pg_ivm: マテリアライズドビューを高速に更新するための PostgreSQL 拡張モジュール

PostgreSQL Conference Japan 2022

2022-11-11

長田悠吾（SRA OSS LLC　OSS事業本部 技術開発室）

自己紹介

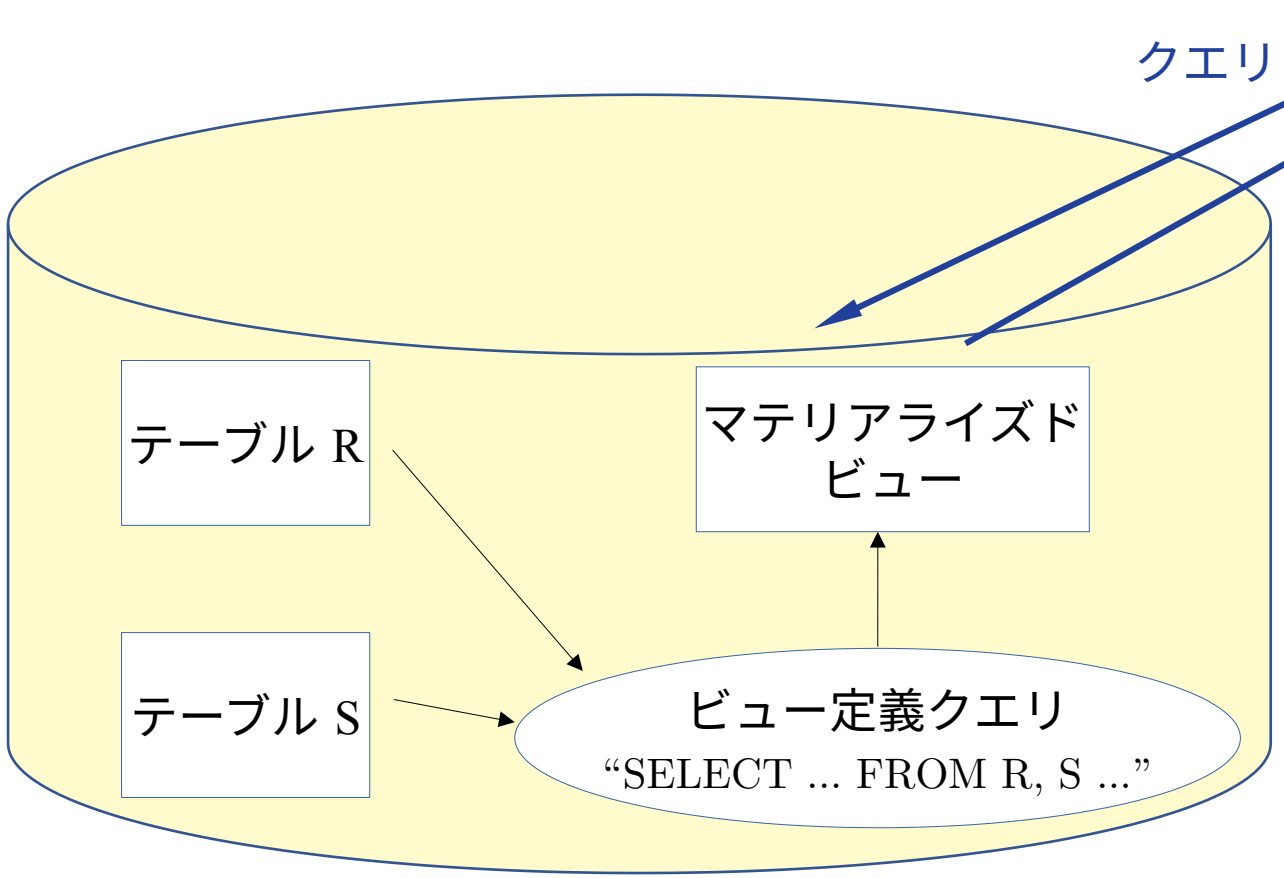
- 長田 悠吾（ながたゆうご）
- SRA OSS LLC　OSS事業本部 技術開発室、
  SRA ホールディングス 先端技術研究所
- PostgreSQL 関連の研究開発
  - 増分ビューメンテナンス機能（Incremental View Maintenance: IVM）
- PostgreSQL インターナル講座

本日の内容

- 拡張モジュール pg_ivm のご紹介
  - PostgreSQL に増分ビューメンテナンス（Incremental View Maintenance, IVM）機能を提供
    IVM = マテリアライズドビューの最新化を高速に行う手法
- あらまし
  - 増分ビューメンテナンス（IVM）とは何か？
  - pg_ivm について
    - 概要、機能、実装、性能評価
  - まとめ

