SRA OSS

pg_ivm: Extensions for Rapid Materialized View Update

Yugo Nagata

PGConf.EU 2024

pg_ivm

- An extension module that provides Incremental View Maintenance (IVM) feature for PostgreSQL.
  - IVM = a way to make materialized views up-to-date rapidly.

Yugo Nagata

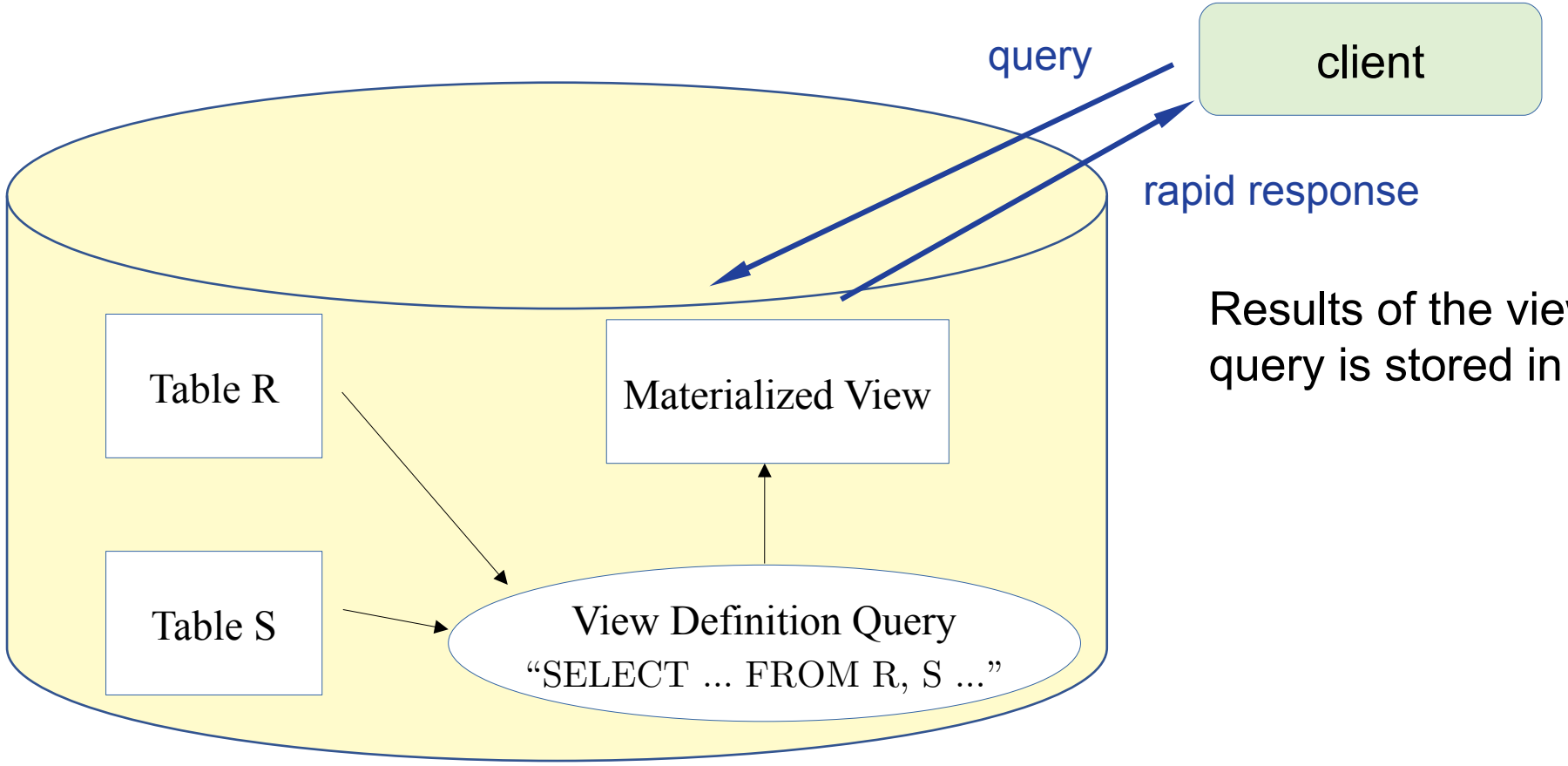
- Software Engineer & Researcher at SRA OSS K.K.
  - R&D related to PostgreSQL
  - Technical support & Consulting
- PostgreSQL Contiributor
- Author of pg_ivm

Outline

- What is Incremental View Maintenance
- What is pg_ivm
  - How to use it
  - How it works
- Performance
- Development and Future Plans

