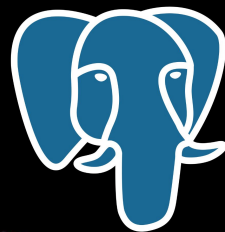


Incremental Materialized Views 101



Engineering

Bloomberg

PGConf India 2024
February 29, 2024

Tushar Amrit
Software Engineer

TechAtBloomberg.com

Imagine this



[TechAtBloomberg.com](https://www.techatbloomberg.com)

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering

You did it!!



TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering

Inevitable happens



TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering

What to do?!



TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering

Agenda

- ⚙ Understanding Materialized Views and why we need them

 - Issues with traditional MVs in PostgreSQL

 - Getting familiar working with Postgres' `pg_ivm` extension

 - Diving deep into `pg_ivm`



Views

Materialised


Incremental

TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering



Views 🧐

Virtual tables for simplifying queries

Provides abstraction

Streamlining complex queries by providing an alias

⚡ Optimizing performance with database views

Schema

```
-- Users Table
```

```
CREATE TABLE users (  
  user_id SERIAL PRIMARY KEY,  
  username VARCHAR(50) NOT NULL,  
  email VARCHAR(100) NOT NULL  
);
```

```
-- Products Table
```

```
CREATE TABLE products (  
  product_id SERIAL PRIMARY KEY,  
  product_name VARCHAR(100) NOT NULL,  
  price DECIMAL(10, 2) NOT NULL  
);
```

Schema

```
-- Orders Table
CREATE TABLE orders (
  order_id SERIAL PRIMARY KEY,
  user_id INT REFERENCES users(user_id),
  product_id INT REFERENCES products(product_id),
  quantity INT NOT NULL,
  order_date DATE NOT NULL
);

-- Indexes for Orders Table
CREATE INDEX idx_user_id ON orders(user_id);
CREATE INDEX idx_product_id ON orders(product_id);
CREATE INDEX idx_order_date ON orders(order_date);
```

Creating Fake Data

Using PostgreSQL 16

We create a database named `mydatabase` and create the tables with indexes

+ We add 11 million rows in the `Users` and `Products` tables and around 4.5 million rows in the `Orders` table

The data is created using a Python script using the `Faker` library to create dummy data and the `psycopg2` Postgres adapter module

The script can be found [here](#)

Running a Simple Query 🏃

Calculate total Order amount for each User:

```
SELECT
    users.user_id,
    username,
    email,
    SUM(price * quantity) AS total_order_amount
FROM
    users
JOIN
    orders ON users.user_id = orders.user_id
JOIN
    products ON orders.product_id = products.product_id
GROUP BY
    users.user_id,
    username,
    email;
```

Time: 26504.869 ms (00:26.505)

Bloomberg

Engineering

user_id	username	email	total_order_amount
388363	melissa95	mbrown@example.org	6186.90
388364	ataylor	heather69@example.com	5774.86
388365	kelly33	angelaclark@example.com	882.85
388366	kathryn57	qmccullough@example.org	1438.36
388368	rjefferson	jeffhernandez@example.net	283.47
388369	donovanashley	ujackson@example.org	1001.61
388371	hannah17	wrightkimberly@example.com	3349.90
388373	wisematthew	natashathompson@example.net	6589.43
388374	josephdalton	coopermatthew@example.net	5536.89
388375	tgreen	samueljones@example.com	3015.24
388376	smithdana	christopherhubbard@example.net	4490.32
388377	kenneth30	heatherwarren@example.com	2183.64
388380	xcampbell	martinezsusan@example.org	11564.15
388385	ashleybrown	johnstonnoah@example.com	5491.92
388387	ryanli	victoriaellis@example.org	996.07
388389	coxrenee	tracy79@example.org	1642.60

All these emails and usernames were generated using the Faker library in Python -

