SRA OSS

PostgreSQL 15 最新情報解説

2022年 9月 2日 13:00-14:30

SRA OSS LLC

高塚 遥

本講演について

- 講演内容
  - 今秋リリース予定の PostgreSQL 15 について新機能や性能向上を解説
  - 動作検証結果を報告
  - 非互換の点についても解説
- 講演者
  - 高塚 遥
  - ヘルプデスク、コンサルティング、など、PostgreSQLの支援業務を各種手掛ける
  - 特定非営利活動法人 日本PostgreSQLユーザ会 理事

会社概要

社名	SRA OSS合同会社
略称	SRA OSS LLC
設立	2022 年 6 月
社長	稲葉　香理
事業内容	• オープンソースソフトウェア関連のテクニカルサポート • オープンソースソフトウェア関連のコンサルティング・構築 • オープンソースソフトウェア関連プロダクトの開発・販売 • オープンソースソフトウェアの教育 • オープンソースソフトウェアの開発 • オープンソースソフトウェアコミュニティの運営支援 • ソフトウェアの研究開発

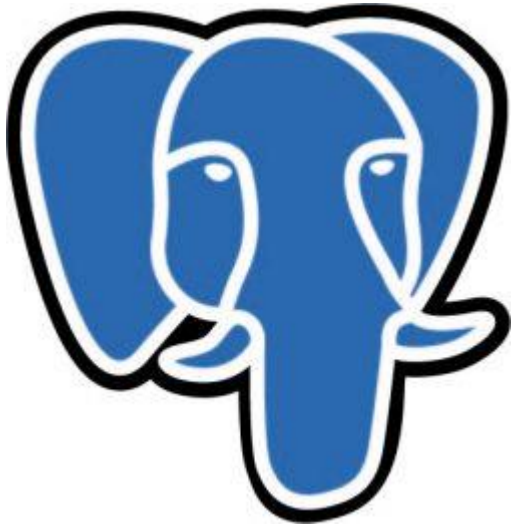
1999 年	日本で初めて PostgreSQL の商用サポートを開始
2003 年	PostgreSQL 完全互換で使いやすさと安心のサポートを提供する「PowerGres」の販売を開始
2005 年	SRA OSS, Inc. 日本支社 設立
2007 年	テキスト変換ライブラリ「libTextConv」の発売を開始
2009 年	メールの高速全文検索ソフトウェアSylpheed Pro の販売を開始
2011 年	OSS ワンストップサポート「OSS プロフェッショナルサポートサービス」を開始
2014 年	PostgreSQLの多機能ミドルウェア 「Pgpool-II」 のサポートサービスを開始
2022 年	**SRA OSS合同会社設立** SRA OSS, Inc. 日本支社よりすべての事業を譲受

