PostgreSQL 18 検証報告

2025年 9月 12日

[SRA OSS PostgreSQL 18 ウェビナー]

株式会社 SRA OSS　高塚 遥

講演内容／講演者

講演内容

- PostgreSQL の概要とこれまで
- PostgreSQL 18 について
  - 性能向上
  - SQL機能の向上
  - 運用性の向上
  - 非互換の変更点

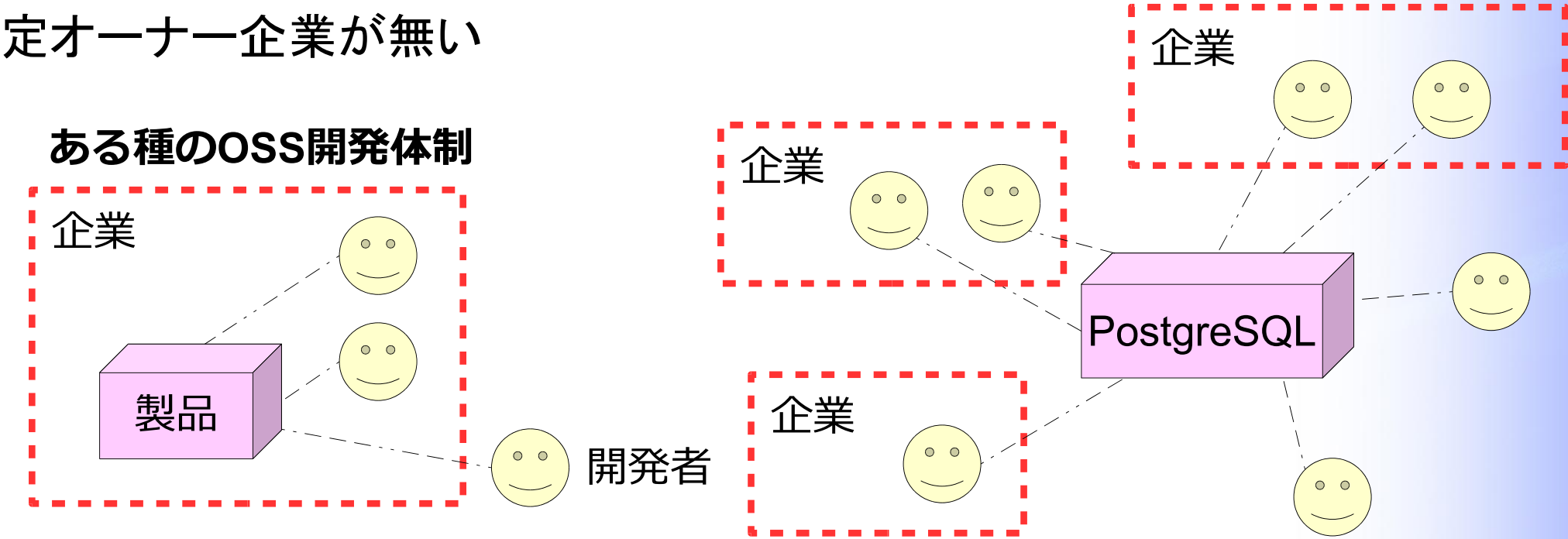
講演者

- 株式会社SRA OSS　高塚 遥
- PostgreSQL のヘルプデスク、コンサルティング、設計構築、トレーナー等をおよそ20年にわたり担当
- PostgreSQL Contributor
- 特定非営利活動法人
  日本PostgreSQLユーザ会 理事

