PostgreSQLのマテリアライズドビューを高速に更新する「pg_ivm」の紹介
～ 開発者が語る機能概要と今後の展望 ～

Open Developers Conference 2026

2026-08-29

長田悠吾（株式会社SRA OSS）

自己紹介

- 長田 悠吾（ながたゆうご）
- 株式会社SRA OSS データベース技術グループ、
  株式会社SRA ホールディングス 先端技術研究所
- PostgreSQL Contributor
- pg_ivm（マテリアライズドビュー増分更新モジュール）作者

会社紹介

株式会社SRA OSS

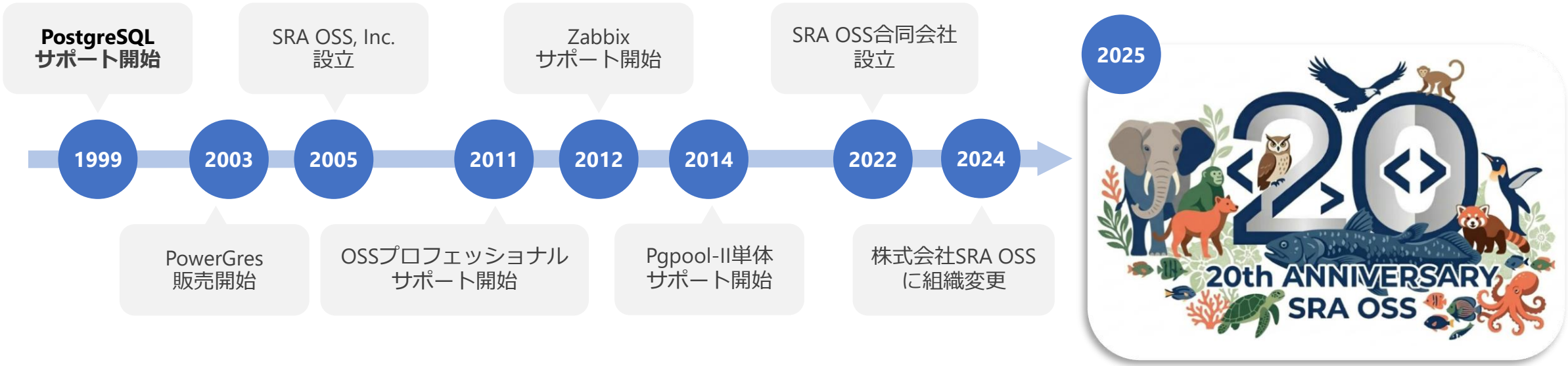
所 在 地: 東京都豊島区南池袋2-32-8
設 立 日: 2022年6月17日
株　　主: 株式会社SRA
　　株式会社NTTデータ
資 本 金: 7,000万円
社　　長: 稲葉 香理

事業内容

- オープンソースソフトウェア (OSS) 関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

顧　　問: 石井達夫

技術顧問:増永 良文 (お茶の水女子大学名誉教授)



PostgreSQL とは

- 多機能、高性能、かつオープンソースの RDBMS
  - 標準SQLの大部分とそのほかの先進的な機能をサポートする
  - BSDタイプの比較的自由度の高いライセンス
    - 広告条項なし
    - 使用、複製、改変、配布の自由
  - 高い拡張性
    - 拡張モジュール（Extension）による機能追加
    - 今回紹介する pg_ivm もその１つ