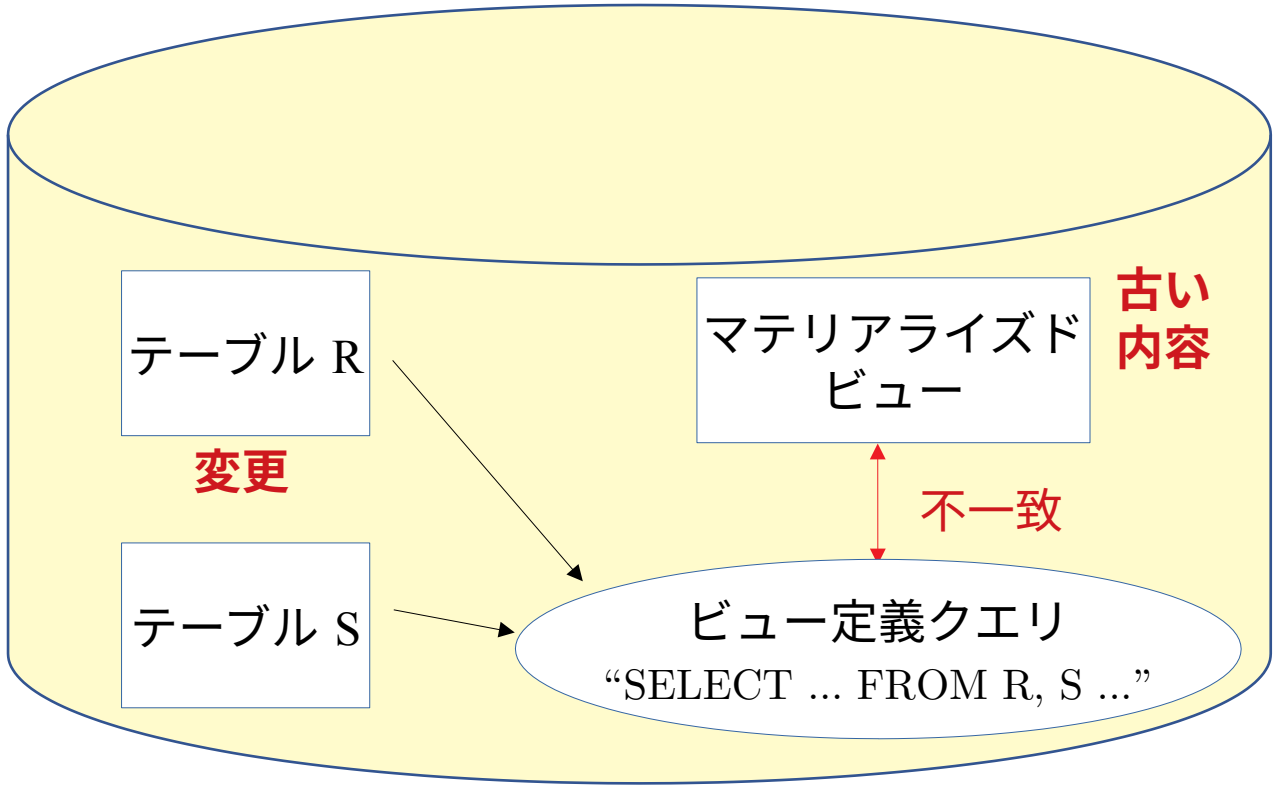
増分ビューメンテナンス
（Incremental View Maintenance: IVM）