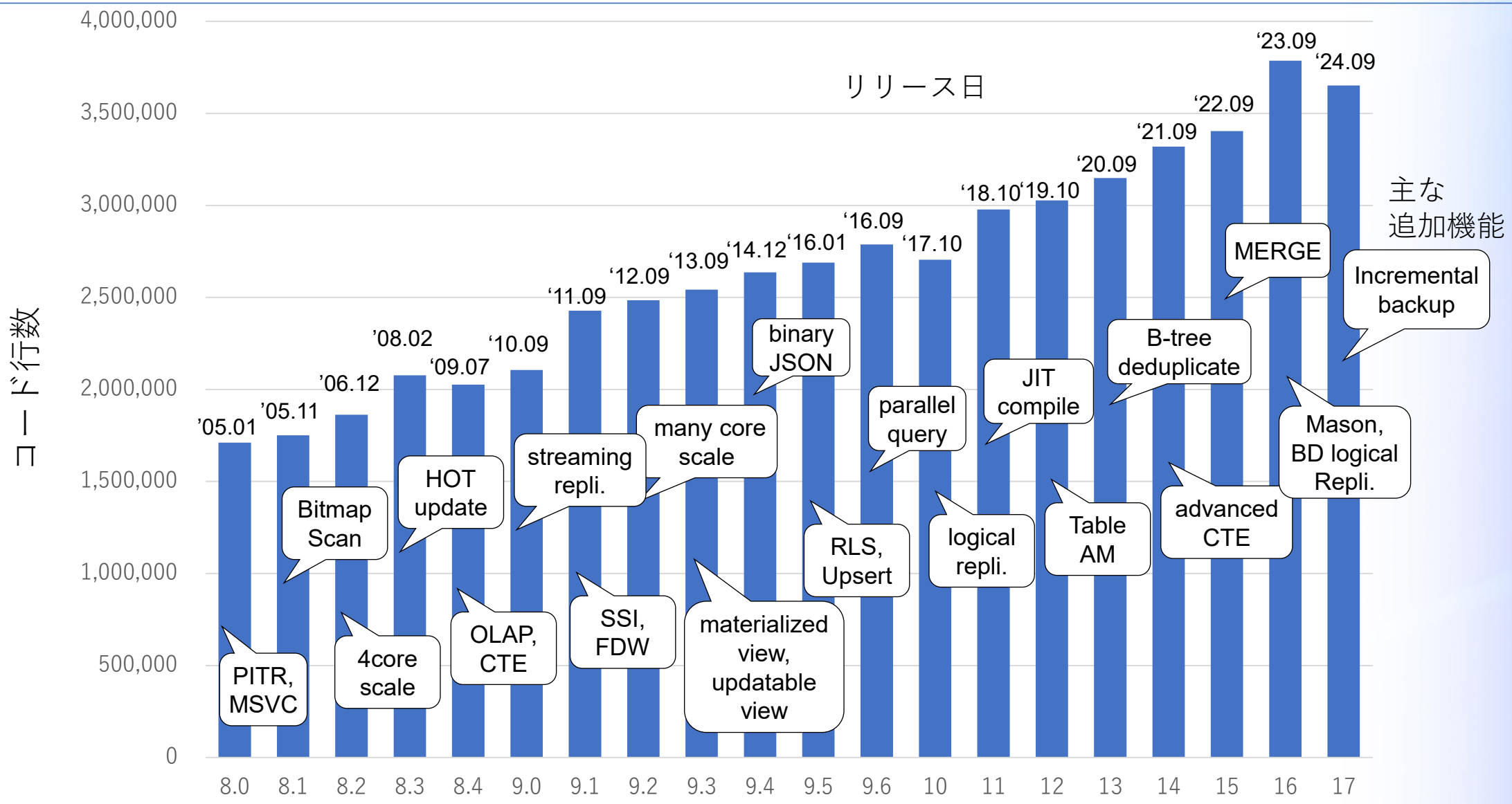
PostgreSQL の概要とこれまで

PostgreSQL とは

- 多機能、高性能、かつオープンソースの RDBMS
  - 伝統的な RDBMS設計
  - INGRES(‘70)、POSTGRES(‘80) から続く長い歴史
  - BSDタイプのライセンス
  - 特定オーナー企業が無い



PostgreSQL リリースの履歴



現状 PostgreSQL の強み／弱み

強み

- 各種用途での圧倒的な実績
  - 各種ソフトウェアからの対応
  - 技術者も多い
- 安定した開発体制
- SQL標準準拠の方針
- 様々な互換製品、拡張製品

弱み

- 伝統的な設計による性能限界
  - ディスク主体、少CPU・小メモリ想定
- 小さい本体と外部機能依存
  - 高可用性クラスタ
  - バックアップ管理
  - 監査ログ
  - プランナヒント