SRA OSS

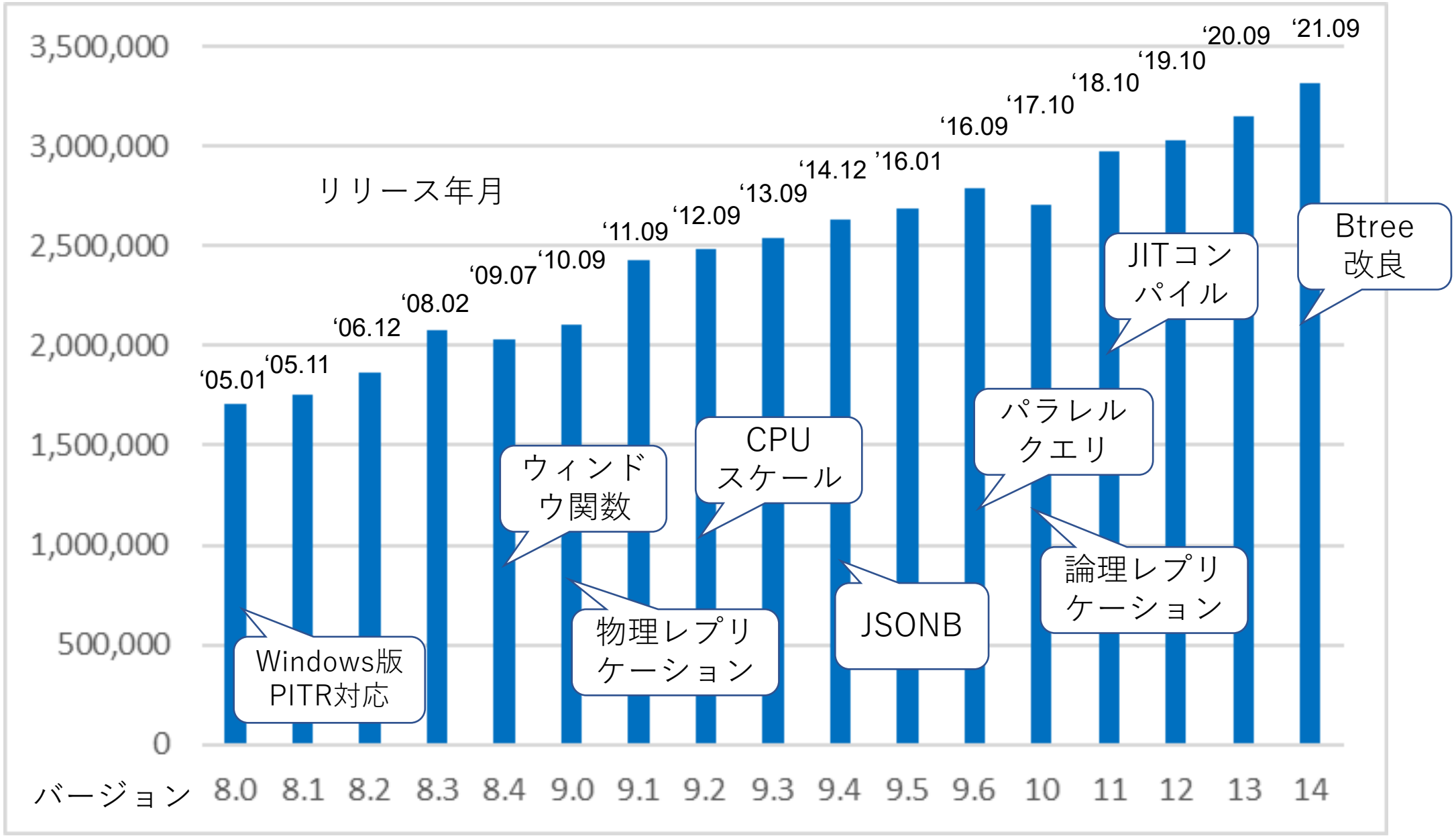
PostgreSQL15の概要

PostgreSQLとは

- 多機能リレーショナルデータベースソフトウェア
- 長い歴史
  - 1970年代 Ingres → 1980年代 POSTGRES → 1996年 PostgreSQL 6.0
- BSDタイプの PostgreSQLライセンス
- 1社主導ではないオープンソースソフトウェア
  - 主要開発者や開発運営者が、
    各PostgreSQLビジネス企業に所属している形態



PostgreSQLのリリース履歴



PostgreSQL 15 の主なエンハンス

- SQL機能追加
  - MERGE文サポート
  - 正規表現関数の追加
  - SQL/JSON 準拠関数の追加
  - 論理レプリケーション機能追加
  - 呼び出し元権限で実行するビュー
- 性能改善
  - 外部ソートアルゴリズム変更
  - ウィンドウ関数
  - ¥copy FROM
  - Zstandard圧縮サポート
- 運用管理
  - JSON形式ログ
  - モニタリングビュー・関数
  - 実行時統計情報の共有メモリ化
  - モジュールによるWALアーカイブとベースバックアップ

SQL機能追加

MERGE文

- テーブルに対して、行または表データを照会して、値に応じた様々な更新を行うことができる SQL文
- SQL標準構文
  - Oracle Database や SQL Server にも
- いわゆる UPSERT機能を実現
  - 同キー行があれば UPDATE、
    同キー行が無ければ INSERT
- 従来の UPSERT 方法
  - WITH .. UPDATE .. RETURNING
  - INSERT .. ON CONFLICT

UPSERT方式	動作
WITH .. UPDATE .. RETURNING	まず UPDATEして、結果が 0行なら、INSERT する。
INSERT .. ON CONFLICT	まず INSERTして、主キー衝突したなら、UPDATEする。
MERGE	まず 外部結合して、値を比較したうえで、INSERTしたり、UPDATEしたりする。

MERGE文

```
db1=# SELECT * FROM t_master;
 id | c1 | c2 |         ts
----+----+----+---------------------
  1 | aa | AA | 2022-08-01 00:00:00
  2 | bb | BB | 2022-08-01 00:00:00
  3 | cc | CC | 2022-08-01 00:00:00
```

```
db1=# SELECT * FROM t_work;
 id | c1  | c2  |         ts          | del
----+-----+-----+---------------------+-----
  1 | aa  | AA  | 2022-08-01 00:00:00 |
  4 | dd  | DD  | 2022-08-02 10:00:00 |
  2 | bbb | BBB | 2022-08-03 11:00:00 |
  3 |     |     | 2022-08-03 12:00:00 | t
```

```
db1=# SELECT * FROM t_master;
 id | c1  | c2  |         ts
----+-----+-----+---------------------
  1 | aa  | AA  | 2022-08-01 00:00:00
  2 | bbb | BBB | 2022-08-03 11:00:00
  4 | dd  | DD  | 2022-08-02 10:00:00
```

```
MERGE INTO t_master m USING t_work w ON m.id = w.id
WHEN MATCHED AND m.ts < w.ts AND w.del IS true THEN
  DELETE
WHEN MATCHED AND m.ts < w.ts THEN
  UPDATE SET c1 = w.c1, c2 = w.c2, ts = w.ts
WHEN NOT MATCHED
  THEN INSERT VALUES (w.id, w.c1, w.c2, w.ts);
```

論理レプリケーション機能追加

様々な機能追加:

```
CREATE PUBLICATION pub1
   FOR TABLE public.t1 WHERE (id < 100);
```

- 行フィルタ
  - PUBLICATION に WHERE句を指定
  - 条件を満たす行だけレプリケーション
  - UPDATE して条件を満たす　→ サブスクリプション側テーブルに行追加
  - UPDATE して条件から外れる　→ サブスクリプション側テーブルで行削除