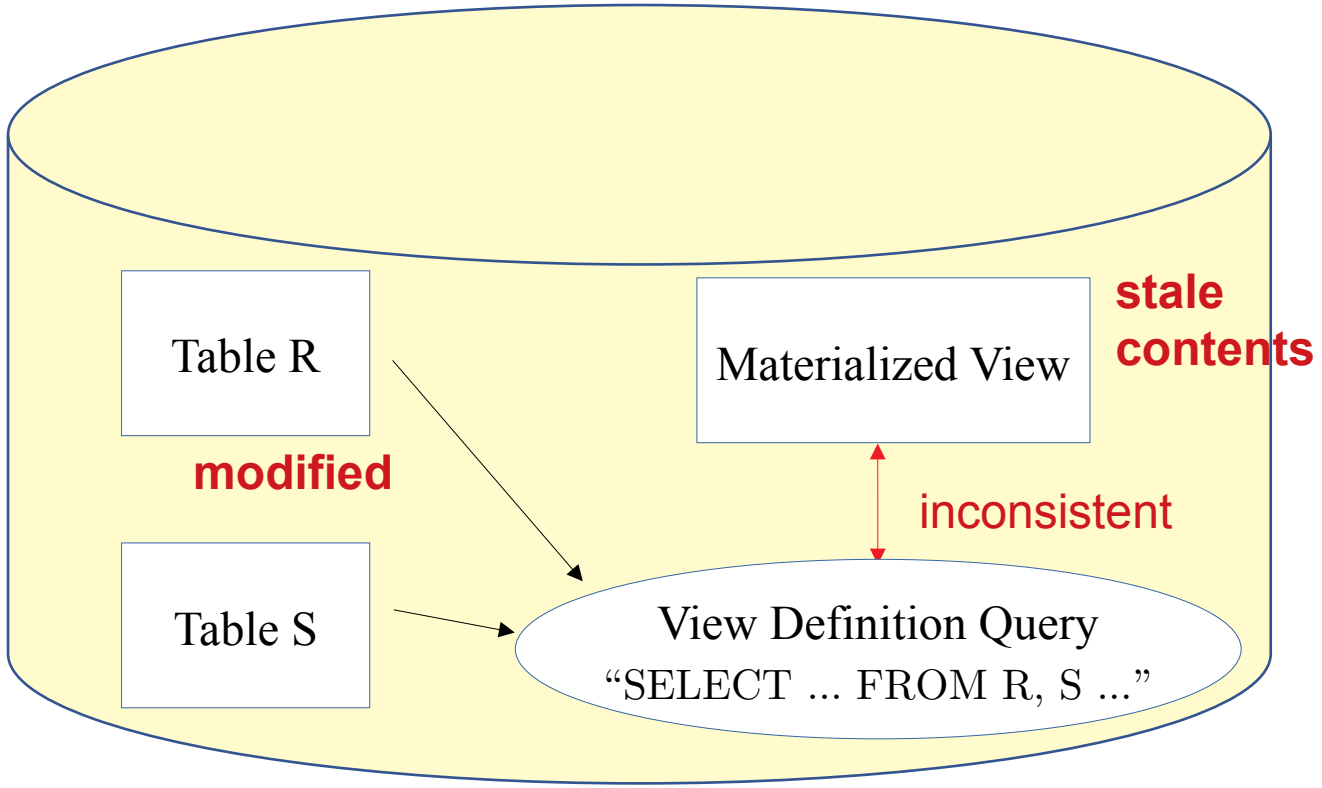
Incremental View Maintenance

Incremental Maintenance of Materialized Views (1)



Results of the view definition query is stored in the database

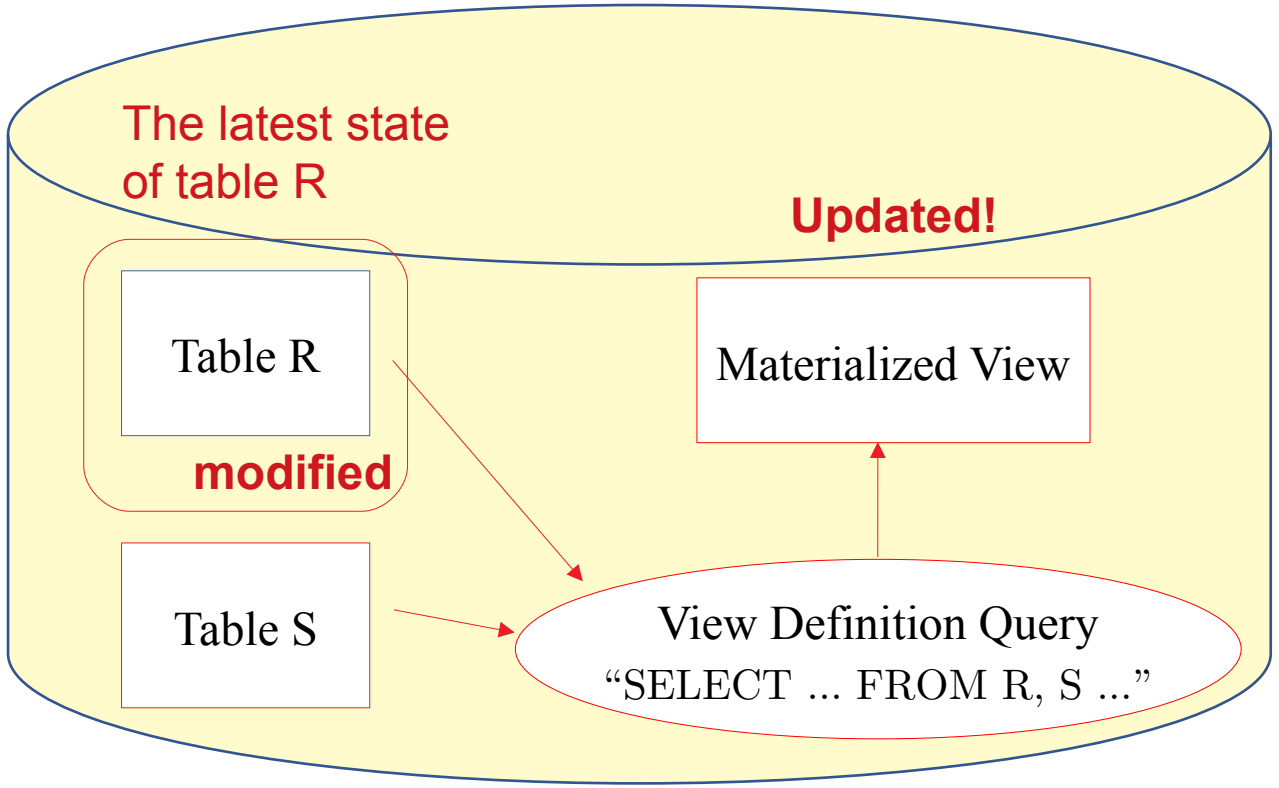
Incremental Maintenance of Materialized Views (2)



Results of the view definition query is stored in the database

Needs maintenance after a table is modified.

