SRA OSS, INC.

The Way for Updating Materialized Views Rapidly

Yugo Nagata, Takuma Hoshiai @ SRA OSS, Inc. Japan.

PGCon2020
- May, 2020

About Us

- Yugo NAGATA
  - Software Engineer at SRA OSS, Inc. Japan
  - Research and Development on PostgreSQL
    - Incremental View Maintenance (IVM) ← Today's topic

- Takuma HOSHIAI
  - Software Engineer at SRA OSS, Inc. Japan
  - a member of IVM project
  - demo

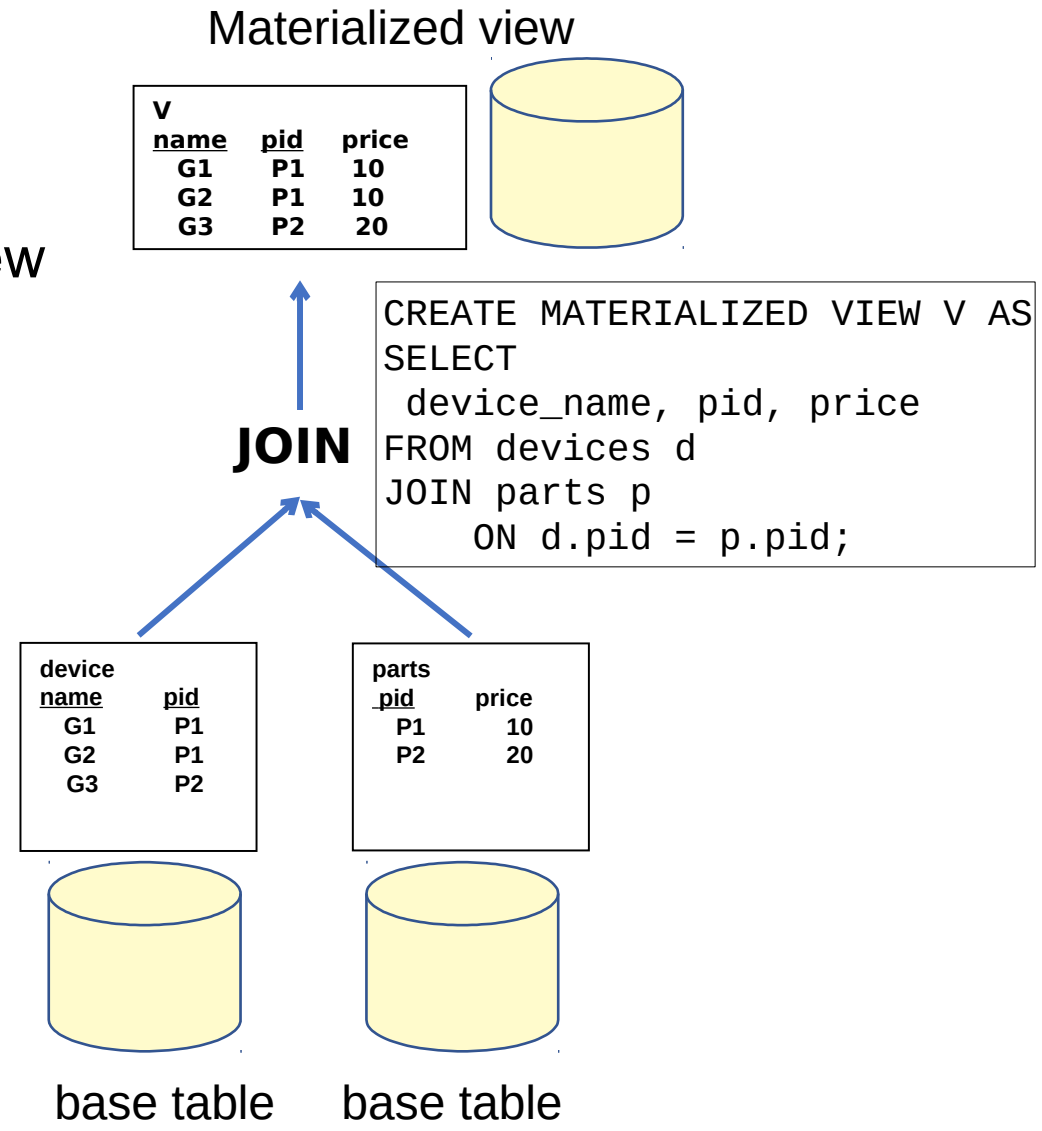
Outline

- Incremental View Maintenance (IVM)
  - The way to refresh materialized views rapidly
- IVM Implementation on PostgreSQL
  - Overview
  - Progress since the initial patch
- Examples
  - Demonstration
  - Performance Evaluation
- Summary

Materialized View

- View
  - Virtual relation defined by a query
  - The query is executed when the view is referred to.
- Materialized view
  - The results are stored in database for quick response.
  - Data warehouse for analyzing a large data

