

pg_ivm: Recent Improvements and Future Directions in Incremental View Maintenance

Yugo Nagata

Author of pg_ivm | PostgreSQL Contributor

SRA OSS K.K.

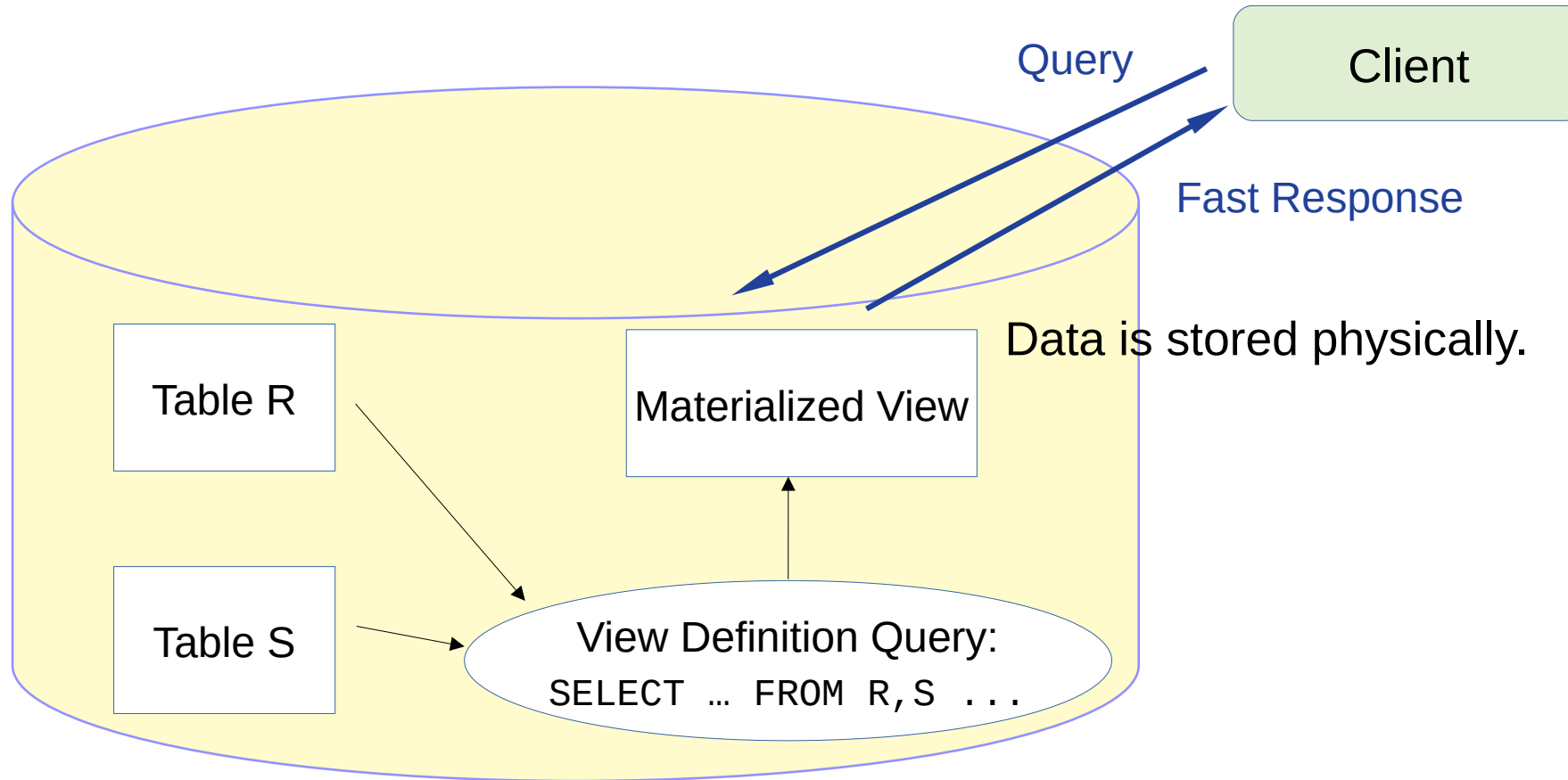
PGConf.EU 2026, October 20

Valencia, Spain

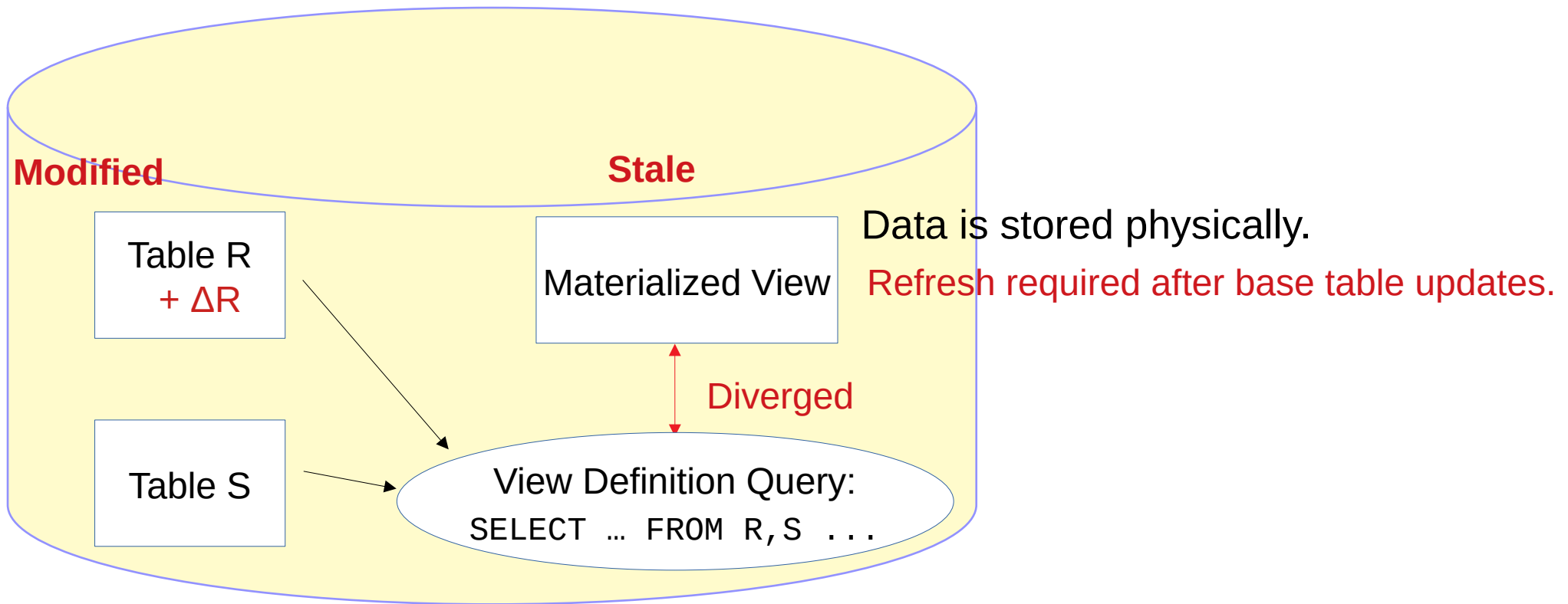
