PostgreSQL 17 とその後

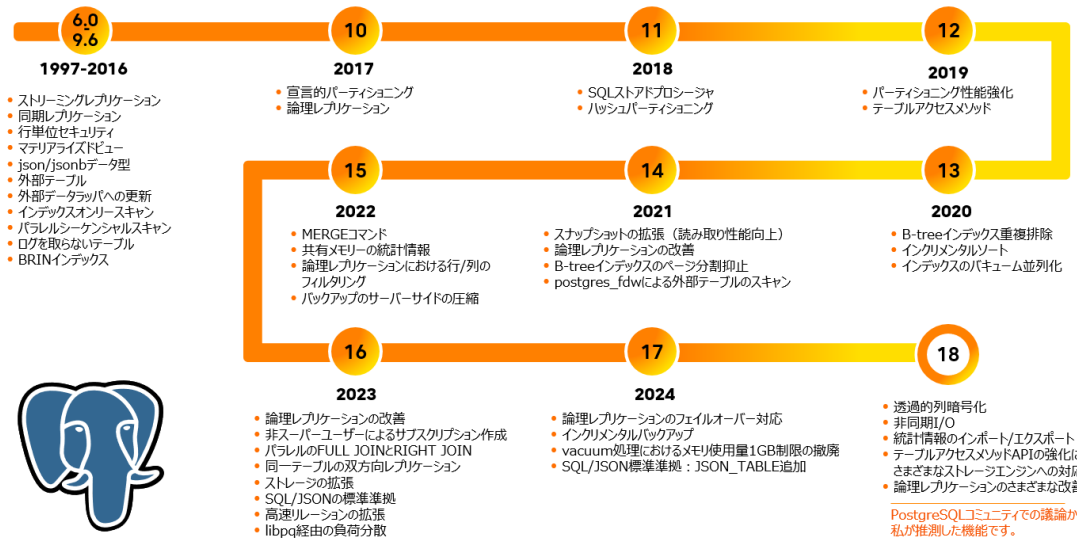
技術者 Blog

Amit Kapila
FUJITSU Limited
Software Products Business Unit Data Management Division Senior Director
PostgreSQL Committer and Major Contributor

PostgreSQL コミュニティは、世界で最も先進的なオープンソースデータベースに数々の新機能と改善を加えた新バージョン PostgreSQL 17 をリリースしました。
これまでのリリースと同様に、PostgreSQL が最も先進的なオープンソースデータベースとして企業に採用され続けるための重要な機能がいくつか含まれています。
それでは、いくつかご紹介しましょう。

PostgreSQL の進化

PostgreSQL 17 に含まれるエキサイティングな新機能について掘り下げる前に、PostgreSQL が長年にわたってどのように進化してきたのか、そして各リリースがどのようにして次のリリースの基盤を築いてきたのかを振り返ってみましょう。



PostgreSQL の進化とその主要なバージョンにおけるマイルストーン

カリフォルニア大学バークレー校のプロジェクトから引き継がれた後、バージョン 6.0 から 9.6 にかけて、PostgreSQL をより大規模で重要なアプリケーションに使用できるように、多くのエンタープライズ機能が追加されました。ストリーミングレプリケーションと同期レプリケーションによるフェイルオーバー、高可用性、JSONB/JSON データ型のサポート、インデックスオンリースキャンの実現、並列処理など、改善された分野は数多くあります。これらすべての機能により、より多くのユーザーの PostgreSQL 採用を促進し、この頃から PostgreSQL が主流のデータベースとなりつつあることがコミュニティで認識され始めました。

勢いを増した PostgreSQL は、バージョン 10 からも進化を続け、今では世界中でエンタープライズクラスのデータベースとして認められるまでに成熟しました。AWS、Microsoft、Google などの主要企業が PostgreSQL を使用し、自社のソリューションやサービスのデータベース層として提供していることは広く知られています。バージョン 10 では、宣言型パーティショニングと論理レプリケーションが 2 つの大きな新機能でした。バージョン 11 の SQL ストアドプロシージャは、Oracle などの他のプロプライエタリなデータベースからの移行を促進しました。バージョン 12 の重要な新機能の 1 つは、PostgreSQL にプラグイン可能なストレージエンジンが独自のストレージニーズに合わせた書き込みを可能にするテーブルアクセスメソッドの導入であると言えるでしょう。これらの 3 つのリリースの間には、パーティショニングの強化が実装されました。バージョン 11 ではハッシュキーによるパーティショニングが導入され、バージョン 12 ではパフォーマンスが向上しました。