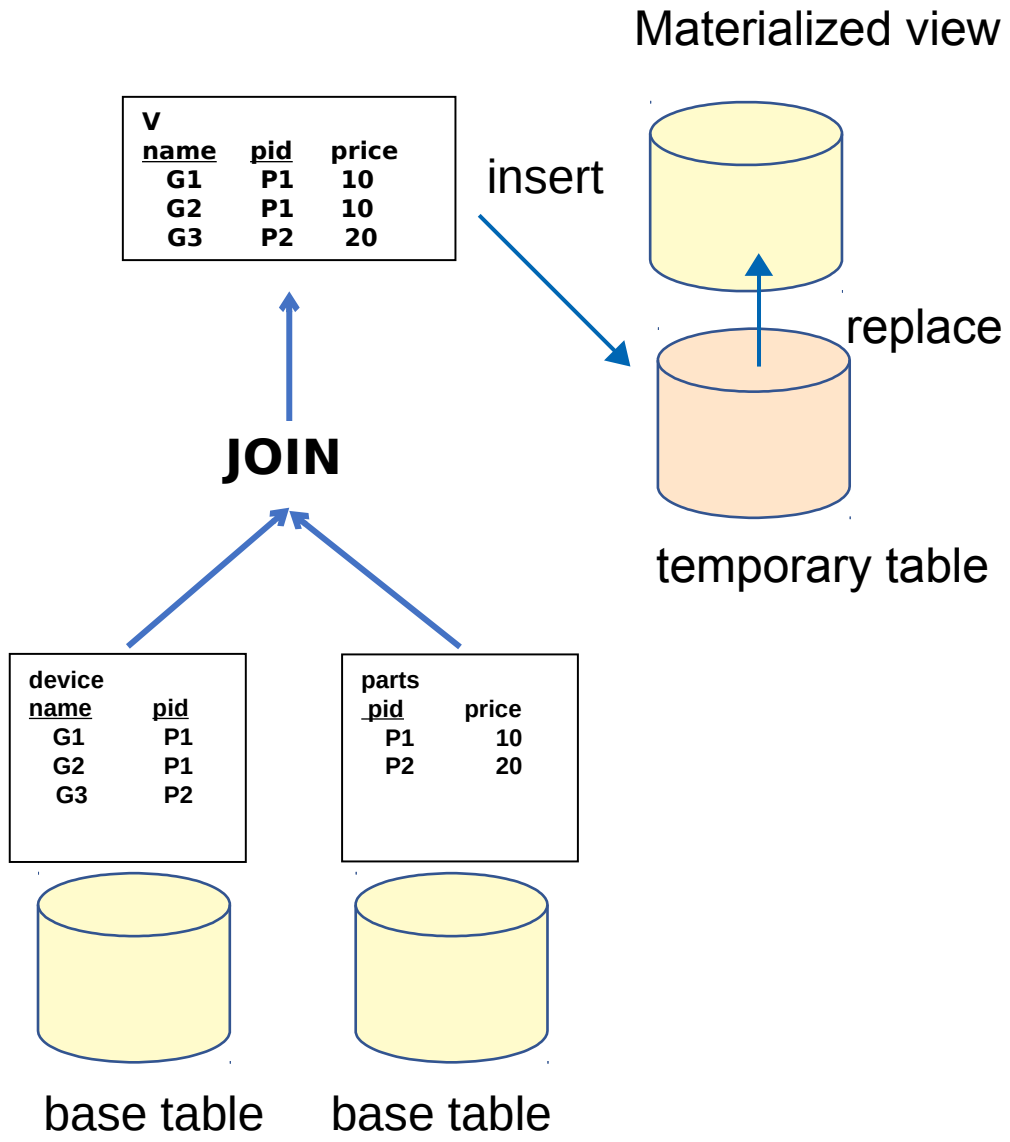
Needs to be maintained after a base table is modified.



Refreshing Materialized Views

```
REFRESH MATERIALIZED VIEW V;
```

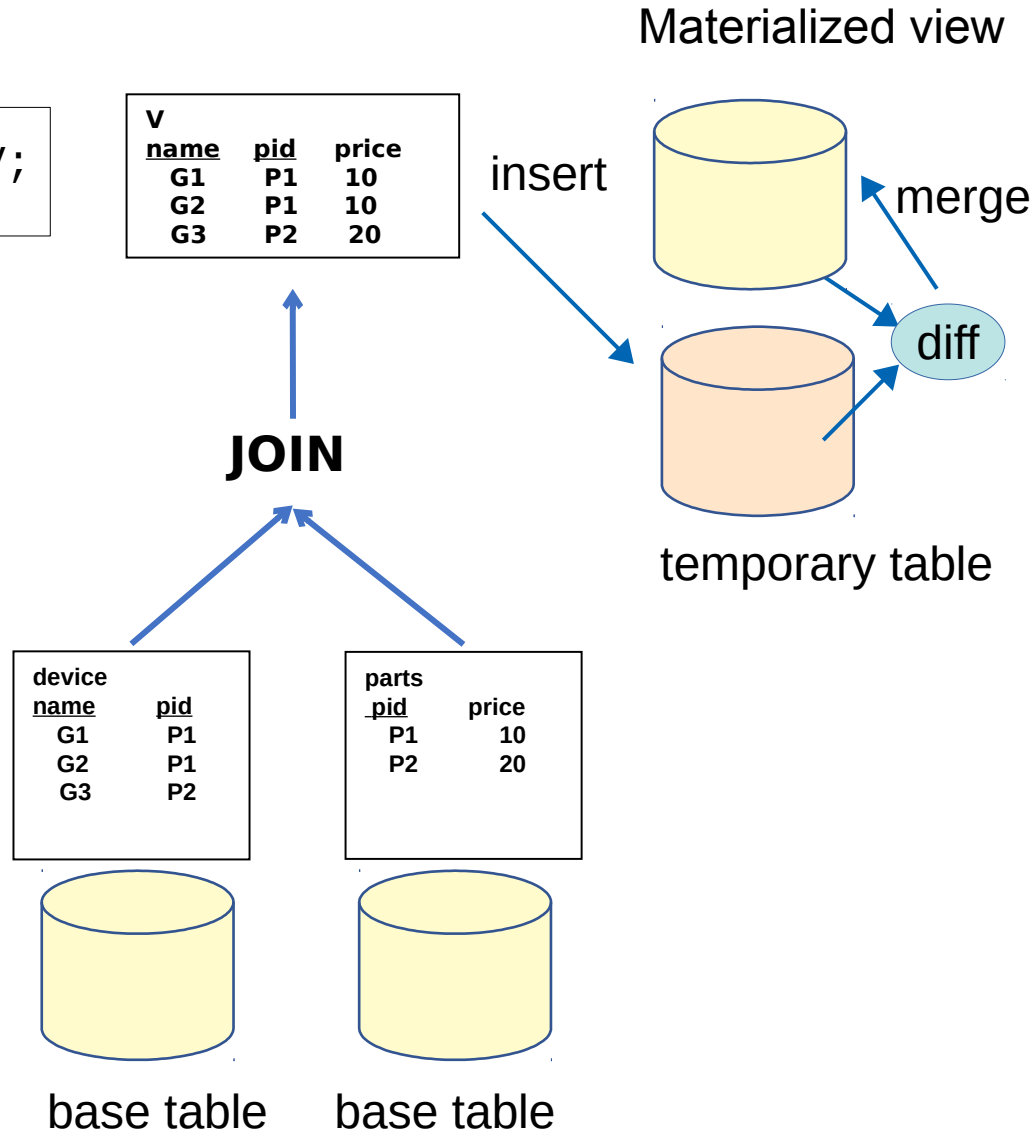
- REFRESH command
  - Recomputing the contents from scratch