PostgreSQL 18 について

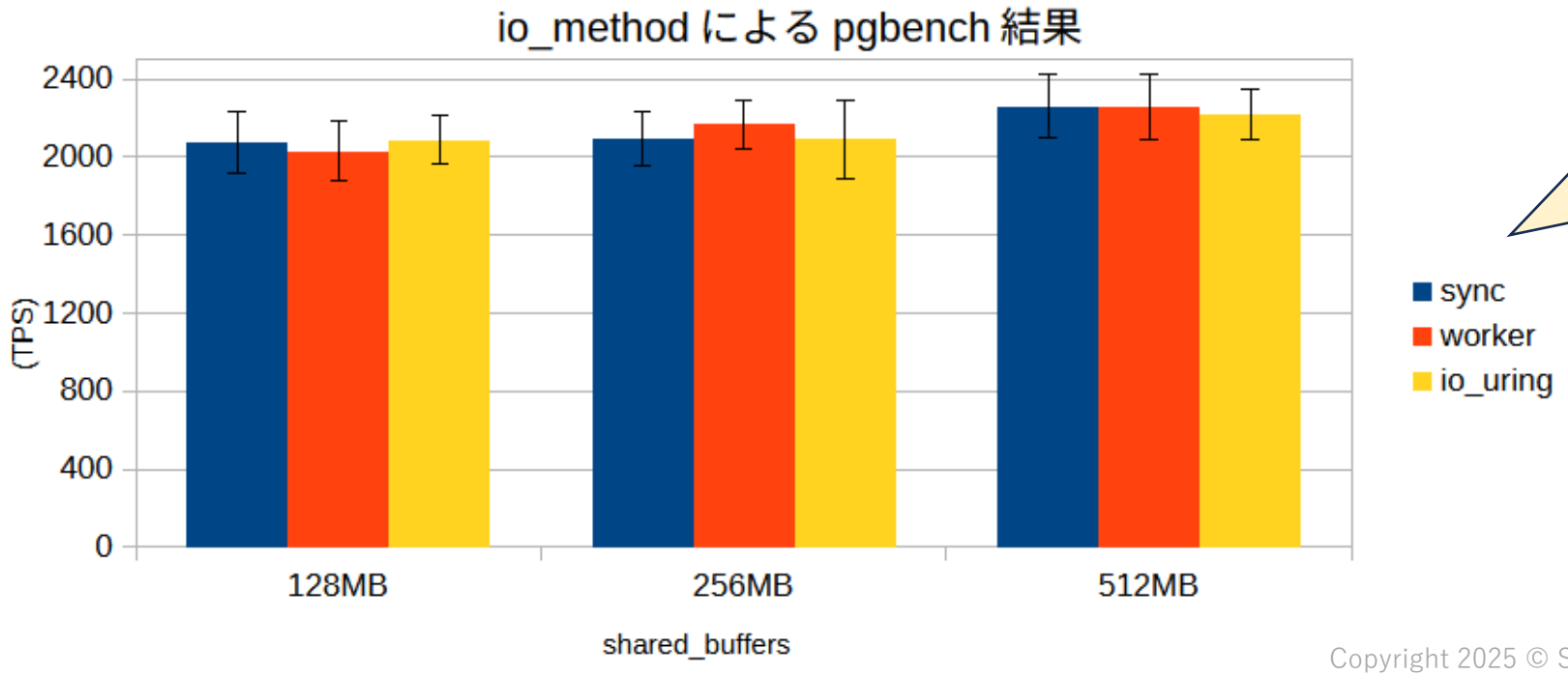
PostgreSQL 18 のエンハンス

- 性能向上
  - 非同期 I/O
  - Skip Scan
  - プランナ改善
  - 多テーブルのロック改善
- SQL機能追加
  - 仮想生成列
  - RETURNING句の拡張
  - UUID v7 関数
  - 照合順序関連の拡張
  - 制約の拡張
  - ラージオブジェクトデフォルト権限
- 運用性向上
  - pg_upgrade 改善
  - SCRAM認証パススルー
  - プランナ統計情報のダンプ
  - EXPLAIN の拡張
  - 論理レプリケーション
    衝突の詳細報告
  - 各種モニタリング拡張

本講演で取り上げる主な項目です。
他にも多数のエンハンスがあります。

【性能向上】非同期 I/O （1）

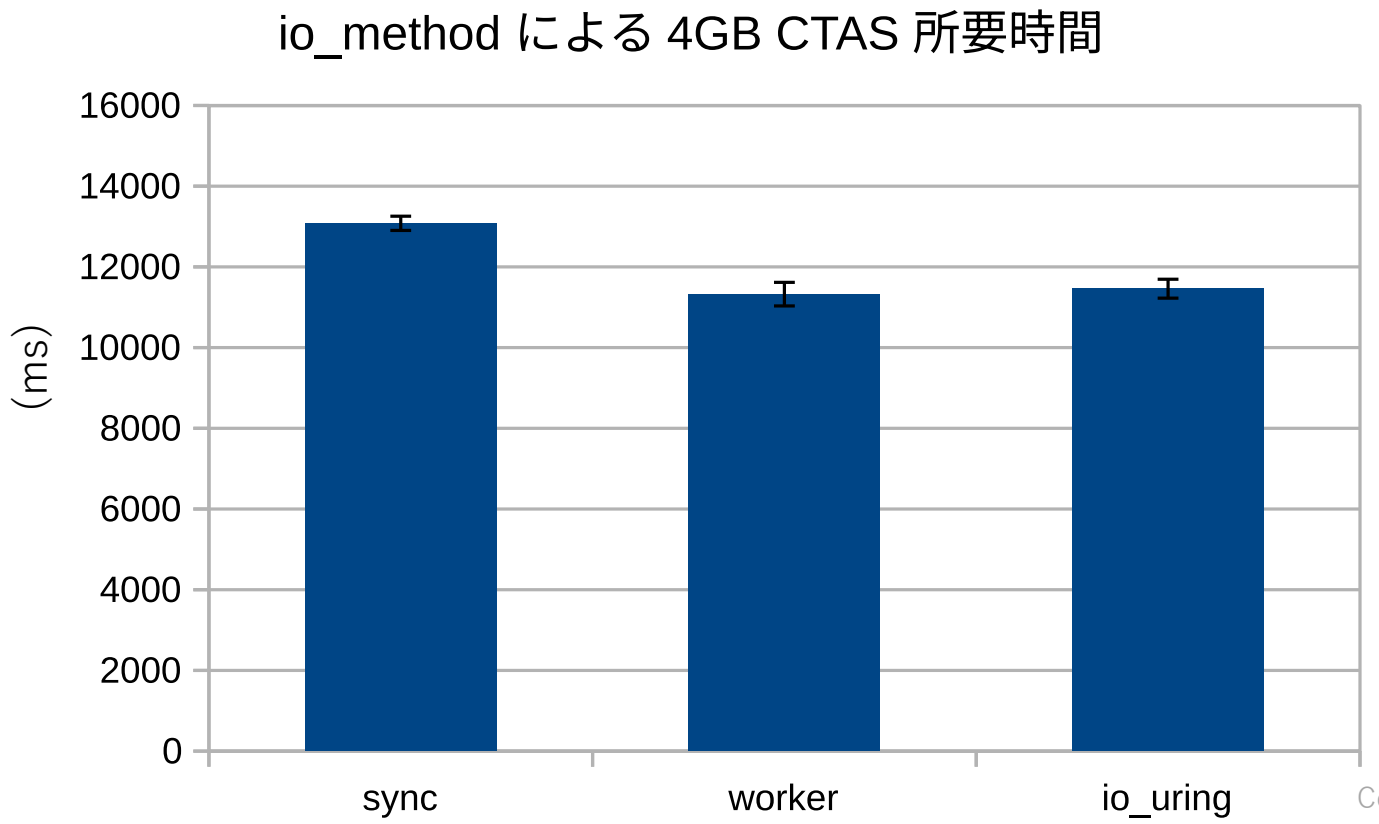
- 新たなファイルアクセスの枠組み
  - sync（従来通り）、worker（ワーカプロセス）、io_uring（Linuxカーネル機能）
  - 今のところ、テーブル・インデックスのファイル読み込みだけで使用
  - pg_aios ビューでブロック読み込み状況を報告