Outline

- Background: Incremental View Maintenance
- How pg_ivm Works
- Performance and Trade-offs
- Recent Improvements
- Challenges and Future Directions

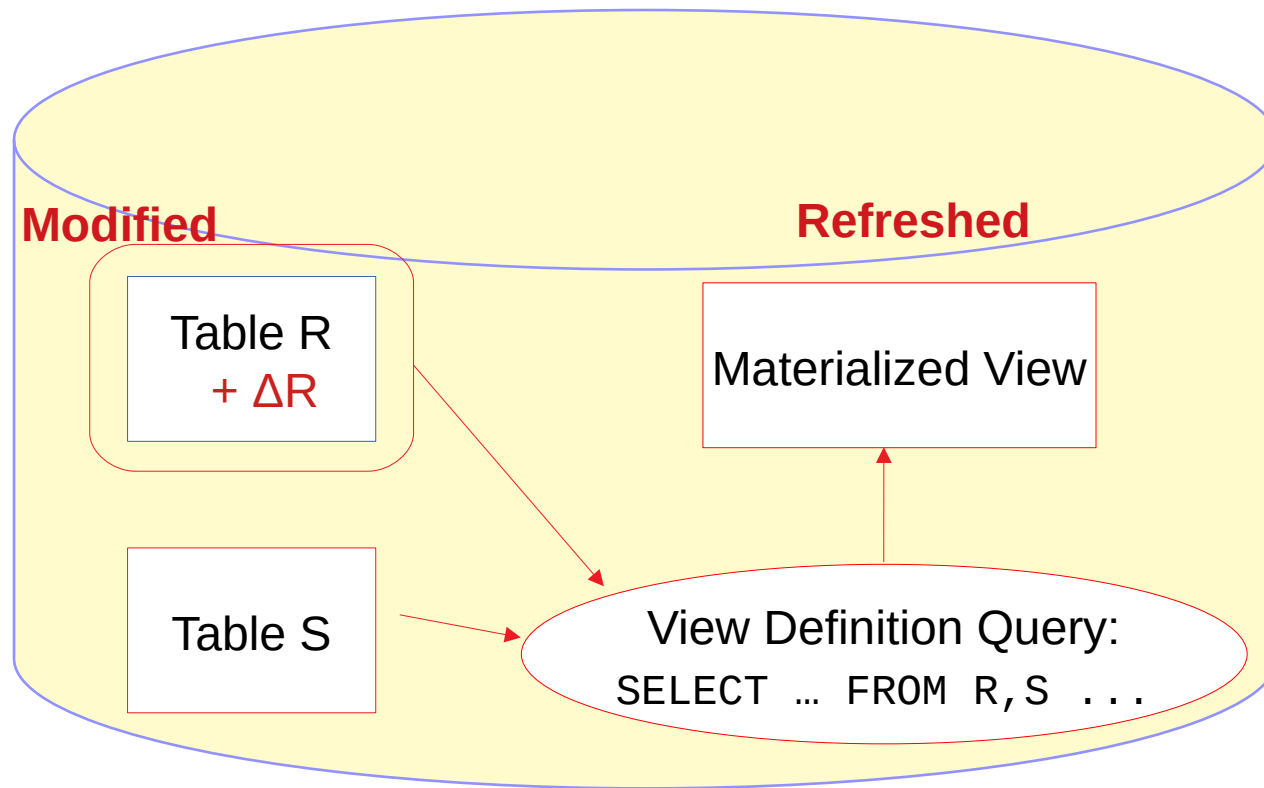
Materialized Views and Refresh Cost (1)