Refreshing Materialized Views

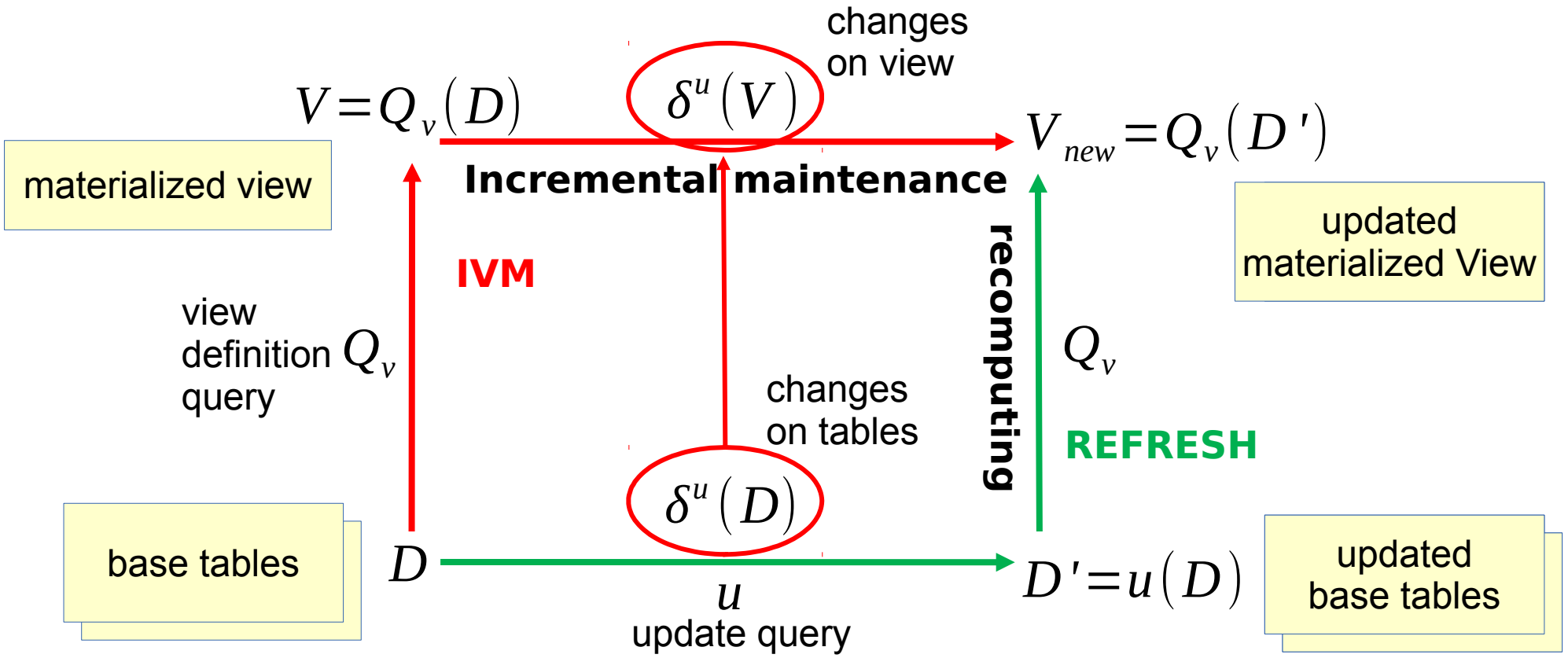
```
REFRESH MATERIALIZED VIEW CONCURRENTLY V;
```

- CONCURRENTLY option
  - Refresh materialized view with a weaker lock
  - Still needs recomputing



Incremental View Maintenance (IVM)

- Compute and apply only the incremental changes to the materialized views → effective maintenance



Our Implementation

- The first patch was submitted a year ago
  - Subject: Implementing Incremental View Maintenance
  - Talk at PGCon 2019

- Materialized views can be updated automatically and incrementally when base tables are updated.

  → You don't need to write trigger functions by yourself!

How Effective?

- TPC-H Q01
  - Aggregates on a large table

Execution time (scale factor = 1)

SELECT on the query	11424.255 ms
SELECT on the materialized view	3.128 ms ← Quick response
REFRESH of the materialized view	24135.419 ms
Incremental View Maintenance (1 row of the base table is updated)	22.315 ms ← Rapid update

Supported Views

- The initial patch
  - Selection, Projection, (Inner) Join,
  - DISTINCT
  - Views with tuple duplicates
- The latest patch
  - Some aggregates with/without GROUP BY
    - count, sum, avg, min, max
  - Self-join
  - Outer Join
  - Sub-queries including EXISTS
  - REFRESH WITH [NO] DATA,
  - Row Level Security (RLS), pg_dump/restore

Basic Theory of IVM

- View definition

```
SELECT * FROM R NATURAL JOIN S;
```

  - Ex.) Natural join view

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

- Change on a base table

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

R, S	base tables
∇R	deleted tuples
ΔR	inserted tuples

- Calculation of change on view

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

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

- Apply the change to the view

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

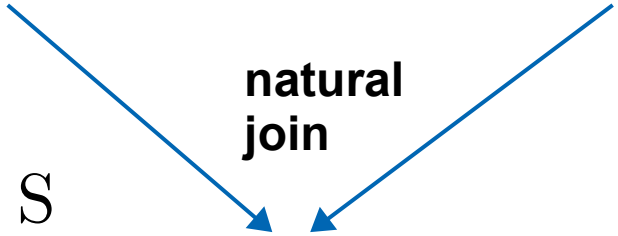
Basic Theory of IVM: Example (1)