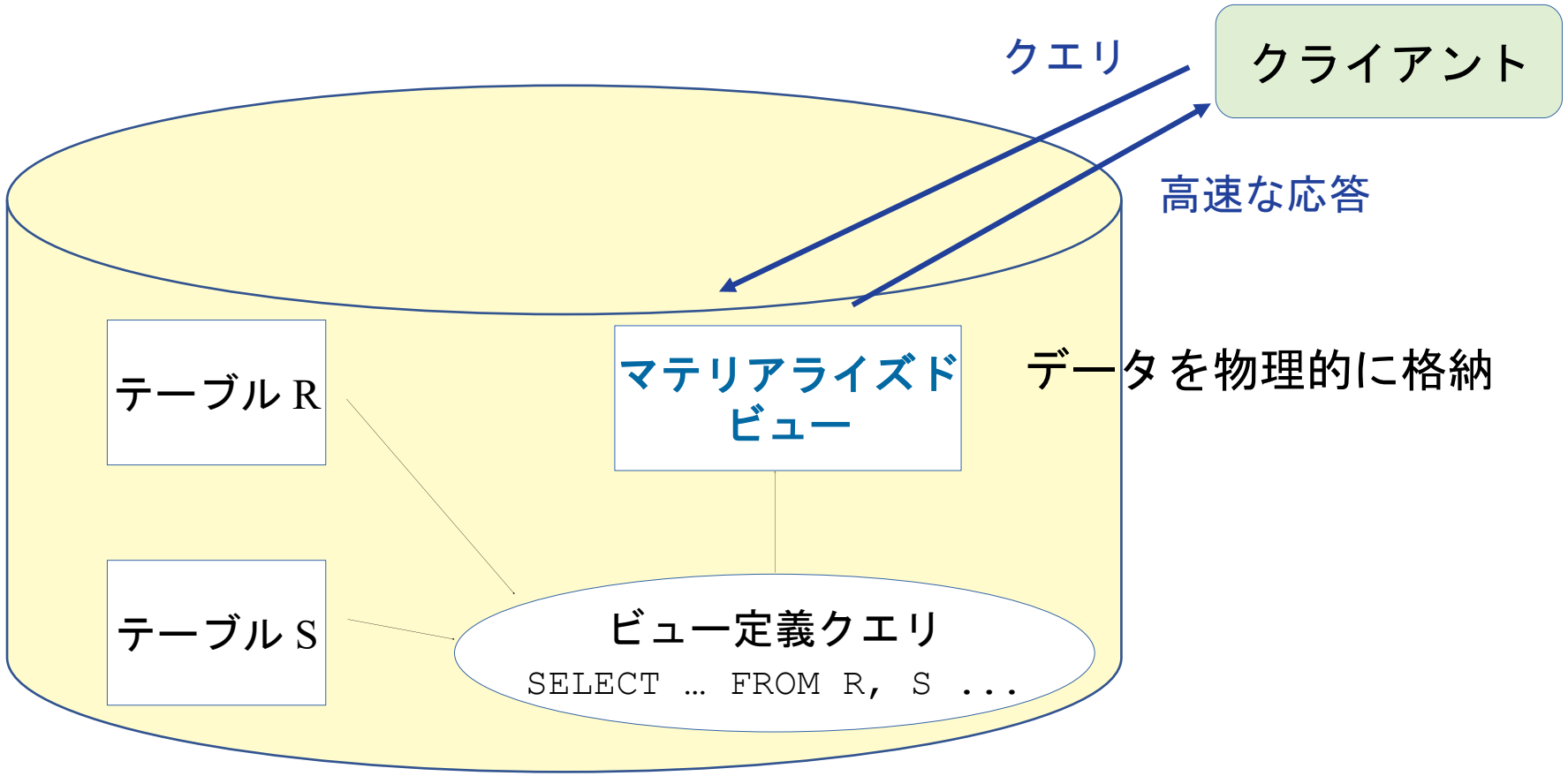
講演概要

- pg_ivm
  - PostgreSQL にマテリアライズドビューの最新化を高速に行う機能を提供する拡張モジュール
  - 増分ビューメンテナンス（Incremental View Maintenance, IVM）機能

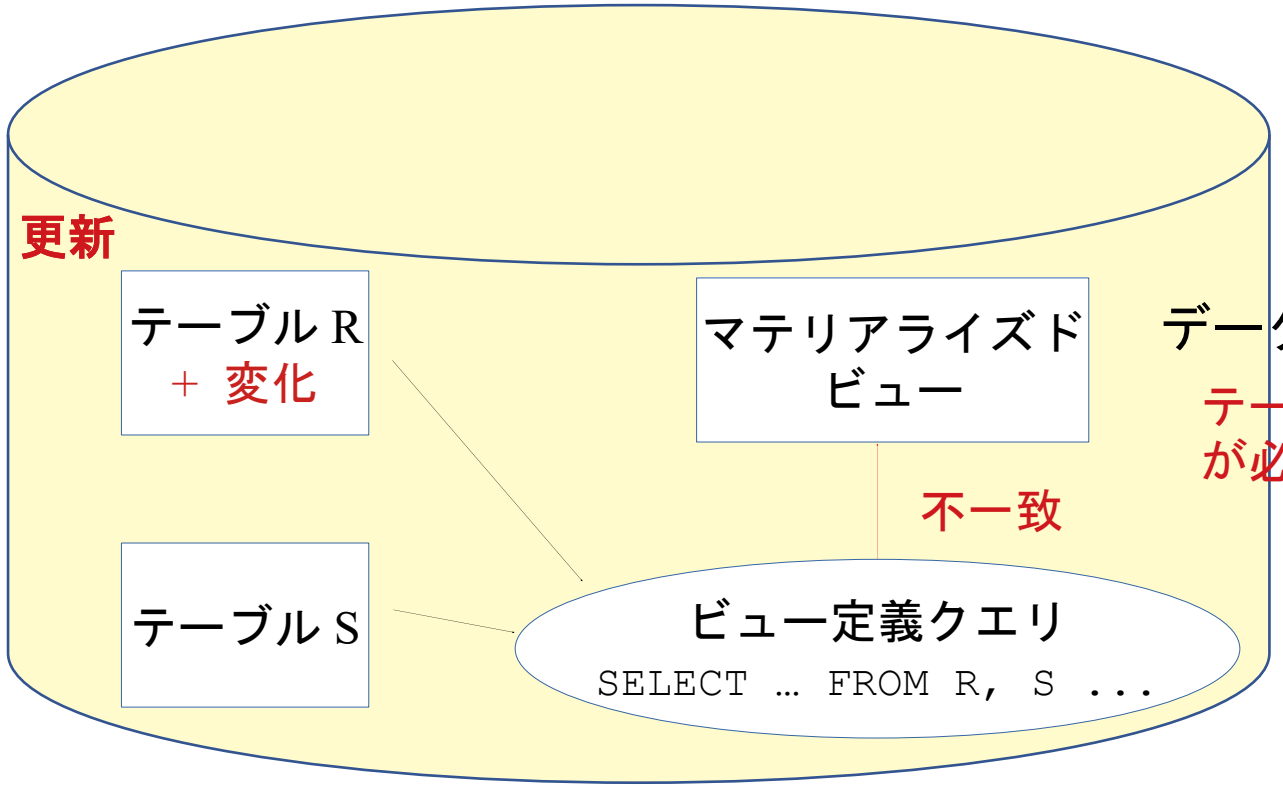
- 内容
  - 増分ビューメンテナンス（IVM）とは
  - pg_ivm について
    - 機能としくみの概要
    - 性能評価と使いどころ
    - 今後の展望