マテリアライズドビューの増分メンテナンス（１）



クエリの実行結果を
データベースに格納

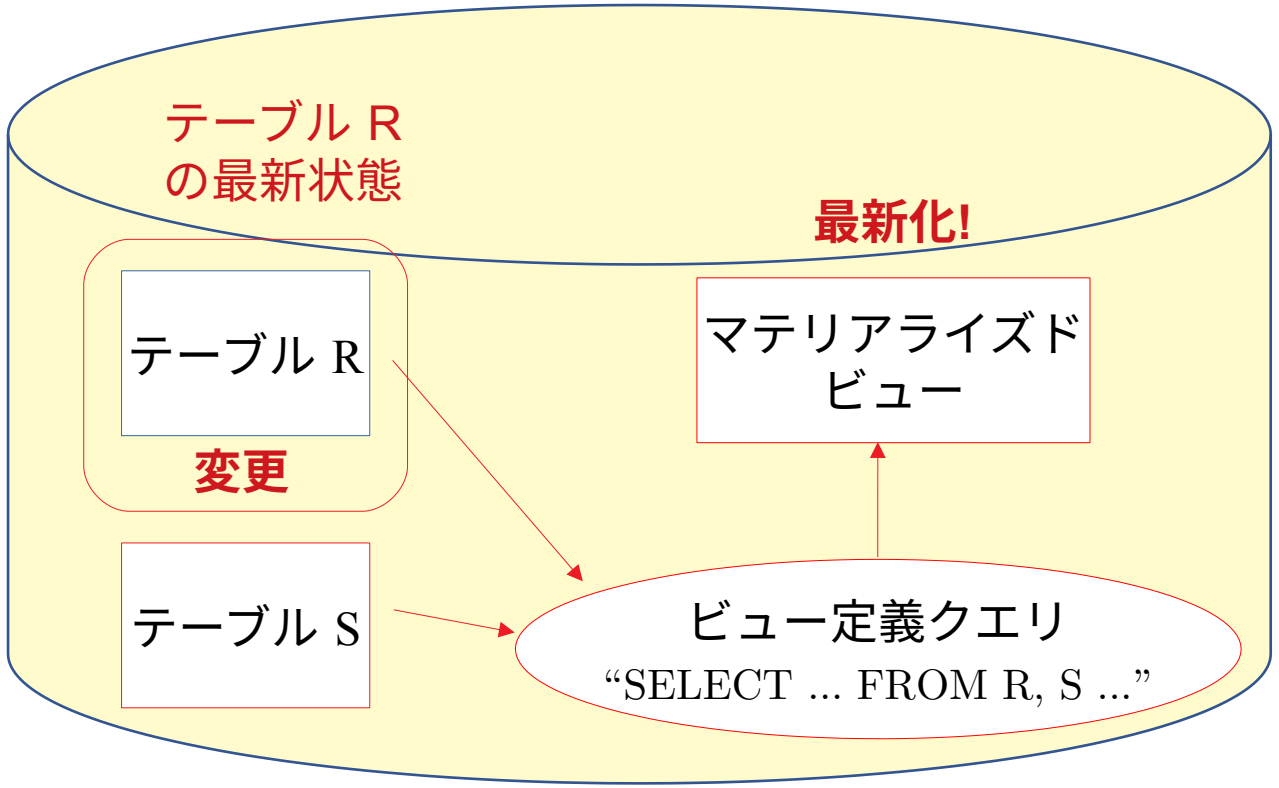
マテリアライズドビューの増分メンテナンス（2）



クエリの実行結果を
データベースに格納

テーブルが変更された後には
メンテナンスが必要

マテリアライズドビューの増分メンテナンス（3）

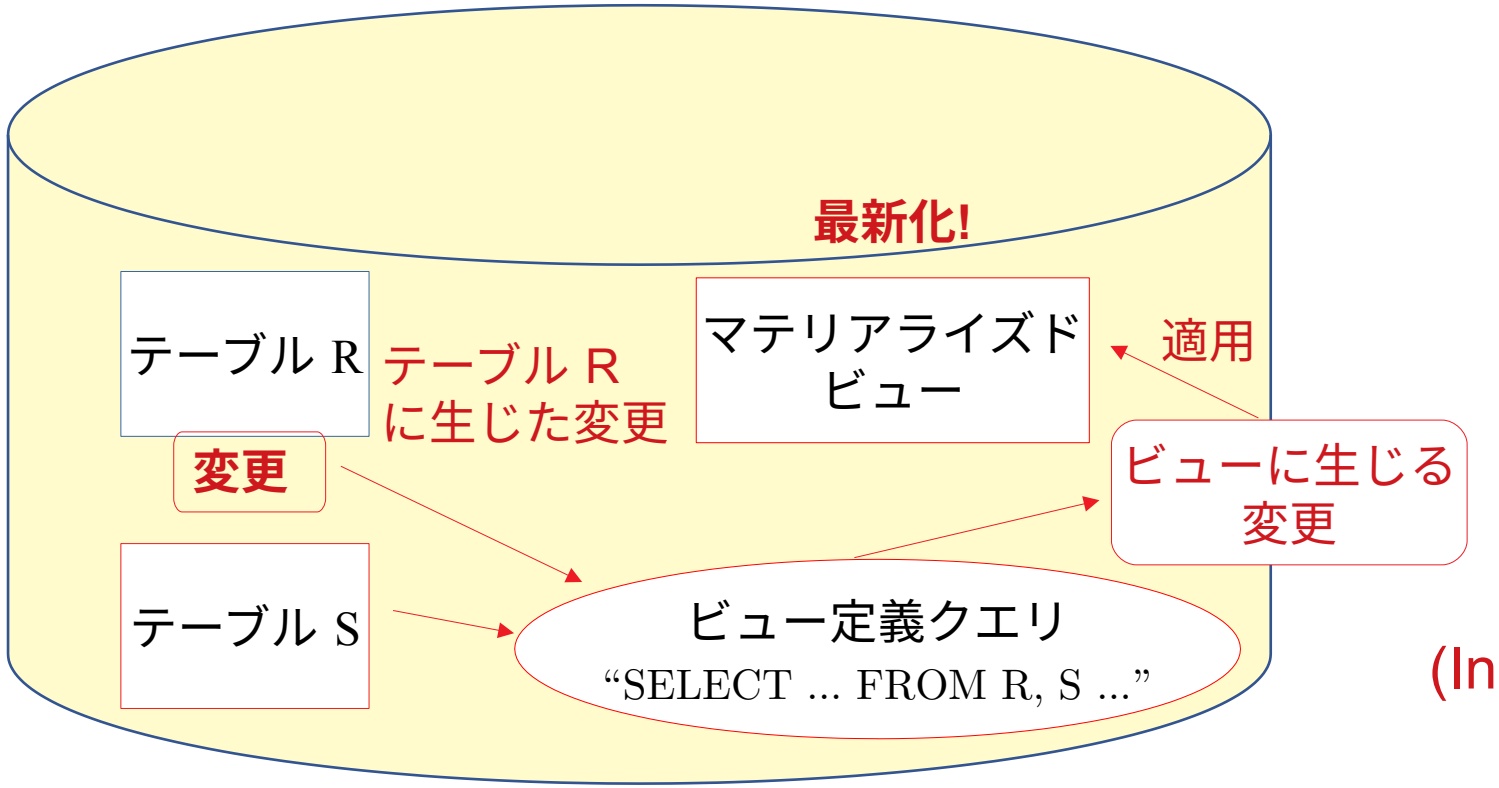


REFRESH MATERIALIZED VIEW

II

1.最新のテーブルの状態を使ってマテリアライズドビューの内容を再計算

マテリアライズドビューの増分メンテナンス（４）



REFRESH MATERIALIZED VIEW

=

1.最新のテーブルの状態を使ってマテリアライズドビューの内容を再計算

2.増分変化だけを計算してビューに適用する

=

増分ビューメンテナンス
(Incremental View Maintenance)

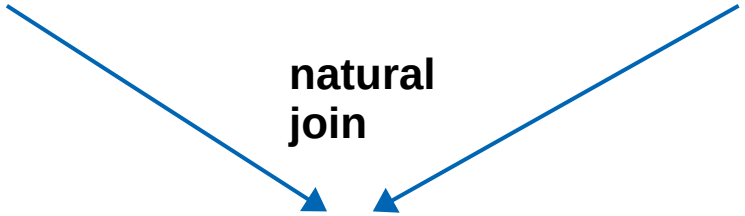
増分ビューメンテナンスの簡単な例（１）

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

増分ビューメンテナンスの簡単な例（２）

テーブルRが更新された場合：

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

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

ビュー V に発生する差分を計算

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

number	english	roman
1	one	I

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

number	english	roman
1	ONE	I

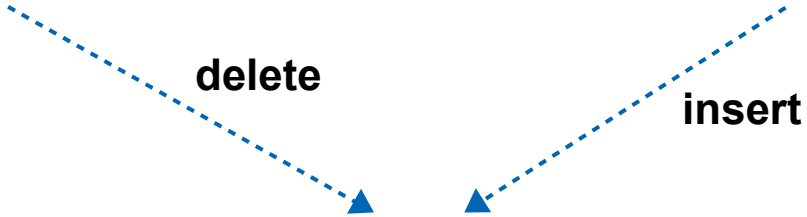
増分ビューメンテナンスの簡単な例（３）

∇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

計算した差分を適用して、
ビュー Vを更新

pg_ivm の概要

PostgreSQLと増分ビューメンテナンス

- 増分ビューメンテナンス（IVM）機能はPostgreSQLには未実装

  → 開発コミュニティにパッチを提案中

- 提案中の IVM 機能を現状の PostgreSQL でも使用したいとの声があった

  → パッチとしてだけではなく拡張モジュール pg_ivm としても提供することにした

  - ねらい
    - IVM 機能に対するフィードバックや認知を得る機会の向上