R

id	text
111	one
222	two
333	three

S

id	value
111	1
222	2
333	3



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

id	text	value
111	one	1
222	two	2
333	three	3

Basic Theory of IVM: Example (2)

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

S

id	text
111	one → first
222	two
333	three

id	value
111	1
222	2
333	3

∇R

id	text
111	one

ΔR

id	text
111	first

natural join

natural join

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

id	text	value
111	one	1

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

id	text	value
111	first	1

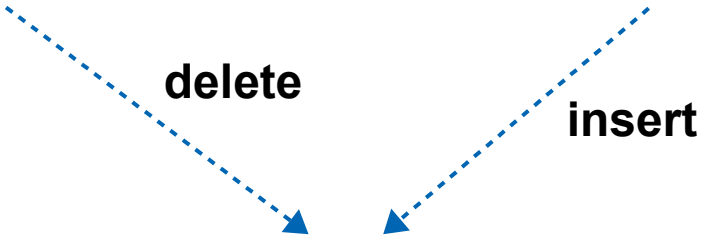
Basic Theory of IVM: Example (3)

∇V

id	text	value
111	one	1

ΔV

id	text	value
111	first	1



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

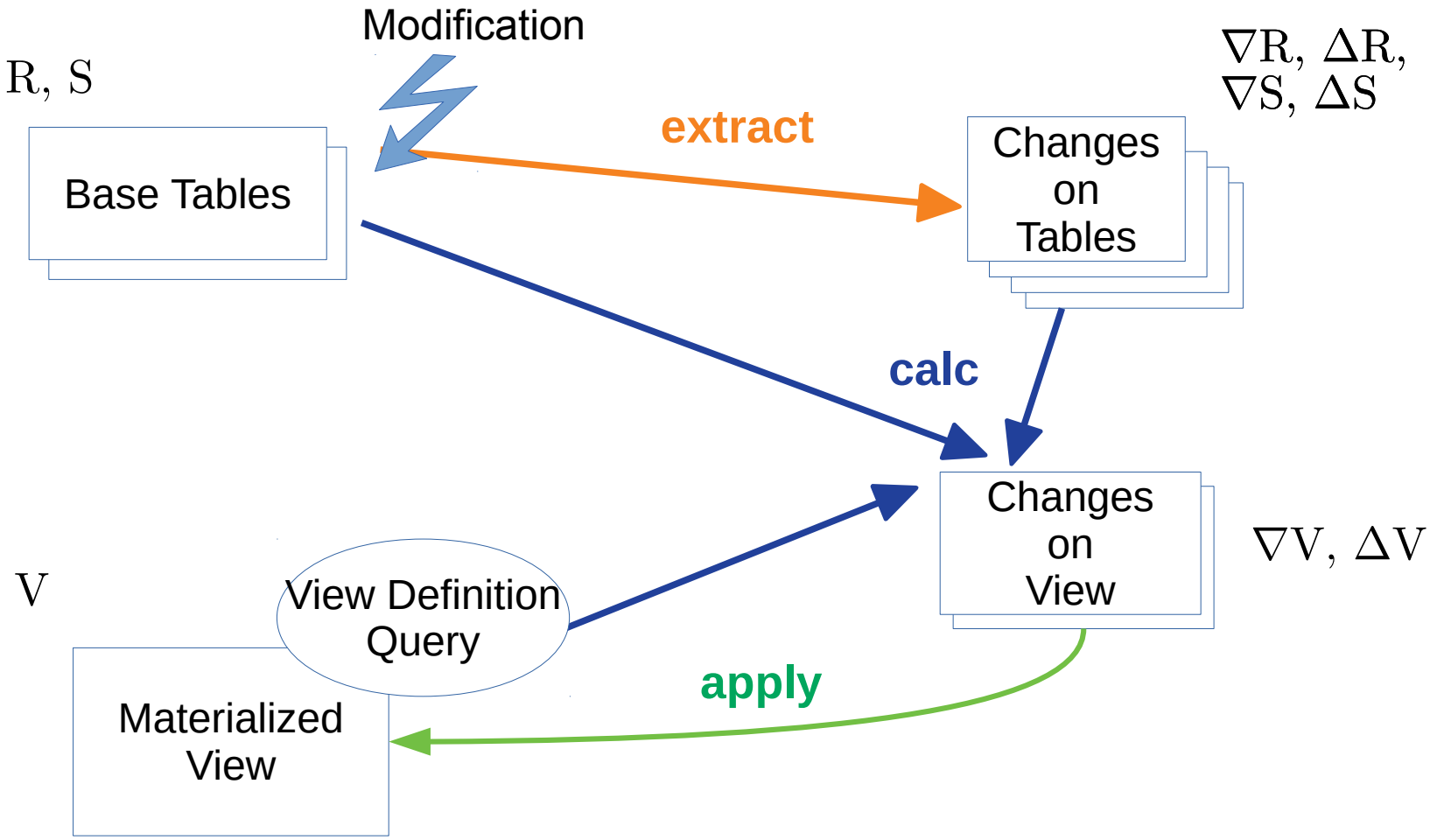
id	text	value
111	one → first	1
222	two	2
333	three	3