Materialized Views and Refresh Cost (2)



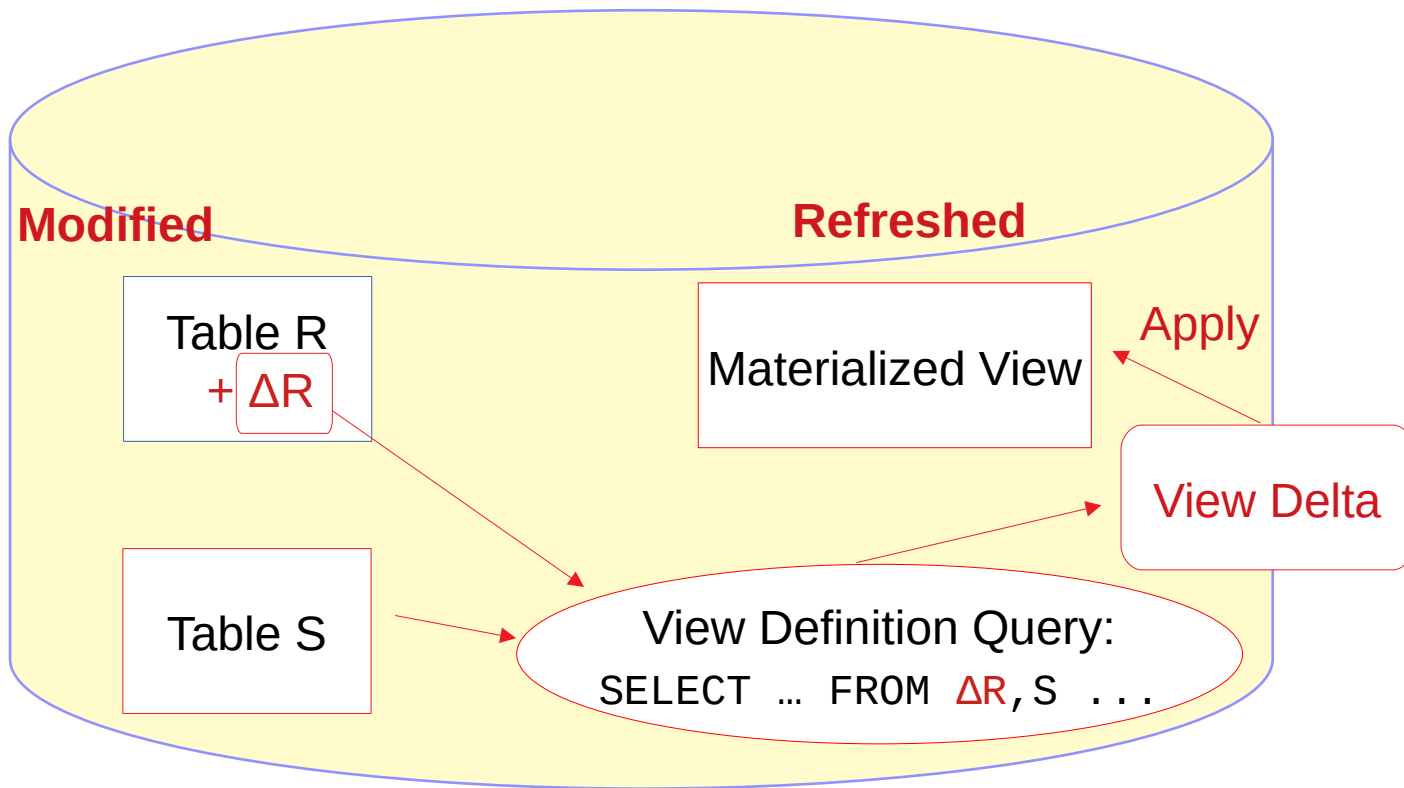
Materialized Views and Refresh Cost (3)



REFRESH MATERIALIZED VIEW

Re-compute the entire view from scratch.