- 列フィルタ
  - PUBLICATION で列リストを指定
  - 列挙した列だけレプリケーション

```
CREATE PUBLICATION pub2
   FOR TABLE public.t2 (id, c1);
```

論理レプリケーション機能追加

- スキーマ単位で対象テーブルを一括指定
  - 従来はテーブル単位、データベース単位のみ

```
CREATE PUBLICATION pub3 FOR ALL TABLES IN SCHEMA scm3;
```

- エラーで自動無効化
  - 制約違反などでサブスクリプション側で行変更の適用がエラーになる
    - するとエラーが繰り返され、後続の変更も適用されない
  - 自動的に SUBSCRIPTION を無効化する設定

```
ALTER SUBSCRIPTION sub1 SET (disable_on_error = true);
```

    - レプリケーションは止まるが、エラー繰り返し発生は回避

論理レプリケーション機能追加

- エラーを起こすトランザクションをスキップ

```
ERROR:  duplicate key value violates unique constraint "t2_pkey"
DETAIL:  Key (id)=(100) already exists.
CONTEXT:  processing remote data for replication origin "pg_16771" during
"INSERT" for replication target relation "scm1.t2" in transaction 1226822
finished at 2/C00FCD0
LOG:  logical replication subscription "sub1" has been disabled due to an
error
```

```
ALTER SUBSCRIPTION sub1 SKIP (LSN = '2/C00FCD0');
ALTER SUBSCRIPTION sub1 ENABLE;
```

問題の変更データを含むトランザクションをスキップ。

エラーメッセージのLSN番号を指定すれば良い。

SUBSCRIPTION を無効化していた場合は、有効化して、レプリケーションを再開。

SQL/JSON 関数の拡充

- SQL/JSON： SQL標準の一つ、SQLでJSONを扱う機能群
- JSONコンストラクタ 関数／構文
  - json型またはjsonb型のデータを生成

関数名	説明
JSON	テキストデータから生成
JSON_OBJECT	テキスト配列から生成
JSON_OBJECTAGG	複数行を JSON オブジェクトに集約
JSON_ARRAY	要素の並びから JSON 配列を生成
JSON_ARRAYAGG	複数行を JSON 配列に集約
JSON_SCALAR	値から JSON スカラ値を生成

```
db1=# SELECT g AS n, g^3 AS cube
        FROM generate_series(1, 3) g;
 n | cube
---+------
 1 |    1
 2 |    8
 3 |   27

db1=# SELECT JSON_OBJECTAGG(n:cube) FROM (
        SELECT g AS n, g^3 AS cube
          FROM generate_series(1, 3) g) v;
        json_objectagg
------------------------------
 { "1" : 1, "2" : 8, "3" : 27 }
```

SQL/JSON 関数の拡充

- JSON問い合わせ 関数
  - JSONデータ内を検索して要素を抽出する機能
  - 同様の機能は従来からあったが、SQL標準に沿った構文で提供
  - JSON_TABLE は JSONデータを表データに変換する

関数名	説明
JSON_EXISTS	JSON PATH で該当する要素があるか
JSON_QUERY	JSON PATH で要素を抽出
JSON_VALUE	JSON PATH で要素を取り出し、SQL スカラ値として返す
JSON_TABLE	JSON データへの問い合わせ結果を表データで返す

SQL/JSON 関数の拡充

```
db1=# SELECT * FROM JSON_TABLE (
      JSON('{"data":[{"name":"Alice", "att":{"age":18}},
                     {"name":"Bob", "att":{"age":21}}]}' RETURNING jsonb),
      '$.data[*]' COLUMNS (id FOR ORDINALITY, name text PATH '$.name',
                           age int PATH '$.att.age'));
 id | name  | age
----+-------+-----
  1 | Alice |  18
  2 | Bob   |  21
(2 rows)
```

- $.data[*] 配列の要素が行に対応
- id 列は FOR ORDINALITY で自動生成の連番
- name列は $.name で 要素内の name値
- age列は $.att.age で要素内の att 内の age値

正規表現関数の拡充

- POSIX正規表現のマッチや置換をする関数群
- 似た機能の関数や演算子は従来からある
  - 商用DB製品と同じ書き方で提供

関数名	説明
regexp_count	正規表現パターンが文字列にマッチする個数
regexp_instr	正規表現パターンが文字列にN番目にマッチする位置
regexp_like	正規表現パターンが文字列にマッチするか
regexp_substr	正規表現パターンにマッチする部分文字列
regexp_replace	正規表現パターンにマッチする部分を指定文字列で置換

呼び出し元権限で実行するビュー

- ビューの機能
  - （1） 問い合わせに別名を付与して、リレーションとして利用可能に
  - （2） 定義された問い合わせをビュー所有者の権限で実行

  - （1）だけ欲しい、（2）は不要、ということがある
    - テーブルに各ロール毎に細かな権限設定をしても、
      ビューを経由されると単一ロールからのアクセスになってしまう