Timing of View Maintenance

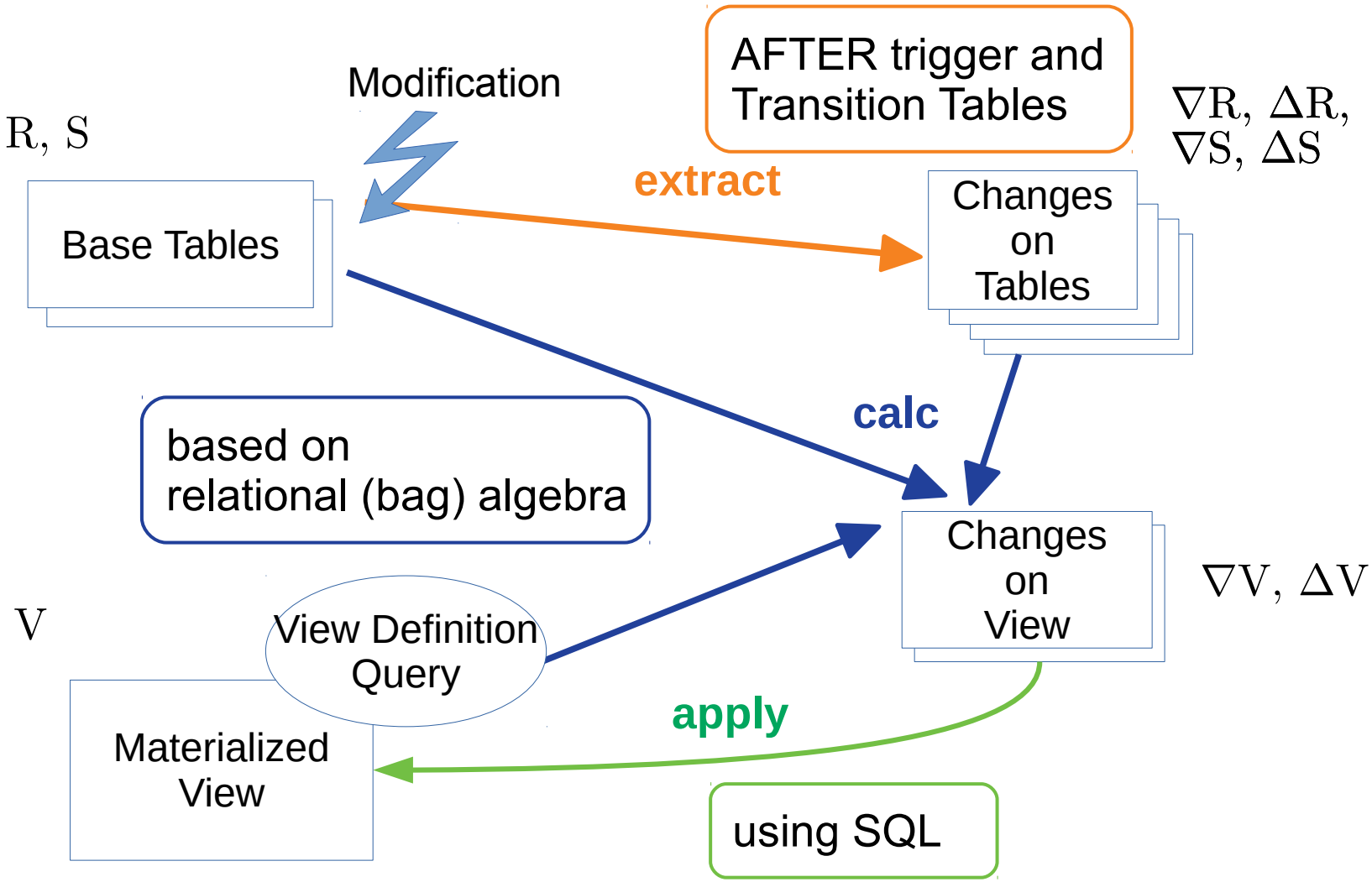
- Immediate maintenance
  - Materialized view is updated in the same transaction where a base table is modified.
- Deferred maintenance
  - Materialized view is updated after the transaction is committed
    - When view is accessed
    - As a response to user command (like REFRESH)
    - Periodically, etc.

- We started from “Immediate” since it requires less number of codes.
  - “Deferred” needs a mechanism to manage “logs” for recording changes of base tables.

Overview



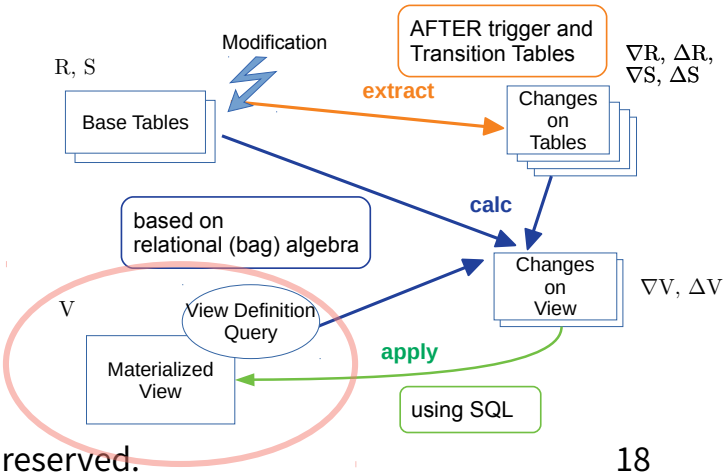
Overview



Creating Materialized Views (1)

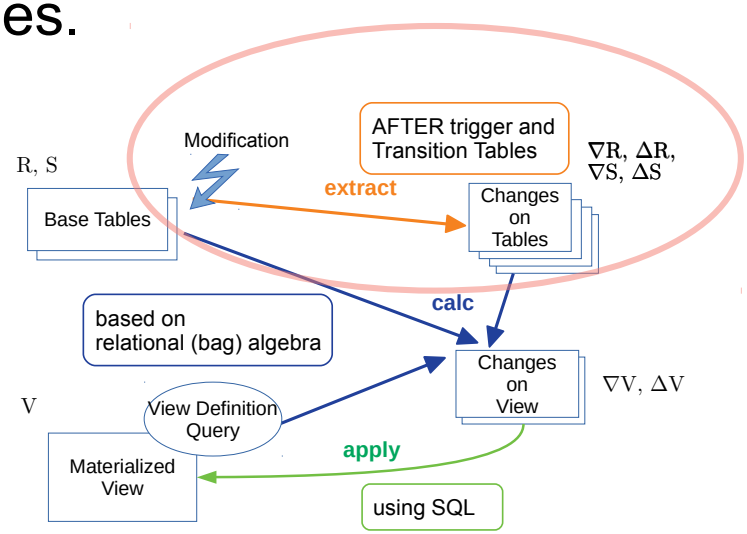
- CREATE INCREMENTAL MATERIALIZED VIEW
  - The tentative syntax to creates materialized views with IVM support

```
CREATE INCREMENTAL MATERIALIZED VIEW mv AS
    SELECT device_name, pid, price
    FROM devices d
    JOIN parts p
        ON d.pid = p.pid;
```



