PostgreSQL再入門
～ この10年で何が変わったのか ～

2026年6月27日
OSC2026 Hokkaido

株式会社SRA OSS
松坂 大地

会社紹介

株式会社SRA OSS

所 在 地: 東京都豊島区南池袋2-32-8
設 立 日: 2022年6月17日
株　　主: 株式会社SRA
　　株式会社NTTデータ
資 本 金: 7,000万円
社　　長: 稲葉 香理

事業内容

- オープンソースソフトウェア (OSS) 関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

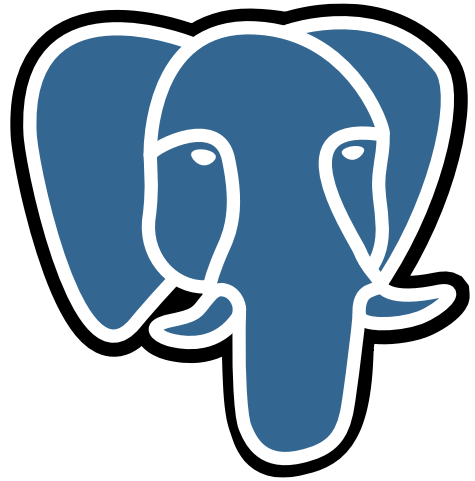
顧　　問: 石井 達夫

技術顧問: 増永 良文 (お茶の水女子大学名誉教授)

PostgreSQLとは

PostgreSQLとは

- ポストグレスキューエルと読む
  - 単にポストグレス、日本ではポスグレとも呼ばれる
- 標準SQLの大部分とそのほかの先進的な機能をサポートする本格的なオープンソースRDBMS
- 30年近くの歴史をもち、現在も活発な開発体制
- 豊富なプラットフォームに対応
  - Unix系OS全般、Windows、macOSなど
- 豊富なサポート言語
  - C, C++, C#, Java, Python, Perl, PHP, Rubyなど



PostgreSQLの特徴

- 対応アーキテクチャ
  - x86, x86_64, SPARC, POWER, ARMなど
- メモリ容量の制限はなし
  - バージョンが新しくなるほど効率的にメモリを使用
- ファイルシステムレベルのI/Oアクセス
- 論理的な容量制限はほぼない
- 追記型による性能特性
  - INSERT、DELETEは (UPDATEに比べ) 速い
  - VACUUMが必要
  - 同時実行でロック待ち少ない
- マルチプロセスモデル

項目	制限
データベースサイズ	制限なし
テーブルサイズ	32TB
行サイズ	400GB
フィールドサイズ	1GB
テーブルの行数	制限なし
テーブルの列数	250-1600

PostgreSQLのライセンス

- The PostgreSQL License
- 広告条項はなしの修正BSDライセンスと同様
  - 使用、複製、改変、配布の自由
  - 複製においては著作権表示、ライセンス条文、免責事項を含めることが条件
- GPLと比較すると、派生物を再配布する際にソースコード公開、ライセンス継承の義務がない

COPYRIGHTファイル

PostgreSQL Database Management System
(formerly known as Postgres, then as Postgres95)

Portions Copyright (c) 1996-2016, PostgreSQL Global Development Group

Portions Copyright (c) 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

PostgreSQLのバージョン番号

- PostgreSQLのバージョンは6.0から始まる
  - POSTGRESがバージョン4.2まで（1986～1994）
    Postgres95をバージョン5と見なした（1994～1995）
- メジャーバージョンでは機能追加・仕様変更、
  マイナーバージョンではバグ修正
- バージョン10以降、メジャーバージョンを表す区切りが変わった

PostgreSQLのリリースサイクル

- マイナーバージョンは四半期ごと (5, 8, 11, 2月) + 緊急時
- メジャーバージョンは毎年第3四半期 (7-9月) にリリース
- メジャーバージョンはリリースから5年間サポート

