

PostgreSQL12新機能紹介

～Enterpriseの観点より～

2019-10-10

OSC2019 .Enterprise

SRA OSS, Inc. 日本支社

北山 貴広

【講演者】

名前

北山 貴広

所属

SRA OSS, Inc. 日本支社

略歴

2019年6月入社の新入社員（社会人経験31年目）

PostgreSQLとZabbix他種々のOSSサポート業務がメイン

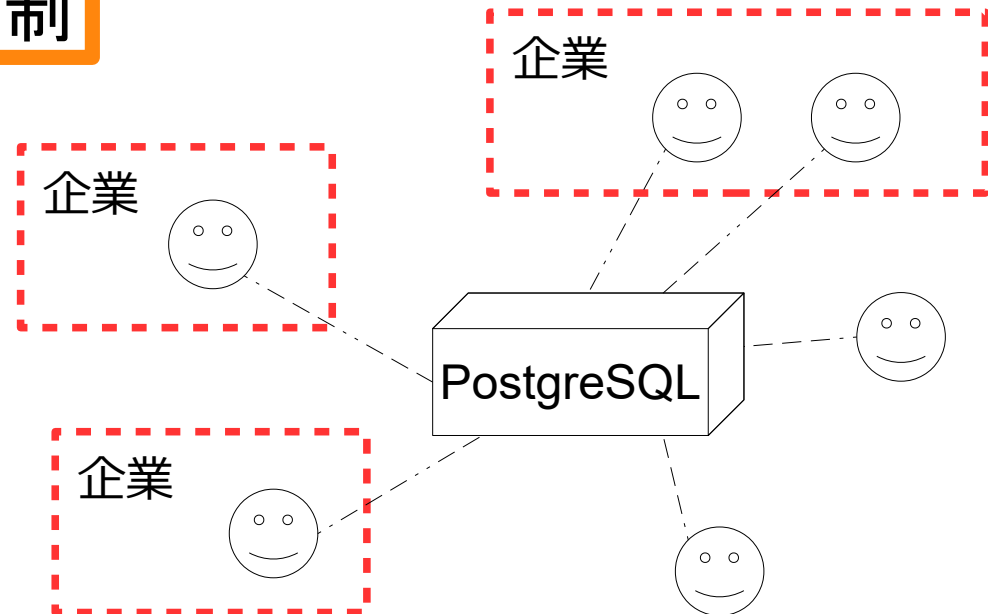
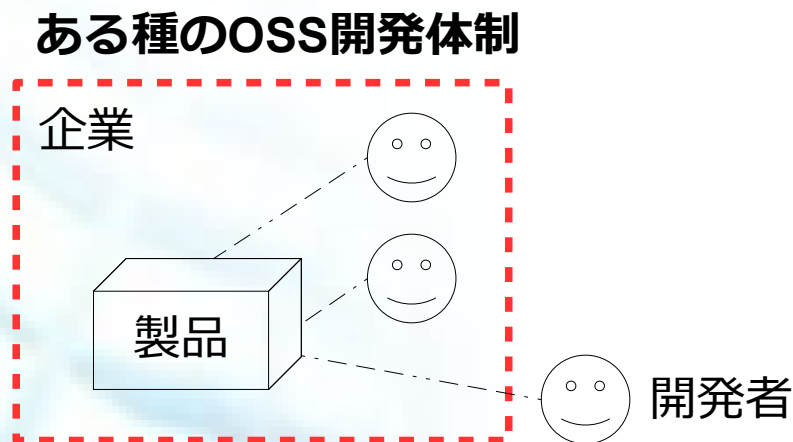
データベーススキルを武器に金融公共系でシステム開発運用経験

