Materialised views now and the future

Thom Brown | PostgreSQL Conference Europe 2013
Disclaimer

“Materialised” vs “Materialized”
“Optimiser” vs “Optimizer”
“Catalogue” vs “Catalog”
“Behaviour” vs “Behavior”
“Customisable” vs “Customizable”
“Favourite” vs “Favorite”
Disclaimer

Elizabeth Windsor
@Queen_UK

For the avoidance of doubt, there is no such thing as 'American English'. There is the English language and there are mistakes.
Materialised Views in PostgreSQL

• Back in 2009, 2\textsuperscript{nd} most-requested PostgreSQL feature on UserVoice was: Materialised Views! (Hot Standby was 1\textsuperscript{st} if you're curious)

• PostgreSQL now has Materialised Views in version 9.3!

• Designed and developed by Kevin Grittner of EDB (major contributor, PostgreSQL committer). (thanks Kevin!)

• Out of 175 people surveyed with the question “What's your favourite 9.3 feature?”, 0 people said Materialized Views... it wasn't in the list of choices though.
Table vs View vs Materialised View

Table

Application

Query directly

Table

Read from table each time
Table vs View vs Materialised View

View

Application

Query indirectly

View

SELECT columns
FROM table
WHERE column = value;

Table

Read from table each time
Table vs View vs Materialised View

**Materialised View**

- Application
  - Query directly
  - Read direct from materialised view each time

- Table
  - Read when MV created or manually refreshed.

**Materialised View**

```sql
SELECT columns
FROM table
WHERE column = value;
```
# Table vs View vs Materialised View

<table>
<thead>
<tr>
<th>Table</th>
<th>View</th>
<th>Materialised View</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Stores data</td>
<td>• Stores a query</td>
<td>• Stores a query</td>
</tr>
<tr>
<td>• Returns its data</td>
<td>• Executes its query</td>
<td>• Executes its query upon creation or refresh</td>
</tr>
<tr>
<td>• Can modify its data</td>
<td>• Returns results</td>
<td>• Stores results</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Returns stored results</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cannot modify its data</td>
</tr>
</tbody>
</table>
-- Create the initial view
CREATE VIEW v_data AS
  SELECT ...;

-- Create a table based on a view
CREATE TABLE mv_data AS
  SELECT * FROM v_data;

-- Refresh the table
BEGIN;
  DROP TABLE mv_data;
  CREATE TABLE mv_data AS
    SELECT *
    FROM v_data;
END;
Previously in PostgreSQL...

```
-- Create supporting tracking tables, functions, triggers...

CREATE TABLE track_mvs;
CREATE FUNCTION create_mv;
CREATE FUNCTION drop_mv;
CREATE FUNCTION refresh_mv;
CREATE TRIGGER t_mv_update;
CREATE TRIGGER t_mv_insert;
CREATE TRIGGER t_mv_delete;
CREATE VIEW v_summary;
```
Using Materialised Views in PostgreSQL 9.3

-- Create a materialised view

CREATE MATERIALIZED VIEW mv_data AS
    SELECT d.id, d.department, count(d.department)
    FROM staff s
    INNER JOIN dept d ON s.dept_id = d.dept_id
    GROUP BY d.id, d.department WITH NO DATA;

-- Refresh a materialised view

REFRESH MATERIALIZED VIEW mv_data;
Should I use a Materialised View?

• For data that doesn't need to be up-to-date.
• For data that takes a long time to query (e.g. requires lots of joins or processing) but frequently needed or needs to be prepared ahead of time.
• Can be based on any read-only query.
• Can have indexes like regular tables.
• Can be useful for caching foreign table data.
• Sacrifice freshness for speed.
• Takes up disk space to store results.
• Returns an error upon querying if created or refreshed using WITH NO DATA.
Materialised Views in 9.3

- CREATE / DROP / ALTER / REFRESH MATERIALIZED VIEW.
- `\dm` command in `psql` to list MVs.
- `pg_matviews` system catalogue.
  - Contains query definition.
- Requires an exclusive lock to refresh.
  - Needs to wait for all active queries to complete.
  - Cannot be used while refreshing.
- Snapshot implementation
- Can produce a lot of WAL data for large refreshes and therefore a lot of replication traffic.
- Cannot be temporary or unlogged, unlike tables.
- Refreshing “freezes” rows.
What about pg_dump with MVs in 9.3?