<https://pypi.org/project/Faker/>

TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering

Using View for our query 🧠

```
CREATE VIEW user_order_summary AS
SELECT
  users.user_id,
  username,
  email,
  SUM(price * quantity) AS total_order_amount
FROM
  users
JOIN
  orders ON users.user_id = orders.user_id
JOIN
  products ON orders.product_id = products.product_id
GROUP BY
  users.user_id,
  username,
  email;
```

```
CREATE VIEW
Time: 98.458 ms
```

```
Time: 28801.565 ms (00:28.802)
```

```
SELECT * FROM user_order_summary;
```



Views

Materialised


Incremental

TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering



Materialized Views - Taking Views to Next Level

Precomputed results for improved query performance

Persistently store query results

Refresh on-demand for real-time accuracy

Optimizing read-heavy workloads

Can be indexed

✗ No support for temporary Materialized Views

✓ Supported in PG since 9.3

Materialized Views in Query Execution

Comes into picture during query rewriting

Cost-based optimization

The SQL query is parsed and rewritten, considering any Views that might be referenced in the query

Implementing MVs in PG

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] table_name
  [(column_name [, ...])]
  [USING method]
  [WITH (storage_parameter [= value] [, ...])]
  [TABLESPACE tablespace_name]
  AS query
  [WITH [NO] DATA];
```

The query is executed and used to populate the View at the time the command is issued (unless WITH NO DATA is used), and may be refreshed later

```
-- Create Materialized View
CREATE MATERIALIZED VIEW user_order_summary_mv AS
SELECT
    u.user_id,
    u.username,
    u.email,
    SUM(p.price * o.quantity) AS total_order_amount
FROM
    users u
JOIN
    orders o ON u.user_id = o.user_id
JOIN
    products p ON o.product_id = p.product_id
GROUP BY
    u.user_id,
    u.username,
    u.email;

-- Index for Materialized View
CREATE UNIQUE INDEX idx_user_order_totals_user_id
ON user_order_summary_mv(user_id);
```

Time: 26504.755 ms (00:26.505)

Bloomberg

Engineering

Running Same Query on MV

```
SELECT * FROM user_order_summary_mv;
```

```
mydatabase=# SELECT * FROM user_order_summary_mv;  
Time: 2085.810 ms (00:02.086)
```

Nearly 10x reduction in execution time

Refreshing MVs

```
REFRESH MATERIALIZED VIEW [CONCURRENTLY] name [WITH [NO] DATA];
```

Does a hard and complete refresh of data

Has option of concurrently refreshing data

Refreshing our MV