過去にSoftware Designでコラム「温故知新 ITむかしばなし」を連載

【本日の講演内容】

- PostgreSQL の概況
- バージョン12新機能をEnterpriseの観点で選定して紹介
- PGEEConsの成果物

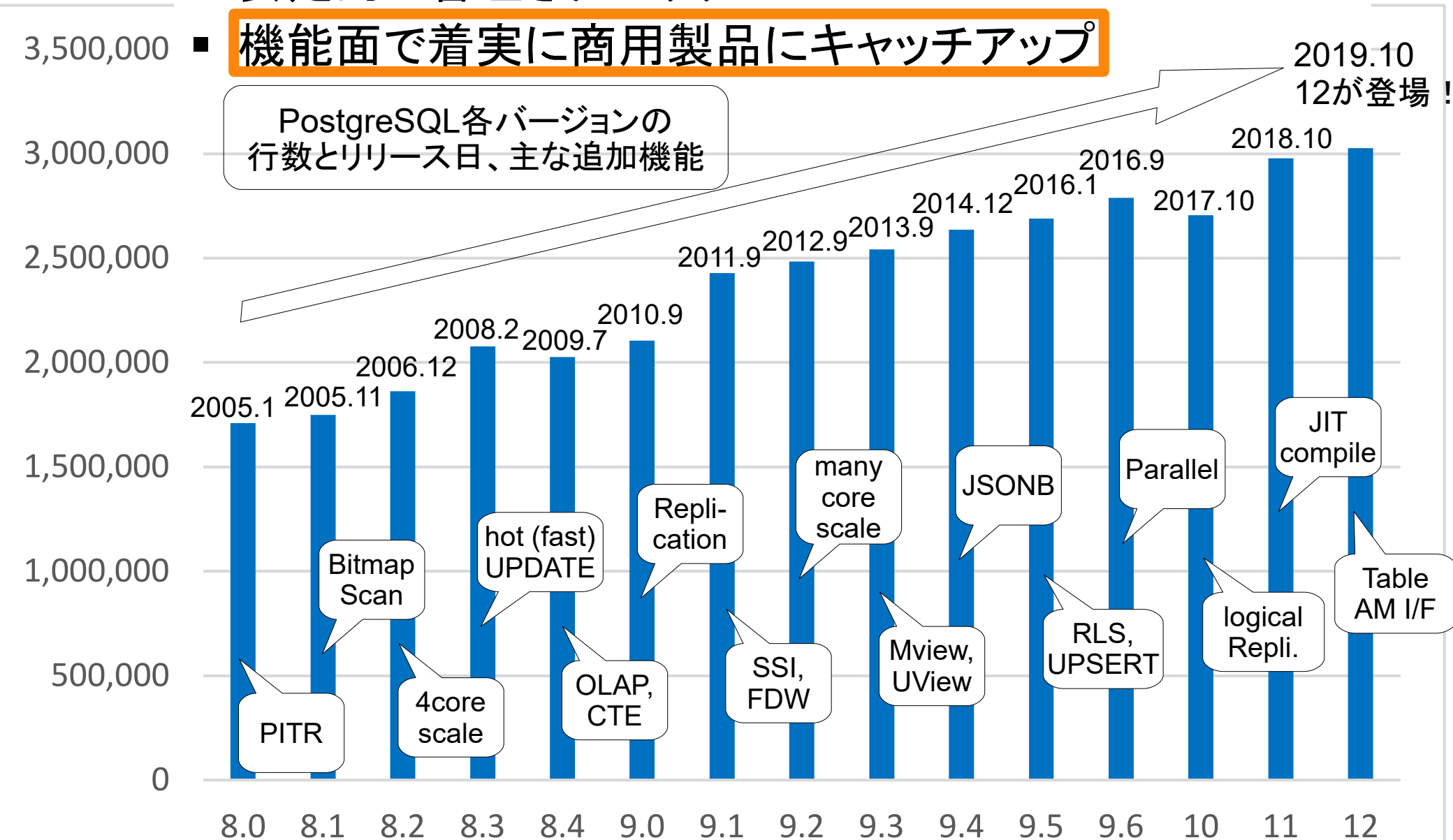
- 多機能、高性能、かつオープンソースの
リレーショナルデータベース管理システム (RDBMS)
- INGRES('70)、POSTGRES('80) 由来の歴史
- 利用、複製、修正、再配布可能なライセンス
- 特定オーナー企業が無い
- 比較的組織だった開発体制



PostgreSQL開発体制

PostgreSQLの概況 (2)

- PostgreSQL 12が2019年10月3日にリリース！
- 安定的で管理されたリリース
- 機能面で着実に商用製品にキャッチアップ



■ 性能面

- • 多数のCPUコアによるOLTP読み書き性能スケール(単純ベンチマーク条件)
- • パラレル問い合わせ/JITに対応 (OLAPむけに)