そこから、ユーザーからの継続的なフィードバックとコミュニティの熱心な技術者によって、PostgreSQL は使いやすさとパフォーマンスを向上させ、大規模データへの対応を強化しました。バージョン 13 では、並列バキューム、B-tree インデックスの重複排除、インクリメンタルソートがリリースされました。バージョン 14 では、クライアント数が非常に多い場合のスナップショットの拡張と、実行中のトランザクションの更新を適用するメカニズムにより、読み取りパフォーマンスが向上し、大規模なトランザクションの遅延が軽減されました。その他の機能の改善としては、B-tree インデックスのページ分割の抑止と、postgres_fdw による外部テーブル並列スキャンが追加されました。

バージョン 15 では、MERGE コマンドの追加や、論理レプリケーションの様々な改善がありました。また、統計情報はバックエンドから共有メモリに移されました。これは、今後数年でより多くの統計情報を追加するための基盤となりました。さらに、行と列のフィルタリングを導入することで論理レプリケーションが改善され、より高速でより容量の少ないバックアップのためにサーバー側でバックアップを圧縮する機能も追加されました。

バージョン 16 では、スタンバイからプライマリへの論理レプリケーションを可能にすることで、論理レプリケーションにおけるプライマリとスタンバイ間のワークロード分散を実現しました。また、大規模トランザクションの適用遅延を軽減するために、並列適用機能が導入されました。さらに、Sequel JSON 標準が改善され、より高速なリレーション拡張と libpq による負荷分散も開発されました。

機能強化におけるすべてのマイルストーンの達成は、今年の PostgreSQL 17 のリリースにつながりました。そのハイライトは次のとおりです。

- 論理レプリケーションのフェイルオーバー対応
- 増分バックアップ（インクリメンタルバックアップ）で大規模バックアップに対応
- vacuum 処理におけるメモリ使用量 1GB 制限の撤廃
- SQL/JSON 標準への準拠性を向上させる JSON テーブル機能

PostgreSQL 17 の機能強化と新機能

バージョン 17 の機能の詳細を見ていきましょう。

ストレージの拡張

- 増分バックアップ
    - 変更されたブロックのみがコピーされ、バックアップデータサイズを削減します。
    - pg_combinebackup を使用して、ベースバックアップと増分バックアップを組み合わせて完全バックアップを再構築します。
    - WAL を収集するために、summarize_wal という新しいバックグラウンドワーカーの有効化が必要です。
- VACUUM の改善
    - インデックスのクリーンアップを高速化し、メモリ使用量を削減します。
      VACUUM 中にデッドタプルを削除する仕組みを変更して、TID ストアを使用してヒープからデッドタプルを収集し、それらを使用してインデックスをクリーンアップすることで実現します。
    - WAL の同期と書き込み時間を短縮します。
      VACUUM プロセスにおけるフリーズとプルーニングのステップを組み合わせることで、単一の WAL レコードを作成します。
    - WAL ボリュームを削減します。
      インデックスを持たないリレーションの VACUUM 処理は、LP_UNUSED を直接マークすることで手順を削減し、最適化されます。
    - ストリーミング API を使用することで読み込みを高速化します。

- SLRU（simple least-recently-used）の上に構築されたサブシステムのパフォーマンスが向上します。
  SLRU キャッシュサイズが設定可能になり、サブキャッシュチャンクはバンクと呼ばれる単位に分割されるため、エビクションバッファ検索と LWLock の影響を特定のバンクに限定します。
- テーブルアクセスメソッドがヒープからブロックのフェッチをスキップできます。
- B-tree インデックスを使用した IN/ANY 句を使用するクエリの実行時間を改善します。
  B-tree インデックスにおける配列のマッチングを最適化します。
- 競合の多い WAL 書き込み（クライアント数 256 以上）のパフォーマンスが向上します。

論理レプリケーションの改善

- 論理レプリケーションがフェイルオーバーに対応します。
  プライマリ/パブリッシャーノードがダウンした場合でも、手動での介入なしに物理スタンバイからデータを取得することで、サブスクリプションを継続できます。

```
CREATE SUBSCRIPTION sub CONNECTION '$connstr'
PUBLICATION pub WITH (failover = 'true')
```

