SRA の技術情報誌
GS LetterNeo
Vol.173　2022 年 12 月

PostgreSQL のマテリアライズドビューを高速に最新化する

長田 悠吾

SRA OSS LLC OSS 事業本部 技術開発室

（株式会社 SRA ホールディングス 先端技術研究所）

はじめに

PostgreSQL は広く使用されているオープンソース・ソフトウェアの RDBMS です。SRA OSS LLC では PostgreSQL に関するサポートやコンサルティングのサービスを提供する一方、技術開発室という部署では PostgreSQL 開発コミュニティへの貢献や PostgreSQL に関する研究開発といった活動を行っています。本稿ではその中から「増分ビューメンテナンス（Incremental View Maintenance, IVM）」と呼ばれる機能の実装について紹介します。これは一言で言うならば、マテリアライズド・ビューの最新化を高速に行う技術です。

マテリアライズド・ビュー

マテリアライズド・ビューとは、そのビューを定義する SQL クエリ（SELECT 文）の実行結果をデータベース内に保存しているようなビューです。PostgreSQL では例えば以下のようなコマンドでマテリアライズド・ビューを作成することができます。

```
CREATE MATERIALIZED VIEW myview AS
 SELECT * FROM weather JOIN cities ON city = name;
```

通常のビューはアクセスされる度に定義クエリの実行を行いますが、マテリアライズド・ビューへのアクセスは保存されている実行結果をクライアントに返すだけなので高速な応答が可能です。その一方で、ビュー定義に使われているテーブルが更新された場合にはマテリアライズド・ビューの内容は古くなり、実際にビュー定義クエリを実行した結果と一致しなくなってしまいます。そのため、テーブルが更新された後にはマテリアライズド・ビューの内容を最新の状態に更新する必要が出てきます。この操作はビューのメンテナンスと呼ばれています。

増分ビューメンテナンスとは

ビューをメンテナンスする最も単純な方法は、マテリアライズド・ビューの内容を「再計算」することです。例えば、PostgreSQL では以下のコマンドを実行するとビュー定義クエリを再実行し、その結果でマテリアライズド・ビューの古い内容を置き換えることができます。

```
REFRESH MATERIALIZED VIEW myview;
```

しかし、この方法ではマテリアライズド・ビューの全てのデータを再生成することになるため、あまり効率的とは言えません。テーブルの更新がごく一部の場合には、マテリアライズド・ビューの全体ではなく一部分を更新するだけで最新の状態を実現した方が効率がよいはずです。このようなビューのメンテンスの方法は、「増分ビューメンテナンス（Incremental View Maintenance, IVM）」と呼ばれています。

PostgreSQL における増分ビューメンテンス機能の実装

増分ビューメンテナンス機能は強力な機能ですが、現在の PostgreSQL には実装されていません。そこで、著者らは PostgreSQL にこの機能を実装して開発コミュニティに提案しているところです。

この提案中の機能を使うと、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 と呼んでいます）を作成することができます。このように作られたマテリアライズド・ビューの内容は、テーブルが更新される度に自動的に増分メンテナンスされるため、常に最新のテーブルの内容を反映した状態になっています。実際のビューのメンテナンスにかかる時間はビュー定義やテーブル更新の内容にもよりますが、例えば REFRESH MATERIALIZED VIEW コマンドの実行に 20 秒程要したビューが、増分メンテナンスを使った場合にはテーブルの更新時間込みで 15 ミリ秒程度で最新化できます。

PostgreSQL への増分ビューメンテナンス機能の提案は pgsql-hackers[1]というメーリングリストで議論中です。ここで行われる設計や実装の議論やバグの報告などを通してコードの修正や改善を進めています。現在提案中の機能では、結合と一部の組み込み集約関数（count, sum, avg, min, max）を含む SELECT 文がビュー定義の中で使用可能です。コードの修正量が大きくなりすぎないよう現在の提案には含めてはいませんが、外部結合やサブクエリへの対応についても実装済です。また、これらの成果は PostgreSQL コミュニティの海外カンファレンス[2][3][4][5][6]および国内の学会発表[7][8]でも報告しています。