Incremental Maintenance of Materialized Views (3)

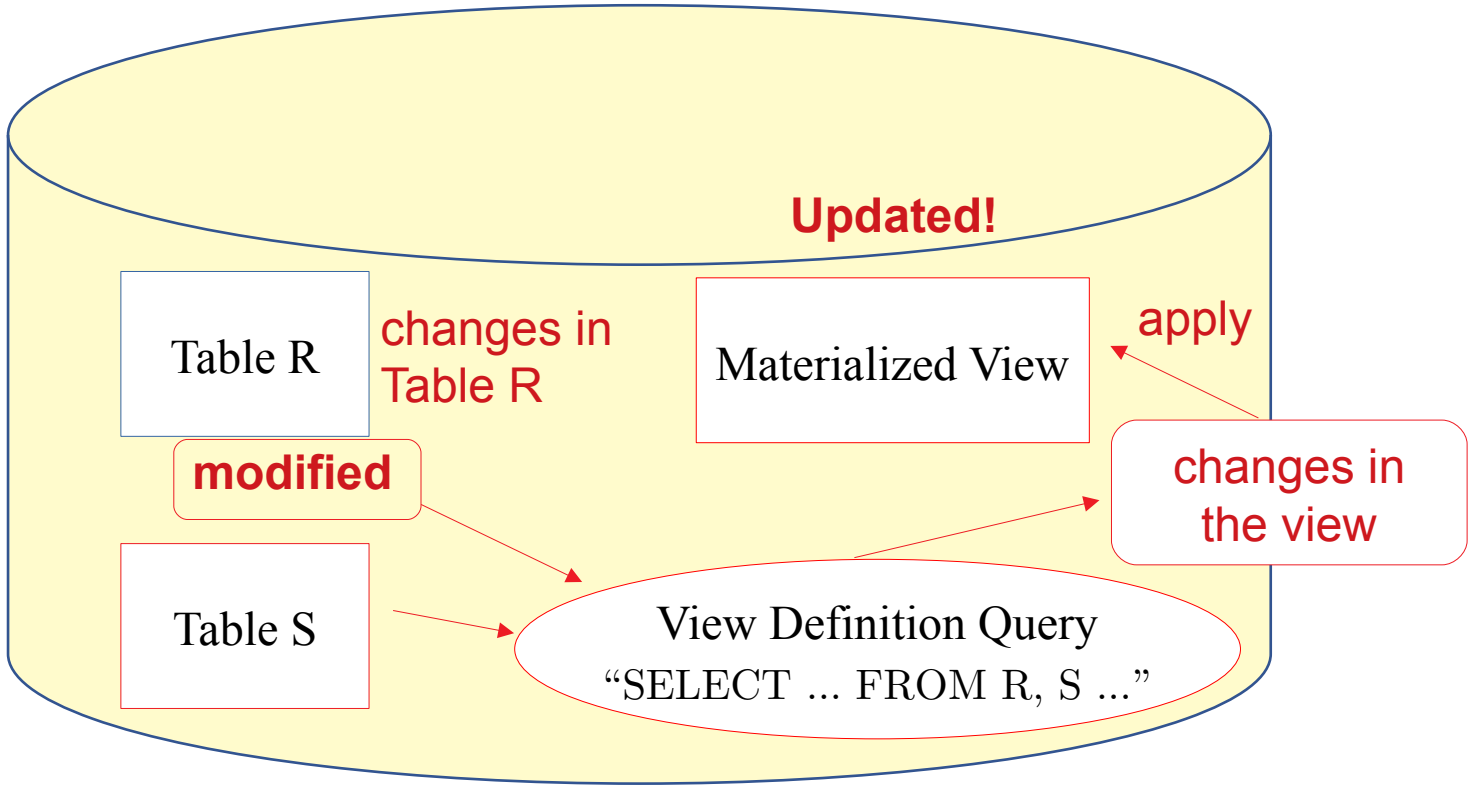


REFRESH MATERIALIZED VIEW

II

Re-compute the materialized view contents using the latest table state

Incremental Maintenance of Materialized Views (4)



REFRESH MATERIALIZED VIEW

II

Re-compute the materialized view contents using the latest table state

Compute only the incremental changes in the view and apply it

II

Incremental View Maintenance

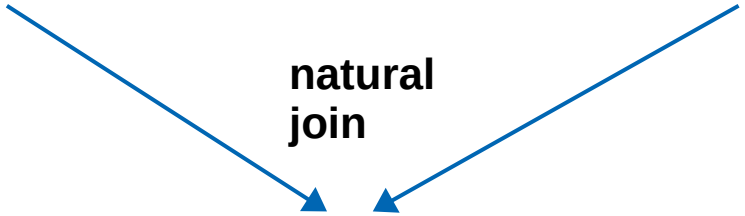
A Simple Example of Incremental View Maintenance (1)

R

number	english
1	one
2	two
3	three

S

number	roman
1	I
2	II
3	III



natural join

$$V \stackrel{\text{def}}{=} R \bowtie S$$

number	english	roman
1	one	I
2	two	II
3	three	III

A Simple Example of Incremental View Maintenance (2)

Table R is modified

$R \leftarrow (R - \nabla R \cup \Delta R)$

number	english
1	one → ONE
2	two
3	three

S

number	roman
1	I
2	II
3	III

∇R

number	english
1	one

ΔR

number	english
1	ONE

natural join

natural join

Calculate the changes in the view

$\nabla V = \nabla R \bowtie S$

number	english	roman
1	one	I

$\Delta V = \Delta R \bowtie S$

number	english	roman
1	ONE	I

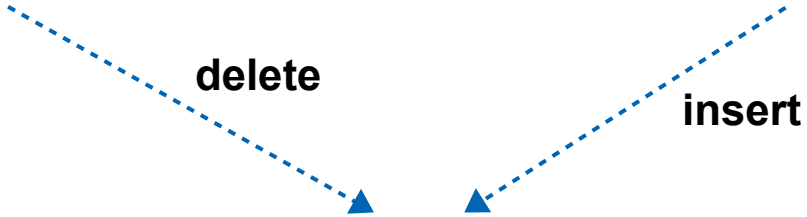
A Simple Example of Incremental View Maintenance (3)

∇V

number	english	roman
1	one	I

ΔV

number	english	roman
1	ONE	I



$V \leftarrow (V - \nabla V \cup \Delta V)$

number	english	roman
1	one → ONE	I
2	two	II
3	three	III

Update the view by applying the changes

What is pg_ivm

PostgreSQL and Incremental View Maintenance

- Incremental View Maintenance (IVM) is not supported in PostgreSQL
  → A set of patches have been proposed.

- Some request to use IVM with the current PostgreSQL
  → pg_ivm developed for providing the feature as not only patches but also an extension module.

