PostgreSQL インサイド

# PostgreSQL 16 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

今年の初め、私はカナダで開催された PGConf 2023 に参加し、PostgreSQL がバークレー校の研究プロジェクトから最も先進的なオープンソースデータベースとしての地位を確立するまでの進化について話し、PostgreSQL 16 で導入された様々な改善点、特に論理レプリケーションについて論じました。

私の講演では、バージョン 16 で導入された魅力的な新機能にフォーカスしましたが、PostgreSQL の過去、つまり過去のバージョンにおける主要な機能の年表や、PostgreSQL 17 での実装に向けてコミュニティが議論してきた内容など、将来についても触れました。

### PostgreSQL の進化- バージョンと主なマイルストーン

PostgreSQL がその初期から歩んできた長い道のりをより深く理解するためには、PostgreSQL の進歩と、その過程で蓄積された素晴らしい機能の一覧を再確認することが重要だと思います。これは、献身的で熱心なコミュニティの活動のおかげです。

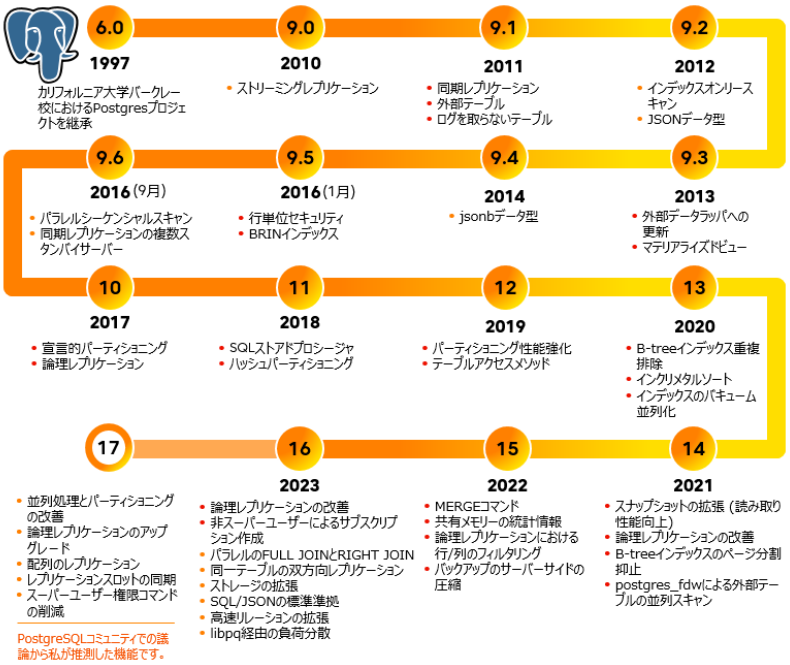


図 1：PostgreSQL のバージョンごとの進化

PostgreSQL の開発プロジェクトは 1997 年にスタートしています。本プロジェクトは、1986 年から稼働していたカリフォルニア大学バークレー校のプロジェクトがもとになっています。以来、毎年主要な機能を追加した新しいバージョンがリリースされています。
興味深いのは PostgreSQL が当初から大量のデータを扱うことを目的としており、以降もその方向に進化してきたということです。
例えば、バージョン 9.0 ではフェイルオーバーに有効な機能であるストリーミングレプリケーションが導入されました。これにより、あるノードがダウンした場合に、別のノードが引き継ぐことができます。バージョン 9.1 では、外部テーブルとログを取らないテーブル、さらに信頼性を向上し価値をさらに高めるのに役立つ同期レプリケーションの機能が導入されました。

 © 2023 Fujitsu Limited

バージョン 9.2 では、JSON データ型のサポートが追加され、ドキュメントとその関連データは、このデータ型を使って格納できるようになりました。このバージョンでは、インデックスオンリースキャンも導入され、様々な種類のクエリの処理速度が向上しました。翌年のバージョン 9.3 では、外部データラッパへの更新とマテリアライズドビューが追加されました。さらにバージョン 9.4 では JSON の改良版である jsonb データ型が導入され、そのデータ型に対する高速な処理とインデックスを付けることが可能となりました。