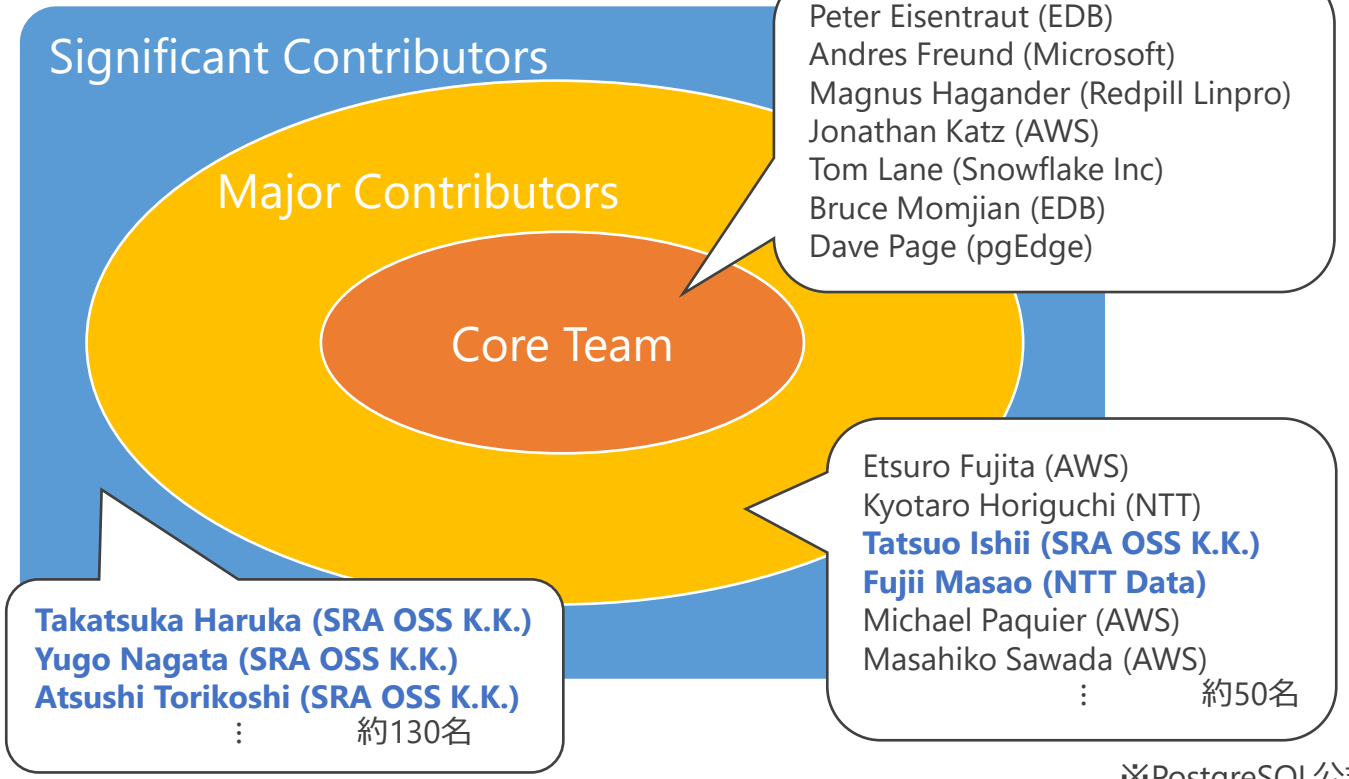
バージョン	サポート可否	初リリース	最終リリース
18	○	2025/9/25	2030/11/14
17	○	2024/9/26	2029/11/8
16	○	2023/9/14	2028/11/9
15	○	2022/10/13	2027/11/11
14	○	2021/9/30	2026/11/12
13	×	2020/9/24	2025/11/13

※PostgreSQL公式サイト Versioning Policyより (2026年6月現在)

PostgreSQLの開発体制

- 特定のオーナー企業をもたない方式
  - 単独の企業に独占されることなく、幅広い層の厚い開発体制

開発コミュニティ



支援企業

Major Sponsor

FUJITSU, Microsoft, aws, PGX POSTGRESQL EXPERTS, INC., crunchy data, PostgresPro, EDB POSTGRES AI, SRA OSS, NTT Group, CONOVA, data egret, Timescale, CYBERTEC, DALIBO, Redpill Linpro

※PostgreSQL公式サイト Contributor Profiles、Sponsorsより (2026年6月現在)

日本のPostgreSQLコミュニティ

ユーザコミュニティ
日本PostgreSQLユーザ会 (JPUG)