Incremental View Maintenance



REFRESH MATERIALIZED VIEW

Re-compute the entire view from scratch.

Incremental View Maintenance

Compute and apply only the incremental changes.

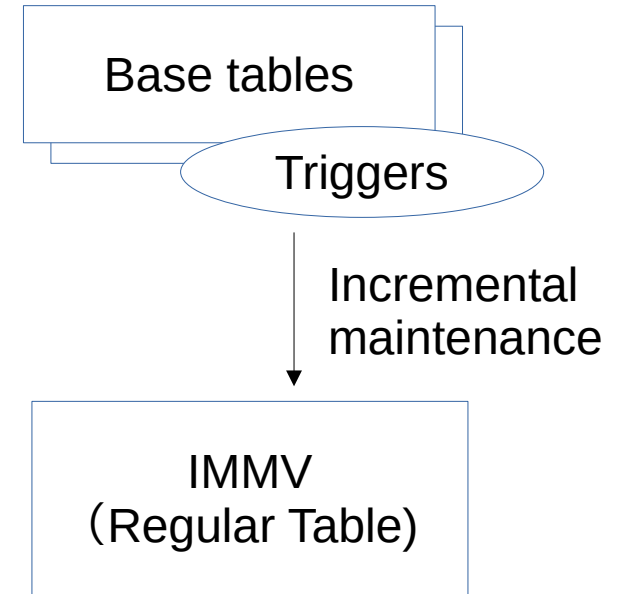
pg_ivm: Overview

- PostgreSQL extension for Incremental View Maintenance (IVM)
 - Open-source software developed and maintained by the IVM Development Group
 - GitHub : https://github.com/sraoss/pg_ivm
 - First Release: Apr 28, 2022
 - Latest Release: Sep 30, 2026 (v1.16)
 - 1.5k GitHub Stars
 - PostgreSQL License



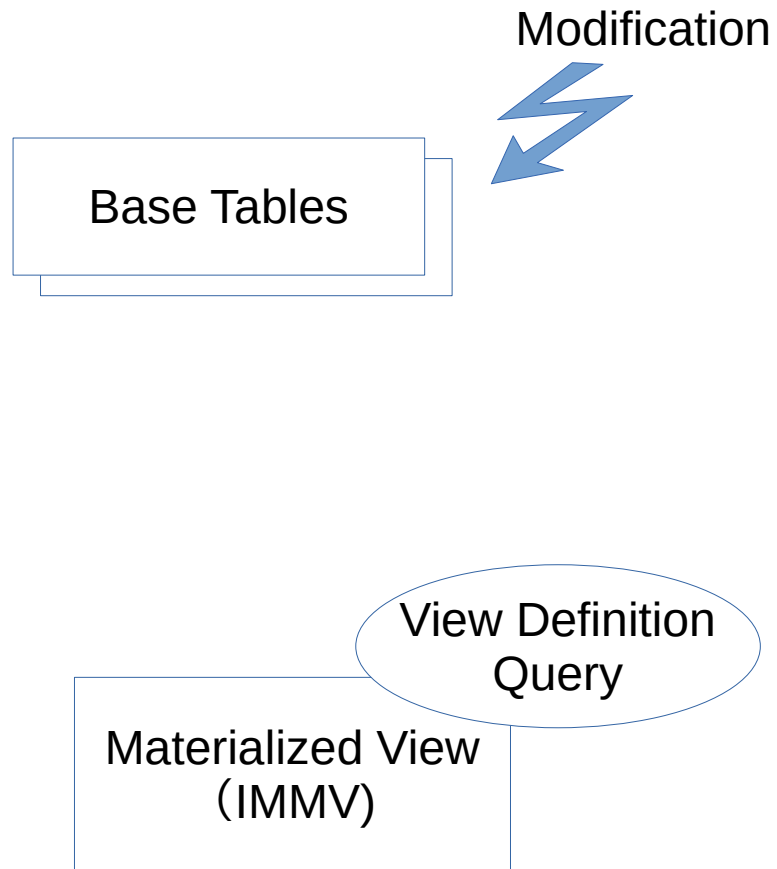
pg_ivm: Features

- IMMV : Incrementally Maintained Materialized View
- Created by `create_immv()`
 - Conceptually similar to `CREATE MATERIALIZED VIEW`
- Implemented as a regular table
- Automatically maintained when a base table is modified