OLTPワークロードでは、特段の性能効果はない

【性能向上】非同期 I/O （2）

- まとまったデータ読み込みで性能向上する
- 高速だが通信往復には時間を要するストレージには特に効果的

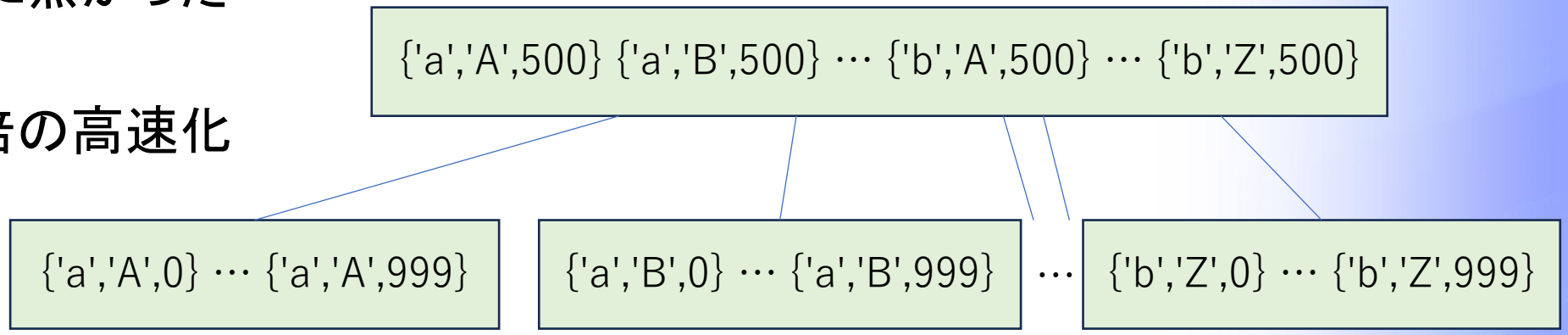


CTAS は「CREATE TABLE AS SELECT」

読み込み処理での性能向上により、worker、io_uring が好結果

【性能向上】Skip Scan

- Btreeインデックスが Skip Scan をサポート
  - 範囲を調べるときに必要箇所だけを走査する
  - 長らく Oracle Database、MySQL 等に在って、PostgreSQLに無かった
  - うまくはまると100倍、1000倍の高速化



複合インデックスの2番目以降の列を条件に検索するとき、インデックスページの読み取り数を少なく済ませられる

```
CREATE INDEX ON tbl1 (c0, c1, c2);

SELECT * FROM  tbl1
  WHERE c1 IN ('A', 'Z') AND c2 BETWEEN 400 AND 500;
```

【性能向上】プランナ改善（1）

- 自己結合の除去

```
db1=# explain SELECT * FROM t413a m, t413a n WHERE m.x = n.x;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=27.50..45.14 rows=1000 width=8)
   Hash Cond: (m.x = n.x)
   ->  Seq Scan on t413a m  (cost=0.00..15.00 rows=1000 width=4)
   ->  Hash  (cost=15.00..15.00 rows=1000 width=4)
         ->  Seq Scan on t413a n  (cost=0.00..15.00 rows=1000 width=4)
(5 rows)
```

同一テーブルの結合であると認識して、余計な処理を割愛している

```
db1=# explain SELECT * FROM t413a m, t413a n WHERE m.x = n.x;
                          QUERY PLAN
------------------------------------------------------------
 Seq Scan on t413a n  (cost=0.00..15.00 rows=1000 width=8)
(1 row)
```

【性能向上】プランナ改善 (2)

- OR句を配列に転換

```
db1=# explain SELECT * FROM t413a
        WHERE x = 1 OR x = 2 OR x = 3 OR x = 5 OR x = 8;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on t413a  (cost=21.42..26.80 rows=5 width=4)
   Recheck Cond: ((x = 1) OR (x = 2) OR (x = 3) OR (x = 5) OR (x = 8))
   ->  BitmapOr  (cost=21.42..21.42 rows=5 width=0)
       ->  Bitmap Index Scan on t413a_pkey  (cost=0.00..4.28 rows=1 width=0)
             Index Cond: (x = 1)
       ->  Bitmap Index Scan on t413a_pkey  (cost=0.00..4.28 rows=1 width=0)
             Index Cond: (x = 2)
                  ⋮
```

同じ列値のOR句の連なりは X = ANY (配列) に書き換えできて、Index Scan が可能

```
db1=# explain SELECT * FROM t413a
        WHERE x = 1 OR x = 2 OR x = 3 OR x = 5 OR x = 8;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Index Only Scan using t413a_pkey on t413a  (cost=0.28..8.63 rows=5 width=4)
   Index Cond: (x = ANY ('{1,2,3,5,8}'::integer[]))
(2 rows)
```

【性能向上】プランナ改善 (3)

- EXCEPT、INTERSECTの改善

```
db1=# explain (SELECT x FROM t413a ORDER BY x)
              EXCEPT (SELECT x FROM t413b ORDER BY x);
                                  QUERY PLAN
-------------------------------------------------------------------------------
 HashSetOp Except  (cost=0.28..99.05 rows=1000 width=8)
   ->  Append  (cost=0.28..94.55 rows=1800 width=8)
       -> Subquery Scan on "*SELECT* 1"  (cost=0.28..45.27
             ->  Index Only Scan using t413a_pkey on t413
              (cost=0.28..35.27 rows=1000 width=4)
       -> Subquery Scan on "*SELECT* 2"  (cost=0.28..40.27
             ->  Index Only Scan using t413b_pkey on t413
              (cost=0.28..32.27 rows=800 width=4)
(6 rows)
```