  - ビューに新たな属性 security_invoker

```
ALTER VIEW v1 SET (security_invoker);
```

性能改善

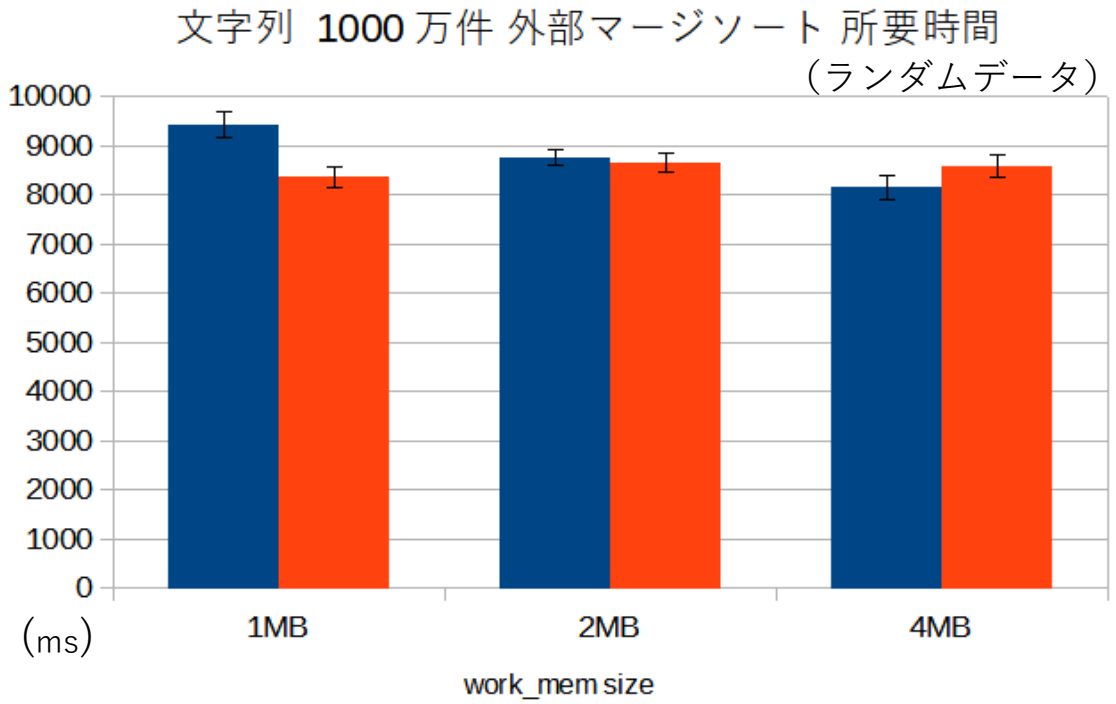
外部ソートアルゴリズム変更

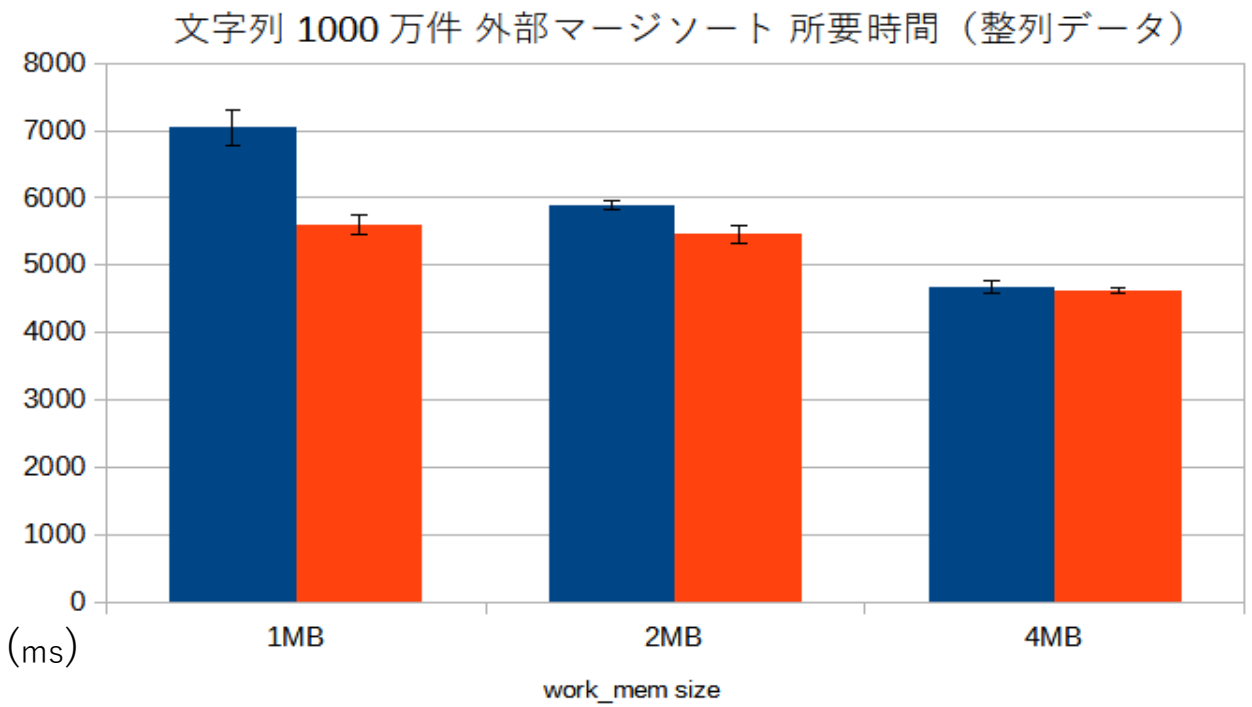
- PostgreSQLのソート機能
  - ORDER BY や GROUP BY、DISTINCT、ウィンドウ関数、マージ結合で
  - 二つのソート方式