    - （将来的には）コミュニティ版では提供できていない機能や、実験的機能の提供

pg_ivm が提供する機能

- 「増分メンテナンス可能なマテリアライズドビュー」
  （Incrementally Maintainable Materialized View: IMMV）の作成

- 実テーブルを更新すると、IMMV が自動的に更新される（= 即時メンテンナンス）
  - 実テーブルにAFTER トリガを作成し、その中でマテリアライズドビューを更新
  - ユーザが自前でトリガ関数を書く必要はない
  - 通常の REFRESH では更新に 20 秒くらいかかるビューが、15ミリ秒程度で更新可能

- PostgreSQL 13, 14, 15 に対応

PostgreSQL に提案中の IVM 機能と pg_ivm の違い

- PostgreSQL に提案中の機能
  - IMMV は本体のマテリアライズドビュー機能を拡張する形で実装
    - CREATE *INCREMENTAL* MATERIALIZED VIEW 構文を使って作成する
  - 結合、組み込み集約（count, sum, avg, min, max）、DISTINCT に対応
- pg_ivm
  - IMMVはテーブルとして実装
    - 独自の関数 create_immv を使って作成
  - 上記に加え、ビュー定義でサブクエリを使用可能
    - FROM句の中の、集約を含まない単純なサブクエリ

pg_ivm の使用法

インストール（１）

- ソースから
  - GitHub（https://github.com/sraoss/pg_ivm）からソースコードを入手
  - PostgreSQLのソース、または、devel パッケージ（postgresql14-devel、postgresql-server-dev-14など）が必要

```
$ make install
```

- RPMパッケージから
  - PostgreSQL Yum Repository（https://yum.postgresql.org/）で提供されている
  - yum や dnf を使ってインストール可能

```
$ sudo yum install pg_ivm_14
```

インストール（２）

- CREATE EXTENSION を実行すると以下のオブジェクトが作成される

```
CREATE EXTENSION pg_ivm;
```

  - テーブル
    - pg_ivm_immv:　IMMV のメタデータを格納するカタログ
  - 関数
    - create_immv :　IMMV を作成する
    - refresh_immv:　IMMV を手動で最新化（refresh）する
    - get_immv_def:　IMMV の定義を出力する