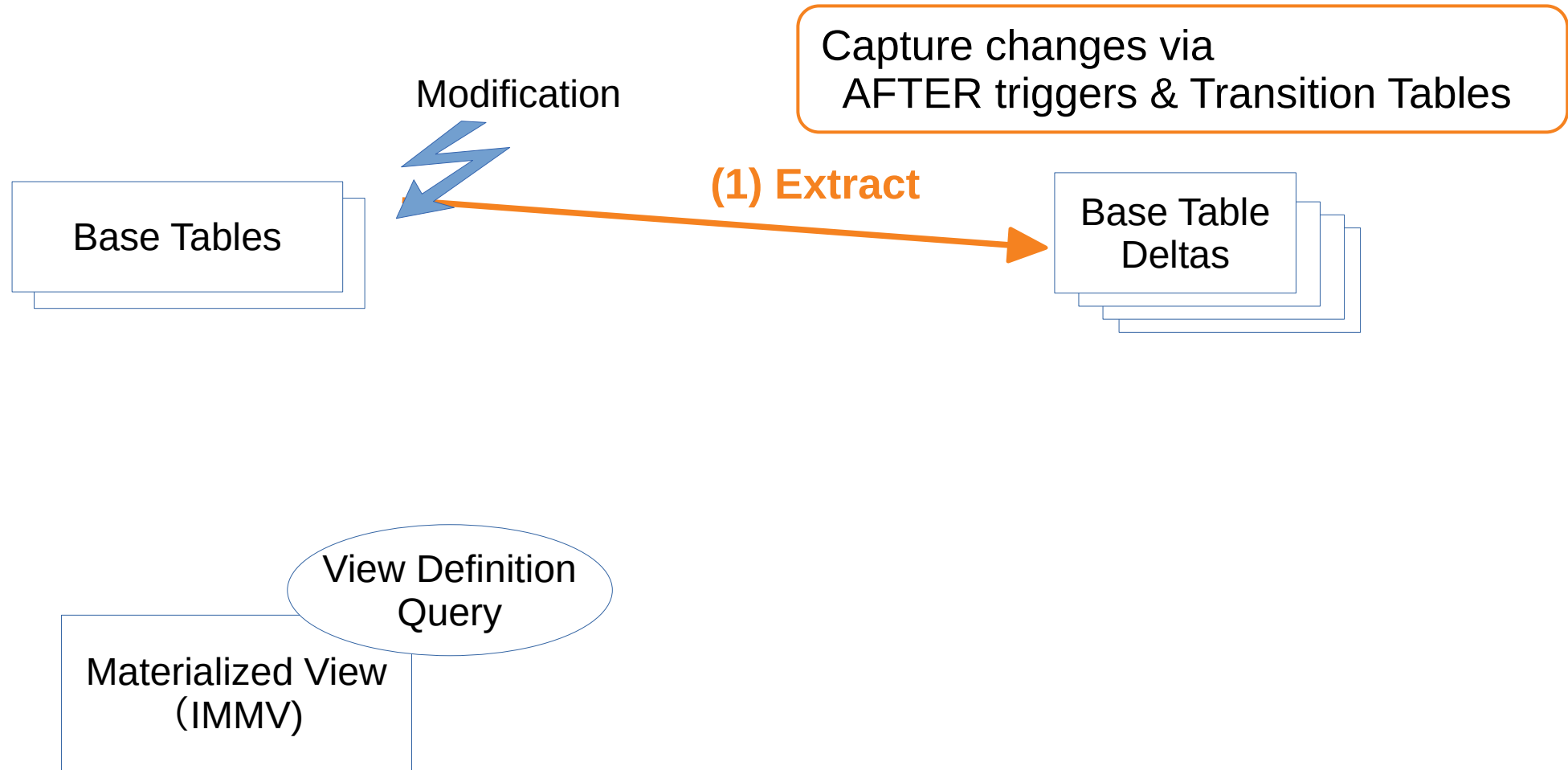


```
SELECT immv.create_immv(  
    'mv(aid, bid, abalance, bbalance)',  
    'SELECT a.aid, b.bid, a.abalance, b.bbalance  
    FROM pgbench_accounts a JOIN pgbench_branches b USING(bid)');
```