  - Other purposes:
    - Improve opportunities to get feedback for IVM features
    - (in future) Provide advanced features

Overview of pg_ivm

- You can create
  Incrementally Maintainable Materialized View (IMMV)

- IMMV is automatically updated when a underlying table is modified
  (= immediate maintenance)
  - AFTER triggers are created on tables, and the view is updated from within them.
  - You don't have to write the trigger functions by yourself.
  - A view that would take 20 seconds with the normal REFRESH could be updated in 15 milliseconds

- Compatible with PostgreSQL 13, 14, 15, 16, and 17.

Proposed feature for PostgreSQL vs. pg_ivm

- The current proposed features for PostgreSQL
  - IMMV is implemented by extending the materialized view feature
    - Use "`CREATE INCREMENTAL MATERIALIZED VIEW`" to create it.
  - (inner) joins、built-in aggregates（count, sum, avg, min, max）, and DISTINCT

- pg_ivm
  - IMMV is implemented as a table
    - Use create_immv() function to create it.
  - Support more complex views
    - A simple subquery in the FROM clause
    - CTE (WITH clause)
    - EXISTS clause

Supported View Definitions and Restriction of pg_ivm

- Supported
  - Inner joins (including self joins)
  - DISTINCT clause
  - Some built-in aggregate functions (count, sum, avg, min, and max)
  - Simple sub-queries in FROM clause, simple CTEs (WITH query)
  - EXISTS sub-queries
- Not Supported
  - Other aggregates, windows functions
  - Outer joins
  - Subqueries containing aggregaet or DISTINCT
  - HAVING, ORDER BY, LIMIT/OFFSET, UNION/INTERSECT/EXCEPT, DISTINCT ON, TABLESAMPLE, VALUES, FOR UPDATE/SHARE
  - IMMVs including other view, materialized view, paritioned table, or foreign table

How to use pg_ivm

Installation (1)

- From source
  - Get the souce code from GitHub（https://github.com/sraoss/pg_ivm）
  - PostgreSQL source code, or devel package（postgresql17-devel, postgresql-server-dev-17, etc.）is required.

```
$ make install
```

- RPM package
  - Provided in PostgreSQL Yum Repository（https://yum.postgresql.org/）

```
$ sudo yum install pg_ivm_17
```

Installation (2)

- `CREATE EXTENSION`

```
CREATE EXTENSION pg_ivm;
```

postgresql.conf

```
shared_preload_libraries = 'pg_ivm'
```

  - the following objects are created.

    - Table
      - pg_ivm_immv: The catalog table that stores IMMV information

    - Functions
      - create_immv : Create an IMMV
      - refresh_immv: Refresh an IMMV manually
      - get_immv_def: Get the view definition query of an IMMV

Create IMMV

- Call `create_immv` function with a relation name and a view definition query
  - Correspond to executing `CREATE MATERIALIZED VIEW` command
  - Create a unique index on the IMMV automatically if possible

```
test=# SELECT create_immv('mv(aid, bid, abalance, bbalance)',
         'SELECT a.aid, b.bid, a.abalance, b.bbalance
          FROM pgbench_accounts a JOIN pgbench_branches b USING(bid)');

NOTICE:  created index "mv_index" on immv "mv"
 create_mv
-------------
    10000000
(1 row)
```

Information on IMMV

- An entry has been added to the `pg_ivm_immv` catalog .

```
test=# SELECT * FROM pg_ivm_immv WHERE immvrelid = to_regclass('mv');
 immvrelid |                      viewdef                       | ispopulated
-----------+----------------------------------------------------+-------------
 mv        | {QUERY :commandType 1 :querySource 0 :canS.| t
           |.etTag true :utilityStmt <> :resultRelation.|
           |. 0 :hasAggs false :hasWindowFuncs false :h.|
           |.asTargetSRFs false :hasSubLinks false :has.|
                              (snip)
           |.nList <> :mergeUseOuterJoin false :stmt_lo.|
           |.cation 0 :stmt_len 0}                      |
(1 row)
```

IMMV View Definition

- The view definition can be checked using the `get_immv_def` function.
  - Correspond to psql's `\d` meta-command.

```
test=# SELECT get_immv_def('mv');
                  get_immv_def
-----------------------------------------
  SELECT a.aid,                         +
     b.bid,                             +
     a.abalance,                        +
     b.bbalance                         +
    FROM (pgbench_accounts a            +
      JOIN pgbench_branches b USING (bid))
(1 row)
```

Automatic Update of IMMV

- When a base table is updated, the IMMV is automatically updated.
  - Takes 15.448 ms
  - `REFRESH` of a materialized view with the same definition takes more than 20 seconds.

```
test=# UPDATE pgbench_accounts SET abalance = 1234 WHERE aid = 1;
UPDATE 1
Time: 15.448 ms

test=# SELECT * FROM mv WHERE aid = 1;
 aid | bid | abalance | bbalance
-----+-----+----------+----------
   1 |   1 |     1234 |        0
(1 row)
```

Manual Refresh of IMMV

- Call `refresh_immv` function with a relation name
  - Correspond to executing `REFRESH MATERIALIZED VIEW` command
  - The second argument specifies if new data is generated (corresponding to `WITH [NO] DATA` option)
    - true: the view definition query is executed to provide the new data, and automatic update is enabled.
    - false: the IMMV is truncated, and automatic update is disabled.

```
test=# SELECT refresh_immv('mv', true);
 refresh_immv
--------------
     10000000
(1 row)
```

An IMMV is actually a table

- psql's `\d` meta-command shows it as "Table".

```
test=# \d mv
                    Table "public.mv"
  Column   |  Type   | Collation | Nullable | Default
-----------+---------+-----------+----------
+----------
 aid       | integer |           |          |
 bid       | integer |           |          |
 abalance  | integer |           |          |
 bbalance  | integer |           |          |
Indexes:
    "mv_index" UNIQUE, btree (aid, bid)
```

IMMV cannot be directly upated

- Same as normal materialized views of PostgreSQL
  - This causes an error.

```
test=# DELETE FROM mv WHERE aid = 1;
ERROR:  cannot change materialized view "mv"
```