- 論理レプリケーションノードをアップグレードします。
  古いバージョンが 17 の場合、メジャーバージョンアップグレード後も論理レプリケーションの設定を保持します。
- サブトランザクション数が多い場合のロジカルデコーディングを高速化します。
- プライマリキーまたはレプリカアイデンティティがサブスクライバーで利用できない場合、適用中の検索にハッシュインデックスが使用できます。
- サブスクリプションのレプリケーション設定を簡易化します。
  新しいツール pg_createsubscriber により、すべてのテーブルを物理スタンバイからコピー可能となります。私の同僚である黒田隼人がこの機能についてブログを公開しています。
  - PGConf.dev 2024 参加レポート – PostgreSQL 17 での論理レプリケーションの新機能

論理レプリケーションをエンタープライズ対応するための今後の方向性としては、私の理解では、2 つの主要な要素があります。具体的なコンフリクト検出と解決の仕組みと、クラスタ全体で一貫性のあるシーケンスです。その他の取り組むべき機能には、DDL レプリケーション機能と、ノード管理 API の提供などがあります。これはエキサイティングな分野であり、将来のリリースではこの方向に進んでいくことを期待しています。

SQL の改善

- パーティショニング
  - パーティションテーブルで IDENTITY 列をサポートします。
  - パーティションテーブルで排他制約を使用できます。
    排他制約がパーティション・キーの列が等価比較する場合に限り、他の列は排他制約特有の比較を使用できます。

```
CREATE TABLE idxpart (a int4range, b int4range, c int4range,
          EXCLUDE USING GIST (b with =, c with &&)) PARTITION BY RANGE (a);
ERROR: unique constraint on partitioned table must include all partitioning columns,
DETAIL: EXCLUDE constraint on table "idxpart" lacks column "a" which is part of the partition key.
```



```
CREATE TABLE idxpart (a int4range, b int4range, c int4range,
          EXCLUDE USING GIST (a with =, b with =, c with &&)) PARTITION BY RANGE (a, b);
```

- BRIN インデックスを構築するために複数ワーカーを使用します。
- initPlan を生成するクエリは、initPlan を実行するために並列ワーカーを使用できます。

```
EXPLAIN (COSTS OFF) SELECT c1 FROM t1 WHERE c1 = (SELECT 1);
              QUERY PLAN
----------------------------------------
  Gather
   Workers Planned: 2
   InitPlan 1
    -> Result
    -> Parallel Seq Scan on t1
        Filter: (c1 = (InitPlan 1).col1
```

- NOT NULL 列に対する IS NOT NULL クエリ制約を削減します。

```
CREATE TABLE pred_tab (a int NOT NULL, b int, c int NOT NULL);
EXPLAIN (COSTS OFF) SELECT * FROM pred_tab t WHERE t.a IS NOT NULL;
        QUERY PLAN
  ------------------------
   Seq Scan on pred_tab t
```

- IS NULL が指定されている場合、NOT NULL 列のスキャンを削減します。

```
EXPLAIN (COSTS OFF) SELECT * FROM pred_tab t WHERE t.a IS NULL;
        QUERY PLAN
  --------------------------
   Result
      One-Time Filter: false
```

- 相関する IN 句サブクエリを結合に変換できます。

```
EXPLAIN (costs off) SELECT * from tenk1 A WHERE hundred in
(select hundred from tenk2 B where B.odd = A.odd);
                      QUERY PLAN
  ------------------------------------------------------------
  Hash Join
    Hash Cond: ((a.odd = b.odd) AND (a.hundred = b.hundred))
    -> Seq Scan on tenk1 a
    -> Hash
        -> HashAggregate
              Group Key: b.odd, b.hundred
              -> Seq Scan on tenk2 b
```


- CTE（Common Table Expression　（共通テーブル式））プランの改善による、統計情報と出力例のソート順序が考慮されます。
- EXISTS と IN のサブクエリを postgres_fdw の外部サーバーにプッシュダウンできるようになりました。

```
EXPLAIN (VERBOSE, COSTS OFF) SELECT t1.c1 FROM ft1 t1 WHERE EXISTS (SELECT 1 FROM ft2 t2 WHERE t1.c1 = t2.c1)
                          ORDER BY t1.c1 OFFSET 100 LIMIT 10;
Foreign Scan
  Output: t1.c1
  Relations: (public.ft1 t1) SEMI JOIN (public.ft2 t2)
  Remote SQL: SELECT r1."C 1" FROM "S 1"."T 1" r1 WHERE EXISTS (SELECT NULL FROM "S 1"."T 1" r2
              WHERE ((r2."C 1" = r1."C 1"))) ORDER BY r1."C 1" ASC NULLS LAST LIMIT 10::bigint OFFSET 100::bigint
```