Creating Materialized Views (2)

- AFTER triggers are created on all base tables.
  - Automatically and internally
  - For INSERT, DELETE, and UPDATE
  - Statement level
  - With Transition Tables



- Transition Tables
  - Changes on tables can be referred to in the trigger function like normal tables.
    - tuples deleted from the table
    - tuples inserted into the table
  - In theory, these two tables are corresponding to ∇R and ΔR respectively.

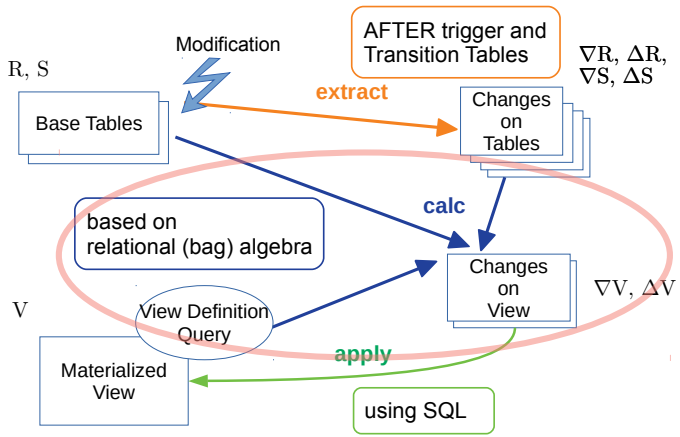
Calculating Delta on Views

- Use the view definition query with some rewrite:
  - Replacing the modified table with the transition table.
  - Using count(*) in order to count the multiplicity of tuples.

View definition:

```
SELECT device_name, pid, price
FROM devices d
JOIN parts p
    ON d.pid = p.pid;
```

modified table



rewrite

Query to calculate view delta:

```
SELECT count(*) AS __ivm_count__, device_name, pid, price
FROM table_delta d
JOIN parts p
    ON d.pid = p.pid
GROUP BY device_name, pid, price;
```

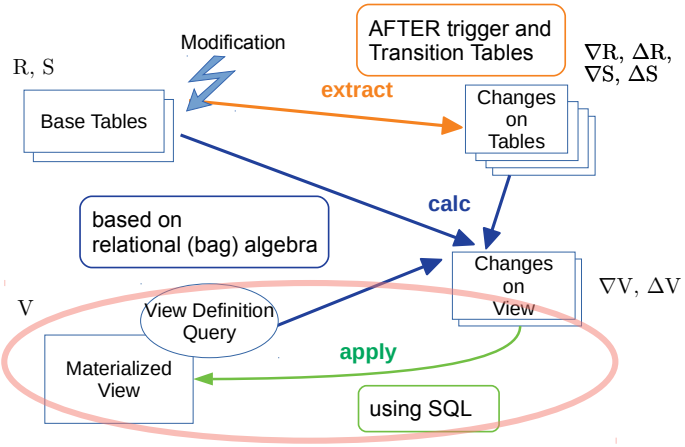
multiplicity

transition table

In theory, the results correspond to ∇V and ΔV.

Applying Delta to View (1)

- Deleting tuples from the view
  - Tuples are identified by joining the delta table.
  - Tuples are deleted as many as the specified multiplicity
    - by numbered using row_number() function.




```
DELETE FROM mv WHERE ctid IN (
  SELECT tid FROM (
     SELECT row_number()
                  OVER (PARTITION BY c1, c2, ...)
                  AS __ivm_row_number__,
            mv.ctid AS tid,
            diff.__ivm_count__
     FROM mv, old_delta AS diff
     WHERE mv.c1 = diff.c1 AND mv.c2 = diff.c2 AND ... ) v
WHERE v.__ivm_row_number__ <= v.__ivm_count__;
```

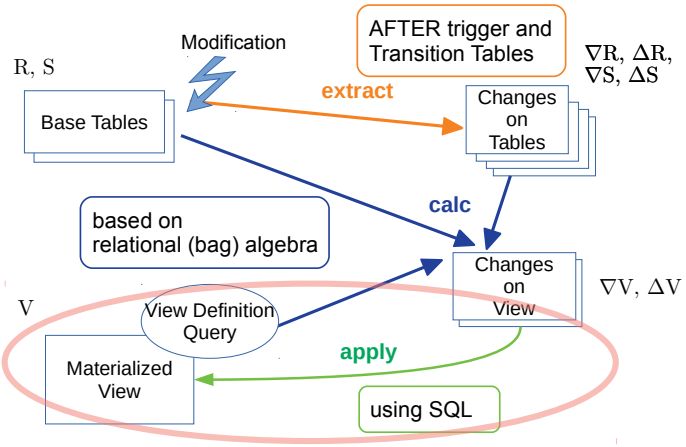
numbering tuples

view delta table

multiplicity

Applying Delta to View (2)

- Inserting tuples into the view
  - Tuples are duplicated to the specified multiplicity
    - using generate_series() function.




```
INSERT INTO mv SELECT c1, c2, ... FROM (
   SELECT diff.*, generate_series(1, diff.__ivm_count__)
     FROM new_delta AS diff) AS v;
```

multiplicity

view delta table

Progress since the Initial Patch

- Supporting Queries:
  - Aggregates
  - Self-Joins
  - Outer Joins
  - Sub-queries including EXISTS clause
- Others:
  - Row Level Security (RLS)
  - REFRESH WITH [NO] DATA
  - pg_dump/restore
  - etc.