バージョン 9.5 では、行単位セキュリティと BRIN インデックスが追加され、特定のタイプのクエリを劇的に高速化できるようになりました。 バージョン 9.6 では、パラレルシーケンシャルスキャンとパラレルジョインが追加され、OLAP アプリケーションで活用できるようになりました。 また、さらに信頼性を高めるために、ストリーミングレプリケーションにおいて複数のスタンバイサーバーと同期レプリケーションができるようになりました。

バージョン 10 では、論理レプリケーションと宣言的パーティショニングが導入され、PostgreSQL に新しい展望が開かれました。バージョン 11 では、ハッシュパーティショニングと SQL ストアドプロシージャが導入され、他のデータベースからの移行を検討している組織にとって魅力が広がりました。バージョン 12 では、パーティショニングの性能を強化し、PostgreSQL と統合できる専用のストレージエンジンが作成可能となるテーブルアクセスメソッドを導入しました。バージョン 13 の主な機能には、B-tree インデックス重複排除、インクリメンタルソート、インデックスのバキューム並列化があります。

バージョン 14 では、多くの重要な機能追加と機能強化が行われました。スナップショット機構を拡張することで、より優れた読み取りスケーラビリティを実現し、進行中のトランザクションの論理レプリケーションを許可することで、大規模トランザクションを適用する際の適用遅延を軽減しました。また、B-tree インデックス更新の肥大化の抑制や、postgres_fdw による外部テーブルの並列スキャンを可能にしました。

バージョン 15 では、MERGE コマンドを導入しました。これは数年前からコミュニティで議論されていたもので、ようやく実装できました。また、以前の統計メカニズムよりも改善された共有メモリーの統計情報を導入し、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を改善しました。最後に、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を追加しました。

## PostgreSQL 16 の機能強化と新機能

バージョン 16 では多くの新機能が導入され、論理レプリケーションの機構もいくつか改善されています。私が考える最も重要なものの 1 つは、スタンバイノードから論理レプリケーションを実行できる機能で、これについては後述します。

### 論理レプリケーションの改善

- レプリケーション中に、origin オプションに基づいてデータをフィルタリング（除外）できるようになりました。

```
CREATE PUBLICATION mypub FOR ALL TABLES;
CREATE SUBSCRIPTION mysub CONNECTION 'dbname=postgres'
       PUBLICATION  mypub WITH (origin = none);
```

  Postgres 15 以前では、テーブルに対してレプリケーションを設定すると無限ループになる場合があり、ノード間で双方向レプリケーションや論理レプリケーションを設定することは困難でした。origin に基づいて、つまりパブリケーションから受け取ったレプリケーションデータと該当サーバーでクエリにて変更されたデータを区別できるようになり、これらのデータをフィルタリングする機能を追加します。 これにより、n-way 論理レプリケーションを設定でき、双方向レプリケーションを実行する際のループを防止できます。

- ロジカルデコーディングがスタンバイサーバーから実行できるようになりました。
  これには、プライマリーとスタンバイの両方で wal_level = logical を設定する必要があります。
  この機能は、プライマリーがビジー状態のときにサブスクライバーがスタンバイからサブスクライブできるようにすることで、負荷を分散させることができます。

- 適用プロセスは、サブスクリプション所有者の権限ではなく、テーブル所有者の権限で実行するよう設定できるようになりました。

 © 2023 Fujitsu Limited

```
CREATE SUBSCRIPTION mysub CONNECTION ...
       PUBLICATION mypub WITH (run_as_owner = false);
```

- 非スーパーユーザーがサブスクリプションを作成できるようになりました。
  非スーパーユーザーは pg_create_subscription ロールを付与されていなければならず、認証のためにパスワードを指定する必要があります。
  スーパーユーザーは、サブスクリプションを所有する非スーパーユーザーに対して password_required = false を設定できます。
- サブスクライバーで大きなトランザクションが並列に適用できるようになりました。

```
CREATE SUBSCRIPTION mysub CONNECTION ...
       PUBLICATION mypub WITH (streaming = parallel);
```

  25～40% の性能向上が確認されています（詳細は「Re: Perform streaming logical transactions by background workers and parallel apply」（pgsql-hackers のメーリングリストへ）。
  https://www.postgresql.org/message-id/CAJpy0uBm0%2ByZs%2B7emKCp2%2BRdvA3Gy_SW0aLfntfHvcEiWq_5Ew%40mail.gmail.com
  各々の大きなトランザクションは、利用可能なワーカーの 1 つに割り当てられます。これは、トランザクション全体がサブスクライバーに受信されるまで待つのではなく、即座に適用することで遅延を改善します。 ワーカーはトランザクションが完了するまで割り当てられたままです。
  参考までに、サブスクリプションあたりの並列適用ワーカーの最大数は、max_parallel_apply_workers_per_subscription で設定します。
- 論理レプリケーションで初期テーブルを同期する際にテーブルをバイナリ形式でコピーできるようになりました。

```
CREATE SUBSCRIPTION mysub CONNECTION ...
        PUBLICATION mypub WITH (binary = true);
```