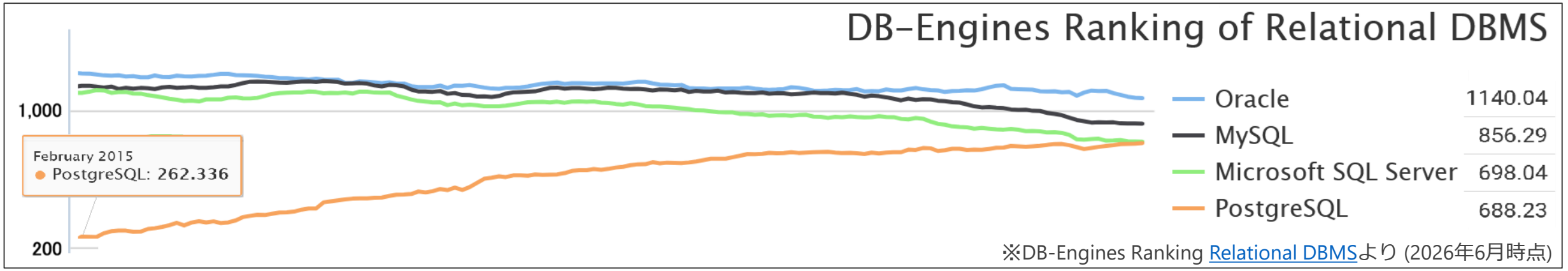
- 1999年設立、2006年NPO法人化
- 全国各地に支部
- 普及活動を中心に、マニュアル日本語訳やカンファレンスを実施

ビジネスコミュニティ
PostgreSQLエンタープライズ・コンソーシアム (PGECons)

- 2012年設立
- 正会員企業16社、一般会員企業68社 (2026年4月現在)
- エンタープライズ用途に向けた共同検証やプロモーション活動

RDBMS市場でのPostgreSQLの位置づけ

- DB-Engines Rankingによる人気ランキングでは
  - Oracle、MySQL、SQL Serverに次ぐ第4位（5位:210ポイント程）
  - ここ十数年、高い伸び率を見せている



※DB-Engines Ranking Relational DBMSより (2026年6月時点)

- Oracleの移行先として選ばれている
  - MySQLに比べると、機能が豊富で移行しやすい
- Oracle以外とは棲み分け
  - MySQLはシンプルなWebシステム向け
  - SQL ServerはWindows向け

ポスグレって HINT句
ってないの？

ポスグレって MERGE文
ってないの？

パーティション機能って
ポスグレはないの…？

10年前と変わった点

～トレーニング講師の時によく聞かれたこと～

レプリケーション
使いたくて来ました

バージョンアップの
案件がありまして…

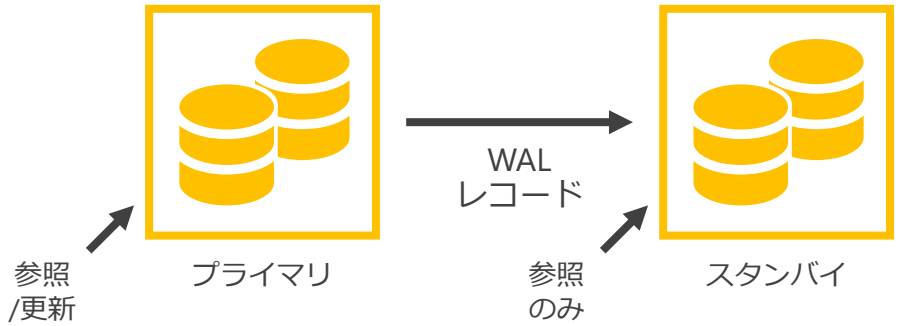
情報収集って
これだけですか？

ポスグレってプロセスモデル
なんでしょ？マルチコア
活かせないんじゃないの？

ストリーミングレプリケーション①

レプリケーション
使いたくて来ました

- ストリーミングレプリケーションとは
  - PostgreSQL 9.0（2010年）に誕生
  - 8.x の PITR をベースに開発
    - PITRではデータの変更内容が記録された16MBの WAL segment file 単位でアーカイブされたものをリカバリする機能
      - ✕ 最新状態までリカバリできない
  - ストリーミングレプリケーションではWALをレコード単位で転送
    - スタンバイへも参照クエリを実行できる