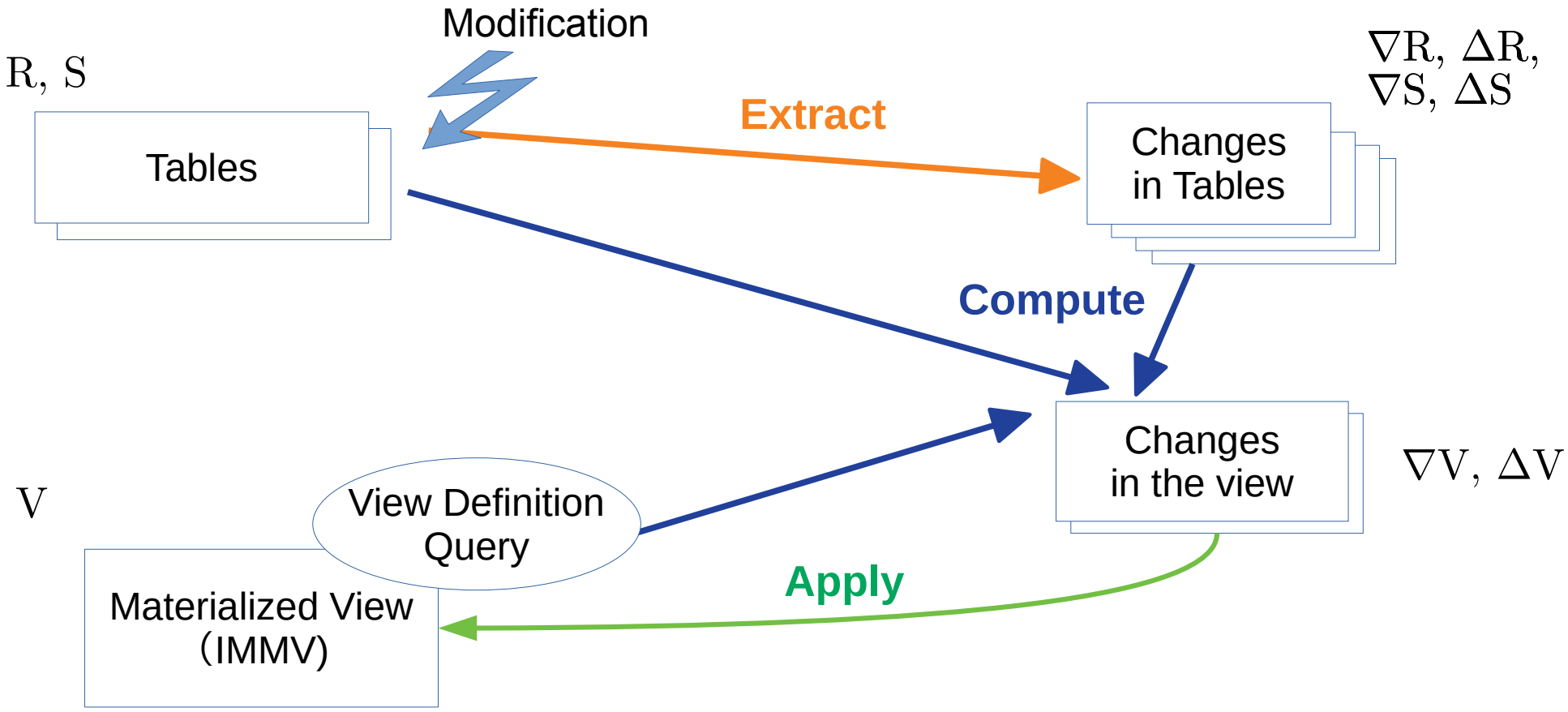
Drop IMMV

- Use `DROP TABLE` command
  - Information in the pg_ivm_immv catalog is automatically deleted.

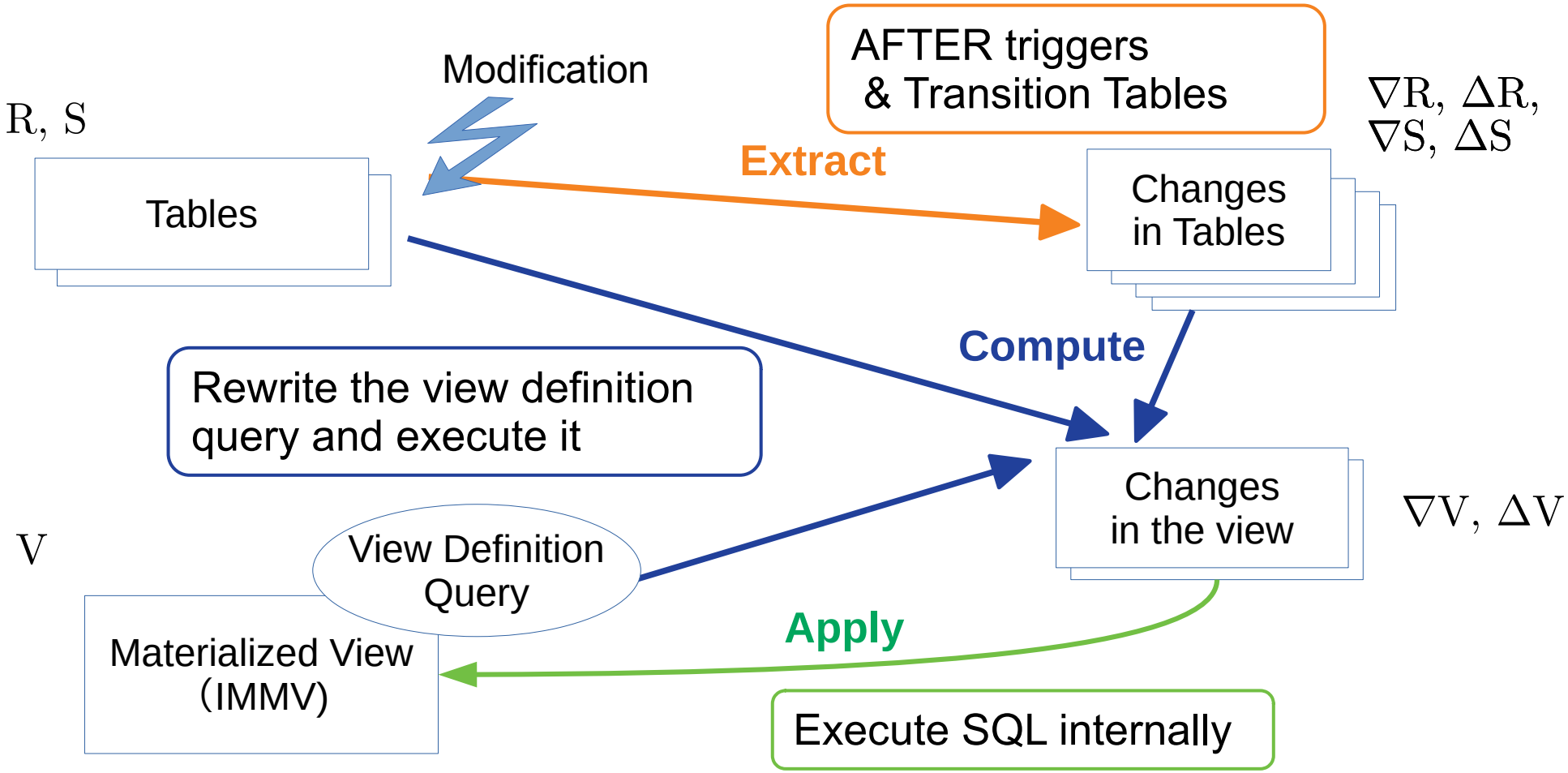
```
test=# DROP TABLE mv;
DROP TABLE

test=# SELECT * FROM pg_ivm_immv WHERE immvrelid = to_regclass('mv');
 immvrelid | viewdef | ispopulated
-----------+---------+-------------
(0 rows)
```