Subquery Scan を省略

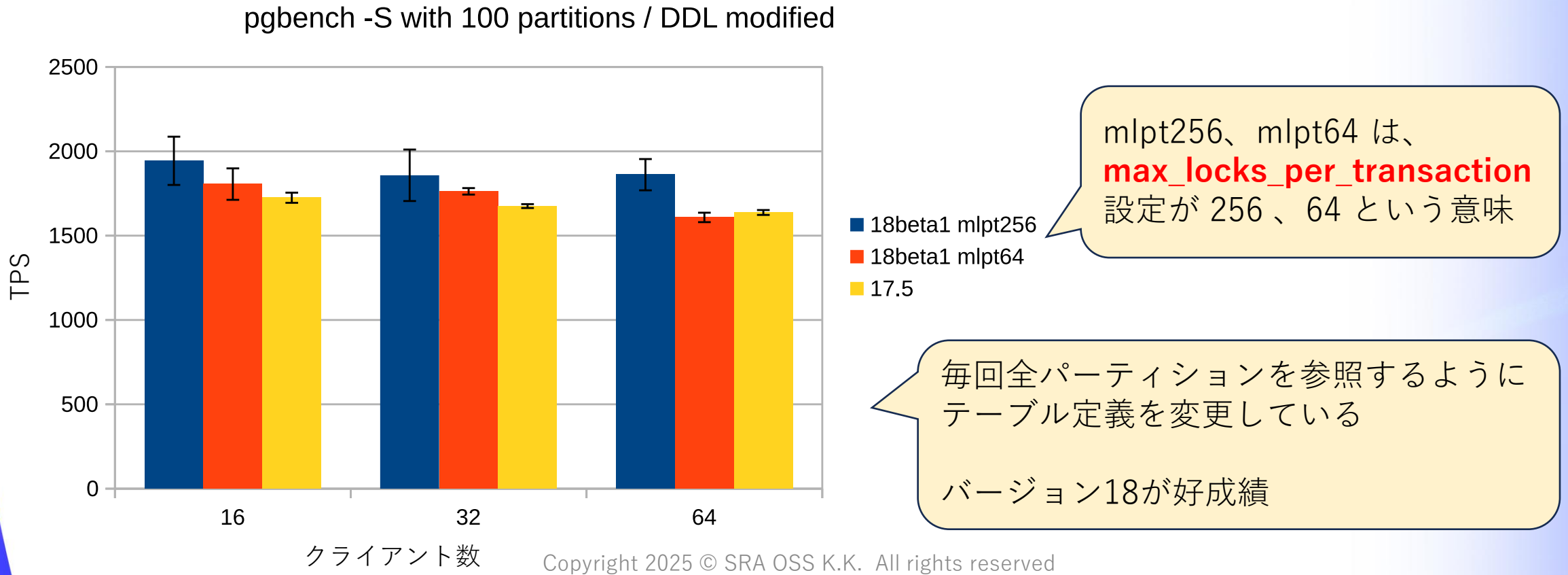
ソート済入力には HashSetOp より SetOp を選好

```
db1=# explain (SELECT x FROM t413a ORDER BY x)
              EXCEPT (SELECT x FROM t413b ORDER BY x);
                                  QUERY PLAN
-------------------------------------------------------------------------------
 SetOp Except  (cost=0.55..74.55 rows=1000 width=4)
   ->  Index Only Scan using t413a_pkey on t413a
       (cost=0.28..35.27 rows=1000 width=4)
   ->  Index Only Scan using t413b_pkey on t413b
       (cost=0.28..32.27 rows=800 width=4)
(3 rows)
```

プランナ改善は他にも数点あり

【性能向上】多テーブルのロック改善

- fast-path ロックの上限数を固定(=16) ではなく設定調整可能に
  - 多テーブルを参照する問い合わせで性能向上



mlpt256、mlpt64 は、
max_locks_per_transaction
設定が 256 、64 という意味

毎回全パーティションを参照するように
テーブル定義を変更している

バージョン18が好成績

【SQL機能】仮想生成列

- 参照時に計算される生成列 – 部分的なビューのようなもの

従来からサポートされていた
STORED な生成列では、
行を挿入・更新した時点で
計算した値を保持する。

```
db1=# CREATE TABLE t_virt(id int PRIMARY KEY, v1 int,
        v2 int GENERATED ALWAYS AS (v1 + 1) VIRTUAL,
        v3 int GENERATED ALWAYS AS (v1 - 1) STORED);
db1=# \d t_virt
                          Table "public.t_virt"
 Column |  Type   | Collation | Nullable |                Default
--------+---------+-----------+----------+---------------------------------------
 id     | integer |           | not null |
 v1     | integer |           |          |
 v2     | integer |           |          | generated always as (v1 + 1)
 v3     | integer |           |          | generated always as (v1 - 1) stored
Indexes:
    "t_virt_pkey" PRIMARY KEY, btree (id)
```

【SQL機能】RETURNING句の拡張

- OLD、NEW エイリアス
  - OLD　更新前の（削除前の）値を返す
  - NEW　更新後の（挿入した）値を返す

```
db1=# UPDATE cities SET city_name = 'Paris' WHERE id = 1
      RETURNING OLD.city_name AS old, NEW.city_name AS new;
  old   |  new
-------+-------
 Tokyo | Paris
(1 row)
```

- これまでは
  - INSERT なら更新後の値
  - UPDATE なら更新後の値
  - DELETE なら削除前の値

NEW、OLD を指定しない場合は旧来通りに動作する

【SQL機能】UUID v7 関数

- UUIDバージョン7
  - タイムスタンプ＋乱数
  - 文字列でソートすると生成順になる

従来の gen_random_uuid() 関数は UUID バージョン4 uuidv4() という関数名で提供

これと別に contrib 拡張の uuid-ossp は引き続き提供

```
(uuidv7()の単純な使用)
db1=# SELECT uuidv7();
                uuidv7
--------------------------------------
 0197afc6-7728-712e-afb2-89602fcd0118
(1 row)

(uuid_extract_timestamp()でUUIDからタイムスタンプを抽出)
db1=# SELECT uuid_extract_timestamp('0197afc6-7728-712e-afb2-89602fcd0118');
   uuid_extract_timestamp
----------------------------
 2025-06-27 14:05:08.904+09
(1 row)
```