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）



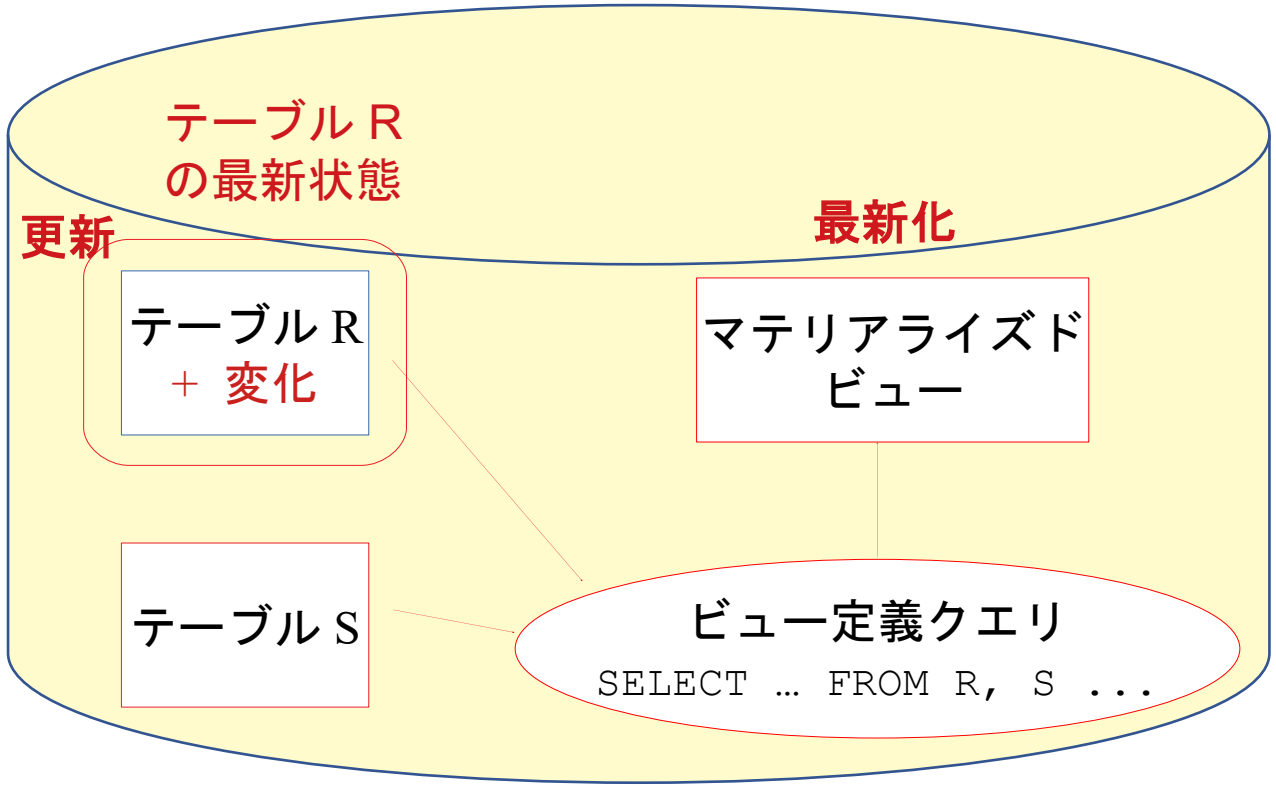
マテリアライズドビューの増分メンテナンス（２）



データを物理的に格納

テーブル更新後にはメンテナンス（最新化）が必要

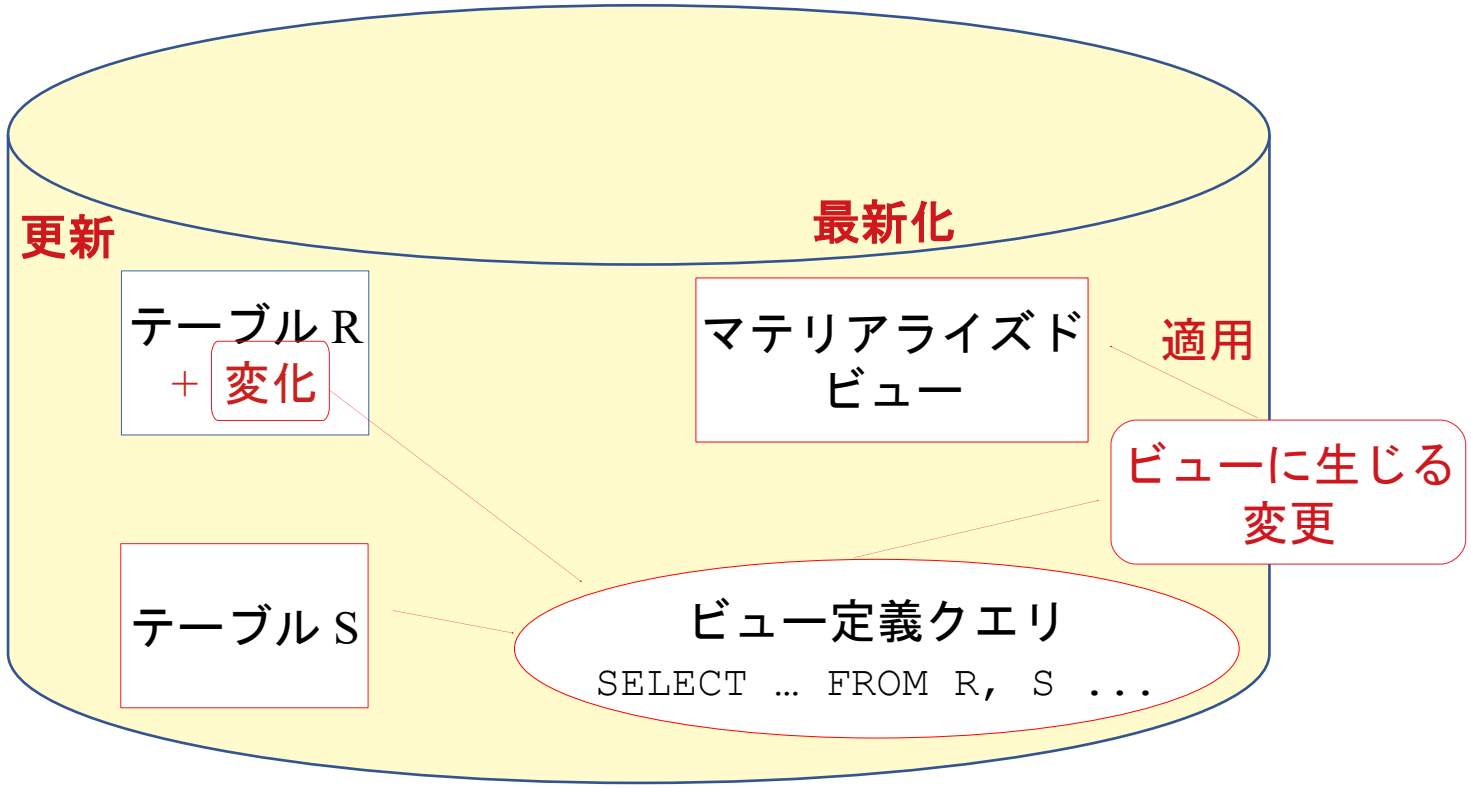
マテリアライズドビューの増分メンテナンス（３）



REFRESH MATERIALIZED VIEW

マテリアライズドビュー全体の再計算 → 時間がかかる

マテリアライズドビューの増分メンテナンス（４）



REFRESH MATERIALIZED VIEW

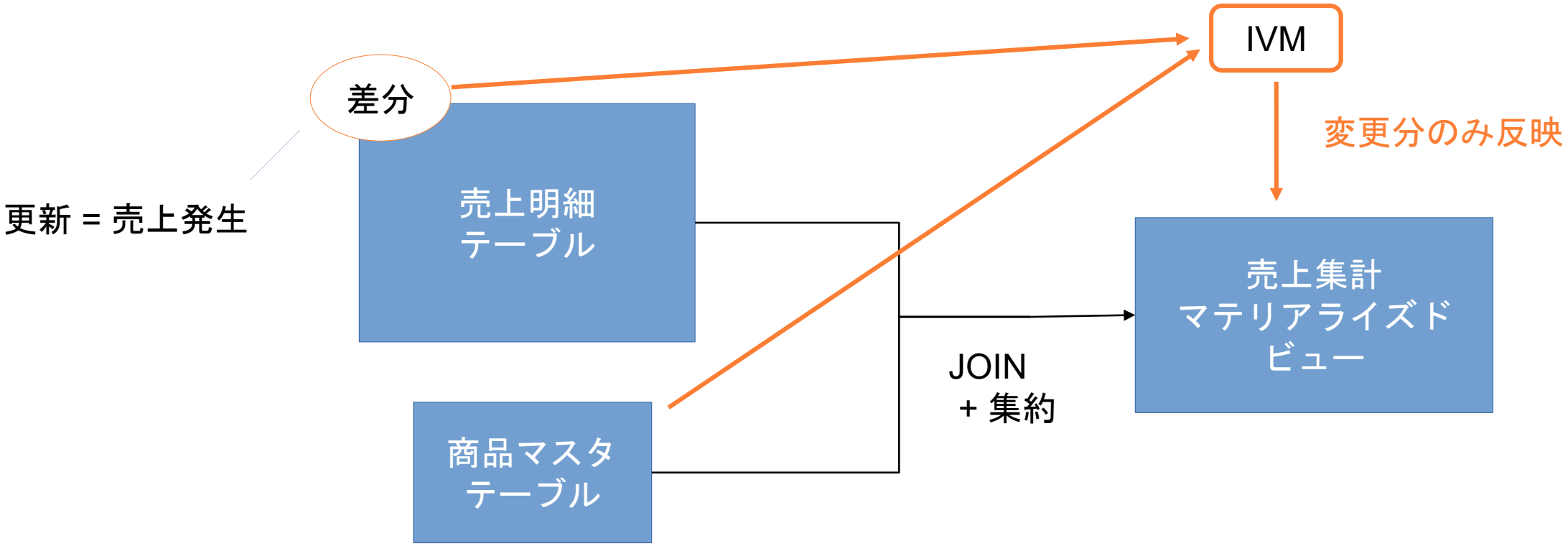
マテリアライズドビュー全体の再計算 → 時間がかかる

増分ビューメンテナンス（IVM）

ビューに生じた差分のみを計算・適用 → 効率的

マテリアライズドビューの増分メンテナンス：具体例

例）大規模な売上データと商品マスタを集約・結合した分析ビュー



pg_ivm の概要

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
  - テーブルを更新すると、ビューが自動的に増分メンテナンスを用いて更新される
  - 通常の REFRESH （クエリの再実行による更新）では 60 秒以上かかるビューが、数ミリ秒で更新可能

- PostgreSQL 13～18 に対応
  - 次期バージョンの PstgreSQL 19 にも対応予定

ビューの作成

- ビューの名前とクエリを指定してcreate_immv 関数を実行する
  - CREATE METERIALIZED VIEW 文の実行に相当

```
test=# SELECT pgivm.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"
```

↑可能な場合はインデックスが自動生成される

```
 create_mv
-------------
   100000000
(1 row)
```

← ビューの行数

ビューの自動更新

- テーブルを更新するとビューは自動で更新される
  - 所要時間はわずか 2.5 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 60 秒以上かかる

```
test=# UPDATE pgbench_accounts SET abalance = 1234 WHERE aid = 1;
UPDATE 1
Time: 2.499 ms

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

ビュー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
    - クエリの再実行によるビュー更新（≠ IVM）
  - 第２引数で、自動更新の有効/無効を切替可能
    - true: ビューを更新、自動更新が有効になる
    - false: ビューの中身が空になり、自動更新は無効になる