IMMVの作成

- ビューの名前と定義を指定してcreate_immv 関数を実行する
  - CREATE METERIALIZED VIEW 文の実行に相当
  - 可能な場合はインデックスが自動生成される

```
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)
```

IMMVの情報

- pg_ivm_immv カタログにエントリが追加されている

```
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.|
                                (中略)
           |.nList <> :mergeUseOuterJoin false :stmt_lo.|
           |.cation 0 :stmt_len 0}                      |
(1 row)
```

IMMVのビュー定義

- ビューの定義は get_immv_def 関数で確認できる
  - psql の \d+ メタコマンドに相当

```
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)
```

IMMVの自動更新

- テーブルを更新するとIMMVは自動で更新される
  - 所要時間はわずか15.448 ms
  - 同じ定義のマテリアライズドビューをREFRESHした場合には20秒以上かかる

```
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)
```

IMMVの手動リフレッシュ

- ビューの名前を指定してrefresh_immvを実行
  - REFRESH MATERIALIZED VIEW 文の実行に相当
  - 第２引数はデータ生成の有無を指定する真偽値（WITH [NO] DATA オプションに相当）
    - true: クエリを実行して新しいデータを生成し、自動更新が有効になる。
    - false: IMMV の中身が空になり、自動更新は無効になる