- 結合条件以外の条件を持つ結合を、外部サーバーへのプッシュダウン、およびカスタムスキャンをできるようになりました。
- MERGE コマンド機能
    - MERGE コマンドは、RETURNING 句をサポートします。

```
MERGE INTO products p USING stock s ON p.product_id = s.product_id
    WHEN MATCHED AND s.quantity > 0 THEN UPDATE SET in_stock = true, quantity = s.quantity
    WHEN NOT MATCHED THEN INSERT (product_id, in_stock, quantity) VALUES (s.product_id, true, s.quantity)
    RETURNING merge_action(), p.*;
Xmerge_action | product_id | in_stock | quantity
--------------+------------+----------+----------
UPDATE        |    1001    | t        |       50
INSERT        |    1003    | t        |       10
```

    - MERGE コマンドは、WHEN NOT MATCHED BY SOURCE をサポートします。
    - MERGE コマンドは、更新可能なビューを変更できます。
- login イベントにトリガーを導入し、ユーザー接続時にすぐにいくつかのアクションを実行できるようにしました。
- 並列集計のシリアル部分を高速化し、並列クエリにおける次の項目をより適切にスケールします。

  `sum(numeric)` `avg(numeric)` `var_pop(numeric)` `variance(numeric)` `stddev_pop(numeric)` `stddev_samp(numeric)` `stddev(numeric)` `array_agg(anyarray)` `string_agg(text)` `string_agg(bytea)`
- builtin 照合プロバイダーを導入しました。

セキュリティ/SQL の改善

- VACUUM、ANALYZE、CLUSTER、REFRESH MATERIALIZED VIEW、REINDEX、LOCK TABLE に pg_maintain ロールを使用することで、スーパーユーザー権限を付与する必要がなくなりました。また、ユーザーにテーブルの MAINTAIN 権限を付与することができます。
- ネットワークラウンドトリップネゴシエーションなしで TLS 接続を確立します。
- ALTER SYSTEM の改善
    - ALTER SYSTEM で認識されないカスタムサーバー変数を設定できるようにします。
    - ALTER SYSTEM を禁止するシステム変数 allow_alter_system を追加します。
      設定が外部ツールによって管理されている環境で役立ちます。

SQL/JSON の改善

- JSON データをテーブル形式に変換するための関数 JSON_TABLE()を導入

```
CREATE TABLE my_films ( js jsonb );

INSERT INTO my_films VALUES (
'{ "favorites" : [
    { "kind" : "horror", "films" : [
    { "title" : "Psycho",
      "director" : "Alfred Hitchcock" } ] }
] }');
SELECT jt.* FROM my_films,
      JSON_TABLE (js, '$.favorites[*]'
        COLUMNS (id FOR ORDINALITY,
                 kind text PATH '$.kind',
                 title text PATH '$.films[*].title',
                 director text PATH '$.films[*].director')) AS jt;
 id |  kind  |  title | director
----+--------+--------+------------------
  1 | horror | Psycho | Alfred Hitchcock
```

- SQL/JSON コンストラクタ関数の追加
  - JSON()：テキストあるいはバイト文字列（UTF8 エンコーディング）として指定された式を JSON 値に変換します。

    `JSON('{"a":123, "b":[true,"foo"], "a":"bar"}')` ➤ `{"a":123, "b":[true,"foo"], "a":"bar"}`

  - JSON_SCALAR()：指定された SQL スカラ値を JSON スカラ値に変換します。

    `JSON_SCALAR(123.45)` ➤ `123.45`

  - JSON_SERIALIZE()：SQL/JSON 式を文字列またはバイナリ文字列に変換します。

    `JSON_SERIALIZE('{ "a" : 1 }' RETURNING bytea)` ➤ `¥x7b20226122203a2031207d20`

- SQL/JSON 問い合わせ関数の追加
  - JSON_EXISTS()：JSON 値に適用された SQL/JSON path_expression が項目を返す場合に true を返します。

```
SELECT JSON_EXISTS(jsonb '{"key1": [1,2,3]}', '$.key1[2]');
------------------------
t
```

  - JSON_QUERY()：JSON 値に SQL/JSON path_expression を適用した結果（JSON、配列、または文字列）を返します。

```
SELECT JSON_QUERY(jsonb '{"a": "[1, 2]"}', '$.a');
------------------------
[1, 2]
```


- JSON_VALUE()：JSON 値に SQL/JSON path_expression を適用した結果（SQL/JSON スカラ）を返します。

```
SELECT JSON_VALUE(jsonb '[1,2]', '$[1]');
------------------------
2
```