How pg_ivm works

Overview of View Maintenance



Overview of View Maintenance



Extracting Changes in Table

- When an IMMV is created, AFTER trigger are created on all base tables.
  - Fired after execution of INSERT, DELETE, UPDATE, and TRUNCATE
  - Changes in the table is extracted, and changes in the view is computed within it.

- Changes in table is extracted as *Transition Tables*
  - Two tables that can be referred to within AFTER triggers, and each contains:
    - Rows deleted from the table
    - Rows inserted into the table

Compute Changes in View

- Rewrite the view definition query and execute it.
  - Modified table is replaced with the transition table

modified table

View Definition Query:

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

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”.

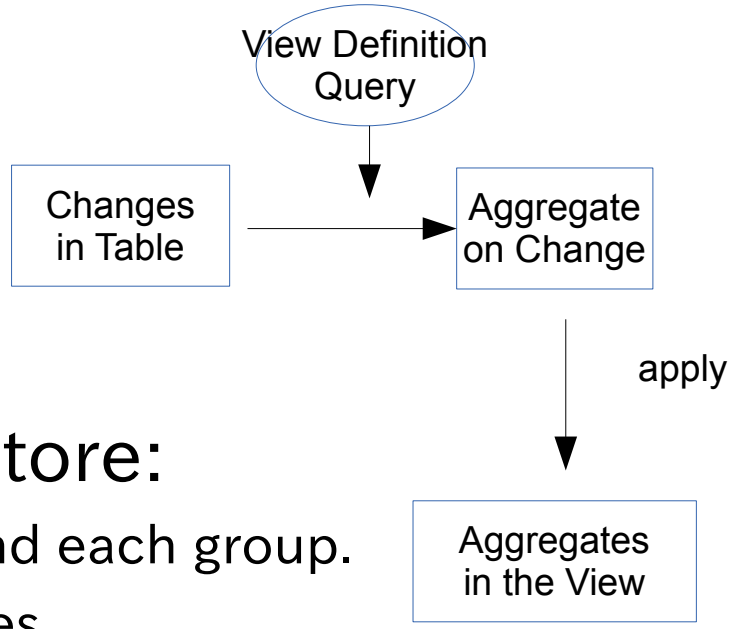
Apply Changes to View

- Execute SQL internally
  - Use DELETE to delete rows from the IMMV.
  - Use INSERT to insert rows into the IMMV.
  - Use UPDATE to update aggregate values in the IMMV.

- Index on an IMMV
  - An appropriate index is necessary on the IMMV to scan target rows of DELETE and UPDATE efficiently.
  - The index is automatically created if possible.
    - If the view contains primary key columns of tables or GROUP BY clause.

Maintenance of View with Aggregates

- Update aggregate values in the view by using aggregate results on table changes.
  - Examples:
    - count(x) ← count(x) ± [count(x) on table changes]
    - sum(x) ← sum(x) ± [sum(x) on table changes]
    - avg(x) ← (sum(x) ± [sum(x) on table changes]) / (count(x) ± [count(x) on table changes])

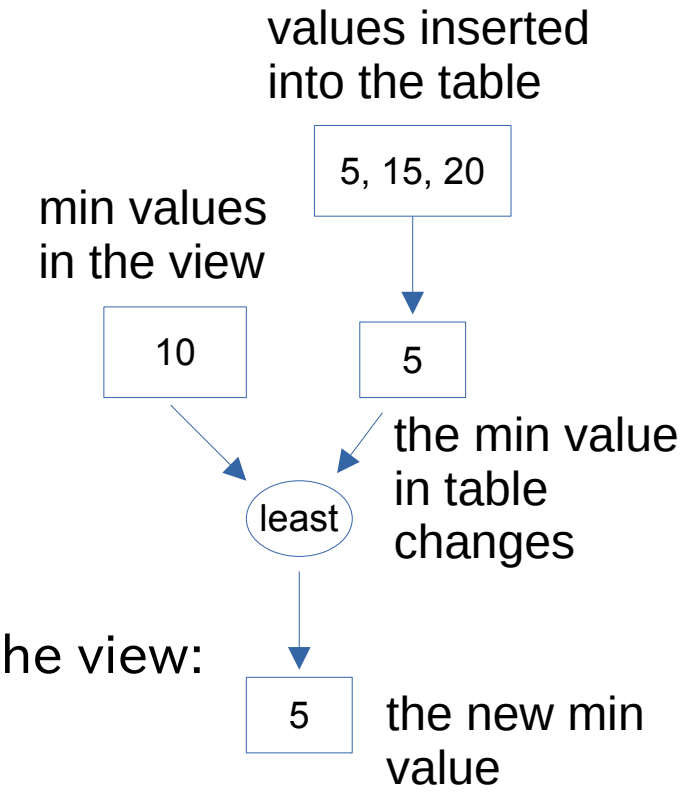


- Some columns are automatically added to store:
  - The number of rows (__ivm_count__) in the whole view and each group.
  - For avg aggregate, the results of sum and count aggregates.