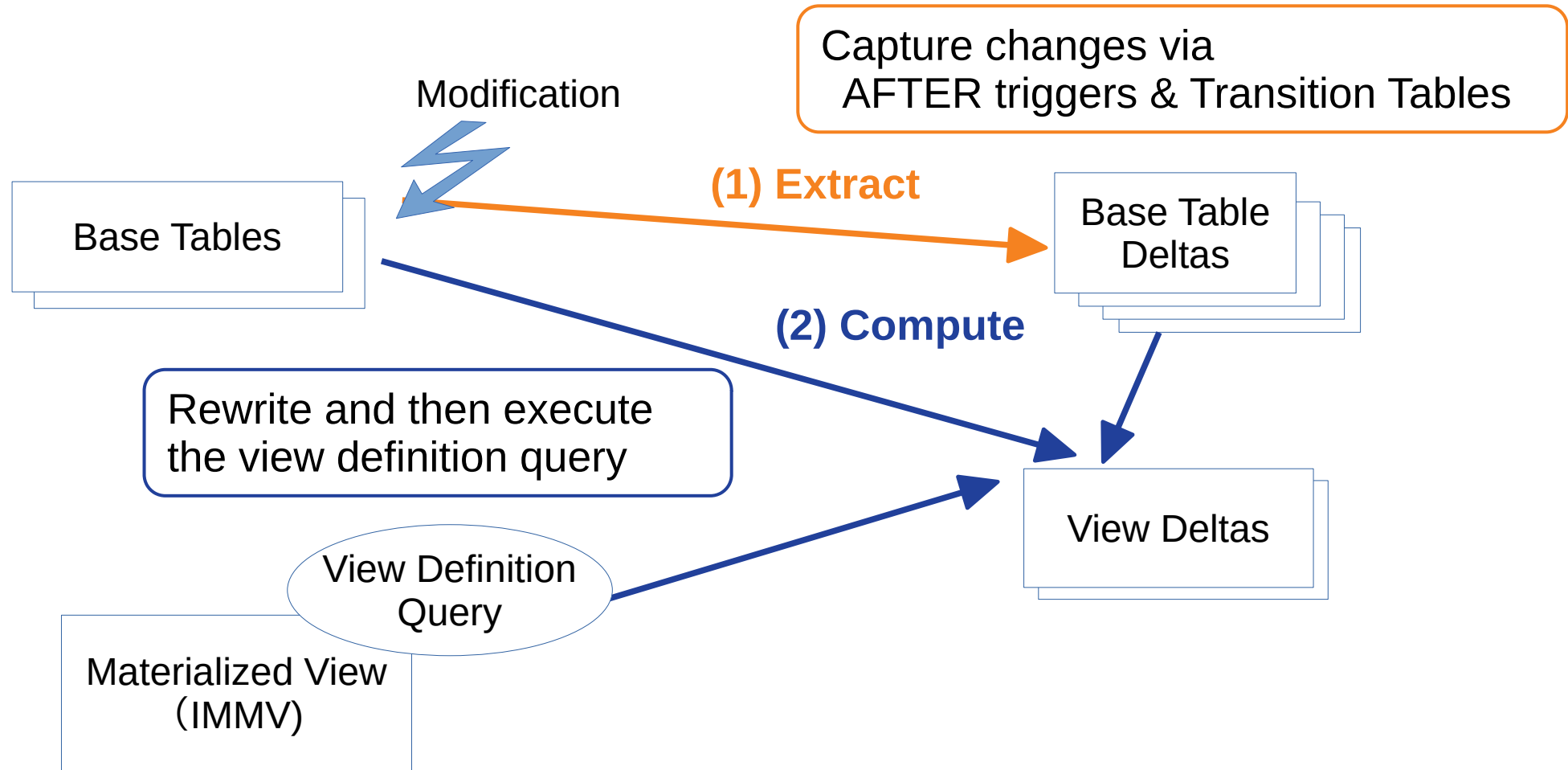
pg_ivm: How It Works



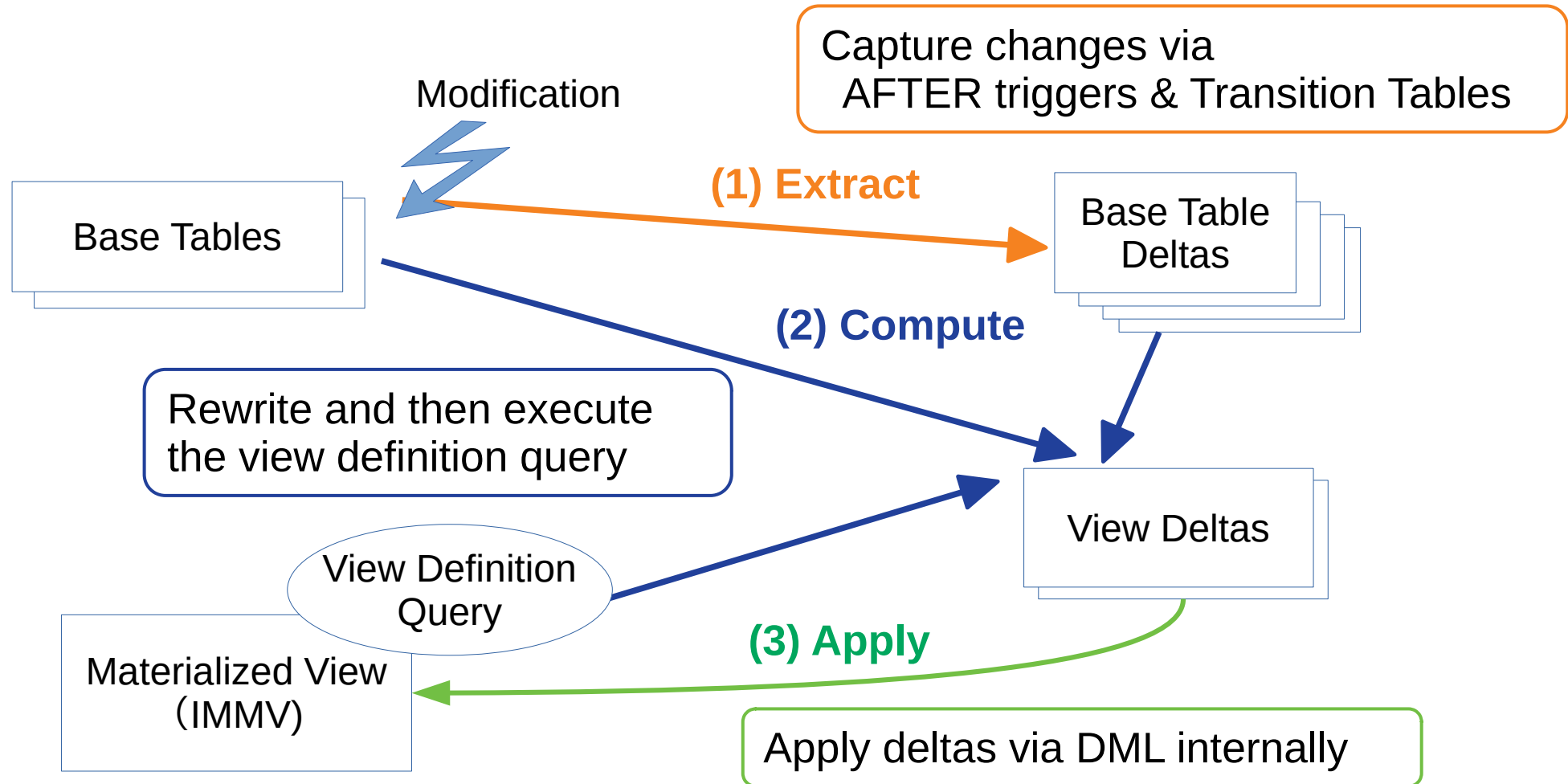
pg_ivm: How It Works



pg_ivm: How It Works



pg_ivm: How It Works



Performance (TPC-H Q01 & Q09)

Query	Measurement Item	Traditional Materialized View	pg_ivm (IMMV)
Q01 (aggregate)	Total Update Time (1-row update in base table: lineitem)	< 1 ms	4 ms (including automatic IVM)
	REFRESH MATERIALIZED VIEW latency	20,000 ms	<i>Not Required*</i>
Q09 (joins + aggregate)	Total Update Time (1-row update in base table: lineitem)	< 1 ms	7 ms (including automatic IVM)
	REFRESH MATERIALIZED VIEW latency	14,000 ms	<i>Not Required*</i>

*Not Required**: The view is updated incrementally and automatically upon base table updates.

Note:

- Approximate values based on TPC-H SF 10 (lineitem: 60M rows)
- Identical SELECT performance to traditional materialized views

Performance Trade-Off

- Table update overhead
 - IVM processing adds latency to each table update.
 - The overhead varies depending on the query complexity and data volume. (Example: 4-7ms in our TPC-H queries (SF=10))
- Exclusive locking for correctness
 - Acquires an exclusive lock on the IMMV to ensure consistency.
 - Concurrent transactions are waited or aborted based on the isolation level.
- Suitable when freshness is important
 - Ideal when real-time data visibility is prioritized over write responsiveness.

Recent Improvements

- Outer Join Support (v1.13)
- pg_dump / pg_upgrade Support (v1.15)
- PostgreSQL 19 Support (v1.16)
- Stability and Correctness Improvements

Outer Join Support (v1.13~)

Developed with support from Mediapart