ソート方式	いつ使われるか？	アルゴリズム
インメモリソート	work_mem サイズ内のメモリで処理できる場合	クイックソート
外部ソート	より大規模なソートの場合、一時ファイルを使って実行	**多層マージソート** ↓ **平衡K-wayマージソート**

  - 多層マージソート: ソート用の領域を使いまわす技法（現代的でない）
  - 均衡K-wayマージソート: 小メモリで大データをソートする際に有利とされる

外部ソートアルゴリズム変更





■ v14.5
■ v15b3

- work_mem が少ない時に V15 が優越の傾向
- ランダムデータ（左）よりも、
  整列済みデータ（右）の方が V15優越が明解

棒グラフに付加された
誤差範囲線は、上下に
標準偏差だけ伸ばしたもの。
同条件ごと 5 回実施。

ウィンドウ関数のプラン改善

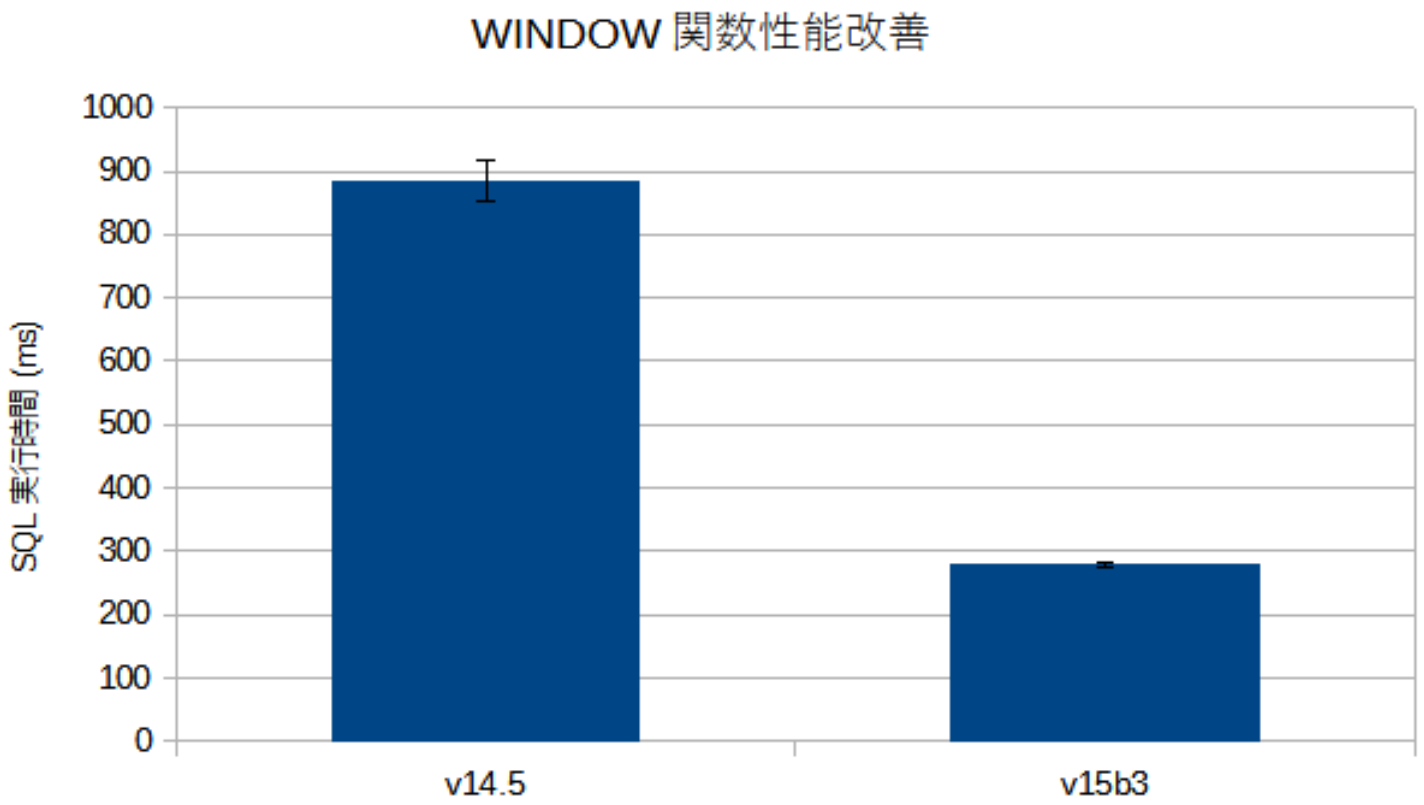
- row_number()、rank()、count() の性能向上
- WindowAgg に「Run Condition」実行方式が追加
  - V14 以前は WindowAgg 結果をすべて出したうえで、
    上位の Subquery Scan にて、これら関数の結果値でフィルタしていた

```
db1=# explain SELECT * FROM (
        SELECT *, row_number() over (order by c1) rn FROM t_win1) t
        WHERE rn <= 10;
                                QUERY PLAN
-------------------------------------------------------------------------
 WindowAgg  (cost=152431.13..171730.62 rows=1102828 width=49)
   Run Condition: (row_number() OVER (?) <= 10)
   ->  Sort  (cost=152431.13..155188.20 rows=1102828 width=41)
         Sort Key: t_win1.c1
         -> Seq Scan on t_win1 (cost=0.00..20374.28 rows=1102828 width=41)
```