✗ • 更新でWALボトルネック有

■ 運用機能

- • PITRバックアップ
- • 監査ログ
- • クラウドによるマネージドDB

✗ • フラッシュバック/時系列

■ 機能面

- • SQL SQL:2011 機能のほとんどをサポート
- • JSON対応での優越
- • PostGIS による GIS対応

✗ • マテビュー差分更新など

■ クラスタ構成

- • 論理・物理のレプリケーション
- • アクティブ=スタンバイ構成
- • 参照負荷分散/シャーディング
- • 両方向論理レプリケーション

✗ • 共有ストレージを用いたアクティブ=アクティブ構成(いわゆるRAC型)

- JSON path
- 生成列(式を使って計算される列)
- 大文字小文字を区別しない比較

アプリケーションのための
SQL機能追加。

- インデックスの機能追加・性能改善
- パーティショニングの改善
- プランナのWITH句の最適化
- MCV拡張統計情報

プラン作成や実行処理の
改善を通した、実行時の
性能アップ。

- ページチェックサム改善
- 認証処理の拡充

運用時の利便性向上。
構成設計上の選択肢が
広がる。

- テーブルAMインタフェース追加
- いくつかの仕様変更

— 将来のPostgreSQL拡張
の布石

- JSON Path ... JSON から要素を抽出する記述方式の一つ
 - SQL:2016 に含まれる / PostgreSQL実装は少し異なる
 - jsonb型のみ対応 (json型は非対応)

```
db1=# SELECT family FROM t_family WHERE id = 1;
      family
```

```
-----
[{"age": 40, "att": "head",    "name": "Tarou"},
 {"age": 35, "att": "spouse",  "name": "Hanako"},
 {"age": 10, "att": "child",   "name": "Shou"}]
```

(1 row)

```
db1=# SELECT family_name,
      jsonb_path_query(family,
        '$[*]?(@.att == "head").name')
      FROM t_family WHERE id = 1;
```

```
family_name | jsonb_path_query
```

```
-----+-----
```

```
Satou      | "Tarou"
```

(1 row)

属性(att)が
筆頭者(head)
である人の
名前(name)

- 式を使って計算される列
 - 生成式の列についても実体が格納される

```
=# CREATE TABLE t_people (id int primary key,  
    height_cm float4,  
    height_in float4 GENERATED ALWAYS AS (height_cm / 2.54) STORED);
```

```
=# INSERT INTO t_people (id, height_cm)  
    VALUES (1, 156), (2, 168), (3, 172), (4, 185), (5, 191);
```

```
=# SELECT * FROM t_people ;  
id | height_cm | height_in
```

id	height_cm	height_in
1	156	61.417324
2	168	66.14173
3	172	67.71654
4	185	72.83465
5	191	75.19685

(5 rows)

height_in には
自動計算された
値が格納される

直接変更は
エラーになる

```
=# UPDATE t_people SET height_in = 61.4 WHERE id = 1;  
ERROR:  column "height_in" can only be updated to DEFAULT  
DETAIL:  Column "height_in" is a generated column.
```

改善箇所は多岐にわたる:

- Btreeインデックス性能改善
 - キー重複が多くある場合に高速化／サイズを小さく
 - 複合インデックスのサイズを小さく
 - 内部ロック軽減で多コア同時処理を効率化
- REINDEX が CONCURRENTLY に対応
- GiST、SP-GiST、GIN の WAL出力量を削減
- GiST が INCLUDE に対応
- SP-GiST で kNN近傍探索に対応
- GiST に対する VACUUM 改善
- 他にもいくつか