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

作成された IMMV の実体はテーブル

- psql の \d メタコマンドでは Table と表示される

```
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 の更新は不可

- 通常のマテリアライズドビューの更新と同様
  - 更新しようとするとエラーが発生

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

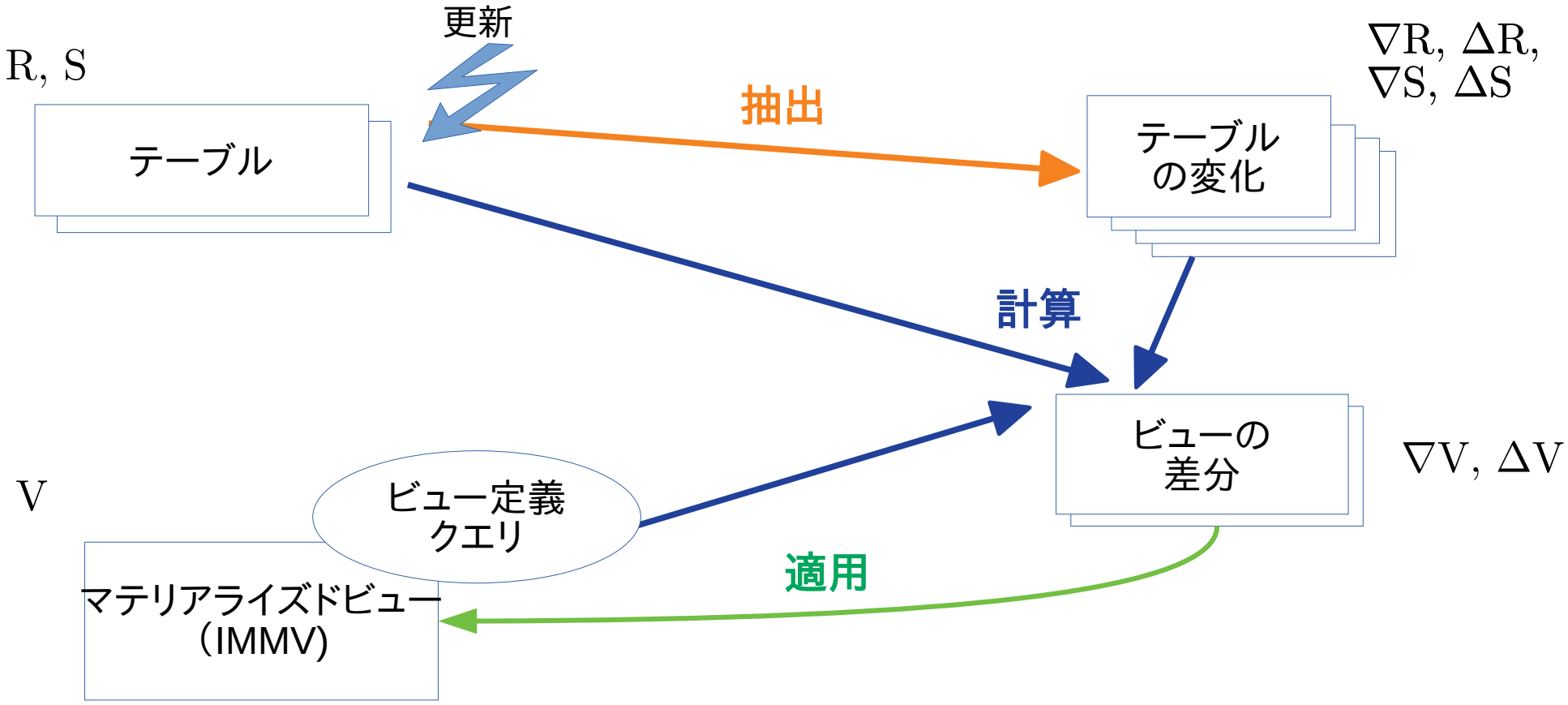
IMMVの削除

- DROP TABLE 文を使用する
  - pg_ivm_immv カタログの情報も自動的に削除される

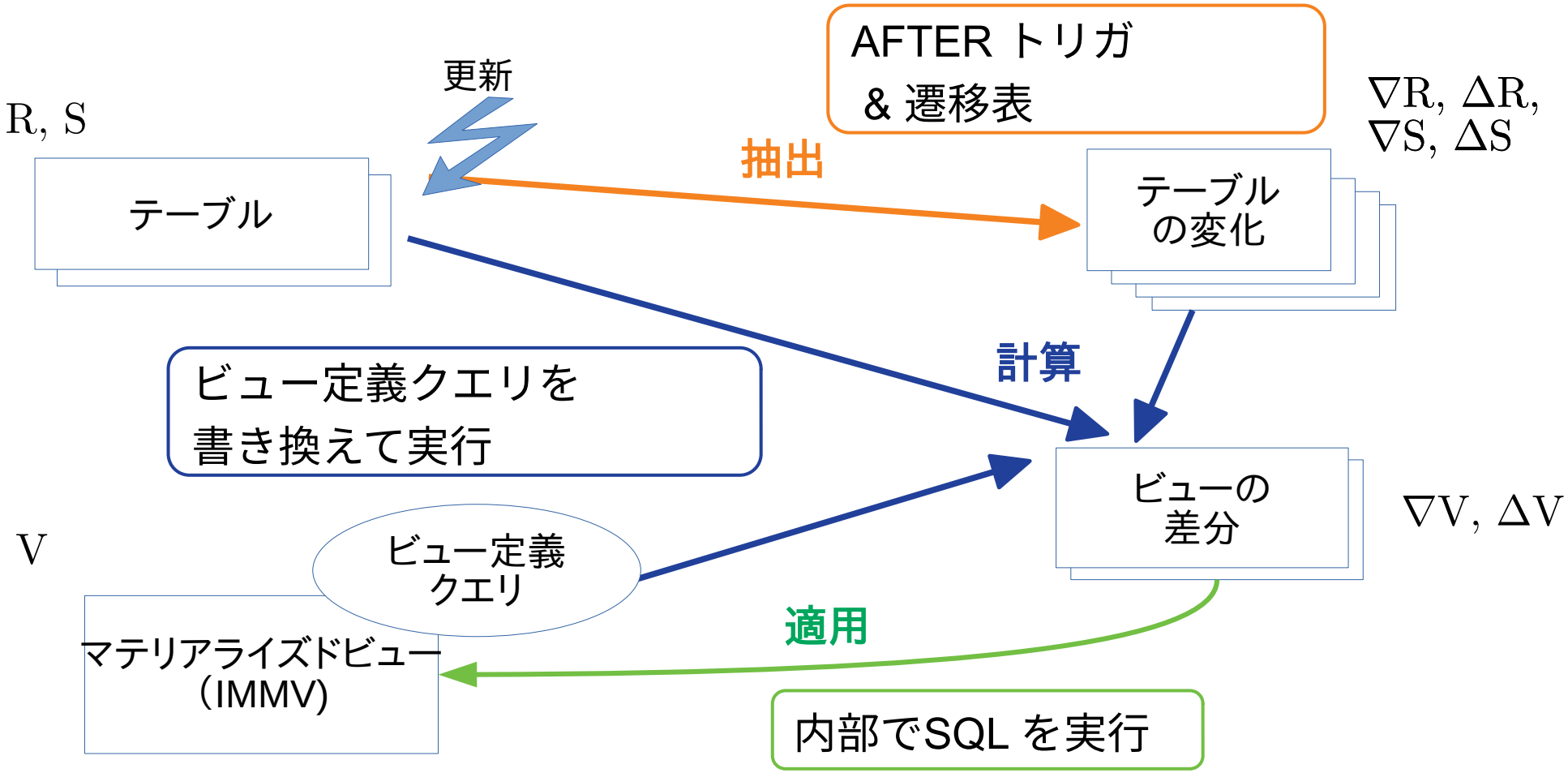
```
test=# DROP TABLE mv;
DROP TABLE

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

pg_ivm の実装

ビュー更新の流れ



ビュー更新の流れ



テーブル変化の抽出

- IMMV の定義する時に、全てのテーブルに AFTER トリガを自動的に作成
  - INSERT, DELETE, UPDATE（および TRUNCATE）の各文の実行直後に実行される
  - その中でテーブル変化の抽出と、ビュー差分の計算および適用を行う

- 遷移表（Transition Tables）を利用して、テーブルの変化を抽出
  - テーブルに発生した変化をトリガ関数の中から参照できる機能
    - テーブルから削除された行の集合
    - テーブルに挿入された行の集合