ウィンドウ関数のプラン改善

```
SELECT * FROM (
  SELECT *, row_number() over (order by c1) rn FROM t_win1) t
  WHERE rn <= 10;
```

100万件データに対して
実行した所要時間を比較
（ソートはクイックソート）

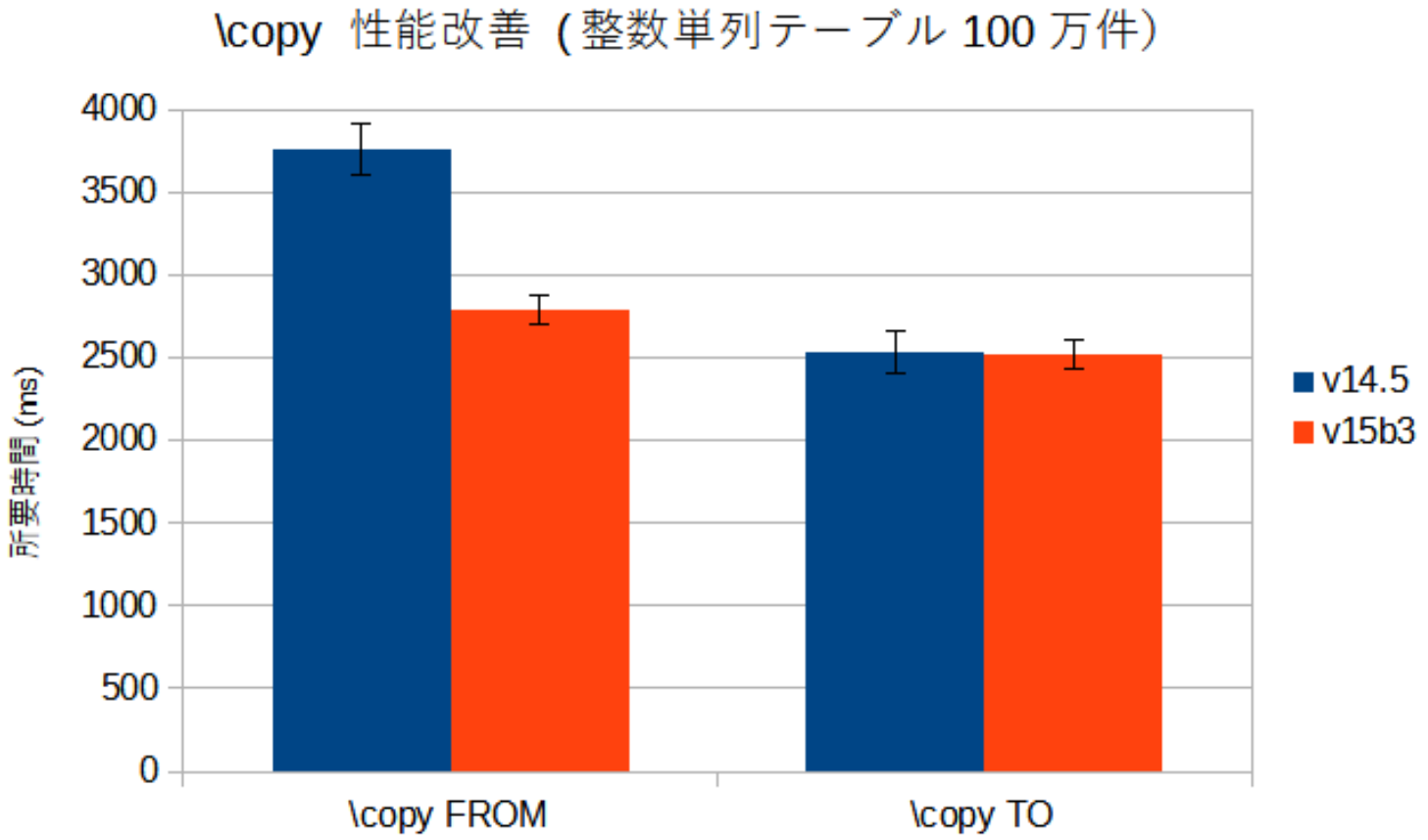


棒グラフに付加された
誤差範囲線は、上下に
標準偏差だけ伸ばしたもの。
同条件ごと 5 回実施。

psql の ¥copy FROM 性能改善

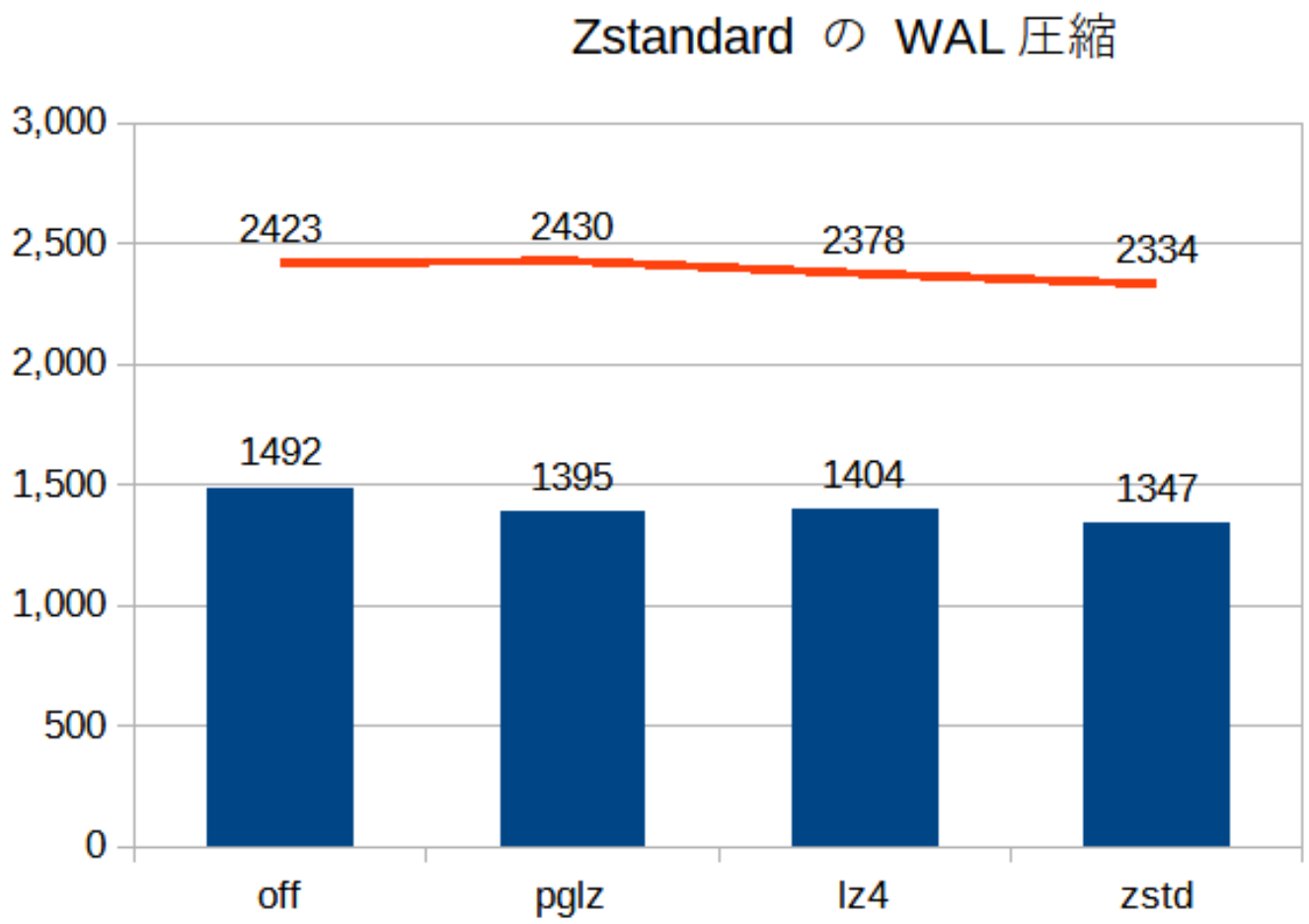
- 列の少ないテーブルの ¥copy FROM で効率改善
  - データを1行ずつ送出していた
    ↓
  - 複数行をまとめて送出

本検証は最も効果が出る条件



Zstandard圧縮サポート

- WAL圧縮とベースバックアップに利用できる



同条件の pgbench を実行、
前後の WAL進行量と、
pgbench のTPS値

WAL圧縮は full page のみ

展開速度のアドバンテージは
本検証には含まれない

運用管理

JSON形式ログ

- log_destination パラメータに新たな選択肢「jsonlog」
  - csvlog の JSON版


```
{"timestamp":"2022-08-30 18:04:51.671 JST",
 "user":"postgres","dbname":"db1",
 "pid":1609,
 "remote_host":"[local]",
 "session_id":"630dd2af.649",
 "line_num":1,
 "ps":"SELECT",
 "session_start":"2022-08-30 18:04:47 JST",
 "vxid":"3/2", "txid":0,
 "error_severity":"ERROR",
 "state_code":"22012",
 "message":"division by zero",
 "statement":"SELECT 1 / 0;",
 "application_name":"psql",
 "backend_type":"client backend",
 "query_id":0}
```

```
{"timestamp":"2022-06-27 17:28:50.270 JST",
 "pid":1999,
 "session_id":"62b96916.7cf",
 "line_num":1,
 "session_start":"2022-06-27 17:23:50 JST",
 "txid":0,
 "error_severity":"LOG",
 "message":"checkpoint starting: time",
 "backend_type":"checkpointer",
 "query_id":0}
```