- Data not dumped, only the query definition.
- Outputs CREATE MATERIALIZED VIEW statement with WITH NO DATA clause.
- Later outputs REFRESH MATERIALIZED VIEW statement if it was populated at the time of backup dump.
Exclusive lock issue explained

```sql
CREATE TABLE data (id serial PRIMARY KEY, value int);
INSERT INTO data (value) VALUES (1);
CREATE MATERIALIZED VIEW mv_data AS SELECT * FROM data;
```

<table>
<thead>
<tr>
<th>Session 1</th>
<th>Session 2</th>
<th>Session 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN;</td>
<td>SELECT *</td>
<td>BEGIN;</td>
</tr>
<tr>
<td></td>
<td>FROM mv_data;</td>
<td>SELECT * FROM mv_data;</td>
</tr>
<tr>
<td>-- Access Share Lock acquired</td>
<td>-- Waiting for T2 to finish to acquire Exclusive Lock</td>
<td>-- Waiting for T1 to finish to acquire Access Share Lock</td>
</tr>
<tr>
<td>INSERT INTO data VALUES (2);</td>
<td>REFRESH MATERIALIZED VIEW mv_data;</td>
<td>COMMIT;</td>
</tr>
<tr>
<td>-- Waiting for T2 to finish to acquire Exclusive Lock</td>
<td>-- -- releases Access Share Lock`</td>
<td>-- Access Share Lock acquired</td>
</tr>
<tr>
<td>-- Exclusive Lock acquired and refresh completes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-- Exclusive Lock acquired and refresh completes

-- Access Share Lock acquired
Exclusive lock issue explained

Key
- Access Share Lock
- Exclusive Lock
- Waiting to acquire lock

Time
- Query 1
- Query 2
- Query 3
- Query 4
- Materialised View Refresh
- Query 5
- Materialised View Refresh
- Query 5
MV exclusive lock mitigation in 9.3

• Shamelessly stolen from Depesz (www.depesz.com)

```
-- Create a copy of the materialised view
DO $$
BEGIN
  EXECUTE 'CREATE MATERIALIZED VIEW mv_new AS ' || pg_get_viewdef('mv'::regclass);
END $$;

-- Replace the MV atomically
BEGIN;
  DROP MATERIALIZED VIEW mv;
  ALTER MATERIALIZED VIEW mv_new RENAME TO mv;
COMMIT;
```

• Transactions don't need to wait for MV build, but still requires Access Exclusive Lock for DROP step.
Back To The Future...

A quirk with using materialised views:

• With a materialised view that is refreshed non-concurrently it's possible for a single transaction to see data in a materialised view that is newer than that of the underlying tables.

• Concurrently-refreshed materialised views don't exhibit this behaviour.
<table>
<thead>
<tr>
<th>Session 1</th>
<th>Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE data (</td>
<td>SET SESSION CHARACTERISTICS AS</td>
</tr>
<tr>
<td>id serial PRIMARY KEY,</td>
<td>TRANSACTION ISOLATION LEVEL</td>
</tr>
<tr>
<td>ts timestamp</td>
<td>SERIALIZABLE;</td>
</tr>
<tr>
<td>INSERT INTO data (ts) VALUES (now());</td>
<td></td>
</tr>
<tr>
<td>CREATE MATERIALIZED VIEW mv_data AS</td>
<td></td>
</tr>
<tr>
<td>SELECT id, ts FROM data;</td>
<td></td>
</tr>
<tr>
<td>UPDATE data SET ts = now();</td>
<td></td>
</tr>
<tr>
<td>REFRESH MATERIALIZED VIEW mv_data;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-- Two rows returned with different ts values.</td>
<td></td>
</tr>
</tbody>
</table>
Materialised Views in 9.4

- REFRESH MATERIALIZED VIEW CONCURRENTLY.
  - Doesn't block reads.
  - Can produce more or less WAL than non-concurrent refresh depending on number of changes.
  - Only one refresh allowed at any one time.
  - MV needs to be already populated.
  - Requires a unique index.
  - VACUUMing becomes relevant.
  - Unlike non-concurrent form, doesn't freeze rows.
  - Cannot be used with WITH NO DATA option (as it wouldn't make sense).
"CONCURRENTLY" implementation

Materialised View
SELECT columns
FROM table
WHERE column = value;

Temp Materialised View
SELECT columns
FROM table
WHERE column = value;

Full Outer Join

Inserts

Deletes

Table
Materialised Views roadmap

(These are not necessarily going to be implemented)

• Unlogged materialised views.
  • Same as WITH NO DATA state upon crash.

• Incremental materialised views.
  • Updates the MV as tables are updated.
  • Customisable level of “eagerness”.
  • Complicated by features such as aggregates and NOT EXISTS.
  • Support for recursive queries will likely arrive more than 1 release later.
Materialised Views roadmap

• **CREATE OR REPLACE MATERIALIZED VIEW**
  • Just an oversight that it wasn't added.

• Updates for concurrent refreshes.
  • Would allow for HOT updates.

• Lazy automatic refresh based on table modification statistics.
  • Staleness testing.

• Optimiser awareness of materialised views.
  • Pull in MV data if fresh enough.
  • Treat MVs like indexes.
Materialised Views roadmap

- Incremental update “eagerness”
  - **Very Eager** – Applied before incrementing command counter so appears up-to-date within the transaction.
  - **Eager** – Applied at commit time, and visible with all other changes in the transaction.
  - **Inbetween** – Queued to apply immediately after transaction commit asynchronously.
  - **Lazy** – Queued to apply on a specified schedule.
  - **Very Lazy** – Queued to be applied on demand.
  - Trade-off: More eager = fresh more frequently but with the price of greater overhead.