```
UPDATE orders  
SET quantity = 10  
WHERE order_id = 19363;
```

```
UPDATE 1  
Time: 9.211 ms
```

```
REFRESH MATERIALIZED VIEW user_order_summary_mv;
```

```
mydatabase=# REFRESH MATERIALIZED VIEW user_order_summary_mv;  
REFRESH MATERIALIZED VIEW  
Time: 29670.218 ms (00:29.670)
```

Issues with the MV Refresh 🚧

Maintenance overhead

Computationally heavy

Locks during refresh



Views

Materialised


Incremental

TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering



Incrementally Maintaining Views - Taking Views to Boss Level

The two types of View maintenance

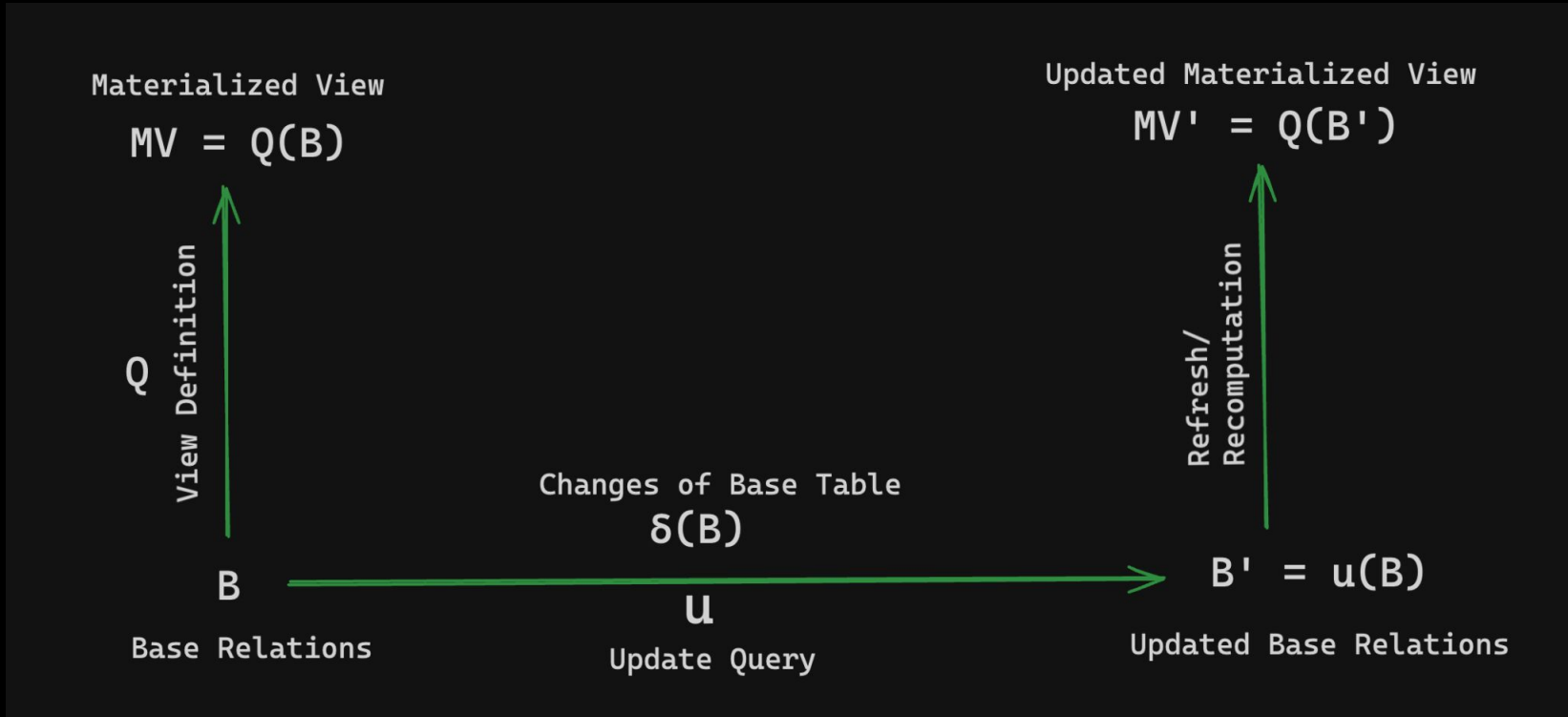
- Immediate
- Deferred

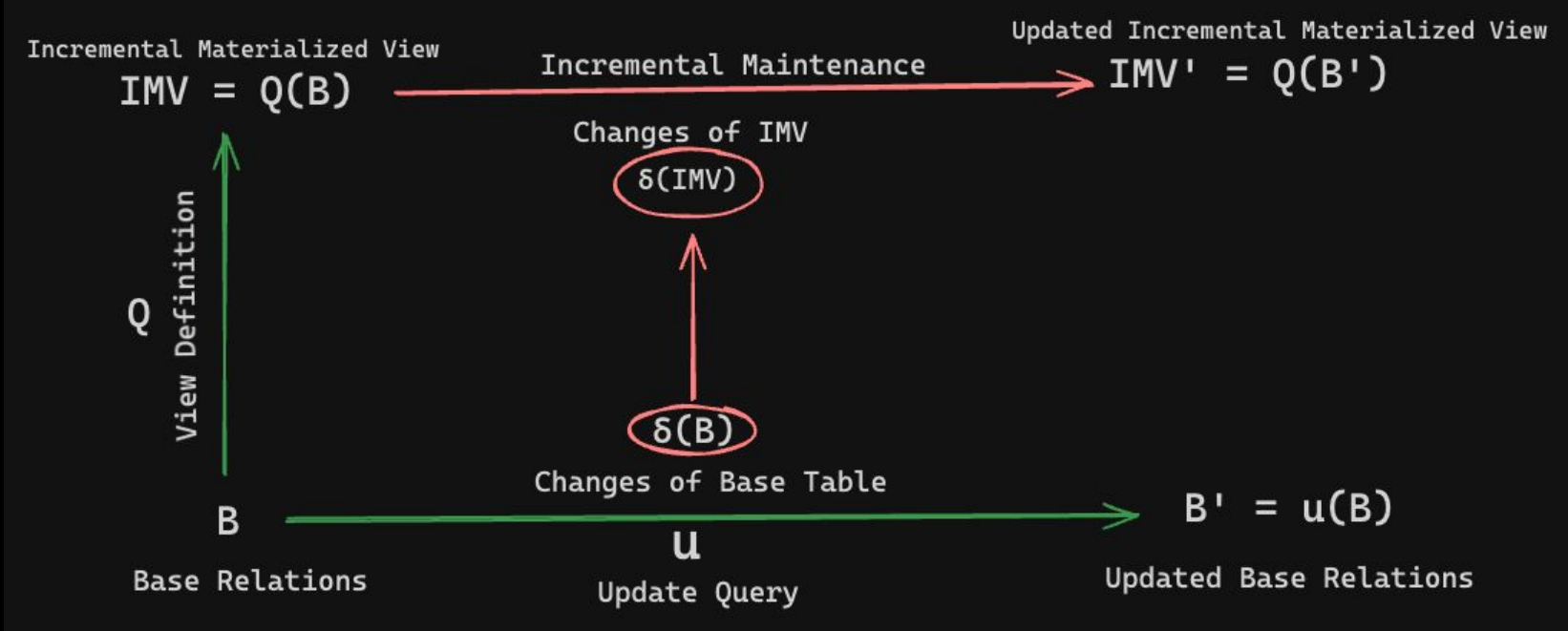
Deferred refresh leads to stale data

Deferred is okay, but we want same transaction

Idea to update only those rows in the base table which are affected

It's time for some ALGEBRA





Basic Theory for IVM for Natural JOIN 🤔

- View definition
 - $V = A \bowtie B$, where A and B are the base tables
- Changes to Base Table A
 - $A \leftarrow (A - \nabla A) \cup \Delta A$, where ∇A are the deleted tuples and ΔA are the tuples inserted by the update query
- Changes on the View
 - $\nabla V = \nabla A \bowtie B$
 - $\Delta V = \Delta A \bowtie B$
- Apply the changes to the View
 - $V \leftarrow (V - \nabla V) \cup \Delta V$

Theory is okay, but how do we implement it? 📚

We have two main approaches:

Using Triggers and Transition Table

Logically decoding WAL

pg_ivm - The Saviour

Extension released by IVM Development Group (led by Yugo Nagata)

Published 1.0 release at the end of April 2022

✓ Compatible with PG 13+

Instead of you having to run REFRESH MATERIALIZED VIEW, this extension performs the updates automatically and incrementally

Installing pg_ivm

- To install pg_ivm, download the [source code](#) and execute this in the module's directory:

```
make install
```

- Execute CREATE EXTENSION command

```
CREATE EXTENSION pg_ivm;
```

- Installing pg_ivm creates the create_immv, refresh_immv and get_immv_def functions

Creating Incrementally Maintained Views 🖋️

We call a Materialized View supporting IVM an Incrementally Maintainable Materialized View (IMMV)

```
SELECT create_immv('myview', 'SELECT * FROM mytab');
```

This is equivalent to :

```
CREATE MATERIALIZED VIEW myview AS SELECT * FROM mytab;
```


Creating IMMV on our Schema

```
SELECT create_immv('user_order_summary_ivm', '  
    SELECT  
        users.user_id,  
        username,  
        email,  
        SUM(price * quantity) AS total_order_amount  
    FROM  
        users  
    JOIN  
        orders ON users.user_id = orders.user_id  
    JOIN  
        products ON orders.product_id = products.product_id  
    GROUP BY  
        users.user_id,  
        username,  
        email;  
' );
```

Time: 48423.240 ms (00:48.423)

Bloomberg

Engineering

Running Same Query on IMMV

```
SELECT * FROM user_order_summary_ivm;
```

```
mydatabase=# SELECT * FROM user_order_summary_ivm;  
Time: 2422.816 ms (00:02.423)
```

```
UPDATE orders  
SET quantity = 15  
WHERE order_id = 19363;
```

```
UPDATE 1  
Time: 13.442 ms
```

How pg_ivm works internally?!



TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering

AFTER Trigger

Enables collection of row sets that include all of the rows inserted, deleted, or modified by the current SQL statement

These row sets can be referred as tables of specified name

Transition Table

Introduced in Postgres 10

Also known as "pseudo-tables"

 Holds data before and after the triggering event (e.g., INSERT, UPDATE, DELETE)

Accessed using NEW and OLD keywords within trigger functions

- Statement-level AFTER triggers are created automatically on all base tables contained in the View definition query
- Changes that will occur in the View are calculated by a rewritten View query with the modified tables replaced by transition tables

View Definition	SELECT ... FROM A, B WHERE ...
Rewritten Query	SELECT ... FROM new_table_A, B WHERE ...

Handling Duplicate Rows for DISTINCT Clause 🚶 🚶

- If there are two of the same tuples in a View and we would like to delete only one tuple rather than both
 - We cannot simply use a DELETE statement, because this will delete both tuples
- Similarly, if MV is defined with DISTINCT and there are duplicate tuples in its base table
 - When deleting tuples from the base table, a tuple in the View should be deleted if *and only if* the duplicity of the tuple becomes zero
 - Otherwise, the tuple must remain in the View

Counting Algorithm

- Algorithm for handling tuple duplicates with DISTINCT clause in IMMV
- When tuples are to be inserted into the View, the count is increased if there is already the same one; otherwise, if the same tuple doesn't exist, a new tuple is inserted
- Similarly, when tuples are to be deleted from the View, the count is decreased; if the count becomes zero, this tuple is deleted from the View
- The IVMs has a special column, `__ivm_count__`, which maintains this count


```
mydatabase=# SELECT * FROM user_order_summary_ivm LIMIT 5;
```

user_id	username	email	total_order_amount	__ivm_count_total_order_amount__	__ivm_count__
388363	melissa95	mbrown@example.org	6186.90	2	2
388364	ataylor	heather69@example.com	5774.86	2	2
388365	kelly33	angelaclark@example.com	882.85	1	1
388366	kathryn57	qmccullough@example.org	1438.36	1	1
388368	rjefferson	jeffhernandez@example.net	283.47	1	1

(5 rows)

All these emails and usernames were generated using the Faker library in Python - <https://pypi.org/project/Faker/>

TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.

Bloomberg

Engineering

Views with Aggregate

- Perform aggregation on the differential rows that occur in the table, and use the results to update the aggregate value in the View
- Update method varies depending on the type of aggregation
- Some columns are added automatically
 - Number of rows in the entire View and within each group (`_ivm_count_...`)
 - For avg, additional columns to store the results of count and sum

Concurrent Execution

Obtains an exclusive lock

READ COMMITTED isolation level causes a wait

At REPEATABLE READ isolation level or higher, one transaction is aborted

pg_ivm Drawbacks 🐱

Loss in performance of update of Base Tables

Currently, only built-in aggregate functions are supported and user-defined aggregates cannot be used

Inner joins including self-join are supported, but outer joins are not supported

Views, materialized Views, inheritance parent tables, partitioned tables, partitions, and foreign tables cannot be used as the base table

Logical replication is not supported

Conclusion

With `pg_ivm` MVs can be updated faster than `REFRESH`, but it affects table update performance
Useful in situations where “the table is not updated often, but you want the latest query results immediately when there is an update”

When loading large amounts of data, it is a good idea to temporarily disable automatic View updates

Thank you!

Contact me: tamrit@bloomberg.net

Engineering

Bloomberg

TechAtBloomberg.com

© 2024 Bloomberg Finance L.P. All rights reserved.