実際の出力には改行は含まれない

メッセージによって含まれる属性が異なる

モニタリングビュー・関数の追加

- pg_ident.conf ファイル、論理レプリケーション関連の
  モニタリング関数・ビューが追加

関数／ビュー	説明
ビュー pg_ident_file_mappings	pg_ident.conf ファイル内容
ビュー pg_stat_subscription_stats	サブスクリプションごと論理レプリケーションの 同期エラー発生数
関数 pg_stat_reset_subscription_stats(oid)	引数はサブスクリプションの oid、 pg_stat_subscription_stats ビュー値をリセット
関数 pg_ls_logicalsnapdir()	$PGDATA/pg_logical/snapshots/ ディレクトリ内容 （ファイル名、サイズ、変更日付時刻）
関数 pg_ls_logicalmapdir()	$PGDATA/pg_logical/mappings/ ディレクトリ内容
関数 pg_ls_replslotdir(text)	引数はスロット名、 $PGDATA/pg_replslot/《スロット》/ ディレクトリ内容

実行時統計情報の共有メモリ化

- 実行時統計情報とは:
  - 各テーブルの各種アクセス回数、推定デッドタプル数、VACUUM記録、チェックポイント実行時の各種カウント、デッドロック発生数、etc ...
- 従来の実行時統計情報
  - プロセス間通信で stats collector プロセスに連携
  - 稼働中は ${PGDATA}/pg_stat_tmp/ 下ファイルに書き出し
  - 停止中は ${PGDATA}/pg_stat/ 下ファイルに保存
  - pg_stat_tmp を RAM disk に割り当てて高速化
- V15 で共有メモリ上で管理するように変更
  - 停止中はファイルに保存

モジュールによるWALアーカイブ

- 一般的なこれまでの WALアーカイブ設定例

```
archive_mode = on
archive_command = 'test ! -f /DATA/arc/%f && cp %p /DATA/arc/%f'
```

  - 外部コマンドを毎回起動しているのは効率的でない

- WALアーカイブ処理を行うモジュールを指定可能に
  - 参考実装 contrib/basic_archive による WALアーカイブ設定例

```
archive_mode = on
archive_library = 'basic_archive'
basic_archive.archive_directory = '/DATA/arc/'
```

モジュールによるベースバックアップ

- pg_basebackupコマンドの --target オプション
  - client　　従来動作、-Dオプション先に取得
  - server:/path/of/basebackup　　サーバ側に取得
  - 新たな ベースバックアップtarget を追加できる内部API
    - BaseBackupAddTarget
- contrib/basebackup_to_shell
  - ベースバックアップの本体処理を任意コマンドで実行
  - pg_basebackup の --target に shell を指定
  - 指定コマンドに各テーブルスペースの tar データが渡される

使わなくても良い

```
basebackup_to_shell.command = '/usr/local/bin/mybackup.sh %f'

shared_preload_libraries = 'basebackup_to_shell'
```

非互換の変更点

排他的ベースバックアップ廃止

- 従来: 2種類のベースバックアップ

方式	使用方法	動作の特長
排他的 ベースバックアップ	pg_start_backup() 関数で *exclusive* =true	pg_stop_backup() 関数を別セッションで実行可。 backup_label 、tablespace_mapファイルを 自動生成。 同時に1つのバックアップ取得しかできない。
非排他的 ベースバックアップ	pg_start_backup() 関数で *exclusive* =false pg_basebackup コマンド	pg_stop_backup() は pg_start_backup() と同セッションで実行。 backup_label や tablespace_map ファイルに書くべき情報が pg_stop_backup() から返される。

- 排他的ベースバックアップ廃止／関数も変更
  - pg_start_backup() → 関数名変更 pg_backup_start()
  - pg_stop_backup() → 関数名変更 pg_backup_stop()
  - pg_backup_start_time() , pg_is_in_backup() → 廃止

publicスキーマへのデフォルト権限

- 従来: publicスキーマには誰でもテーブル等を作成可能
  - セキュリティバグを突いた攻撃に利用されがち
    - 例: オブジェクトを作る→ VACUUM 実行を通してスーパーユーザ権限でコード実行
- publicスキーマのデフォルト権限変更
  - データベース所有者かスーパーユーザ以外はオブジェクト作成権限無し
  - publicスキーマの所有者が postgres から pg_database_owner に変更
- 考えられる影響
  - バックアップリストア失敗
  - パッケージソフトがインストールで失敗

推奨はできないが、これで
旧デフォルト権限に戻せる

```
template1=# GRANT CREATE ON SCHEMA public TO PUBLIC;
```

その他の非互換

- PL/Python で Python 2 サポート終了
- array_to_tsvector() 関数で挙動変更
  - 空文字列の配列要素に対してエラーになる
  - 関数インデックスのダンプリストアで失敗する可能性
- hash_mem_multiplier のデフォルト値 1.0 → 2.0
  - メモリ使用量が増える
- interval型の出力関数が IMMUTABLE → STABLE

```
db1=# CREATE INDEX ON t_interval ((interval_type_col::text));
ERROR:  functions in index expression must be marked IMMUTABLE
```

非互換修正点は他にも多数

まとめ

- バージョン15に見る PostgreSQLの現在
  - 意欲的な機能追加
    - MERGE文や論理レプリケーション機能追加
  - 実用的な利用者メリット提供
    - 商用DB互換関数、SQL/JSON準拠機能、など
  - 拡張性の拡大
    - 拡張モジュールによる実現余地を拡大
  - 不断の旧実装の見直し
    - ソートアルゴリズム差し替え、publicスキーマ権限変更、など