モニタリングの改善

- 新しいモニタリングビュー：pg_wait_events
  待機イベントのタイプとイベント名を表示します。
- 新しいモニタリングビュー：pg_stat_checkpointer
  すべてのチェックポイントに関する稼働統計を表示します。
- pg_stat_progress_vacuum におけるインデックス VACUUM の進捗状況を表示します。
- パラメータ old_snapshot_threshold を削除しました。

すべての機能一覧

新機能や強化された機能、およびその他の変更に関するすべての機能一覧はここにあります。

- PostgreSQL:Documentation: 17: E.1. Release 17（PostgreSQL オフィシャルのページへ）
  https://www.postgresql.org/docs/17/release-17.html

PostgreSQL 18 とその後

最後に、コミュニティで活発に行われている議論を紹介して、この記事を締めくくりたいと思います。なお、これらの機能が次のリリースまたは将来のリリースに必ず提供されるものではないことをご承知おきください。

- 透過的列暗号化：クライアントにおける特定の列の自動的で、透過的な暗号化と復号
- 非同期 I/O：インデックスのパフォーマンスを向上させるインデックス・プリフェッチ、システム・パフォーマンスを向上させるデータのプリフェッチ、一括書き込みのためのベクトル化 I/O
- アップグレード後、最初に ANALYZE を実行せずにクエリを実行できるよう、統計情報のインポート/エクスポートをサポート
- テーブルアクセスメソッド API の強化によるさまざまなストレージエンジンへの対応
- amcheck が Gist と Gin インデックスに対応
- 論理レプリケーションにおける様々な改善：DDL レプリケーション、シーケンスのレプリケーション、競合の検出と解決、ノード管理 API、未使用スロットのスロット無効化など
- エクゼキュータの改良：一般的な組み合わせに対する特殊ケースのエクゼキュータ式ステップ（JIT 生成コードの簡素化）、プランノードごとの JIT コンパイル、SQL 標準の行パターン認識（RPR）
- 特に多数のパーティションが存在する場合のプルーニングにおけるパーティショニング技術の改善
- 特に NBTree における、インデックスの改善
- 様々な種類のクエリをより適切に動作させるための、オプティマイザーの改善
- wire_protocol_level での圧縮の導入
- 並列処理：テーブルに対する VACUUM の並列化、GIN インデックスの Create Index の並列化、相関のあるサブクエリの並列化、TID 範囲スキャン
- 64 ビット XID によるフリーズ回避と autovacuum の必要性の減少

- WAL ヘッダーを小さくすることで、WAL サイズを縮小
- カスタムフォーマットや辞書式圧縮法などの TOAST の改良
- インデックスとテーブルの統計を異なるタイプの統計に分割し、より多くの統計を追加
- CI とビルドシステムの改善

最後に

PostgreSQL の未来を見据えると、バージョン 17 はパフォーマンス、スケーラビリティ、セキュリティの強化により、その進化における重要なマイルストーンであることは明らかです。コミュニティは、絶え間なく変化する現代のデータ管理要求に PostgreSQL を適応させ続けており、イノベーションと卓越性への取り組みにより、PostgreSQL が開発者と企業にとっての礎であり続けることを保証しています。
PostgreSQL の長年のユーザーであっても、初めて使用するユーザーであっても、バージョン 17 のリリースは、プロジェクトでその機能を活用するための素晴らしい機会を提供します。
これらの新機能を受け入れるとともに、情熱的で献身的なコミュニティによって推進される PostgreSQL の継続的な成長と発展にも期待しています。これからも PostgreSQL コミュニティに引き続き参加し、貢献できることを楽しみにしています。

参考

富士通のウェブサイト「PostgreSQL インサイド」では、富士通の技術者による最新動向を紹介するブログや、PostgreSQL を利用するうえで知っておきたい技術情報・豆知識、および Fujitsu Enterprise Postgres に関する記事を掲載しています。

Fujitsu Enterprise Postgres（エンタープライズ ポストグレス）は、OSS（オープン・ソース・ソフトウェア）の PostgreSQL をエンジンとし、富士通のデータベース技術とノウハウで導入・運用のしやすさを向上し、「セキュリティ」「性能」「信頼性」を強化したデータベースです。
「Fujitsu Enterprise Postgres の製品サイト」では、製品の概要、サービス、技術資料、お客様の導入事例に関する情報を掲載しています。

2024 年 10 月 9 日