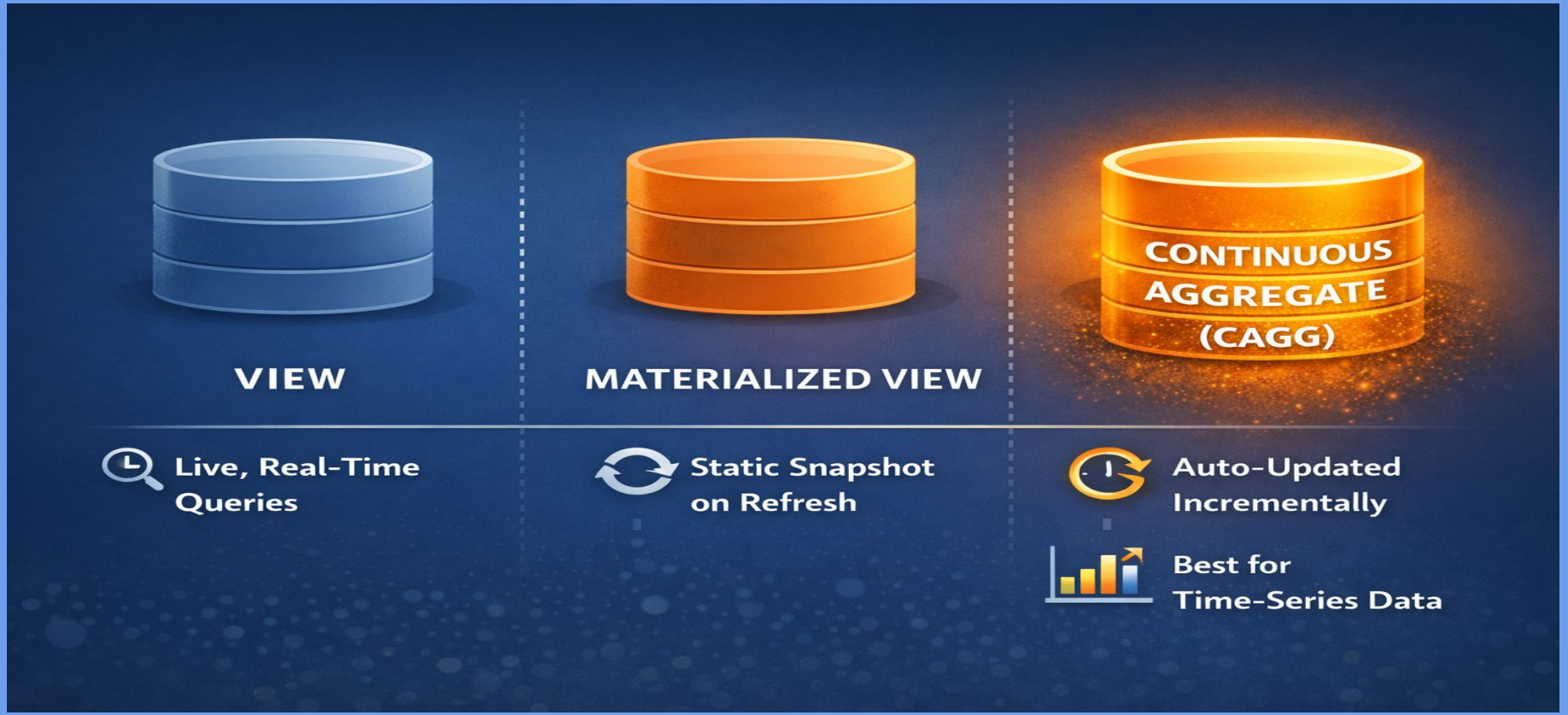
MAXIMIZE INSIGHTS AND PERFORMANCE WITH THE RIGHT TIME-SERIES VIEW

RAJNISH KUMAR
DATA ARCHRTECT

Content

Different Types Of View



What is a View?

A **view** is a **virtual table** in a database.
It **does not store data physically**; instead, it stores a **saved SQL query**.

How a View Works

when you query a view, the database:

1. Takes the SQL query stored in the view
2. Executes it on the underlying base tables
3. Returns the result **as if it were a real table**

For User's perspective – it behaves just like a table.

What is a Materialized View?

A **materialized view** is a **physical snapshot of a query's result**.
Unlike a standard view (which stores only a query), a materialized view **stores the actual data on disk**.

