Incremental View Maintenance

A Fast Method for Updating Materialized Views

Yugo Nagata
nagata@sraoss.co.jp

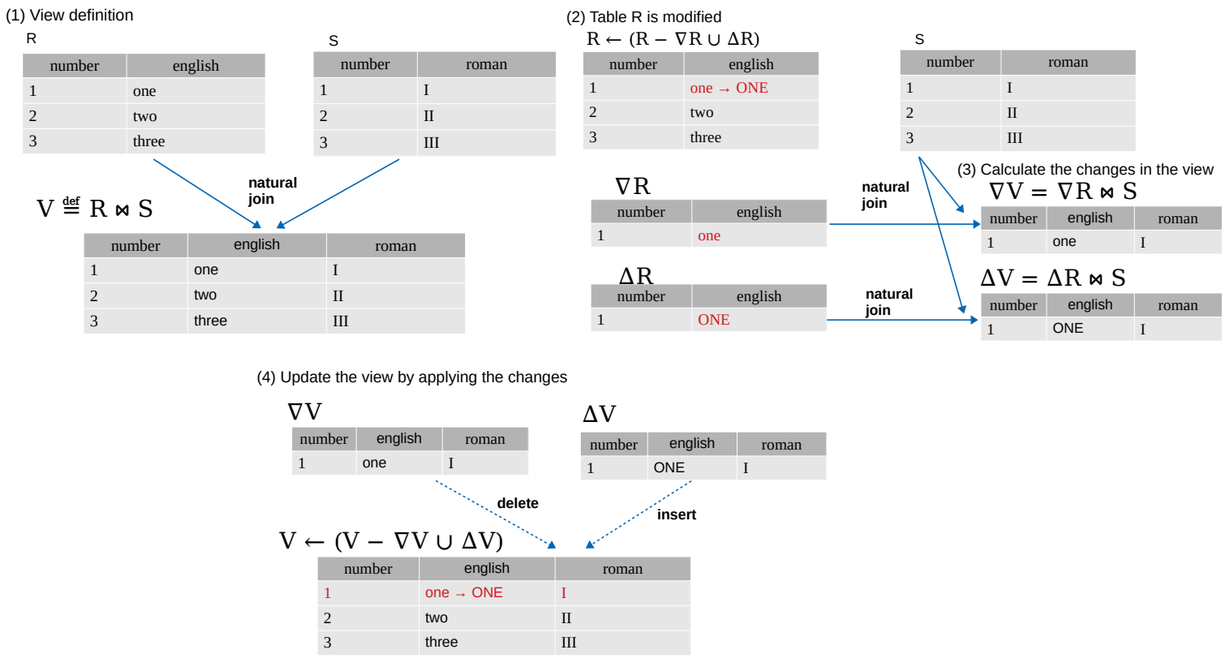
PGConf.dev 2025

Materialized View:

- The results of the view definition query are stored in the database.
- Requires maintenance when base tables are modified.
- REFRESH MATERIALIZED VIEW re-computes the materialized view contents using the latest table state.

Incremental View Maintenance (IVM):

- Only the incremental changes to the view are computed and applied.



Proposed Patch: Adds support for materialized views that are updated automatically and incrementally when base tables change.

```
test=# CREATE INCREMENTAL MATERIALIZED VIEW mv_ivm AS
        SELECT aid, bid, abalance, bbalance
        FROM pgbench_accounts JOIN pgbench_branches USING (bid)
        WHERE abalance > 0 OR bbalance > 0;

test=# UPDATE pgbench_accounts SET abalance = 1000 WHERE aid = 1;
UPDATE 1
Time: 18.634 ms
test=# SELECT * FROM mv_ivm WHERE aid = 1;
 aid | bid | abalance | bbalance
-----+-----+----------+----------
   1 |   1 |     1000 |       10
(1 row)
```

Takes 18 ms to update vs 10 seconds using REFRESH (pgbench scale factor 100)

Design:

- AFTER triggers are automatically created on base tables.
- Uses Transition Tables to extract changes.
- Rewrites view definition queries and executes them to calculate incremental view changes.

View Definition Query (modified table)

```
SELECT ... FROM R, S WHERE ...
```

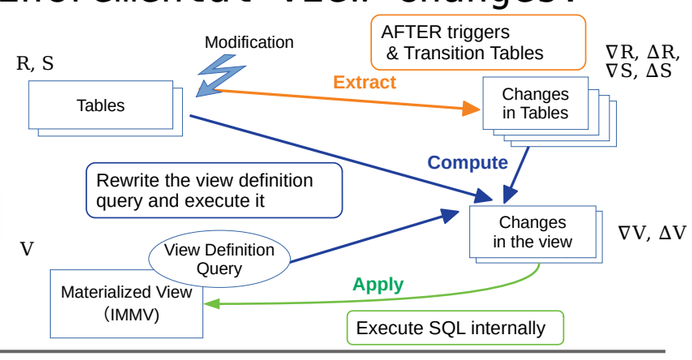
rewrite

After Rewritten

```
SELECT ... FROM new_table_R, S WHERE ...
```

Transition Table (rows inserted into R)

→ The results are "rows to be inserted into the view".



Simultaneous Multiple Tables Modification:

- Multiple tables could be modified in a statement. (ex. foreign key constraint, CTE)
- Self-join shares the same situation
  (Tables appearing in a query repeatedly ≒ Different tables with the same contents)
- Requires both *pre-* and *post-update* states of tables.

View definition	V ≝ R ⋈ S
Tables modifications	R _new = R_old ∪ ΔR, S_new = S_old ∪ ΔS
Incremental change	ΔV = (ΔR ⋈ S_old) ∪ (R_new ⋈ ΔS)

- *Pre-update* state of a table is reconstructed using:

```
SELECT… FROM tbl
 WHERE ivm_visible_in_prestate(t.tableoid, t.ctid, matview_oid)
UNION ALL SELECT … FROM deleted_tuples_from_tbl;
```

- *ivm_visible_in_prestate* function returns true if a row is visible with a snapshot before the table modification acquired in the BEFORE trigger.

Duplicate Tuples in View

- The multiplicity of each tuple in the incremental view changes is determined using count(*).
- For deletion, tuples are deleted according to their specified multiplicity using the row_number() window function.
- For insertion, tuples are duplicated to match specified multiplicity using the generate_series() function.

Commitfest item:

pg_ivm:

Maintenance of View with DISTINCT

- A hidden column *_ivm_count_* tracks row multiplicity.
- Row is deleted from view when count reaches zero.

Aggregate Support

- Supports built-in count, sum, avg, min, and max.
- More than one hidden column is created per aggregate.

pg_ivm:

- An extension module version of IVM implementation
- Supports more complex views (sub-queries, CTEs.)
- Includes fixes not yet applied to the patch but to be.

Open questions:

1. Trigger-based Design: Should we avoid relying on triggers like declarative partitioning?
2. Features Scope: Exclude aggregate, DISTINCT, and tuple duplicate supports in the first release to simplify the patch and improve its reviewability?
3. Hidden Columns: How should they be handled?
4. Pre-update State of Table: Need infrastructure to scan a table using a specified snapshot, instead of using the crafted sub-query?
5. Syntax: “CREATE INCREMENTAL MATERIALIZED VIEW” is tentative. Is "CREATE MATERIALIZED VIEW … WITH (reloptions)" preferable?.
6. Other issues: EXPLAIN outputs, CONTEXT in an error message, etc.

More design review and community discussion are needed.