Aggregates Support

- count, sum, min, max, avg (with or without GROUP BY)
- Expressions specified in GROUP BY must appear in the target list.
- Views have one or more hidden columns. For example:
  - count(*) result value for multiplicity of tuples
  - Additional count(x) and sum(x) values for avg(x) aggregates.
- Aggregates are performed on table deltas, and aggregated values in the view are updated using the results
  - The way of updating depends on the kind of aggregate function.

Updating Aggregated Values

- count(x) ← count(x) ± [count(x) from delta table]
- sum(x) ← sum(x) ± [sum(x) from delta table]
- avg(x)
  ← (sum(x) ± [sum(x) from delta]) / (count(x) ± [count(x) from delta])
- min(x), max(x)
  - When tuples are inserted:
    - min(x) ← least (min(x), [min(x) from delta table])
    - max(x) ← greatest (max(x), [max(x) from delta table])
  - When tuples are deleted:
    - If the old min(x) or max(x) is deleted from the view, it needs recomputing the new value from base tables.

Performance Evaluation: Aggregate View (1)

- Materialized views of aggregates on pgbench_accounts

Scale factor of pgbench: 1000

```
CREATE MATERIALIZED VIEW mv_normal2 AS
        SELECT bid, count(abalance), sum(abalance), avg(abalance)
        FROM pgbench_accounts GROUP BY bid;
```

```
CREATE INCREMENTAL MATERIALIZED VIEW mv_ivm2 AS
        SELECT bid, count(abalance), sum(abalance), avg(abalance)
        FROM pgbench_accounts GROUP BY bid;
```

Performance Evaluation: Aggregate View (2)

```
test=# REFRESH MATERIALIZED VIEW mv_normal2 ;
REFRESH MATERIALIZED VIEW
Time: 30494.729 ms (00:30.495)
```

REFRESH took **30 seconds**.

```
test=# SELECT * FROM mv_ivm2 WHERE bid = 1;
 bid | count  |  sum  |           avg
-----+--------+-------+-------------------------
   1 | 100000 | -1855 | -0.018550000000000000
(1 row)
```

```
test=# UPDATE pgbench_accounts SET abalance = abalance + 1000 WHERE aid = 1;
UPDATE 1
Time: 30.215 ms
```

Updating pgbench_accounts took **30 ms**.

x 1000 faster!

```
test=# SELECT * FROM mv_ivm2 WHERE bid = 1;
 bid | count  | sum  |           avg
-----+--------+------+-------------------------
   1 | 100000 | -855 | -0.008550000000000000
(1 row)
```

Updated automatically and correctly!

Multiple Tables Modification and Self-Join (1)

- Possible cases:
  - Modifying CTEs
  - Triggers
  - Foreign key constrains

```
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 is essentially the same situation
  - Same tables in a query ≒ Different tables with the same contents

- View maintenance with simultaneous multiple table modifications.
  - Pre-update state of tables is required as well as post-update state.

Multiple Tables Modification and Self-Join (2)

- Pre-update table state
  - Available by applying table deltas inversely
  - Inserted tuples can be removed by filtering with xmin, cmin

```
SELECT t.* FROM mytbl t
  WHERE (age(t.xmin) - age(pre_xid::text::xid) > 0) OR
        (t.xmin = pre_xid AND t.cmin::text::int < pre_cid)
 UNION ALL
SELECT * FROM old_delta
```

removing inserted tuples

appending deleted tuples

- Simultaneous modifications of multiple tables
  - AFTER triggers are fired more than once.
    - Each trigger extracts and stores changes of each table.
    - View is updated incrementally in the final AFTER trigger call.

Outer Join Support

- Outer Joins
  - Null-extended tuples (= dangling tuples) appear when the join condition does NOT meet.
- Outer join view maintenance:
  - Insert into a table → dangling tuples might be deleted
  - Delete from a table → dangling tuples might be inserted

- View maintenance steps:
  1. Calculate and apply deltas as similar to inner join case
  2. Additional dangling tuples handling
- Implemented based on an algorithm in Larson & Zhou (2007)
  - With theory extension to allow tuple duplicates

Performance Evaluation: Inner and Outer Join View (1)

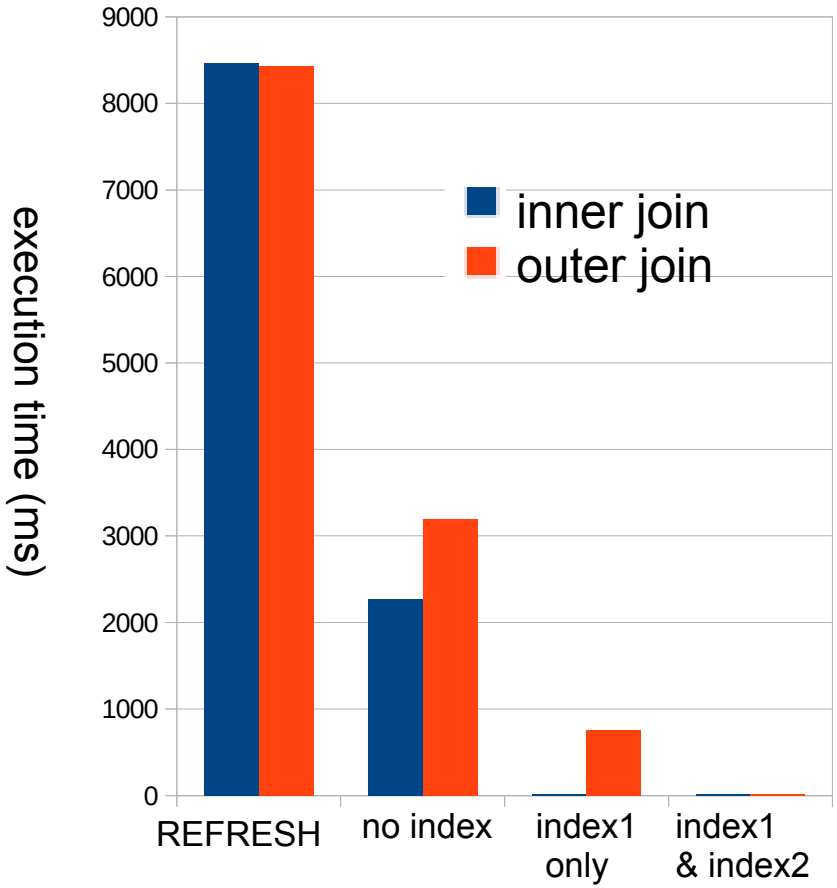
- Materialized views joining pgbench tables: Scale factor of pgbench: 100
  - Inner join & full outer join