  バイナリ形式でテーブルをコピーすると、カラムの種類によっては時間短縮される場合があります。
- PRIMARY KEY と REPLICA IDENTITY 以外のインデックスをサブスクライバーで使用できるようになりました。
  REPLICA IDENTITY または PRIMARY KEY インデックスが使用できない場合に、パブリッシャーで REPLICA IDENTITY FULL を使用すると、サブスクライバーでタプル（レコード）の変更ごとにテーブルをフルスキャンすることになります。
  使用できるインデックスは B-Tree インデックスです。部分インデックスは利用できません。また、左端のフィールドは、リモートのリレーションの列を参照する列（式ではありません） でなければなりません。
  テーブル内のデータ量に比例して性能は向上します。

**ストレージの拡張**

- リレーション拡張のパフォーマンスが向上しました。
  バージョン 16 では、1 つのリレーションへの同時 COPY が大幅に改善されました（16 クライアントで 3 倍）。
  リレーション拡張のロックについて、以前はシステムが以下の処理を実行する間も保持されましたが、バージョン 16 ではリレーション拡張時のみロックするようになりました。
    - 新しいページのために書き出されるバッファの捕捉（WAL をフラッシュする必要があるなど、古いページの内容をさらに書き出す必要があるかもしれません）。
    - 拡張中の 0 ページの書き込みと実ページの内容の書き出し（これは書き込み速度が約 2 倍になります）。
- BRIN インデックスが付けられた列のみが更新される場合、ヒープ専用タプル（Heap-Only Tuples、以降 HOT）更新が許可されます。

 © 2023 Fujitsu Limited

対応する列が更新された場合でも、BRIN インデックスを更新します。これは、インデックスの述部で参照される属性には適用されず、このような属性の更新は常に HOT を無効にします。

- ダイレクト I/O を使用可能とします。
  これにより、リレーションデータと WAL ファイルのキャッシュ効果を最小化するようカーネルに要求できます。現在、この機能はシステム性能を低下させる上、エンドユーザー向けではない（開発者向けのオプション）ため、デフォルトでは無効になっています。 このオプションは、GUC パラメーターの debug_io_direct（有効な値は、data、wal、wal_init）により有効にできます。
  今後の計画は、カーネルがこのオプションで無効にする機能を置き換えるために、先行読み込みなどのような独自の I/O 機構を導入することです。ダイレクト I/O でより良いパフォーマンスを得られるように、すべての I/O バッファを 4096 に揃えます。

- バキューム中にページレベルでフリーズできるようにします。
  これにより、WAL の量を減らし、フリーズ処理のコストを削減します。

- 詳細な I/O 統計を表示する pg_stat_io ビューを追加します。
  このビューには、バックエンドの種別、ターゲット I/O オブジェクトおよび I/O コンテキストの組合せが 1 行含まれており、クラスタ全体の I/O 統計情報が表示されます。

  - バックエンドタイプの例：バックグラウンドワーカー、オートバキュームワーカー、チェックポインターなど
  - ターゲット I/O オブジェクトの可能なタイプ： 永続または一時のリレーション
  - I/O コンテキストの取りうる値：normal, vacuum, bulkread, bulkwrite

  ビューは、reads、writes、extends、hits、evictions、reuses、fsyncs など、さまざまな I/O 操作を追跡します。evictions の回数が多い場合は、共有バッファを増やす必要があることを示しています。 クライアントバックエンドによる fsyncs が多い場合は、共有バッファまたはチェックポインターの設定ミスを示している可能性があります。
  この統計は、ディスクからフェッチしなければならなかったデータと、カーネルのページキャッシュにすでに存在していたデータを区別しません。