Aggregate: min(x) & max(x)

- Tuples are inserted into a table
  - min(x) ← least (min(x), [min(x) on inserted rows])
  - max(x) ← greatest (max(x), [max(x) on inserted rows])

- Tuples are deleted from a table:
  - When existing min(x) or max(x) value need to be deleted from the view:
    → Re-compute the new value from the latest table.
  - Otherwise: nothing to do



Maintenance of View with DISTINCT

- A column (_ivm_count_) that stores the number of rows for each distinct row is automatically added.
  - The count value is updated when a table is modified.
  - When the count becomes zero, the row is removed from the view.

TRUNCATE

- When a base table is TRUNCATEed, the IMMV is also TRUNCATEed.
  - If any base table is empty, the contens of the view also must be empty.
  - Exception: if the view contains an aggregates without GROUP BY;
    - The view always has only one row.
    - After a base table is truncated, the view has a row whose column values are NULL.

Simultaneous Multiple Tables Modification

- Possible situations
  - Updating CTEs
  - Triggers
  - Foreign key constraint

```
WITH x AS (INSERT INTO r VALUES(1,10) RETURNING 1),
     y AS (INSERT INTO s VALUES(1,100) RETURNING 1)
SELECT 1;
```

- Self-join shares the same situation
  - Tables appearing in a query repeatedly ≒ Different tables with the same contents

- In this case, both *pre-update* and *post-update* states of tables are needed to compute incremental changes in the view.
  - Example: JOIN view
    - View definitnion $V \stackrel{\text{def}}{=} R \bowtie S$
    - Tables modifications $R \leftarrow R \cup \Delta R,\ S \leftarrow S \cup \Delta S$
    - Incremental change $\Delta V = (\Delta R \bowtie S\_old) \cup (R\_new \bowtie \Delta S)$

How to get “pre-update” state of table?

- Only “post-update” state is available in AFTER trigger functions
- “pre-update” state can be obtained by using a saved “snapshot”
  - Save the current snapshot before the table modification (in BEFORE trigger)
  - When scanning the table, call `ivm_visible_in_prestate` function in WHERE clause

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

  - This function returns true if a row is visible with the given snapshot.

    → The results with deleted rows append is “pre-update” state

Performance

Performance Evaluation Using TPC-H Query (1)

※ scale factor = 1
(lineitem: 6M rows)

- Q01: Aggregate on a large table
  - When we use a normal materialized view:
    - Update one row in lineitem : 2 min sec.
    - REFRESH: 11 sec.

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 2.077 ms

tpch=# REFRESH MATERIALIZED VIEW mv01;
REFRESH MATERIALIZED VIEW
Time: 11277.638 ms (00:11.278)
```

```
CREATE MATERIALIZED VIEW mv01 AS
select
     l_returnflag,
     l_linestatus,
     sum(l_quantity) as sum_qty,
     sum(l_extendedprice) as sum_base_price,
     sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
     sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as
sum_charge,
     avg(l_quantity) as avg_qty,
     avg(l_extendedprice) as avg_price,
     avg(l_discount) as avg_disc,
     count(*) as count_order
from
     lineitem
where
     l_shipdate <= date '1998-12-01' - interval '78' day
group by
     l_returnflag,
     l_linestatus;
```

Panasonic Let's note CF-SV7
CPU: Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz (8 core)
DRAM: 16GB
Storage: SSD
OS: Ubuntu 22.04.4 LTS (64bit), linux kernel 5.15.0-124-generic
PostgreSQL 17 + pg_ivm 1.9

Performance Evaluation Using TPC-H Query (2)

※ scale factor = 1
(lineitem: 6M rows)

- Q01: Aggregate on a large table
  - When we use a normal materialized view:
    - Update one row in lineitem: **2 min sec.**
    - REFRESH: **11 sec.**
  - pg_ivm:
    - Update one row in lineitem
      & automatic update of IMMV: **20 min sec.**

View update: 500x+ faster

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 19.635 ms
```

```
SELECT create_immv('immv01',
'select
     l_returnflag,
     l_linestatus,
     sum(l_quantity) as sum_qty,
     sum(l_extendedprice) as sum_base_price,
     sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
     sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as
sum_charge,
     avg(l_quantity) as avg_qty,
     avg(l_extendedprice) as avg_price,
     avg(l_discount) as avg_disc,
     count(*) as count_order
from
     lineitem
where
     l_shipdate <= date ''1998-12-01'' - interval ''78'' day
group by
     l_returnflag,
     l_linestatus');
```

Performance Evaluation Using TPC-H Query (3)

※ scale factor = 1
(lineitem: 6M rows)

- Q09: Aggregate on six tables join
  - When we use a normal materialized view:
    - Update one row in lineitem: 2 **min sec.**
    - REFRESH: 5 **sec.**
  - pg_ivm:
    - Update one row in lineitem
      & automatic update of IMMV: 32 **min sec.**

View update: 150x+ faster

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 32.474 ms
```

```
SELECT create_immv('immv09',
'select
     nation,
     o_year,
     sum(amount) as sum_profit
from
     (
          select
               n_name as nation,
               extract(year from o_orderdate) as o_year,
               l_extendedprice * (1 - l_discount) -
                    ps_supplycost * l_quantity as amount
          from
               part, supplier, lineitem, partsupp, orders, nation
          where
               s_suppkey = l_suppkey
               and ps_suppkey = l_suppkey
               and ps_partkey = l_partkey
               and p_partkey = l_partkey
               and o_orderkey = l_orderkey
               and s_nationkey = n_nationkey
               and p_name like ''%sandy%''
     ) as profit
group by
     nation,
     o_year');
```

Performance Trade-Off

- IMMV can be updated more rapidly than REFRESH, but it affects table update performance.
  - Suitable use case:
    - The response time of table updates is not so critical, and the latest view state is required immediately after the table update.
  - When large data is loaded to a table, disable automatic update of IMMV.