```
SELECT a.aid, a.bid AS abid, b.bid AS bbid,
       a.abalance, b.bbalance
  FROM pgbench_accounts a
    [FULL OUTER] JOIN pgbench_branches b
    ON a.bid = b.bid;
```

- Conditions of indexes on materialized view:
  - No index
  - Index1:
    - An index on the primary key columns of base tables: (aid, abid)
  - Index2:
    - Two Indexes on columns used in join conditions : (abid) & (bbid)

Performance Evaluation: Inner and Outer Join View (2)

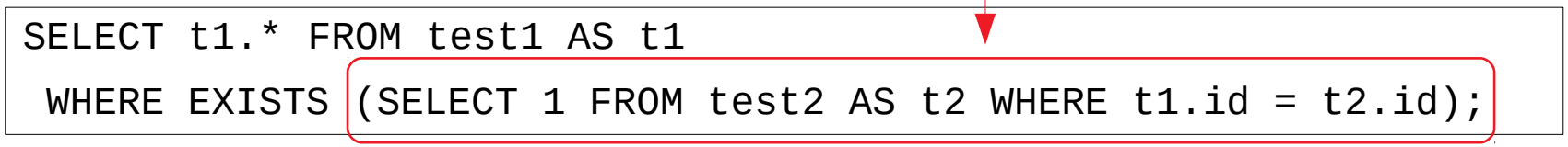
- Execution time of updating a tuple in pgbench_accounts
  - Compared re-computation of the view (REFRESH)



  - IVM is faster than REFRESH
  - For effective maintenance, an appropriate index is required.
    - To search tuples to be deleted
  - Comparable performance between inner and outer join views
    - For outer join views, indexes on join condition columns are required.
    - To search dangling tuples to be deleted or inserted

Sub-queries Support

- EXISTS in WHERE
  - The row number count of the EXISTS sub-query is stored in the view as a hidden column.



```
SELECT t1.* FROM test1 AS t1
 WHERE EXISTS (SELECT 1 FROM test2 AS t2 WHERE t1.id = t2.id);
```

  - When a table in the sub-query is modified:
    - If the count becomes 0, the tuple in the view should be deleted.
    - Otherwise, the tuple remains.

- Simple sub-queries in FROM clause are also supported.
  - Only selection, Projection, or Inner join is allowed in a sub-query.

Other Progress (1)

- WITH [NO] DATA
  - An option for CREATE [INCREMENTAL] MATERIALIZED VIEW or REFRESH
  - If WITH NO DATA is specified, the triggers are dropped from base tables.
    - The view is not automatically updated even when a base table is modified.
    - Also, the materialized view becomes not scannable.
- Row Level Security (RLS)
  - When a view is updated incrementally, base tables are accessed with the view owner's privilege.
- pg_dump/restore
  - Support for "CREATE INCREMENTAL MATERIALIZED VIEW"

Other Progress (2)

- Not use temporary tables
  - Before: Temporary tables were used to store view deltas.
    → System catalog bloat, prepared transaction unavailable, etc.
  - After: Tuplestores (ephemeral named relations) are used.

- Performance improvement of view access
  - Before: generate_series function is used to output duplicated tuples.
    → Huge overhead, row estimate mistake of planner, etc.
  - After: Changed to not use generate_series at SELECT on views

demo

About Us

- Yugo NAGATA
  - Software Engineer at SRA OSS, Inc. Japan
  - Research and Development on PostgreSQL
    - Incremental View Maintenance (IVM) ← Today's topic

- Takuma HOSHIAI
  - Software Engineer at SRA OSS, Inc. Japan
  - a member of IVM project
  - demo

Current Restrictions (1)

- Supported:
  - Selection,
  - Projection
  - Inner and Outer join
  - DISTINCT
  - Aggregates
    - count, sum, avg, min, max
    - with/without GROUP BY
  - Self-join
  - EXISTS
  - Simple sub-queries
- Not supported:
  - Other aggregates
  - HAVING
  - Complex sub-queries
    - including aggregates, outer-join, etc.
  - CTE
  - window functions
  - Set operations
    - UNION, EXCEPT, INTERSECT
  - ORDER BY
  - LIMIT/OFFSET

Current Restrictions (2)

- Major restrictions of outer joins
  - Support only simple equijoin.
  - Can not be used together with aggregates or sub-queries.
- Major restrictions of EXISTS
  - “OR EXISTS” or “NOT EXISTS” is not supported.
  - Can not be used together with aggregates.

- TPC benchmark queries:
  - TPC-H : 9 / 22 queries are supported.
  - TPC-DS: 20 / 99 queries are supported.

(ORDER BY and LIMIT/OFFSET are ignored.)

- 33 queries fail due to CTEs
- 11 queries fail due to aggregates in a sub-query

Summary

- Incremental View Maintenance on PostgreSQL
  - Rapid and automatic update of materialized views
  - Progress:
    - Aggregates, Outer joins, Self-joins, Sub-queries, ...
- Future works:
  - Relaxing restrictions
    - CTE, Sub-queries including aggregates, and so on.
  - Deferred Maintenance using table change logs
  - Performance improvement and optimizations

- The patch is proposed and discussed in pgsql-hackers
  - Subject: Implementing Incremental View Maintenance
  - Github: https://github.com/sraoss/pgsql-ivm/

Thank you