【SQL機能】照合順序関連の拡張

- LIKE句で非決定論的照合順序に対応
  - 従来は「ERROR: nondeterministic collations are not supported for LIKE」

```
CREATE COLLATION case_insensitive
  (provider = icu, locale = 'und-u-ks-level2', deterministic = false);

SELECT  'ABCDE' LIKE '%bcd%' COLLATE case_insensitive;  ⇒ t
```

- casefold関数
  - 「大文字小文字を揃える」関数、lower() と似た動作
  - 従来は upper() / lower() で代用
  - 出現位置による複数通りの小文字や特殊文字での非対応などを回避
    - lower('Σ Σ') ⇒ σς
    - casefold('Σ Σ') ⇒ σ σ

【SQL機能】制約の拡張（1）

- WITHOUT OVERLAPS
  - 排他制約を主キー制約、ユニークキー制約に含めて使用できる

```
（排他制約で使用する btree_gist 拡張を導入）
db1=# CREATE EXTENSION btree_gist;


（「同uid で range が重ならない」条件の主キーを持つ、スケジュールテーブルを作成）
db1=# CREATE TABLE t_schedule (uid int, range tsrange, description text,
        PRIMARY KEY (uid, range WITHOUT OVERLAPS));
```

```
（以前からある排他制約を使った書き方）
db1=# CREATE TABLE t_schedule2
        (uid int NOT NULL, range tsrange NOT NULL, description text,
         EXCLUDE USING gist (uid WITH =, range WITH &&));
```

【SQL機能】制約の拡張（2）

- CHECK制約と外部キー制約に ENFORCED / NOT ENFORCED
  - NOT ENFORCED 制約が無いのと同じ動作

- NOT NULL に制約名
- NOT NULL 制約を
  NOT VALID で作成可に
- NOT NULL 継承指定
  - 「ALTER TABLE ..
    ALTER CONSTRAINT ..
    NO INHERIT」で
    継承対象外

```
db1=# CREATE TABLE t436p (id int PRIMARY KEY, v text NOT NULL);
db1=# \d+ t436p
                                                Table "public.t436p"
 Column |  Type   | Collation | Nullable | Default | Storage  | Compression |
 Stats target | Description
--------+---------+-----------+----------+---------+----------+-------------
+--------------+-------------
 id     | integer |           | not null |         | plain    |             |
           |
 v      | text    |           | not null |         | extended |             |
           |
Indexes:
    "t436p_pkey" PRIMARY KEY, btree (id)
Not-null constraints:
    "t436p_id_not_null" NOT NULL "id"
    "t436p_v_not_null" NOT NULL "v"
Access method: heap
```

【SQL機能】ラージオブジェクトのデフォルト権限

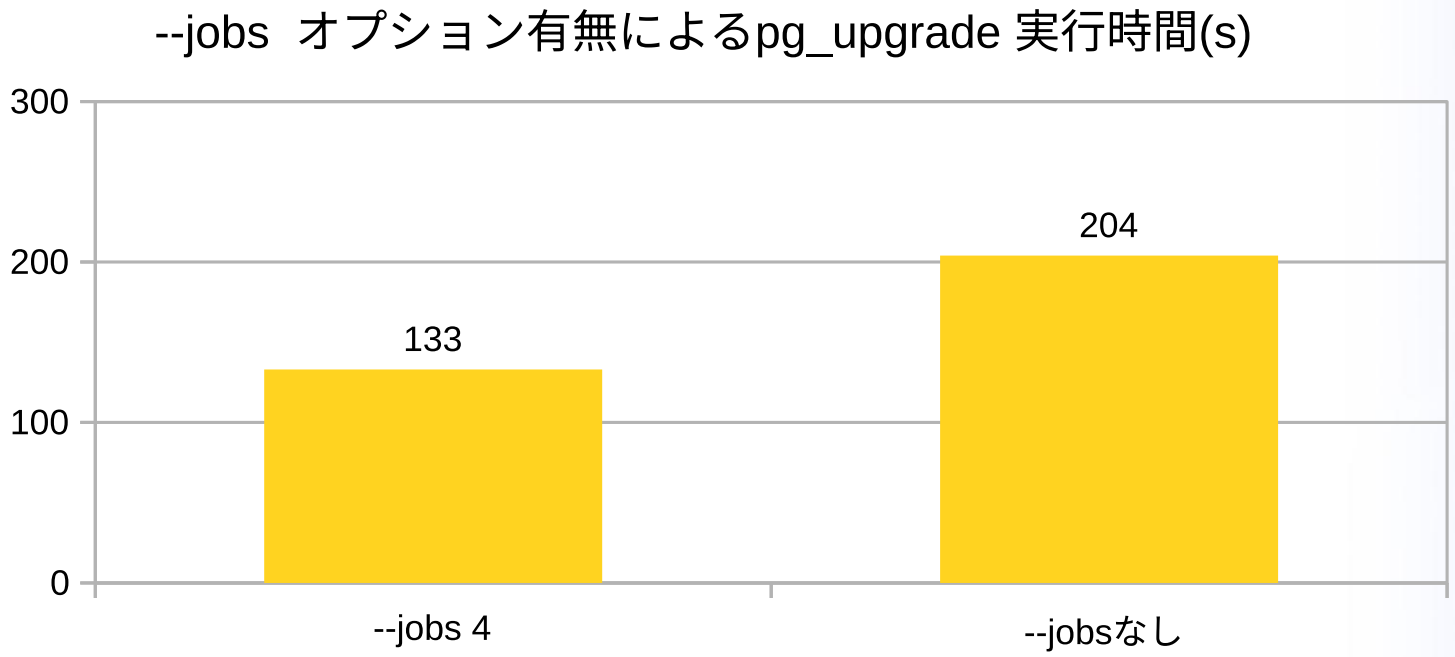
- ALTER DEFAULT PRIVILEGES 文がラージオブジェクトに対応
  - オブジェクト(テーブル等)を作ったときのアクセス権限のデフォルトを指定

```
ALTER DEFAULT PRIVILEGES FOR USER foo
  GRANT SELECT ON LARGE OBJECTS TO bar;
```