How Materialized View Works

1. Creation

- The database executes the query **once**
- The result set is stored as a **physical table behind the scenes**

How To Create,

```
CREATE MATERIALIZED VIEW view_user_daily_activity AS
SELECT date_trunc('month', order_date) AS month,
       SUM(steps) AS total_step
FROM device_data_table
GROUP BY 1;
```

2. Access

- Querying a materialized view **reads precomputed data**
- No joins or aggregations are recalculated at query time

```
SELECT * FROM view_user_daily_activity;
```

This is why it’s **fast**.

3. Refresh

- Materialized views **do not update automatically**
- They remain static until refreshed

Postgre:

```
REFRESH MATERIALIZED VIEW CONCURRENTLY view_user_daily_activity;
```

NOTE: avoids blocking reads, we need to create required index

What `CONCURRENTLY` does

It allows **reads to continue while the materialized view is being refreshed**.

Without it:

- PostgreSQL **locks the materialized view**
- All `SELECT` queries **wait or fail**
- Bad for dashboards & live apps

With it:

- Users keep querying the **old data**
- PostgreSQL builds the **new data in the background**
- Once done, it **atomically swaps** old → new

Quick Comparison: View vs Materialized View

Feature	View	Materialized View
Data Stored	✗ No	✓ Yes
Storage	Minimal (query only)	Uses disk space
Freshness	Always fresh	Stale until refreshed
Query Speed	Depends on query complexity	Very fast ⚡
Use Case	Abstraction, security, simplicity	Analytics, reporting, heavy queries

View = live query shortcut • Materialized View = precomputed snapshot

Pros & Cons of Materialized Views

Pros

- Speed
- Offloading
- Predictable Query Time
- Fost services

Cons

- Data Staleness
- Disk Space Usage
- Manual Refresh Needed
- Expensive Refresh

Continuous Aggregate (CAGG)

A **Continuous Aggregate (CAGG)** is a **materialized view in TimescaleDB that is continuously and incrementally updated** as new data is added to a **hypertable**. Unlike a regular materialized view that requires manual refreshes, a continuous aggregate automatically keeps aggregated data up-to-date with minimal overhead.

Key points:

- Works on **hypertables** (TimescaleDB's optimized time-series tables).
- Performs **incremental refreshes**, meaning only new or changed data is aggregated.
- Can **downsample data**, e.g., daily, weekly, monthly from a high-frequency time-series table.
- Improves **query performance** for large datasets.

Continuous Aggeriates (CAGGS) in TimescaleDB

Powerful Time-Series Data Aggegation

- ✓ Incremetinal updates as new data arrives.
- ✓ Improved query performance for analytical worklads.
- ✓ Automatic refresh polices (e..gl, daily, monthly).

- ✓ Downsample & summiarize high-frequency data easily

```
CREATE MATERIALIZED VIEW cagg_user_daily_activity
WITH (timescaledb.continuous) AS
SELECT
    time_bucket('1 month', order_date) AS month,
    SUM(steps) AS total_step
FROM device_data_table
GROUP BY month;
```

`WITH (timescaledb.continuous)`

- Marks this as a **Continuous Aggregate** that refreshes automatically and incrementally.

Refresh Policy

You can also set a **refresh interval**, so TimescaleDB updates the aggregate automatically:

```
SELECT add_continuous_aggregate_policy('cagg_user_daily_activity',
    start_offset => INTERVAL '1 month',
    end_offset => INTERVAL '1 day',
    schedule_interval => INTERVAL '1 hour');
```

- `start_offset` → how far back to start refreshing.
- `end_offset` → leave small buffer for late-arriving data.
- `schedule_interval` → how often TimescaleDB updates the CAGG.

Continuous Aggregate (CAGG)

AUTO & INCREMENTAL UPDATE

- ✔ Incremental Updates
- ✔ Always Up-to-Date!

Regular Materialized View (MV)

MANUAL & FULL REFRESH

- ⚠ Manual Refresh
- ⚠ Full Re-Aggregation

Conclusion:

Continuous Aggregates (CAGGs) in TimescaleDB provide a **powerful and efficient way to aggregate time-series data**. Unlike regular materialized views that require manual refreshes and can be slow on large datasets, CAGGs **incrementally update** as new data arrives, significantly improving query performance.

By using features like `time_bucket` and **automatic refresh policies**, developers can **downsample, summarize, and analyze high-frequency data** (daily, monthly, or yearly) without worrying about stale results or heavy computation.

THANK YOU

FOR YOUR TIME AND ATTENTION

RAJNISH KUMAR,
Data Architect