ストリーミングレプリケーション②

Before

15年ほど前…
ちょうど私が一番
登壇していた時代

- 9.0 - 誕生、Hot Standby により待機系に参照クエリ実行可能
- 9.2 - カスケード構成が可能に、pg_basebackup 強化（backup+WAL stream）
  - 実用化を開始したかに見えたが…
    - ✕ WAL消失問題 – スタンバイの遅延でWALが消える
    - ✕ Promoteしたら孫スタンバイが新timelineを追えない

バージョンアップで
改修されるが、
毎回新たな問題が
生まれていた

⇒ base backup から構築し直すしかなかった
※でも他のHA製品は高価なので、頑張って使おうとしていた

After

- レプリケーションスロット（待機系に確実にWALを届ける仕組み）
- pg_rewind （WAL巻き戻しにより旧プライマリをスタンバイ化）

⇒ 機能が充実、フェイルオーバー運用が容易に
柔軟なスタンバイ構成にも対応

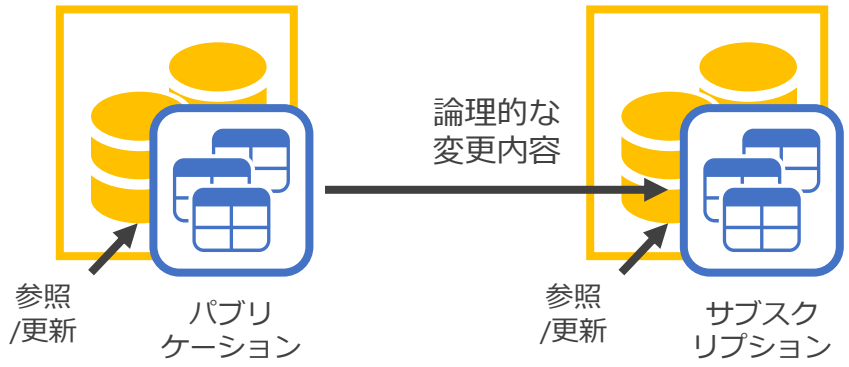
WALが途切れなく
なって実用レベルに

- ロジカルレプリケーション機能も登場

ロジカルレプリケーション

After

- ロジカルレプリケーションとは
  - PostgreSQL 10（2017年）に誕生
  - WALからデコードされた論理的な変更内容を転送
  - テーブル単位から可能　（SR: データベースクラスタ単位）
  - 異バージョン／アーキテクチャ間でも可能　（SR: 不可）
  - 複製先でも更新が可能　（SR: 不可）
- 用途
  - データ配布
    - 受注システム⇒メール配信システムに顧客テーブルだけ配布
    - 本番DB⇒分析DB に必要なテーブルだけデータ配布
      （分析DBで INDEX 追加、BIツール接続などが可能）
  - メジャーバージョンアップ
  - 双方向レプリケーションの基盤