pg_ivm: 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

前節では PostgreSQL 本体の新機能として増分ビューメンテナンス機能を提案していることを紹介しましたが、提案の過程で「増分ビューメンテナンス機能を既存の PostgreSQL でも使用したい」という要望を受けることがありました。そこで、この機能をリリース済の PostgreSQL でも使用できるように拡張モジュールとして開発したのが pg_ivm[9]です。現状では、PostgreSQL 13, 14, そして最新の 15 に対応しています。

pg_ivm はさらに、PostgreSQL における増分ビューメンテナンス機能に対するフィードバックや認知を得る機会の向上、および、PostgreSQL 本体への提案には含まれていない追加的な機能の提供も目的としています。実際に、最新の pg_ivm 1.3 では本体に提案している機能では対応していないサブクエリに部分的に対応しています。

pg_ivm の使用例

pg_ivm は GitHub レポジトリ[9]からソースコードを入手してインストールすることもできますし、PostgreSQL コミュニティで提供されているレポジトリ[10]で提供されている rpm ファイルを利用して yum や dnf コマンドでインストールすることもできます。

PostgreSQL で pg_ivm を有効にするにはまず CREATE EXTENSION コマンドを実行します。

```
CREATE EXTENSION pg_ivm;
```

pg_ivm で増分メンテナンス可能なマテリアライズド・ビューを作成するには、create_immv という関数を実行します。これは CREATE MATERIALIZED VIEW コマンドに相当する関数で、第 1 引数にビューの名前と列のリスト、第 2 引数にビューの定義を指定します。実行結果として生成されたビューの行数（以下の例では 10,000,000 行）が返されます。

```
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)
```

このビューの定義で使われているテーブルの 1 行を更新してみます。著者の環境では、その所要時間は 15.448 ミリ秒でした。その後にビューの内容を確認してみると、更新されたテーブルの内容が反映されていることがわかります。これと同じ定義の通常のマテリアライズド・ビューを作成して REFRESH MATERIALIZED VIEW コマンドを実行した場合には 20 秒以上かかりました。これと比較すると、今回の例では増分メンテナンスが効果的に働いていることがわかると思います。

```
test=# UPDATE pgbench_accounts SET abalance = 1234 WHERE aid = 1;
UPDATE 1
Time: 15.448 mstoeic  "ubuntu"  ip windows
test=# SELECT * FROM mv WHERE aid = 1;
 aid | bid | abalance | bbalance
-----+-----+----------+----------
   1 |   1 |     1234 |        0
(1 row)
```

pg_ivm の性能

増分メンテナンスの性能を示す別の例として、業界標準のベンチマークである TPC-H のクエリを用いた性能評価を紹介します。TPC-H のクエリの 1 つ Q01 は 1 つの巨大なテーブルに対する集約です。

```
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;
```

このクエリで定義される通常のマテリアライズド・ビューを REFRESH MATERIALIZED VIEW コマンドで最新化した場合、筆者の環境では 10 秒以上かかりました。なお、TPC-H のスケールファクタ は 1（lineitem テーブルの行数は 600 万行程度）を使用しました。一方、pg_ivm を使って増分メンテナンス可能なマテリアライズドビューを作って、lineitem テーブルを 1 行更新した場合の所要時間は 20 ミリ秒程度でした。

別のクエリ Q09 は 6 つのテーブルを結合して集約したクエリです。

```
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;
```

このクエリの場合、通常のマテリアライズド・ビューの REFRESH MATERIALIZED VIEW コマンド実行は 3 秒程度かかりましたが、pg_ivm を使用した場合の lineitem テーブルの 1 行更新の所要時間は 45 ミリ秒程度でした。

以上のように、増分メンテナンスに要する時間はビュー定義など状況によって異なりますが、増分メンテナンスが REFRESH MATERIALIZED VIEW によるクエリの再実行より効果的であることがわかります。

一方で、現在の機能ではテーブルの更新の度にビューのメンテナンスが発生するため、テーブル更新の性能にはオーバヘッドがあります。実際に上の例では、増分ビューメンテナンス機能を使用しない場合は lineitem の 1 行更新は 10 ミリ秒ほどで完了していますが、増分ビューメンテナンスを行う場合には Q01 の場合で 20 ミリ秒、Q09 の場合で 45 ミリ秒とかかっています。その他にも、データ不整合の発生を防止するため、複数トランザクションが同時にビューをメンテナンスしようとする場合には排他処理を行うなど、同時実行性能にも影