```
test=# SELECT pgivm.refresh_immv('mv', true);
 refresh_immv
--------------
    100000000
(1 row)
```

pg_ivm のしくみ

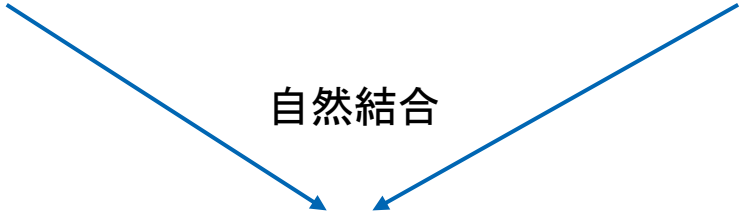
増分ビューメンテナンスの簡単な例（１）

R

number	english
1	one
2	two
3	three

S

number	roman
1	I
2	II
3	III



自然結合

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

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 テーブルRから削除されたタプル

number	english
1	one

ΔR テーブルRに挿入されたタプル

number	english
1	ONE

自然結合

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

自然結合

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

number	english	roman
1	ONE	I

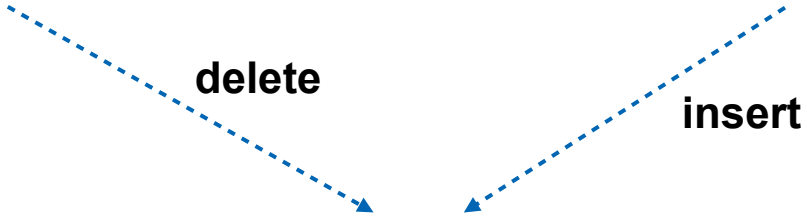
増分ビューメンテナンスの簡単な例（３）

∇V　ビュー V から削除されるタプル

number	english	roman
1	one	I

ΔV　ビュー V に挿入されるタプル

number	english	roman
1	ONE	I