■ 性能面

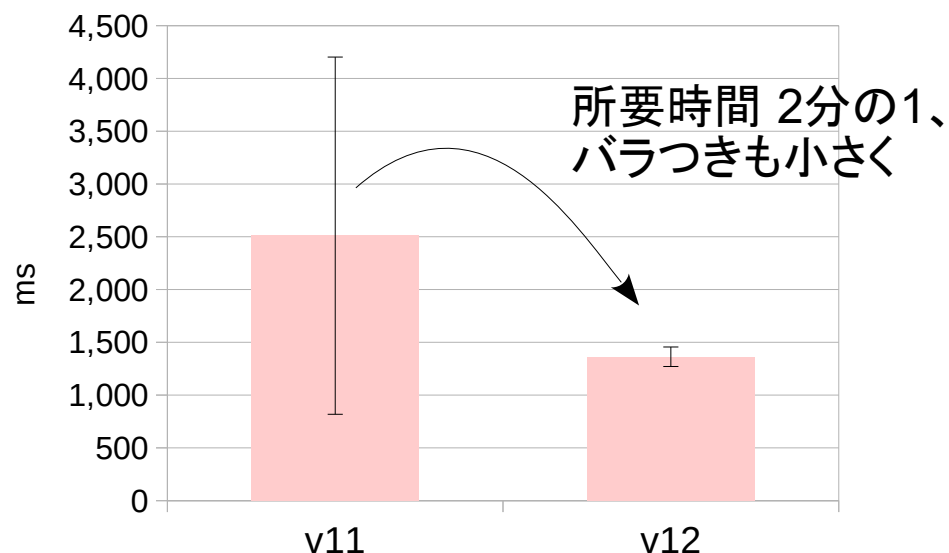
- 多パーティションに対する検索の高速化
- COPY TO によるデータ投入の高速化
- SQL実行で無駄なソートを省略する

■ 機能面

- 外部キー制約で被参照側でもパーティションテーブル対応
- パーティションを調べるビュー、関数、psqlコマンドの拡充
- パーティション定義構文で定数でなく式を使用可能に
- パーティションテーブルでテーブルスペース指定

■ 他にもいくつか

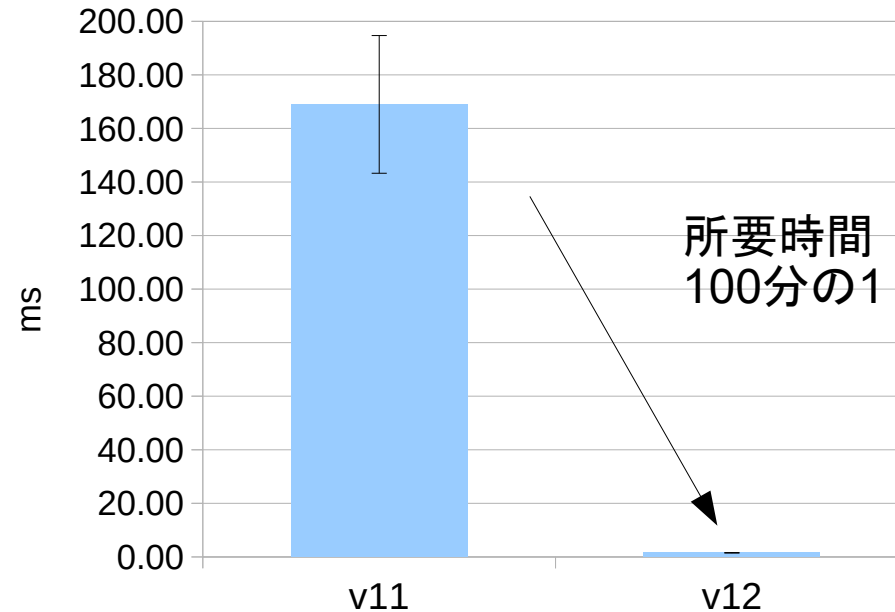
多パーティションに COPY データ投入



- 3,000パーティションを持つパーティションテーブルに各10件ずつ 30,000件のデータを投入

```
CREATE TABLE oya (id int primary key, v text) PARTITION BY RANGE (id);  
CREATE TABLE ko1 PARTITION OF oya FOR VALUES FROM (0) TO (10);  
CREATE TABLE ko2 PARTITION OF oya FOR VALUES FROM (10) TO (20);  
..
```

一部パーティションのみ参照する SQL



- 同テーブルで以下SQLを実行。
3000パーティション中、3パーティションのみ該当。

```
SELECT * FROM oya
WHERE id IN (100, 1000, 10000);
```

id		v
100		f899139df5e1059396431415e770c6dd
1000		a9b7ba70783b617e9998dc4dd82eb3c5
10000		b7a782741f667201b54880c925faec4b

(3 rows)

■ WITH句の節(CTE、共通テーブル式)も最適化される

従来のプラン: WITH句は書いた通りに独立して処理される

EXPLAIN

```
WITH cte AS (SELECT * FROM t_customer)
SELECT * FROM t_sales JOIN cte USING (customer_id)
WHERE cte.info ~ '^abcde';
```