```
SELECT refresh_immv('mv', false);
```

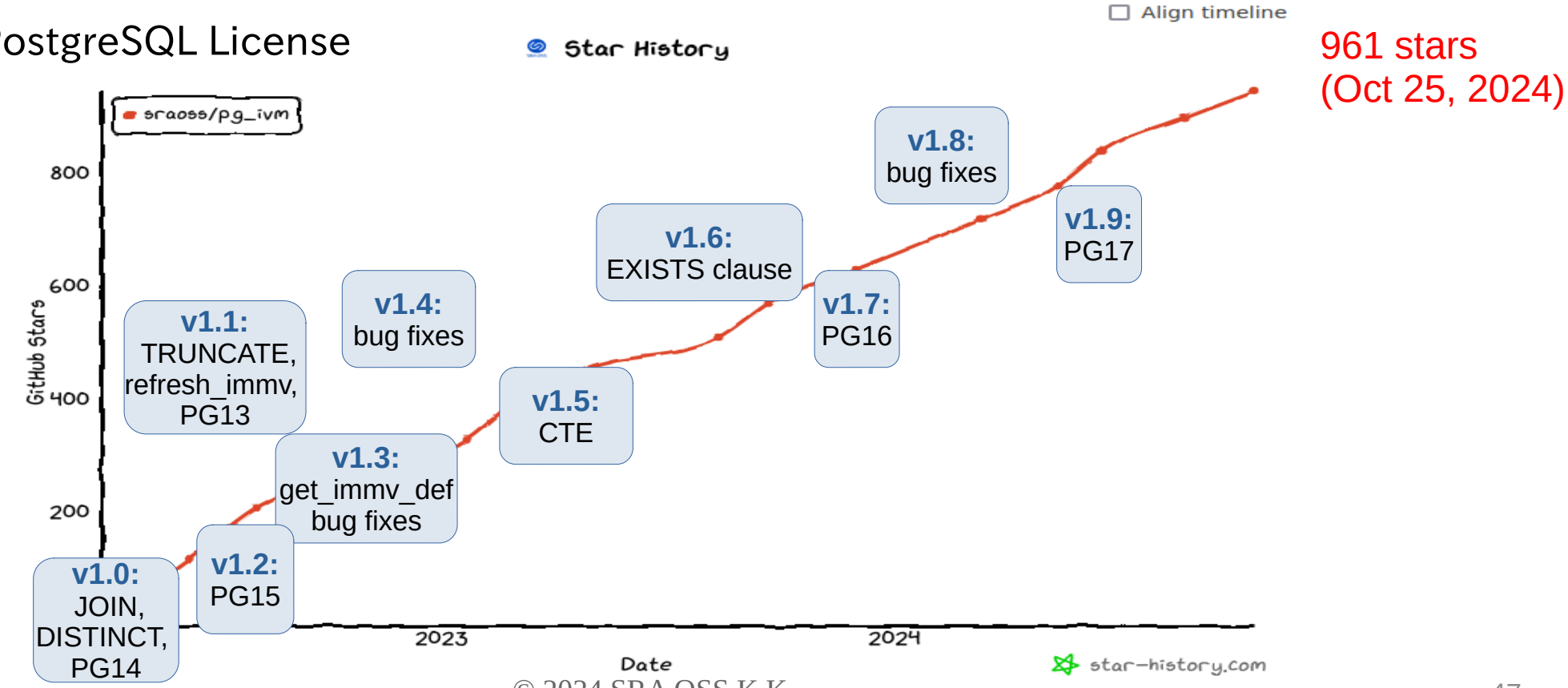
- Performance with Concurrent Transactions
  - When concurrent transactions update an IMMV, an exclusive lock is acquired to avoid inconsistent results.
    - READ COMMITTED isolation level: one transaction is waited.
    - REPEATABLE READ isolation level: one transaction is aborted.

Development of pg_ivm

Development of pg_ivm

- GitHub：https://github.com/sraoss/pg_ivm
- Developer： IVM Development Group
- License： PostgreSQL License

Issues & Pull Requests are always welcome!



961 stars
(Oct 25, 2024)

Future Plans (1)

- Outer Joins Support
  - Outer join has null-extended tuples (= dangling tuples) when the join condition does NOT meet.
  - View maintenance needs additional dangling tuples handling
    - Insert into a table → dangling tuples might be deleted
    - Delete from a table → dangling tuples might be inserted

```
SELECT *
    FROM weather LEFT OUTER JOIN cities ON weather.city = cities.name;

     city      | temp_lo | temp_hi | prcp |    date    |     name      | location
---------------+---------+---------+------+------------+---------------+----------
 Hayward       |      37 |      54 |      | 1994-11-29 |               |
 San Francisco |      46 |      50 | 0.25 | 1994-11-27 | San Francisco | (-194,53)
 San Francisco |      43 |      57 |    0 | 1994-11-29 | San Francisco | (-194,53)
(3 rows)
```

dangling tuple

Future Plans (2)

- Partitioned Table Support
  - Started investigating what could be a problem
- Other requests from GitHub issues
  - Logical Replication Support
    - Allow an IMMV in the subscriber be automatically updated, when a table in the publisher is modified.
  - Deferred maintenance
    - Instead of automatically update at table changes, allow an IMMV be updated incrementally later by a manual command etc., with low impact on table update performance.
  - Aggregate with GROUPING SET, CUBE, ROLLUP

Summary

- pg_ivm: an extension module that provides Incremental View Maintenance feature for PostgreSQL.
  - Incrementally Maintainable Materialized View (IMMV)
  - When a base table is modified, the IMMV is automatically and incrementally updated.
  - Materialized views can be updated more rapidly than REFRESH
    - As a trade-off, table update performance is affected
  - Development is ongoing on GitHub
    - Issues & Pull Requests are welcome

Thank you!

Feedback