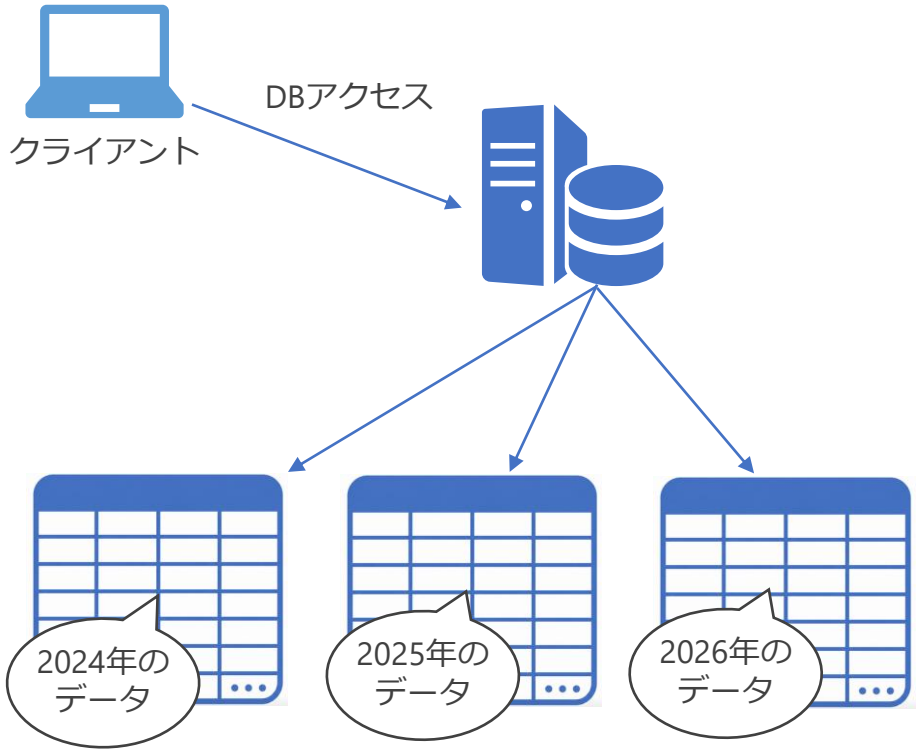


パーティショニング①

パーティション機能って
ポスグレはないの…？

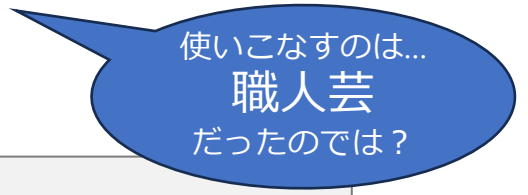
- パーティショニングとは
  - １つの巨大なテーブルを、複数の小さな単位に分割して管理
  - クライアント向けには1つのテーブルに見せる

  - 目的：
    - 検索の高速化
    - メンテナンスの容易化
    - バックアップやアーカイブ
    - VACCUM / REINDEX の負荷分散
      - 特にPostgreSQLでは重要



Before

- テーブル継承、CHECK制約、トリガーなどの組み合わせ
  - パーティション機能として提供されているものではなかった
- 構築時のSQL文が長文になりがち（初心者には辛い）
  - それを、cron で自動作成されるよう頑張っていた




```
CREATE TABLE sales_201604 (
  PRIMARY KEY (id),
  FOREIGN KEY (customer_id) REFERENCES customer (id),
  FOREIGN KEY (item_id) REFERENCES item (id),
  CHECK (time >= ‘2016-04-01’ AND time < ‘2016-05-01’)
) INHERITS (sales);
CREATE INDEX sales_201604_time_idx ON sales_201604(time);
CREATE RULE sales_insert_201604 AS
  ON INSERT TO sales WHERE time >= ‘2016-04-01’ AND time < ‘2016-05-01’
  DO INSTEAD INSERT INTO sales 201604 VALUES (new.*);
```

- テーブルを跨いだ UNIQUE 制約が付けられない
- Prepared Statement だと効果が得られない
  プラン作成時点ではどのパーティションか不明なため

パーティショニング③

After

- PARTITION 句により
  格段に使いやすくなった

```
CREATE TABLE sales ( … ) PERTITION BY RANGE (time);

CREATE TABLE sales_202604 PERTITION OF sales
  FOR VALUES FROM (‘2026-04-01’) TO (‘2026-05-01’);
```

現在は
簡単に使える

- 親テーブルに CREATE INDEX しておけば、子テーブルにも作られる
- Partition Pruning 導入
  - プラン作成時だけでなく、SQL実行段階で不要な子テーブルを除外
    - 以前はCHECK 制約で不要な子テーブルを除外していた
  - Prepared Statement でも効く
  - INSERT/COPY 文だけでなく、MERGE 文にも対応
  - UPDATEで Partition 移動する
  - Attach/Detach 時も Lock 改善で高速に
  - VACUUM/ANALYZE 改善

⇒ 数千Partitionでも耐えられる本格的な機能に生まれ変わった

パラレルクエリ

Before

- PostgreSQL ＝ multi-process model

top 見ても CPU 1個しか使ってないな...