- Challenge

- Null-extended tuples (= **dangling tuples**) which have no matching row

```
SELECT c.c_custkey, c.c_name, o.o_orderkey, o.o_orderdate
FROM customer c LEFT OUTER JOIN orders o ON c.c_custkey = o.o_custkey;
```

c_custkey	c_name	o_orderkey	o_orderdate
1	Customer#01	10001	2024-01-15
2	Customer#02	10002	2024-01-20
3	Customer#03		

(3 rows)

dangling tuple

Maintaining Outer Join Views

- Handling dangling tuples

- Insert into a table → dangling tuples deleted
- Delete from a table → dangling tuples inserted

```
INSERT INTO orders (...)  
VALUES (3, 10003, '2025-03-10', ...);
```

```
DELETE FROM orders  
WHERE o_orderkey = 10002;
```

```
SELECT c.c_custkey, c.c_name, o.o_orderkey, o.o_orderdate  
FROM customer c LEFT OUTER JOIN orders o ON c.c_custkey = o.o_custkey;
```

c_custkey	c_name	o_orderkey	o_orderdate
1	Customer#01	10001	2024-01-15
2	Customer#02	10002	2024-01-20
3	Customer#03		

(3 rows)

delete

insert

2	Customer#02		
---	-------------	--	--

Outer Join Support: Key idea

- An outer join can be represented as:
 - Inner Join
 - Rows produced by Anti-Joins

```
SELECT c.c_custkey, c.c_name, o.o_orderkey, o.o_orderdate  
FROM customer c LEFT OUTER JOIN orders o ON c.c_custkey = o.o_custkey;
```

c_custkey	c_name	o_orderkey	o_orderdate
1	Customer#01	10001	2024-01-15
2	Customer#02	10002	2024-01-20
3	Customer#03		

(3 rows)

Inner join of customer and orders
(reuses existing join maintenance)

Anti-join of customer and orders
(additional maintenance for dangling tuples)

- Each component is maintained incrementally and combined to produce the final result.