- VACUUM/ANALYZE でバッファ使用量制限を指定できるようになりました。
  VACUUM コマンドと ANALYZE コマンドに新しいオプション BUFFER_USAGE_LIMIT が追加され、共有バッファのサイズを制御できるようになりました。 より大きな値を指定することで、同時に実行される他の問合せの速度を低下させる代わりに、バキュームをより高速に実行できます。
  本処理を制御する別の方法として、GUC パラメーターの vacuum_buffer_usage_limit がありますが、VACUUM コマンドと ANALYZE コマンドの BUFFER_USAGE_LIMIT の指定が優先されます。 GUC パラメーターは自動バキュームでも指定された制限値を使用することができます。
  また、vacuumdb コマンドに buffer-usage-limit オプションを追加しました。

**SQL の新機能**

新しいバージョンには、照合順序や JSON データの操作などのオプションをユーザーに提供する多数の新機能が搭載されています。

- テキストの照合順序（テキストをどのように並べ替えるかのルール）のサポートが改善されました。

```
CREATE COLLATION en_custom (provider = icu, locale = 'en', rules = '&a < g');
```

  上記のコマンドラインでは、g を a の後、b の前に置きます。
  ルールを設定するための新しいオプションが CREATE COLLATION、CREATE DATABASE、createdb、および initdb に追加されました。

- ICU をデフォルトの照合順序のプロバイダーに設定できるようにしました。
  環境に応じて ICU のデフォルトのロケールを決定します。

 © 2023 Fujitsu Limited

- 定義済の照合順序である unicode および ucs_basic のサポートが追加されました。
- SQL/JSON 標準に準拠した JSON 型用のコンストラクタが追加されました。
    - JSON_ARRAY()関数 - 一連の value_expression パラメーターまたは query_expression の結果から JSON 配列を構築します。

```
SELECT json_array(1,true,json '{"a":null}');
     json_array
-----------------------
[1, true, {"a":null}]
```

    - JSON_ARRAYAGG()関数 – JSON_ARRAY 関数と同じように動作しますが、集約関数なので value_expression パラメーターを 1 つだけ受け取ります。

```
SELECT json_arrayagg(v NULL ON NULL) FROM (VALUES(2),(1),(3),(NULL)) t(v);
 json_arrayagg
-----------------
[2, 1, 3, null]
```

    - JSON_OBJECT()関数 - 指定されたすべてのキーと値のペアの JSON オブジェクトを作成します。何も指定されていない場合は空のオブジェクトを作成します。

```
SELECT json_object('code' VALUE 'P123', 'title': 'Jaws', 'title1' : NULL ABSENT ON NULL);
            json_object
-------------------------------------
{"code" : "P123", "title" : "Jaws"}
```

    - JSON_OBJECTAGG()関数 – JSON_OBJECT 関数と同じように動作しますが、集約関数なので 1 つの key_expression と 1 つの value_expression パラメーターだけ受け取ります。

```
SELECT json_objectagg(k:v) FROM (VALUES ('a'::text,current_date),('b',current_date + 1)) AS t(k,v);
              json_objectagg
-------------------------------------------
{ "a" : "2023-05-19", "b" : "2023-05-20" }
```

- SQL 標準の IS JSON 述語を実装しました。
  IS JSON [VALUE], IS JSON ARRAY, IS JSON OBJECT, IS JSON SCALAR

```
SELECT js, js IS JSON "json?", js IS JSON SCALAR "scalar?",
       js IS JSON OBJECT "object?", js IS JSON ARRAY "array?"
FROM (VALUES ('123'), ('"abc"'), ('{"a": "b"}'), ('[1,2]')) foo(js);
    js     | json? | scalar? | object? | array?
-----------+-------+---------+---------+--------
123        | t     | t       | f       | f
"abc"      | t     | t       | f       | f
{"a": "b"} | t     | f       | t       | f
[1,2]      | t     | f       | f       | t
```

 © 2023 Fujitsu Limited

- パラレルのハッシュ結合において full join をサポートしました。

```
EXPLAIN (COSTS OFF)
 SELECT COUNT(*)
 FROM simple r FULL OUTER JOIN simple s USING (id);
                      QUERY PLAN
------------------------------------------------------------
Finalize Aggregate
  ->  Gather
        Workers Planned: 2
        ->  Partial Aggregate
              ->  Parallel Hash Full Join
                    Hash Cond: (r.id = s.id)
                    ->  Parallel Seq Scan on simple r
                    ->  Parallel Hash
                          ->  Parallel Seq Scan on simple s
```