響があります。これらの性質から、PostgreSQL に現在提案中の増分ビューメンテナンス機能、および pg_ivm は「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」という状況に有用なものと言えます。

pg_ivm については先月開催された PostgreSQL Conference Japan 2022 でも紹介しました[11]。講演資料も公開されていますのでぜひ参照してください。

今後の予定

PostgreSQL 本体への増分ビューメンテナンス実装は開発コミュニティに提案中の機能です。PostgreSQL への機能追加はコミュニティでの議論を経てコンセンサスを得る必要があるため時間がかかるのですが、これからも引き続き本体への採用を目指して議論と改善を重ねていきます。

pg_ivm はこれまでに何回かのリリースで、ビュー定義として対応できるクエリの種類の拡充やバグ修正を行ってきました。今後も引き続き EXISTS サブクエリ、CTE（WITH 句）、外部結合といったクエリに対応していくと共に、バグ修正などの改善を続けていきます。また、さらに進んだ機能も追加していきたいと考えており、その中の 1 つが「遅延メンテナンス」です。現在の提供しているのはビューのメンテナンスをテーブルの更新と同時に行う「即時メンテナンス」のみであるため、テーブルの更新性能への影響が目立ちます。これに対して遅延メンテナンスはテーブル更新とは別のタイミング（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）で増分ビューメンテナンスを行う方法で、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ます。その他にも性能全般の改善やパーティション・テーブルなどの特殊なテーブルへの対応なども考えています。

おわりに

本稿では PostgreSQL における増分ビューメンテナンス機能の提案と、拡張モジュール pg_ivm について紹介しました。pg_ivm は GitHub レポジトリ[9]、PostgreSQL Yum レポジトリ[10]から入手可能な他、最近販売開始されました PowerGres Plus V13[12]に同梱されています。Linux 版の提供はもちろん、Windows 版にも Windows で使用可能なビルド済バイナリが含まれています。評価版もありますので、是非試してみてください。

また、本プロジェクトはオープンソースプロジェクトです。開発コミュニティでの議論はもちろん、GitHub への Issue, Pull Request の投稿なども大歓迎ですので、何かしらのフィードバックを寄せて頂けましたら幸いです。

参考文献・リンク等

[1] pgsql-hackers: https://www.postgresql.org/list/pgsql-hackers/

[2] Yugo Nagata, "Toward Implementing Incremental View Maintenance on PostgreSQL", PGCon 2019.

[3] Yugo Nagata, "Toward Implementing Incremental View Maintenance on PostgreSQL", PGConf.Asia 2019.

[4] Yugo Nagata, Takuma Hoshiai, "The Way for Updating Materialized Views Rapidly", PGCon 2020.

[5] Yugo Nagata, "The Way for Updating Materialized Views Rapidly", PostgresConf.CN & PGConf.Asia 2020.

[6] Yugo Nagata, "Updating Materialized Views Automatically and Incrementally", PGConf.Online 2021/PGConf.Russia 2021.

[7] 長田悠吾, 星合拓馬, 石井達夫, 増永良文, "タプル重複のもとで外部結合および準結合を含むビューの増分メンテナンスとその PostgreSQL への実装," 第 12 回データ工学と情報マネジメントに関するフォーラム (DEIM Forum), 2020.

[8] 長田悠吾, 星合拓馬, 石井達夫, 三島健, 増永良文, "PostgreSQL におけるビューの増分メンテナンス機能の実装とその評価," 第 13 回データ工学と情報マネジメントに関するフォーラム (DEIM Forum), 2021.

[9] pg_ivm: https://github.com/sraoss/pg_ivm

[10] PostgreSQL Yum Repository: https://yum.postgresql.org/

[11] 長田悠吾, "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. (https://www.postgresql.jp/jpug-pgcon2022#A3)

[12] PowerGres Plus V13: https://powergres.sraoss.co.jp/product/plusv13/

GSLetterNeo Vol.173
2022 年 12 月 20 日発行
発行者　株式会社 SRA 技術本部 先端技術研究室

編集者　熊澤努　方 学芳
バックナンバー https://www.sra.co.jp/public/sra/gsletter/
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8

夢を。Yawaraka Innovation
やわらかいのべーしょん