ビューに発生する差分の計算

- ビュー定義クエリを書き換えて実行する
  - 変更のあったテーブルを遷移表で置き換える

更新のあったテーブル

ビュー定義：

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

書き換え後：

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

遷移表（テーブルRに挿入された行の集合）

→ 実行結果は「ビューに新しく挿入する行の集合」となる

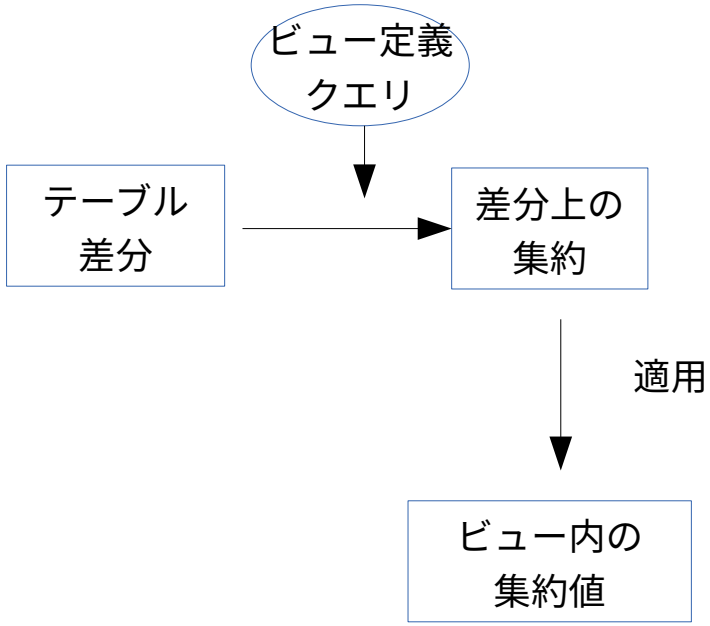
差分をビューに適用

- 内部でSQLを実行
  - DELETE を実行して IMMV から行を削除
  - INSERT を実行して IMMV に行を挿入
  - 集約を含むビューの場合は、 UPDATE を実行して集約値を更新する

- IMMV 上のインデックス
  - DELETE や UPDATE の対象行を検索するため適切なインデックスが IMMV 上に必要
  - 可能な場合には自動で作成する
    - 主キーカラムが存在する場合や、GROUP BY がある場合

集約を含むビューのメンテナンス

- テーブルに発生した差分行に対して集約を行い、その結果を使ってビュー内の集約値を更新する
  - 更新の方法は、集約の種類によって様々
  - 例）
    - count(x) ← count(x) ± [差分上の count(x)]
    - sum(x) ← sum(x) ± [差分上の sum(x)]
    - avg(x) ← (sum(x) ± [差分上の sum(x)]) / (count(x) ± [差分上の count(x)])



- いくつかのカラムが自動で追加される
  - ビュー全体および各グループの中に含まれる行数（_ivm_count_...）
  - avg の場合はさらに、count および sum の結果を格納するカラム

集約: min(x) および max(x) 集約値の更新

- タプル挿入時：
  - min(x) ← least (min(x), [差分上の min(x)])
  - max(x) ← greatest (max(x), [差分上の max(x)])
- タプル削除時：
  - 既存の min(x)（または max(x)）がビューから削除される場合は、新しい最小値・最大値をベーステーブルから計算し直す必要がある
  - それ以外の場合、何もする必要がない

DISTINCTを含むビューのメンテナンス

- 各行の行数を格納するカラム（_ivm_count_）を自動で追加
  - 集約の場合と同様に、テーブルに変化が生じたら行数の情報を更新
  - 行数が0になった行はビューから削除する

TRUNCATE

- ビュー定義に含まれるテーブルが TRUNCATE された場合は、IMMVもTRUNCATEする
  - 通常の結合の場合、クエリ実行の結果が必ず0行となる
  - ただし、GROUP BY がない集約クエリの場合は例外
    - 常に１行の結果がビューに含まれる
    - クエリの再実行によりビューの内容を最新化する

      → 結果として、列の値が NULL である１行が生成される

複数テーブル更新＆自己結合

- 可能な状況
  - 更新 CTE
  - トリガー
  - 外部キー

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

- 自己結合は本質的には複数テーブル更新と同じ
  - クエリ中の同一テーブル ≒ 同じ内容の異なるテーブル

- ビューの差分更新時には、テーブルの「更新前」と「更新後」の両方の状態が必要
  - JOINビューの場合の例
    - 定義　　$V \stackrel{\text{def}}{=} R \bowtie S$
    - 更新　　$R \leftarrow R \cup \Delta R, S \leftarrow S \cup \Delta S$
    - 差分計算　$\Delta V = (\Delta R \bowtie S\_old) \cup (R\_new \bowtie \Delta S)$

テーブルの「更新前」の状態の取得

- AFTER トリガの中では「更新後」の状態しか参照できない
- 「更新前」の状態は、スナップショットを使用して取得
  - テーブル更新前時点のスナップショットを保存しておく(BEFORE トリガ)
  - 更新前の状態を得るために WHERE 句で特別な関数を呼ぶ