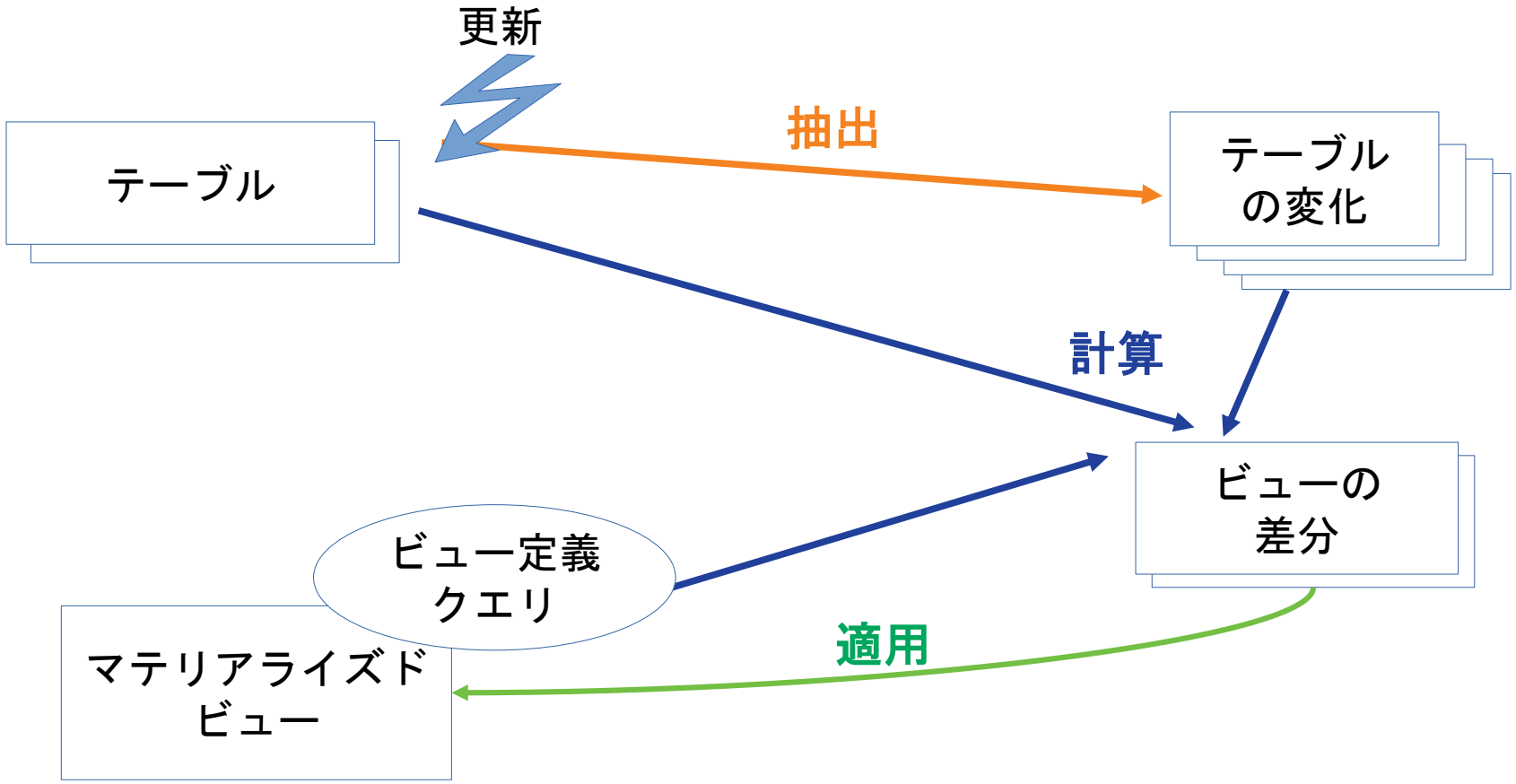


V ← (V − ∇V ∪ ΔV)

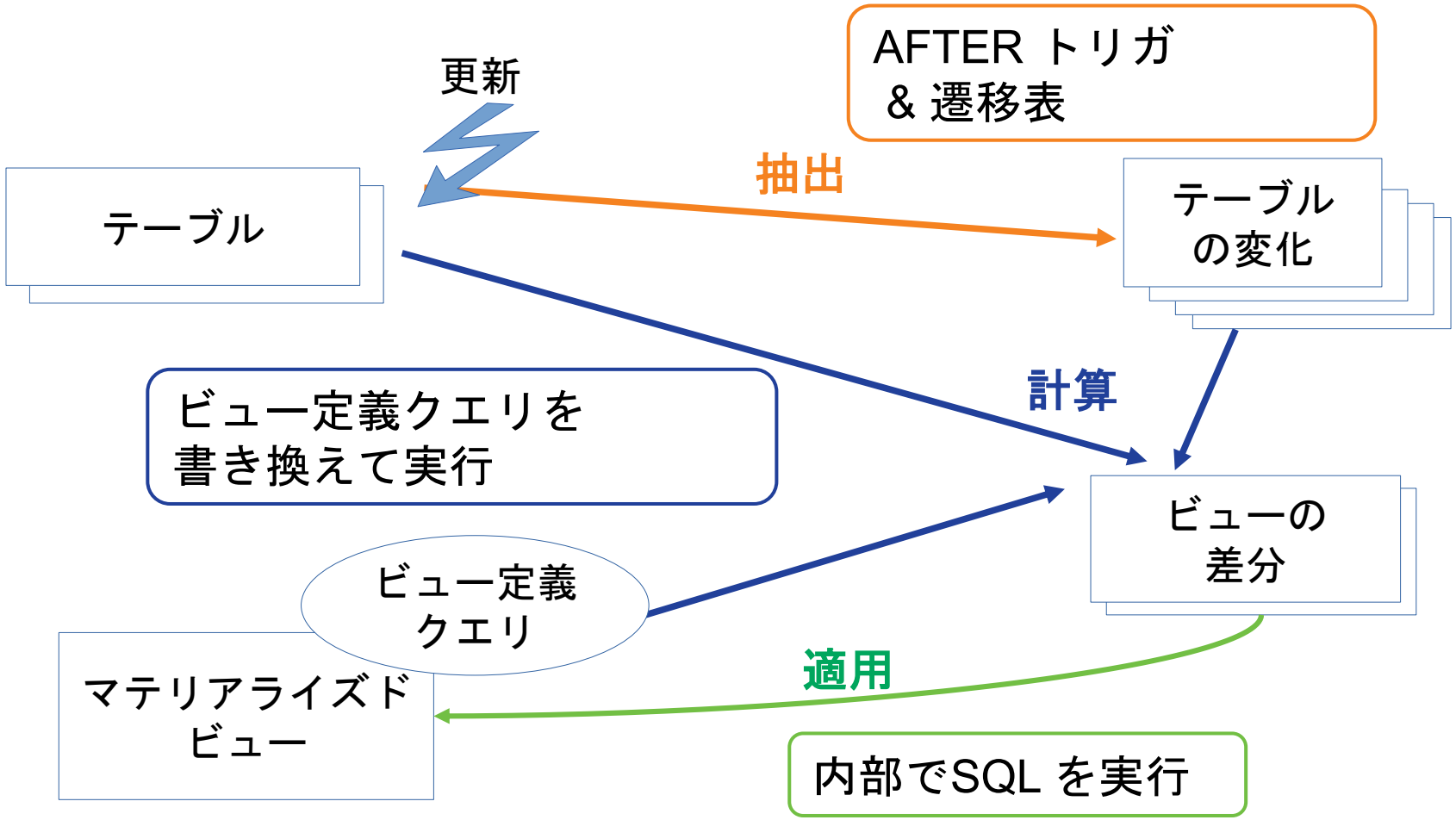
number	english	roman
1	one → ONE	I
2	two	II
3	three	III

計算した差分を適用して、
ビュー Vを更新

ビュー更新の流れ

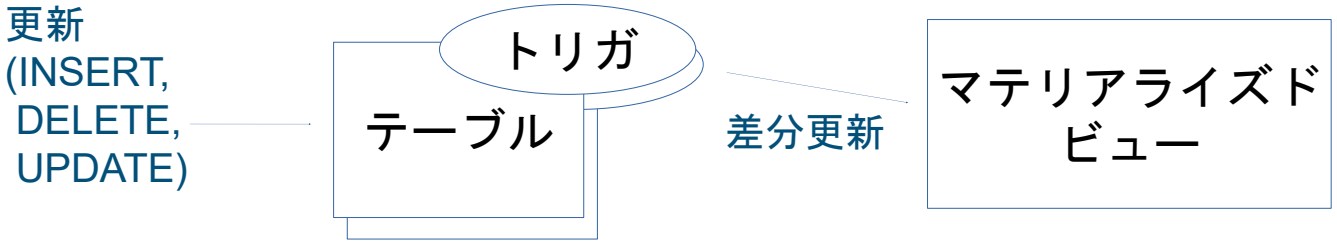


ビュー更新の流れ



テーブル変化の抽出

- ビュー作成時、各テーブルに AFTER トリガを作成
  - INSERT, DELETE, UPDATE の直後に実行される
  - テーブル変化の抽出、ビュー差分の計算、および適用を行う



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

- 「遷移表」でテーブルの変化を抽出
  - 「テーブルに発生した変化」を格納しているテーブル
    - 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

遷移表

∇R テーブルRから削除されたタプル

number	english
1	one

ΔR テーブルRに挿入されたタプル

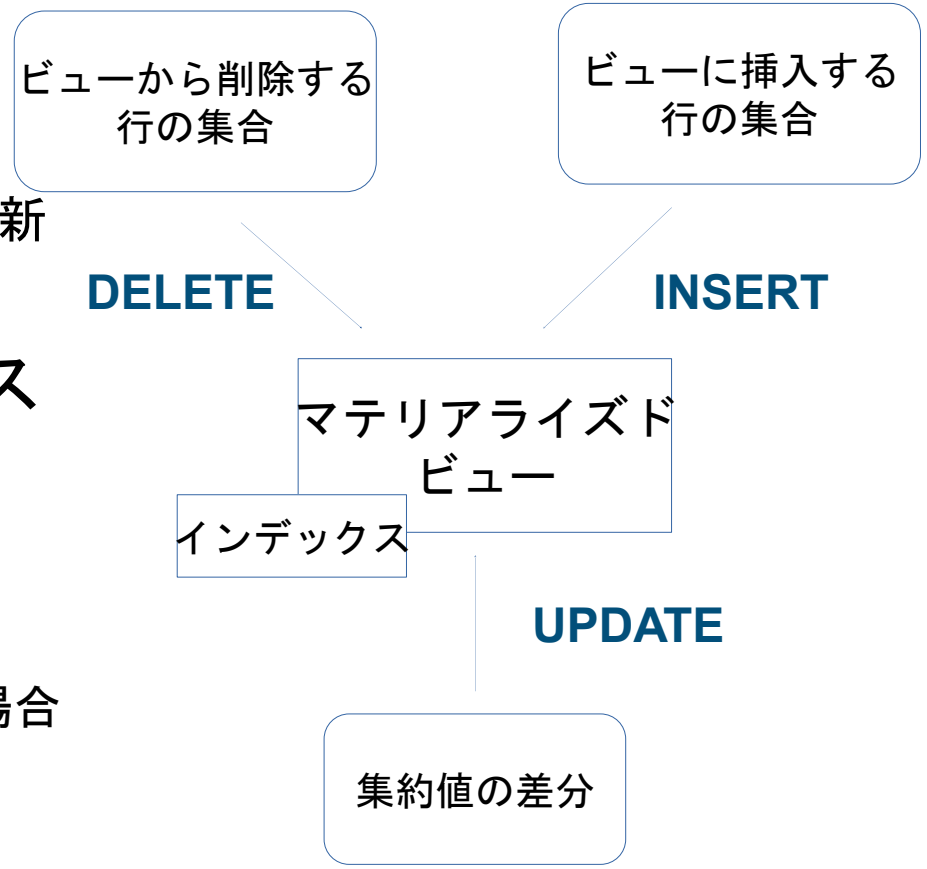
number	english
1	ONE

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

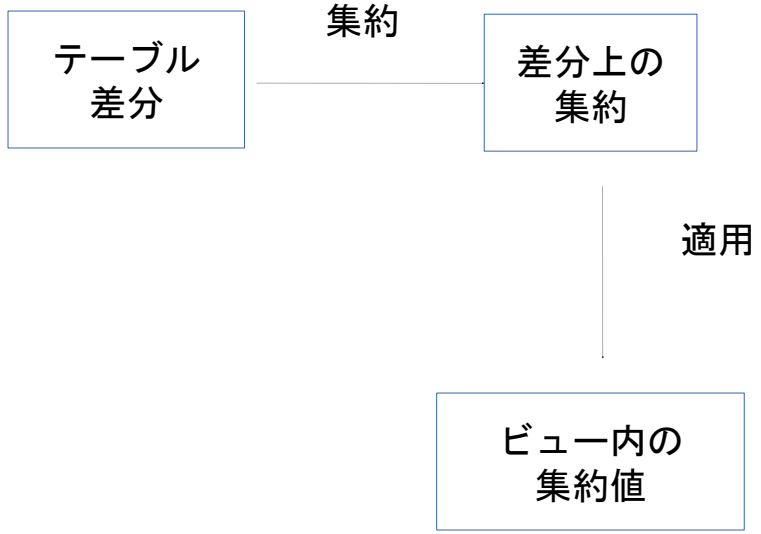
差分をビューに適用

- 内部でSQLを実行
  - DELETE、INSERT を実行して差分を適用
  - 集約を含むビューの場合は、 UPDATE 文で集約値を更新
- マテリアライズドビュー上のインデックス
  - DELETE や UPDATE 時に、ビュー内の行の検索が発生
    → 適切なインデックスが必要
  - 可能な場合には自動で作成
    - 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」に対して集約を実行
- その結果を使ってビュー内の集約値を更新
- SUM, COUNT, AVG, MIN, MAX に対応
  - 例）
    - count(x) ← count(x) ± [差分上の count(x)]
    - sum(x) ← sum(x) ± [差分上の sum(x)]



- 必要な補助情報もビューに保持
  - 例）ビュー全体および各グループの中に含まれる行数

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 = 10
（lineitem: 6000万行程度）

- 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： 1ms 未満
    - REFRESH: 20秒程度
  - pg_ivm を使った場合：
    - lineitem テーブルを１行更新
      ＆ビューの自動更新： 4ms 程度

ビューの更新性能は約5000倍

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
     ...（略）...
    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');
```