QUERY PLAN

```
-----
Hash Join  (cost=42470.00..53695.48 rows=5123 width=73)
  Hash Cond: (cte.customer_id = t_sales.customer_id)
  CTE cte
    -> Seq Scan on t_customer  (cost=0.00..1834.00 rows=100000 width=37)
    -> CTE Scan on cte        (cost=0.00..2250.00 rows=500 width=36)
        Filter: (info ~ '^abcde'::text)
    -> Hash  (cost=19346.00..19346.00 rows=1000000 width=41)
        -> Seq Scan on t_sales  (cost=0.00..19346.00 rows=1000000 width=41)
(8 rows)
```

info列にインデックス
が在っても、全件取得
してから絞り込む、
賢くないプラン

最適化のために以下のようにサブクエリに書き直す必要があった

```
SELECT * FROM t_sales JOIN
  (SELECT * FROM t_customer) c USING (customer_id)
WHERE c.info ~ '^abcde';
```

PostgreSQL12 のプラン: WITH句も最適化の対象となる

EXPLAIN

```
WITH cte AS (SELECT * FROM t_customer)
SELECT * FROM t_sales JOIN cte USING (customer_id) WHERE cte.info ~ '^abcde';
QUERY PLAN
```

```
-----
Gather  (cost=1008.57..15625.03 rows=100 width=74)
  Workers Planned: 2
    -> Hash Join  (cost=8.56..14615.03 rows=42 width=74)
        Hash Cond: (t_sales.customer_id = t_customer.customer_id)
        -> Parallel Seq Scan on t_sales
            (cost=0.00..13512.67 rows=416667 width=41)
        -> Hash  (cost=8.44..8.44 rows=10 width=37)
            -> Index Scan using t_customer_info_idx on t_customer
                (cost=0.42..8.44 rows=10 width=37)
                Index Cond: ((info >= 'abcde'::text) AND (info < 'abcdf'::text))
                Filter: (info ~ '^abcde'::text)

(9 rows)
```

WITH cte_name MATERIALIZED AS (...)

従来動作 / 必ず独立プラン

WITH cte_name AS (...)

最適化を許す

WITH cte_name NOT MATERIALIZED AS (...)

複数回評価する最適化も許す

■ 拡張統計情報とは:

- CREATE STATISTICS コマンドでプランナ統計情報項目を追加
- 作成後は ANALYZE 処理で統計情報が収集される
- v11 までのサポート項目

dependencies: 列間の依存関係

ndistinct: 複数列の組み合わせに対する値の種類数

■ MCV

- 複数列の組み合わせに対する Most Common Values (最頻値)

列 location の最頻値:

最頻値	東京	大阪	京都	...
出現率	0.10	0.09	0.08	...

列 age の最頻値:

最頻値	20代	30代	40代	...
出現率	0.20	0.15	0.10	...

列 location、age の組み合わせの最頻値:(MCVで複数列の統計情報を取得)

最頻値	京都, 40代	東京, 20代	東京, 30代	大阪, 20代	大阪, 30代	...
出現率	0.05	0.02	0.01	0.01	0.01	...

個別の統計情報
では見えてこない
偏りを把握

従来は0.008と
計算されていた
(0.08 × 0.10)

- プラガブルなテーブル格納インタフェース (= Access Method)
 - 拡張モジュールでテーブル用のストレージエンジンを追加できる
 - 従来から インデックスAM は任意に追加可能 (多種のインデックスはこれのおかげ)、これに加えて、テーブルAM を追加できるようになった
 - テーブル毎にテーブルAM指定できる
- 今のところインタフェースのみ
- 開発されている実装
 - zheap: UNDOログを使って VACUUMレスを実現
 - zedstore: 列指向格納で OLAP最適化を実現

- recovery.conf 廃止
 - リカバリ設定も postgresql.conf に記述する
 - リカバリ起動を指示するには、recovery.signal ファイルを置く
 - スタンバイ起動を指示するには、standby.signal ファイルを置く
- テーブルの行ごとの OID 廃止
 - CREATE TABLE で WITH OIDS 指定のサポートが除去
- システムテーブルの oid 列が隠されなくなった
 - `SELECT * FROM pg_class` で oid 列も出力される。
- その他、様々な細かな非互換修正
 - 設定パラメータ名変更 (trigger_file → promote_trigger_file)
 - 設定値のデフォルト変更 (jit = on...)
 - システムテーブル／ビューの定義変更 (pgclass.relhasoids列の削除...)
 - etc.