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

  - この関数は、tableoid, ctid で指定されたタプルの可視性を保存済みのスナップショットを使って判定し、真偽値を返す

現状の制約

- 以下を含むビューは非対応
  - 組み込みの count, sum, avg, min, max 以外の集約、ウィンドウ関数
  - 外部結合
  - CTE（WITH句）
  - 集約やDISTINCTを含むサブクエリ
  - HAVING, ORDER BY, LIMIT/OFFSET, UNION/INTERSECT/EXCEPT, DISTINCT ON, TABLESAMPLE, VALUES, FOR UPDATE/SHARE
  - ビュー、マテリアライズドビュー、継承テーブル、パーティションテーブル、外部テーブル

性能評価

TPC-H クエリを用いた性能評価（１）

※ スケールファクタ＝１
（lineitem: 600万行程度）

- Q01: １つの巨大テーブルに対する集約
  - 通常のマテリアライズドビューの場合：
    - lineitem テーブルを１行更新： **10ms** 程度
    - REFRESH: **10秒**程度

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 9.995 ms

tpch=# REFRESH MATERIALIZED VIEW mv01;
REFRESH MATERIALIZED VIEW
Time: 10438.409 ms (00:10.438)
```

```
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
ストレージ: SSD
OS: Ubuntu 20.04.5 LTS (64bit), linux kernel 5.4.0-126-generic
PostgreSQL 15rc1 + pg_ivm 1.3（以下同様）

TPC-H クエリを用いた性能評価（２）

※ スケールファクタ＝１
（lineitem: 600万行程度）

- Q01: １つの巨大テーブルに対する集約
  - 通常のマテリアライズドビューの場合：
    - lineitem テーブルを１行更新： **10ms** 程度
    - REFRESH: **10秒**程度
  - pg_ivm を使った場合：
    - lineitem テーブルを１行更新
      ＆IMMVの自動更新： **20ms** 程度

ビューの更新
性能は約500倍

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 19.525 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');
```

TPC-H クエリを用いた性能評価（３）

※ スケールファクタ＝１
（lineitem: 600万行程度）

- Q09: ６つのテーブルを結合して集約
  - 通常のマテリアライズドビューの場合：
    - lineitem テーブルを１行更新： **10ms** 程度
    - REFRESH: **3秒**程度
  - pg_ivm を使った場合：
    - lineitem テーブルを１行更新
      ＆IMMVの自動更新： **45ms** 程度

ビューの更新
性能は約65倍

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 45.688 m
```

```
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');
```

性能のトレードオフ

- REFRESHより高速にマテリアライズドビューを更新可能だが、テーブルの更新性能に影響がある
  - 「テーブル更新頻度は低いが、更新があった際にはすぐに最新のクエリ結果が欲しい」という状況に有用
  - 大量データをロードする場合には、一時的にビューの自動更新を無効にするのが良い

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

- 同時実行性能
  - 複数トランザクションが同時にビューを更新する際に不整合が発生する可能性があるため、ビューの更新は排他ロックを取得して行う
    - READ COMMITTED 分離レベルでは待ちが発生
    - REPEATABLE READ 分離レベル以上では片方のトランザクションがアボートされる

まとめ

pg_ivm：まとめ

- PostgreSQL に増分ビューメンテナンス（IVM）機能を提供する拡張モジュール
  - 増分メンテナンス可能なマテリアライズドビュー（IMMV）の作成
  - テーブルを更新すると、IMMV が自動的かつ高速に更新される
    - 結合、集約（count, sum, avg, min, max）、DISTINCT、単純なサブクエリに対応
  - REFRESHより高速にマテリアライズドビューを更新可能
    - テーブルの更新性能には影響があり
  - PostgreSQL 13, 14, 15 で使用可能

PowerGres Plus V13

- PostgreSQL をベースに独自に拡張したデータベース
  - 安心のサポートサービス
  - 使いやすい GUI 管理ツール
  - 透過的データ暗号化によるセキュリティ
  - トランザクションログ二重化による信頼性

    （https://powergres.sraoss.co.jp/product/plusv13/）

- V13 から pg_ivm が PowerGres Plus で利用可能
  - Windows 対応版あり

pg_ivm の開発

- GitHub：https://github.com/sraoss/pg_ivm
- 開発元：IVM Development Group
- ライセンス：PostgreSQL License
  - 1.0 (2022/4)
    - 基本的な結合（内部結合、自己結合）、DISTINCT に対応
    - PostgreSQL 14 対応
  - 1.1 (2022/6)
    - 集約（count, sum, avg）対応
    - TRUNCATE 対応
    - 手動リフレッシュ（refresh_immv()）対応
    - PostgreSQL 13 対応
  - 1.2 (2022/7)
    - min/max 集約対応
    - 単純なサブクエリに対応
    - PostgreSQL 15 対応
  - 1.3（2022/9）
    - ビュー定義確認関数（get_immv_def()）
    - バグ修正
      - XID周回時の問題など

pg_ivm 開発の今後の予定

- 対応ビュークエリの拡充
  - EXISTS サブクエリ、CTE（WITH句）、外部結合
- その他
  - 性能の改善
  - 遅延メンテナンス
  - パーティションテーブルの対応
  - ・・・

Pull Request や Issue は大歓迎！

Thank you!