After

- CPUを複数使って並列処理
  - Parallel Seq Scan
  - Parallel Aggregate
  - Parallel Index Scan
  - Parallel Bitmap Heap Scan
  - Parallel Hash Join
    - 各Workerが共有 Hash Table を作成、メモリ効率 & JOIN性能UP

```
EXPLAIN ANALYZE
SELECT count(*) FROM huge_table;
```

```
Finalize Aggregate
  -> Gather
       Workers Planned: 4
       -> Partial Aggregate
            -> Parallel Seq Scan on huge_table
```

おぉ！！
PostgreSQLが
CPU全部使ってる

⇒ 巨大データでも使えるDBへ進化

ポスグレって HINT句
ってないの？

Before

- PostgreSQL コミュニティとしては、Oracle のような Hint 句は PostgreSQL 本体には入れない方針だった
  （理由）
  - SQLが DBバージョン／データ量 依存になる
    - 今日正しいHint句が、次バージョンでは正しいと限らない
      （プランナが改善されたり、データ量が大きく変わるとメンテナンスが必要になる）
  - 根本原因を調査しなくなる
    - SQLの遅い原因である、統計情報不足、インデックス設計不足、SQL記述ミスなどを調査せず、単に「ここはHashJoinで」などとHINTを書きごまかしてしまう

⇒ とは言っても、利用者としては、今このクエリが遅いのを何とかしたい…という現場では、どう対応していたのか…

HINT句②

Before

- HINT句がない代わりにプランナのコストを調整していた
  - JOIN 方式／Scan方式を禁止
    - 該当のクエリ直前でSET文発行…

```
SET enable_seqscan = off;
SET enable_nestloop = off;
SET enable_hashjoin = off;
SET enable_mergejoin = off;
```

  - 統計情報を増やす
    - 全体的にANALYZEの頻度を調整したり
    - 特定のカラムだけサンプリング数を増やしたり

```
ALTER TABLE t ALTER COLUMN c
  SET STATISTICS 1000;
```

  - random_page_cost 設定を調整したり
    - ディスクへのランダムアクセスがシーケンシャルアクセスに比べて何倍と見積もるか（ちょうど、SSD が増えてきた時代だったため重要だった）
- それでもだめなら pg_hint_plan
  - NTT OSSセンタさんが開発しているサードパーティ

HINT句③

After

- HINT句を本体に入れない方針は10年前と変わらず
  - メンテナンスなしに、その時々で最速プランを選択／実行を目指す
- プランナの精度を上げる機能が充実
  - 拡張統計（Extended Statistics）
  - パラレルクエリ改善
  - JIT 導入で実行速度そのものを改善
  - パーティションプルーニング改善（Partition Pruning）
  - Memoize – 検索結果をキャッシュ、Nested Loop の弱点軽減 …など

pg_hint_plan で
頑張っていた部分
が不要に

- pg_plan_advice 登場

V19

  - プランナが作成した良いプランを “Advice” として保存、次回利用する
  - 本体に入ったわけではなく、contribモジュール

Before

- PostgreSQL9.2 当時の JSON型は内部的には、ほぼ text 型

After

- パース済みのバイナリ形式、GIN Index で高速検索
- JSON Subscript で配列のようなアクセス
- 部分更新可能に
- SQL標準対応

⇒ 検索キーは列、変動データはJSONB という設計でもそこそこ使えるDBへ

```
SELECT data['address']['city'] FROM users;
```

```
UPDATE users SET data = jsonb_set(
    data, '{address, city}', '"Tokyo"'
);
```

```
CREATE TABLE orders (
    id SERIAL,
    customer_id int,
    details jsonb
);
```

UPSERT と MERGE ①

ポスグレって MERGE文
ってないの？

- INSERTの競合解決
  - もし、ID:1 のデータが存在していたら UPDATE
    存在していなかったら INSERT したい
  - MySQL には UPSERT、
    Oracle には MERGE があったが…

```
UPDATE users SET (name = ‘Alice’) WHERE id = 1;
```

```
INSERT INTO users(id, name) VALUES (1, ‘Alice’);
```

Before

- PostgreSQL には機能がなかった
  - PL/pgSQL を使って頑張って IF 文書いてみたり…
  - アプリから SELECT … FOR UPDATE ⇒ INSERT／UPDATE を実装したり…
    - それでも、存在しない行はロックできなくて困ったり…
    - 例外処理を頑張って書いたり…