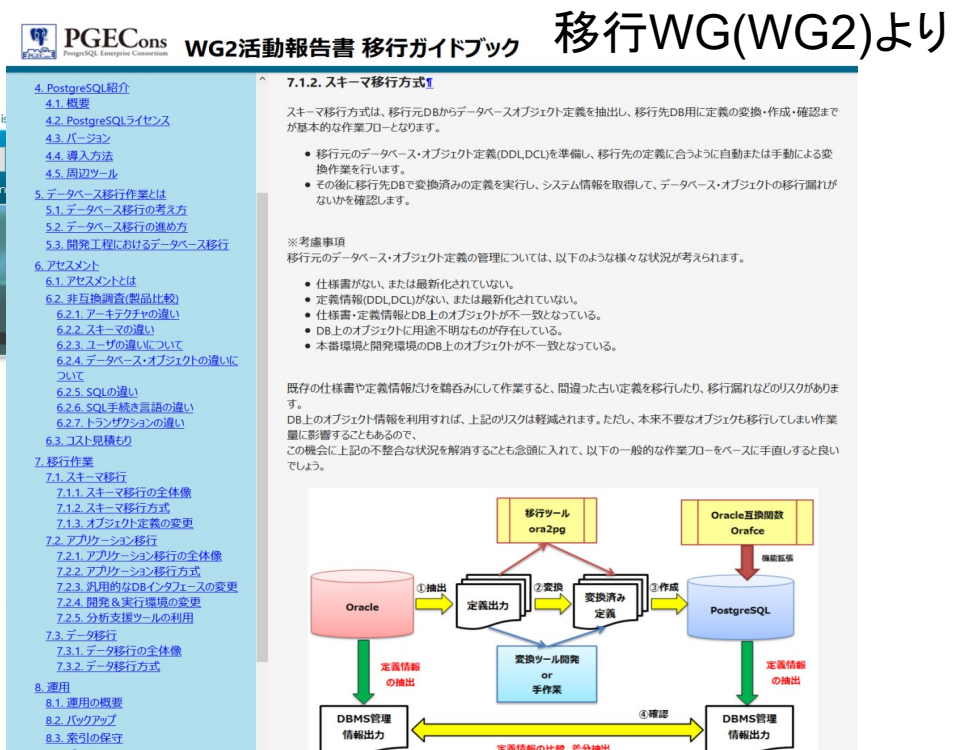
- 既存機能の性能向上が主→言い換えると実用性が向上
 - ◎最も大きいのが多数(数千パーティションで)の性能向上→パーティション機能がより実用的となった
 - WITH CTE句インライン対応(サブクエリへの書き換えが不要)
 - ジャストインタイムコンパイラ(JIT)がデフォルトで有効
- 運用面での改善も
 - ◎REINDEX CONCURRENTLY対応(CREATE INDEX CONCURRENTLY時に必要な制約の削除付加やRENAMEが不要)
 - ページチェックサム機能がinitdb時以外でも変更可能
- 新たに追加された便利な機能も
 - ◎MCV拡張統計情報→複数列で統計情報を保持出来るようになったことで、より良い実行計画の生成が可能となった
 - JSON/Path式→この機能追加で使いたくなった人がいるかも
 - テーブルAM I/F実装→将来的にはVACUUMレスや列指向なども
- 非互換機能への対応は必要(特にrecovery.conf廃止)

PostgreSQLエンタープライズコンソーシアム(PGEECons)

- PostgreSQL がエンタープライズの業務システムに適用できるように活動
- 2013年から継続的にエンタープライズな各社が集まって毎年成果物を作成
- 3つのワーキンググループで活動
新技術検証WG(WG1)/移行WG(WG2)/課題検討WG(WG3)

⇒成果物がすごいのでぜひ知って欲しい(出来れば見て欲しい)

PGEEConstoppページから「活動報告と成果物のダウンロード」か
「成果物総索引」をクリック



オープンソースとともに



URL: <http://www.sraoss.co.jp/>
E-mail: sales@sraoss.co.jp
Tel: 03-5979-2701