ユーザー foo がラージオブジェクトを作ったとき、
デフォルトでそのラージオブジェクトの参照権限をユーザ bar に付与

【運用性向上】pg_upgrade改善（1）

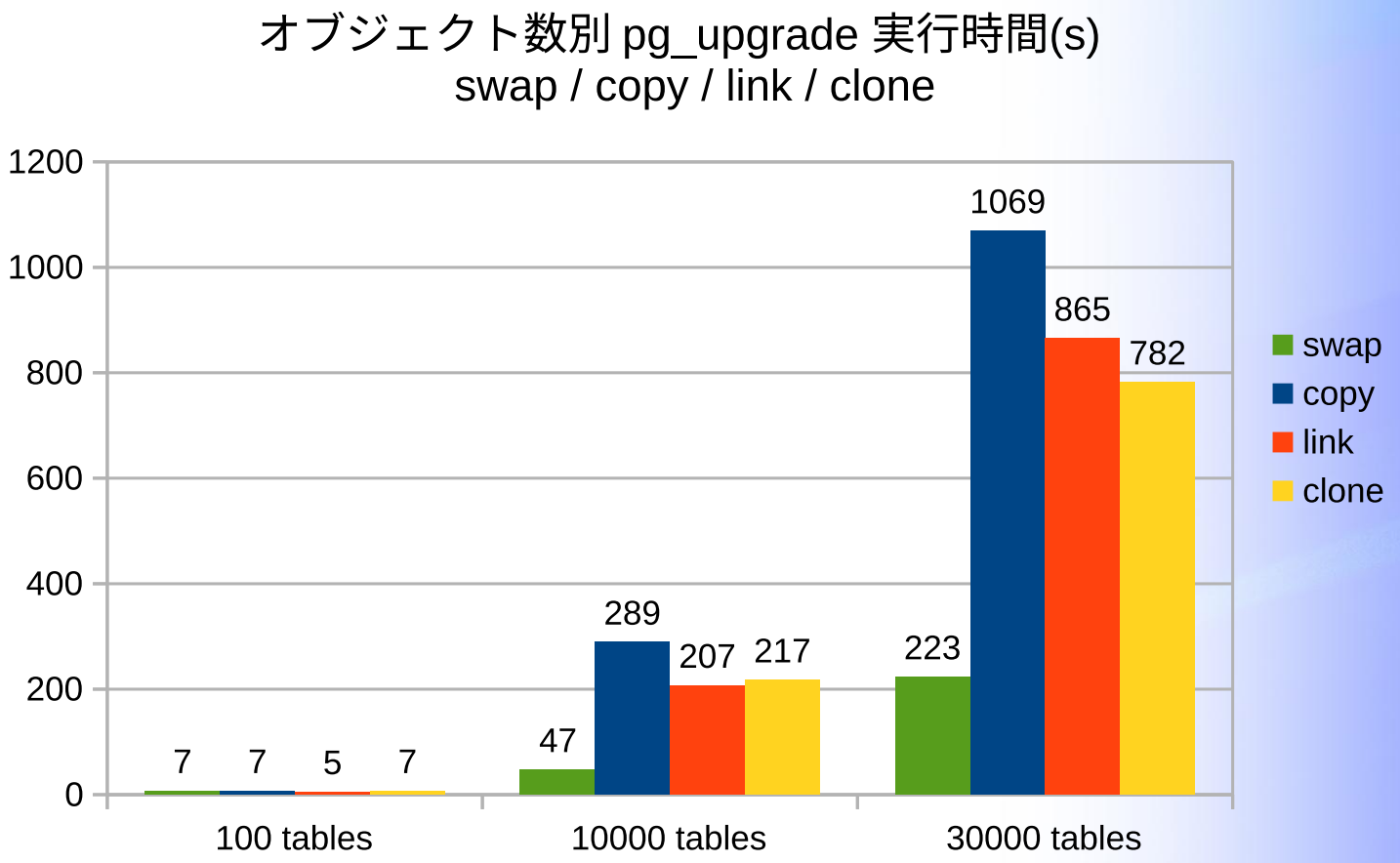
- メジャーバージョンアップのためのツールコマンド
- pg_upgrade の機能追加
  - プランナ統計情報の移行（アップグレード後の ANALYZE が不要に）
  - データベース毎のチェック処理についても並列化



100データベース、
100テーブルスペース、
でテストした場合

【運用性向上】pg_upgrade改善（2）

- pg_upgrade の機能追加
  - --swapオプションの追加
  - 新バージョンに
    テーブルファイルを
    用意する新たな方法
  - ディレクトリ全体を
    差し替えるため高速、
    多テーブルのとき効果的



【運用性向上】SCRAM認証パススルー

- postgres_fdw、dblink の新たな認証方式
  - ローカル側 PostgreSQL への接続で使った scram-sha-256 認証情報を、そのままリモート側 PostgreSQL への接続で使用
  - USER MAPPING にパスワード文字列を設定しなくてすむ

  - FOREIGN SERVER の新オプション use_scram_passthrough 'true'
  - データベース接続で使うユーザについて、pg_authid テーブルの rolpassword 列の値をローカルとリモートで同一にする

```
db1=# ALTER ROLE foo PASSWORD 'SCRAM-
SHA-256$4096:MXBDm4NWwh4Q45iPEuujjg==$P1szXKsLJzUzm4lZp1jsshSeK73QtwiFQ8U4Rv
U+oHk=:RuqLICsNXeLooVi2Yj4w9gSzVU1hOwSWm2fAjTKcKio=';
```