UPSERT と MERGE ②

After

```
INSERT INTO users(id, name) VALUES (1, ‘Alice’)
  ON CONFLICT (id) DO UPDATE
  SET name = EXCLUDED.name;
```

- UPSERT (INSERT … ON CONFLICT)
  - UNIQUE 制約ベースの機能
  - 各トランザクションは UNIQUE INDEX に仮INSERT
    - MVCC的に未来のトランザクションは見えないが、 UNIQUE INDEX を使って競合を確認
      競合の場合、先行トランザクションのCOMMIT結果を待ってから INSERT/UPDATE 判断
  - 並列実行に強く、シンプルで軽量、Serializable に動作

```
MERGE INTO users u USING staging_users s ON u.id = s.id
  WHEN MATCHED THEN
    UPDATE SET name = s.name, updated = now()
  WHEN NOT MATCHED THEN
    INSERT (id, name) VALUES (s.id, s.name);
```

- MERGE
  - 標準SQLのため実装された
  - JOIN ベース、表全体をマージ同期するような使い方に向いている
  - DELETE も書ける

```
  WHEN MATCHED AND s.deleted = true THEN DELETE
```

メジャーバージョンアップ手順

Before

- 基本は、旧環境で pg_dump ⇒ 新環境で pg_restore / psql –f
  - 時間はかかるが、確実でトラブル少ない
  - サービス停止が必要
- pg_upgrade
  - テーブル／インデックス／システムカタログ を新バージョン向けに変換
  - サービス停止必要、短時間で終わる
  - 実績不足、本番ではあまり使いたくない

```
pg_upgrade ¥
  --old-bindir=/usr/pgsql-16/bin ¥
  --new-bindir=/usr/pgsql-18/bin ¥
  --old-datadir=/pg16/data ¥
  --new-datadir=/pg18/data
```

After

- pg_upgrade が主流に
  - DBクラスタのハードリンクオプションやCoWクローンオプションも強力
  - 停止時間が許容できなければ、ロジカルレプリケーションを検討

統計情報ビュー

情報収集って
これだけ？

Before

- 統計情報ビュー(pg_stat_*) や OSコマンドにて手動で収集が必要
- pg_statsinfo が重宝していた
  - PostgreSQLの稼働状況やOSリソースを定期的に蓄積し可視化
  - サードパーティ製 (NTT OSSセンタ)

After

- 統計情報ビューが充実
  - pg_stat_progress_vacuum: 長時間VACCUMが可視化、進捗がわかる
  - pg_stat_create_index: INDEX作成も可視化, CONCURRENTLYも見える
  - pg_stat_wal: WALの生成量、ディスクへの書き込み回数など
  - pg_stat_io: バックエンドごとに I/O が可視化

autovacuum

- VACUUM とは
  - データの更新／削除を行うと発生する古いデータを削除する操作
  - autovacuum はこれを自動で行う機能

Before

- autovacuumは有効、手動でも定期的に実行が必要

After

- autovacuum に任せておけば基本的にOK
  - 並列化や最適化が進み、VACUUM 不足がほとんど起きなくなった

まとめ

変わった点／変わってない点

- １０年前と変わった点
  - レプリケーション、パーティショニング、パラレルクエリなど組み込み機能が大幅に強化された
  - 性能や運用性が向上し、より大規模システムへ対応できるよう進化
- １０年前と変わってない点
  - MVCCベース主義
    - 更新前後の複数バージョンを保持、旧バージョンを残すため VACCUM が必要
    - autovacuum は止めてはいけない
  - WAL中心のアーキテクチャ
    - レプリケーション、PITR、Local Recovery などすべてWALベース
    - WAL容量、archiveを管理する文化
  - HINT句に頼らない、統計情報ベースのプランナ

SRA OSS 技術情報

- SRA OSS Tech Blog
  - https://www.sraoss.co.jp/tech-blog/
  - PostgreSQLのリリースノートの日本語訳など様々なOSS技術情報を掲載中
- SRA OSS 公式 Youtube チャンネル
  - https://www.youtube.com/c/sraoss-official
  - 過去のセミナー動画を公開中

SRA OSS

www.sraoss.co.jp sales@sraoss.co.jp 03-5979-2701