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 4.081 ms
```

CPU: Intel Core i7-1370P (14 cores, 20 threads)
Memory: 32GB LPDDR4X
Storage: SSD
OS: Ubuntu 24.04.1, linux kernel 6.17.0-35--generic
PostgreSQL 18.6 + pg_ivm 1.15（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 = 10
（lineitem: 6000万行程度）

- 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： 1ms 未満
    - REFRESH: 14秒程度
  - pg_ivm を使った場合：
    - lineitem テーブルを１行更新 ＆ ビューの自動更新： 7ms 程度

ビューの更新性能は約2000倍

```
tpch=# UPDATE lineitem
       SET l_quantity = l_quantity * 2
       WHERE (l_orderkey, l_linenumber)
                          = (3653,1);
UPDATE 1
Time: 7.420 ms
```

```
SELECT
    nation,
    o_year,
    sum(amount) as sum_profit
FROM
     (
        SELECT
        ... (略) ...
        FROM
            part,supplier,lineitem,partsupp,orders,nation
        WHERE
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            ... (略) ...
        ) AS profit
GROUP BY
    nation,
    o_year
```

TPC-H クエリを用いた性能評価：まとめ

クエリ	計測項目	通常のマテリアライドビュー	pg_ivm	ビュー更新の高速化
Q01 (集約)	テーブル更新所要時間 (lineitemテーブルを１行更新)	< 1 ms	4 ms (ビューの自動更新含む)	約5000倍
	REFRESH MATERIALIZED VIEW （クエリ再実行）	20,000 ms	*（必要なし）*	
Q09 (結合 + 集約)	テーブル更新所要時間 (lineitemテーブルを１行更新)	< 1 ms	7 ms (ビューの自動更新含む)	約2000倍
	REFRESH MATERIALIZED VIEW （クエリ再実行）	14,000 ms	*（必要なし）*	

性能のトレードオフ

- 更新のオーバヘッド
  - テーブル更新ごとにビューメンテナンスを実行
  - オーバヘッドの程度はクエリの複雑さやデータのサイズによって異なる
    （前ページの例では、数ms 程度）

- 排他ロック
  - 複数トランザクションが同時にビューを更新する際の不整合防止
  - 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

pg_ivm が活用できる場面（１）

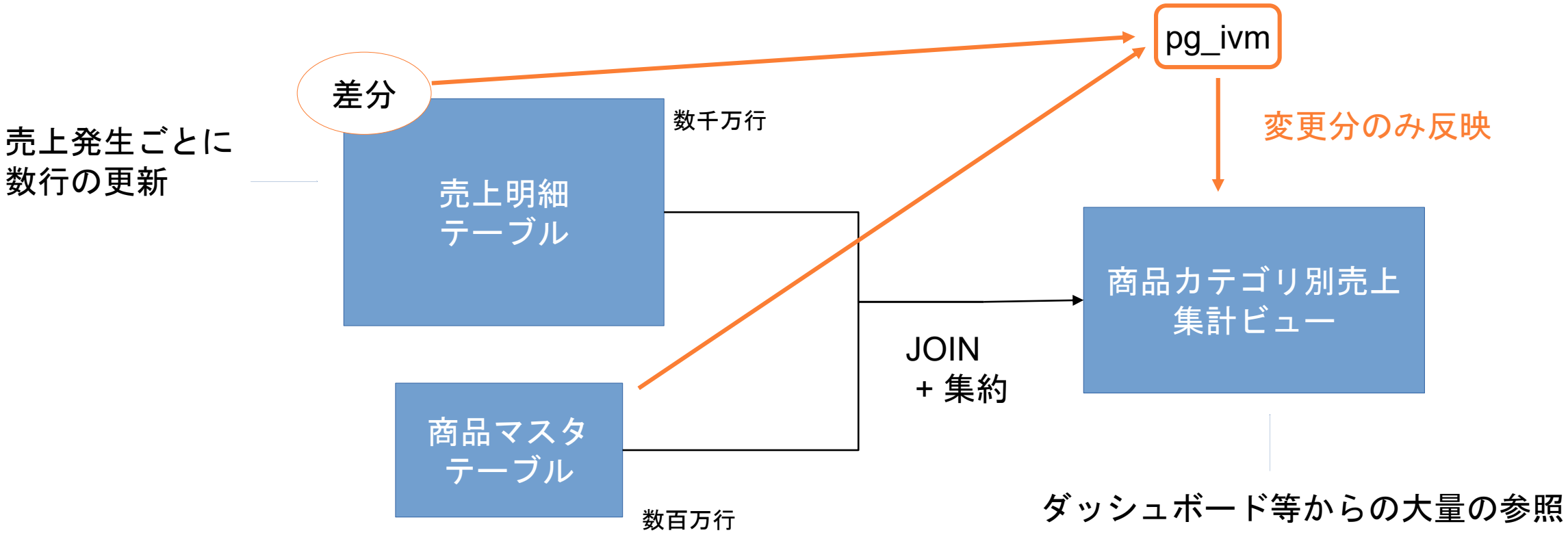
- ビューでの「最新データの取得」が重要なケース
  - 書き込み性能よりもデータの最新性が重視される
  - 大規模テーブルに対する集約や結合
  - 少数の更新に対して多数の参照が発生する

- 大量データのロードが必要な場合には、一時的にビューの自動更新を無効にするのが良い

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

pg_ivm が活用できる場面（２）

例）大規模な売上データと商品マスタを集約・結合した分析ビュー



他にも…

- 利用ログ + ユーザマスタ → 顧客別の利用状況
- センセデータ + 機器マスタ → 拠点/機器種別の最新集計、など

pg_ivm の開発

pg_ivm の開発

- 開発元：IVM Development Group
- ライセンス：PostgreSQL License
- GitHub：https://github.com/sraoss/pg_ivm
  - 2022年4月 の最初のリリースから継続的に開発
  - 1.5k GitHub Stars
- 最近のリリース
  - PostgreSQL 18 対応（pg_ivm 1.12 (2025-09-04)）
  - 外部結合ビュー対応（pg_ivm 1.13（2025-10-20））
  - データのダンプ／リストア、PostgreSQL のアップデートへの対応（pg_ivm 1.15 (2026-06-30)）

外部結合サポート（pg_ivm 1.13～）

- Mediapart（フランスの報道機関）のスポンサーシップを受けて開発が進められた
- 課題：
  - 結合条件を満たさないときに発生するNULL 拡張されたタプル(= ダングリングタプル)
  - これを正しく扱う必要がある

```
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)
```

ダングリングタプル

外部結合ビューの増分メンテナンス

- ダングリングタプルの扱い
  - テーブルへのタプル挿入 → ダングリングタプル消失
  - テーブルからのタプル削除 → ダングリングタプル生成

```
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)
```

削除

新しく挿入

```
           2 | Customer#02    |            |
```

外部結合ビュー：現在の制約

- 主な制約
  - ビューに結合に使われているすべてのカラムが含まれている必要がある
  - 単純な等価結合のみサポート(e.g., A.id = B.id).
- 現時点で非サポート
  - 集約、サブクエリ
  - SELECT ターゲットリスト内での一部の関数使用
    - NULL 入力に対してNULL以外を返すような（strict でない）関数
  - 複雑な WHERE 句
    - OR や、NULL が含まれるときに false とならない（IS NULL のような）関数

データのダンプ／リストア、 PostgreSQL アップグレード対応 (pg_ivm 1.15〜)

- pg_ivm 1.14 以前：
  - pg_dump でダンプされたデータのリストア
  - pg_upgrade による PostgreSQL バージョンのアップグレード

    → pg_ivm で作成したビューの「メタデータ」が不全のため、最初から作り直す必要があった

- pg_ivm 1.15 以降：
  - ビューの「メタデータ」の再作成が可能に

→ ビューの作り直しが不必要に

pg_ivm 開発の今後の予定

- PostgreSQL 19 対応
- 遅延メンテナンス対応
- パーティションテーブルや、論理レプリケーションへの対応
- 外部結合ビューの制限緩和

今後の開発テーマ: 遅延メンテナンス

- 現在サポートしているのは「即時メンテナンス」のみ
  - テーブルが更新される毎にビューが自動的に更新される
    → テーブル更新性能への影響が大きい

- 遅延メンテナンス（検討中）
  - テーブル更新とは別のタイミング増分ビューメンテナンスを行う方法
    - コマンドの実行や、バックグラウンドによる定期処理など
  - テーブル更新性能への影響を抑えられることが期待される

まとめ

- pg_ivm は PostgreSQL に増分ビューメンテナンス機能を提供する拡張モジュール
- ビュー差分のみを計算することで、マテリアライズドビューを高速に最新化できる
- 最新性が重要で、集約・結合結果への参照が多いワークロードで特に有効
- 今後は遅延メンテナンスなどの機能拡張を検討している

SRA OSS 技術情報

- SRA OSS Tech Blog
  - https://www.sraoss.co.jp/tech-blog/
  - PostgreSQLのリリースノートの日本語訳など様々なOSS技術情報を掲載中
- SRA OSS 公式 Youtube チャンネル
  - https://www.youtube.com/c/sraoss-official
  - 過去のセミナー動画を公開中