- 集約関数 string_agg と array_agg はパラレル集約できるようになりました。

```
EXPLAIN (COSTS OFF)
 SELECT y, string_agg(x::text, ',') AS t, array_agg(x) AS a
 FROM pagg_TEST GROUP BY y;
                  QUERY PLAN
-----------------------------------------------
Finalize HashAggregate
  Group Key: y
  ->  Gather
        Workers Planned: 2
        ->  Partial HashAggregate
              Group Key: y
              ->  Parallel Seq Scan on pagg_test
```

- ORDER BY または DISTINCT を持つ集約は、ソート済のデータが使用できるようになりました。
  これまでは、集約を行う前にタプルをソートする必要がありました。
  PostgreSQL 16 では、インデックスによって事前にソートされた入力データが提供され、それが集約に直接使用されます。

```
EXPLAIN (COSTS OFF)
 SELECT SUM(c1 ORDER BY c1), MAX(c2 ORDER BY c2) FROM presort_test;
                      QUERY PLAN
------------------------------------------------------------
Aggregate
  ->  Index Scan using presort_test_c1_idx on presort_test
SET enable_presorted_aggregate=off;
EXPLAIN (COSTS OFF)
 SELECT SUM(c1 ORDER BY c1), MAX(c2 ORDER By c2) FROM presort_test;
         QUERY PLAN
```

 © 2023 Fujitsu Limited

```
--------------------------------
Aggregate
  ->  Seq Scan on presort_test
```

- RANGE パーティションおよび LIST パーティションの検索で最後に見つかったパーティションがキャッシュされます。
  これにより、多くの連続したタプルが同じパーティションに属するパーティションテーブルへのバルクロードのオーバーヘッドが削減されます。
- パーティション化テーブルでの LEFT JOIN の削除と等価結合ができるようになりました。

```
CREATE TEMP TABLE a (id int PRIMARY KEY, b_id int);
CREATE TEMP TABLE parted_b (id int PRIMARY KEY) PARTITION BY RANGE(id);
CREATE TEMP TABLE parted_b1 PARTITION OF parted_b FOR VALUES FROM (0) TO (10);

EXPLAIN (COSTS OFF)
 SELECT a.* FROM a LEFT JOIN parted_b pb ON a.b_id = pb.id;
 QUERY PLAN
---------------
Seq Scan on a
```

**セキュリティ/権限の追加**

- 各種コマンドでスーパーユーザー権限を追加する必要がなくなりました。
  - reserved_connections パラメーターは非スーパーユーザーによる接続スロットを予約する方法を提供します。
  - 定義済のロール pg_use_reserved_connections は reserved_connections パラメーターで予約された接続スロットを使用します。
- Kerberos 資格情報の委任のサポートが追加されました。
  これにより PostgreSQL サーバーはこれらの委任された資格情報を使用して、postgres_fdw や dblink、または理論的には Kerberos を使用して認証できる他のサービスなどの別のサービスに接続できるようになります。
- 許可される認証メソッドの一覧を指定する新しい libpq 接続オプション require_auth が追加されました。
  指定できるメソッドは、password、md5、gss、sspi、scram-sha-256、または none です。
  このオプションは、メソッドの先頭に!を付けることによって、特定の認証メソッドを許可しないようにすることもできます。
  サーバーが必要な認証要求を使用しない場合、接続試行は失敗します。
- libpq が証明書の検証にシステム証明書プールを使用できるようになりました。
  これは sslrootcert=system で有効になります。設定値 system は、同じ名前のローカル証明書ファイルよりも優先されます。
- GRANT ... SET オプションが追加されました。
  SET オプションを TRUE に設定すると、メンバは SET ROLE コマンドを使用して付与されたロールに変更できます。別のロールが所有するオブジェクトを作成したり、既存のオブジェクトの所有権を別のロールに付与したりするには、そのロールに SET ROLE する権限が必要です。 それ以外の場合、ALTER ... OWNER TO または CREATE DATABASE ... OWNER などのコマンドは失敗します。

**その他のパフォーマンスの改善**

- pg_strtointnn 関数のパフォーマンスが向上しました。
  テストの結果、2 つの INT カラムを含むテーブルへの COPY で、約 8%の高速化が確認されました。