Outer Join Support: Current Restrictions

- Requirements
 - Simple equijoins only (e.g., $A.id = B.id$)
 - Target list must include all attributes used in join conditions.
- Not supported
 - Aggregates or subqueries
 - Non-strict functions in the target list
 - Complex WHERE clauses (e.g., OR or non-null-rejecting predicates)

Support pg_dump / restore, and pg_upgrade (v1.15~)

- Before 1.15:
 - IMMV metadata is lost after restore, or incompatible between major versions after upgrade.
 - IMMV had to be recreated manually
- Since v1.15
 - IMMV metadata can be restored by `restore_immv()`

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

- Incremental maintenance continues **without recreating IMMV**s

Future Directions

- Deferred Maintenance
- Partitioned Tables
- Logical Replication
- Relaxing Outer Join Restrictions

Future Directions

- Deferred Maintenance
- Partitioned Tables
- Logical Replication
- Relaxing Outer Join Restrictions

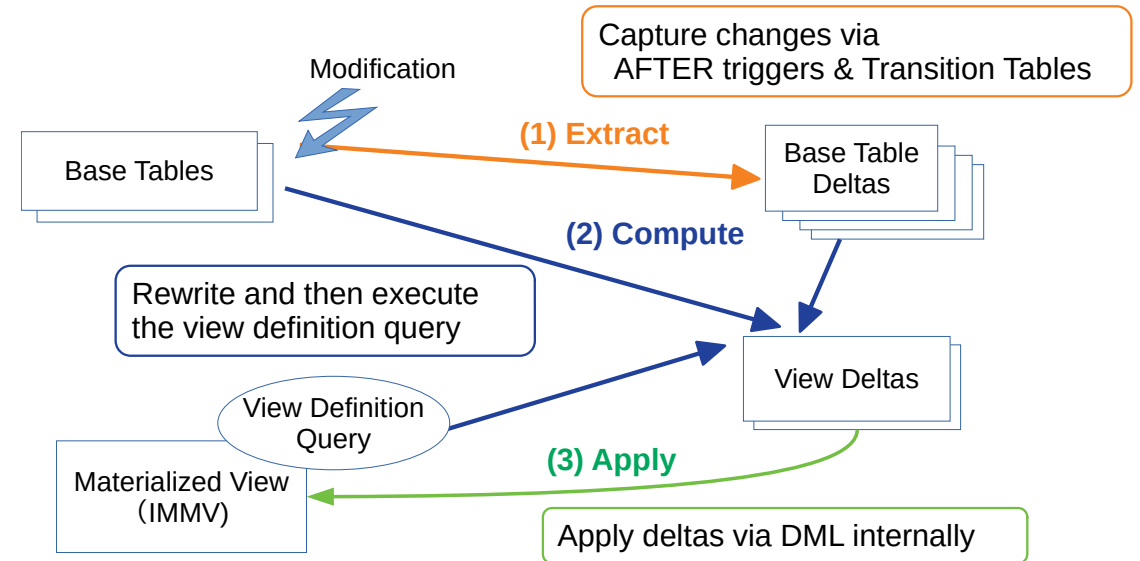
Deferred Maintenance: Motivation

- Immediate Maintenance (Current pg_ivm)
 - Views are updated immediately after each base-table update
 - Always-fresh results
 - Additional update overhead
- Deferred Maintenance
 - Capture changes first, maintain views later
 - Maintenance performed manually or periodically
 - Reduced impact on update performance

Deferred Maintenance: Capturing and Storing Deltas

- Capturing Base Table Delta

- Statement trigger & Transition Tables
 - closest to current architecture
 - practical for an initial PoC
- Alternative approaches
 - Row-level trigger, WAL decoding
- Storing it into dedicated tables



- Computing View Deltas using the table deltas

- Instead of transition tables

Deferred Maintenance: Delta Representation

- What should be stored?
 - Full tuples
 - Primary keys only
 - Minimal required columns
- Goals
 - Minimize change-capture overhead
 - Preserve enough information for maintenance

cf. Oracle Materialized View Logs

Deferred Maintenance: Correctness Challenges

- Maintaining correctness guarantees
 - Deferred maintenance must provide the same correctness guarantees as immediate maintenance.
 - Examples:
 - Repeated updates in the same table
 - User trigger-generated updates

Conclusion

- pg_ivm brings Incremental View Maintenance for PostgreSQL
- Recent releases added:
 - Outer Join Support
 - pg_dump/pg_upgrade Support
 - PostgreSQL 19 Support
- Exploring future directions including deferred maintenance

Thank you!



https://github.com/sraoss/pg_ivm