【運用性向上】プランナ統計情報のダンプ／リストア

- pg_dump でデフォルトでプランナ統計情報も含まれる

- プランナ統計情報だけのダンプ、リストアが可能
  - 「--statistics-only」オプション
  - 応用例：
    - プラン変動を避けたい SQL を実行する前に、問い合わせ対象テーブル群について、毎回プランナ統計情報をリストアする

【運用性向上】EXPLAINの拡張

- BUFFERSオプションがデフォルト有効
  - ANALYZEオプションが指定されている場合
- インデックス参照回数
- enable_* = off を明示
  - 「Disabled: true」
  - 従来はコスト値増大が報告されていた

```
db1=# explain analyze SELECT * FROM t452 WHERE id IN (100, 1000, 10000);
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using t452_pkey on t452  (cost=0.29..16.93 rows=3 width=9)
                           (actual time=0.034..0.053 rows=3.00 loops=1)
   Index Cond: (id = ANY ('{100,1000,10000}'::integer[]))
   Index Searches: 3
   Buffers: shared hit=9
 Planning Time: 0.118 ms
 Execution Time: 0.074 ms
```

プランナとしても
コスト加算で判断する
方式でなくなった

【運用性向上】論理レプリケーション衝突報告

- コンフリクト発生時のログが詳細化

```
2025-07-03 09:51:42.077 JST [1537] ERROR:  conflict detected on relation
"public.t455": conflict=insert_exists    ←コンフリクト種別
2025-07-03 09:51:42.077 JST [1537] DETAIL:  Key already exists in unique
index "t455_pkey", modified locally in transaction 111972803 at 2025-07-03
09:50:52.060684+09.    ← 競合する行の作成トランザクションIDや作成日時を含む報告
      Key (id)=(101); existing local tuple (101, X); remote tuple (101, Y).
2025-07-03 09:51:42.077 JST [1537] CONTEXT:  processing remote data for
replication origin "pg_38779" during message type "INSERT" for replication
target relation "public.t455" in transaction 111972799, finished at
42/D5018EA0    ← オリジン名、処理種別、テーブル、トランザクションID、LSN を報告
2025-07-03 09:51:42.077 JST [1462] LOG:  background worker "logical
replication apply worker" (PID 1537) exited with exit code 1
```

- 停止を要さない場合の LOG報告
  - delete_missing
  - update_missing
  - delete_origin_differs
  - update_origin_differs

→ これらの報告には track_commit_timestamp = on 設定が必要

【運用性向上】各種モニタリングの拡張

- pg_stat_*_tables で VACUUM、ANALYZE 総所要時間
- ログ と pg_stat_progress_* にVACUUM、ANALYZE 休止時間
  - track_cost_delay_timing = on で有効化
- プロセス毎の I/O量、WAL量の統計
  - pg_stat_get_backend_io(pid) 関数
  - pg_stat_get_backend_wal(pid) 関数
  - pg_stat_reset_backend_stats(pid) 関数（リセット用）
- pg_stat_io ビューが項目拡充
  - pg_stat_wal から wal_write、wal_sync、wal_write_time、wal_sync_time が移転

非互換の変更点

非互換変更点（1）

- タイムゾーン省略形の動作変更
  - 二つの定義元：
    ① timezone_abbreviations設定（と timezonesets/ の時間帯省略形ファイル）
    ② IANA時間帯データベース（timezone/ 以下のファイルで定義）
  - 従来は省略形について ① だけ使われていた
    → これからは「② IANA時間帯データベース」が優先

そのために、生成した省略形を解釈できないエラーケースが発生

```
db1=# SELECT '1000-01-01'::timestamptz::text::timestamptz;
ERROR:  invalid input syntax for type timestamp with time zone: "Wed Jan 01
00:00:00 1000 LMT"
```

非互換変更点 (2)

- パーティション／継承テーブルに対する VACUUM / ANALYZE
  - ONLYオプション追加

対象／コマンド	VACUUM	ANALYZE
継承ツリーの親テーブル	**PG17:** 子テーブルは処理対象外	**PG17:** 子テーブルは処理対象外
	PG18: 子テーブルも処理対象、 ONLY指定で子テーブルが処理対象外	**PG18:** 子テーブルも処理対象、 ONLY指定で子テーブル処理が対象外
パーティションテーブル	**PG17:** 子テーブル（パーティション）が処理対象	**PG17:** 子テーブル（＝パーティション）も必ず処理対象
	PG18: ONLYを指定すると何もしない （ONLY指定無しなら従来通り）	**PG18:** ONLYを指定すれば、親テーブルのみを処理対象にできる

非互換変更点 (3)

- COPY の EOFマーカーの動作変更
  - CSV形式のファイルから読む場合に ¥. は通常文字列扱い
    （STDIN から読む場合には EOFマーカー扱いされる）
  - 行末に書く ¥. は通常文字列扱い
- UNLOGGEDパーティションテーブル禁止
- md5パスワードが非推奨に
- RULE権限の完全廃止
- 外部キー制約で非決定論的照合順序の使用に制限

まとめ

まとめ

- 今秋リリース予定の PostgreSQL 18 の特徴:
  - 現代的な利用法で生じたニーズに対応したエンハンス
    - クラウドサービスでの提供 → 非同期 I/O
    - 大規模パーティションテーブル利用 → 多テーブル性能改善
    - 分散構成でのニーズ → UUIDv7 、ロジカルレプリケーション衝突報告
  - ユーザ視点での地道な改善の継続
    - プランナ改善、スキップスキャン、ラージオブジェクトデフォルト権限、ほか多数
  - SQL標準対応の継続
    - 仮想生成列、制約の拡張、など

TEL：03-5979-2701
E-mail: sales@sraoss.co.jp
https://www.sraoss.co.jp/