 © 2023 Fujitsu Limited

- ハッシュインデックス構築のパフォーマンスが向上しました。
  最初のデータソートで、バケット番号が同じ場合は、次にハッシュ値でソートします。不要なバイナリ検索を省略することで、ビルドが高速化されます。ハッシュインデックスの作成速度が 5～15%向上しました。
- メモリー管理のパフォーマンスが向上し、オーバーヘッドが削減されました。
  各割り当てのヘッダーサイズを 16+から 8 バイトに削減し、ロジカルデコーディング中にメモリーを割り当てるために使用されるスラブメモリーアロケータの性能を向上させました。
- x86 アーキテクチャと ARM アーキテクチャの両方で、SIMD を使用した CPU アクセラレーションのサポートが追加されました。
  これにより、サブトランザクション検索と ASCII および JSON 文字列の処理が最適化されます。
- libpq の接続ロードバランシングの性能が向上しました。
  load_balance_hosts=random を設定すると、ホストとアドレスをランダムな順序で接続できます。このパラメーターを target_session_attrs と組み合わせて使用すると、スタンバイサーバーでのみ負荷分散を行うことができます。
  選択したノードが応答しない場合に他のノードが試行されるように、connect_timeout にも適切な値を設定することをお勧めします。
- pg_dump に LZ4 と Zstandard 圧縮オプションが追加されました。
- バッチで行を追加するための外部テーブルへの COPY がサポートされました。
  これは postgres_fdw の batch_size オプションで制御されます。

**互換性**

- Windows インストールにおいて、Windows 10 の最小バージョンをサポートしました。
- スタンバイサーバーの昇格を有効にする promote_trigger_file オプションを削除します。
  スタンバイを昇格させるには pg_ctl promote コマンドまたは pg_promote() 関数を使用しなければなりません。
- サーバー変数 vacuum_defer_cleanup_age が削除されました。
  hot_standby_feedback とレプリケーションスロットが追加されたので、これは不要になりました。
- libpq による SCM 資格証明認証のサポートが削除されました。
- 最終的に Autoconf に取って代わる Meson ビルドシステムを導入しました。

**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: 16: E.1. Release 16（PostgreSQL オフィシャルのページへ）
  https://www.postgresql.org/docs/16/release-16.html

 © 2023 Fujitsu Limited

## PostgreSQL 17 とその後

最後に、PostgreSQL 17 以降のような PostgreSQL の将来のバージョンについて PostgreSQL コミュニティが現在議論している変更点をいくつか紹介したいと思います。 このリストは確定したものではなく、これらすべての機能が最終候補に入る保証はないことに注意してください。これは PostgreSQL コミュニティ内で行われている議論から得た私自身の見解に基づいています。

- 論理レプリケーションにおける様々な改善
    - DDL コマンドのレプリケーション
    - シーケンスのレプリケーション
    - フェイルオーバーを可能にするレプリケーションスロットの同期化
    - 論理レプリケーションノードのアップグレード
    - テーブル同期ワーカーの再利用
- スーパーユーザー権限を必要とするコマンドの数の削減
- 標準規格への準拠性を向上させるための SQL/JSON の改善
- 増分バックアップ
- クライアント内の特定の列の透過的な列暗号化と復号化
- 非同期 I/O によりデータのプリフェッチを可能にし、システムのパフォーマンスを向上
- 大量のファイルデイスクリプタの open/close を削減するための大規模リレーションファイル
- テーブルアクセスメソッド API のエンハンス
- amcheck モジュールによる GiST インデックスおよび GIN インデックスの検査
- ロックの改善によるスケーラビリティの向上
- パフォーマンスデータ構造の使用によるバキューム技術の改善
- パーティショニング技術の改善
- 統計/監視の改善
- 多くの組織におけるセキュリティ・コンプライアンスへの対応に役立つ透過的データ暗号化
- 64 ビットのトランザクション ID（XID）により凍結を発生しにくくし、自動バキュームの必要性を削減
- 並列処理
    - InitPlan が付加されたプランノードのパラレル安全化
    - 相関するサブクエリの並列化
- WAL のヘッダーを小さくすることで WAL 全体のサイズを削減
- SLRU をメインバッファプールへ移動
- TOAST の改善：カスタム形式と辞書の圧縮
- CI およびビルドシステムの改善

 © 2023 Fujitsu Limited

**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、無料体験版、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。

2023 年 10 月 6 日


© 2023 Fujitsu Limited