

PostgreSQL 13のご紹介

2020-08-28

OSC 2020 Online/Kyoto

SRA OSS, Inc. 日本支社 高塚 遥

講演の概要:

- 2020年秋にリリースが見込まれる、
PostgreSQL 13
の新機能や性能向上を解説します。
- 非互換の点についても説明します。

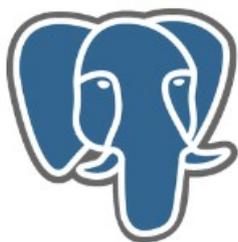
講演者:

- 高塚 遥
- 普段は 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
- (特非)日本PostgreSQLユーザ会 理事



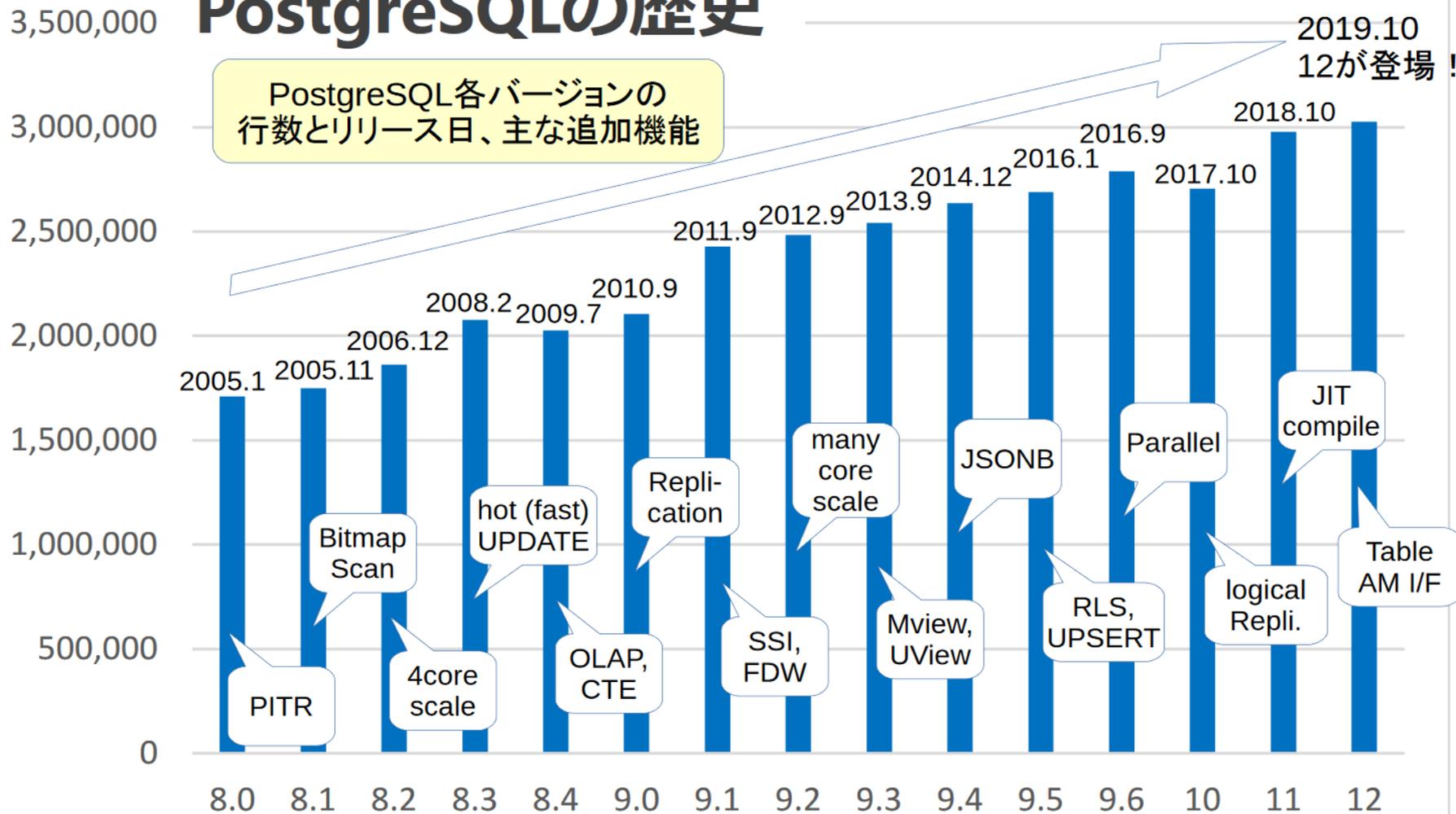
PostgreSQL 12 まで

- PostgreSQLとは
 - 多機能 RDBMS
 - 20年以上の歴史
 - POSTGRES から 30年以上
 - 1社主導でない OSS
- これまでの主要機能追加
 - 8.0 PITR / Windows
 - 9.0 レプリケーション
 - 9.1 Serializable SI
 - 9.2 多CPUコア適応
 - 9.4 バイナリ JSON型
 - 9.6 パラレルクエリ
 - 10 論理レプリケーション
 - 11 JITコンパイル
 - 12 テーブルAM



PostgreSQLの歴史

PostgreSQL各バージョンの
行数とリリース日、主な追加機能



PostgreSQL 13の概要

性能向上	Btreeインデックス性能向上、 インクリメンタルソート、 ストレージハッシュ集約
SQL機能	FETCH FIRST WITH TIE、 JSON Path の datetime、 パーティションテーブル関連の各種拡張
運用管理機能	進捗レポート対応拡張、並列VACUUM、 pg_rewind改良、pgbench拡張、pg_verifybackup
含まれなかった機能	セッション変数、グローバル一時テーブル、 オンラインチェックサム検証、 マテビュー差分更新 、 テンポラルテーブル、組込コネクションプーラ、 新たなテーブルアクセスメソッド

SQL機能の追加(1)

- FETCH FIRST ... WITH TIES

```
db1=# SELECT * FROM t_rank ORDER BY score DESC  
      OFFSET 0 ROWS FETCH FIRST 3 ROWS WITH TIES;
```

id	score
1	100
2	90
3	90
4	90

(4 rows)

3件指定だが
同順位(タイ)なら、
全てを出力する。

SQL機能の追加(2)

- JSON Path の datetime() メソッド追加
 - json_path_query_tz() 等の追加

```
db1=# SELECT jsonb_path_query(  
    ' ["12:30:54", "9:10:00", "13:20:10"] ',  
    '$[*].datetime() ? (@ > "12:00:00".datetime()) ');
```

```
jsonb_path_query
```

```
-----  
"12:30:54"
```

```
"13:20:10"
```

日付や時刻、
日付時刻の
比較ができる。

to_timestamp() と
同様の書式指定。

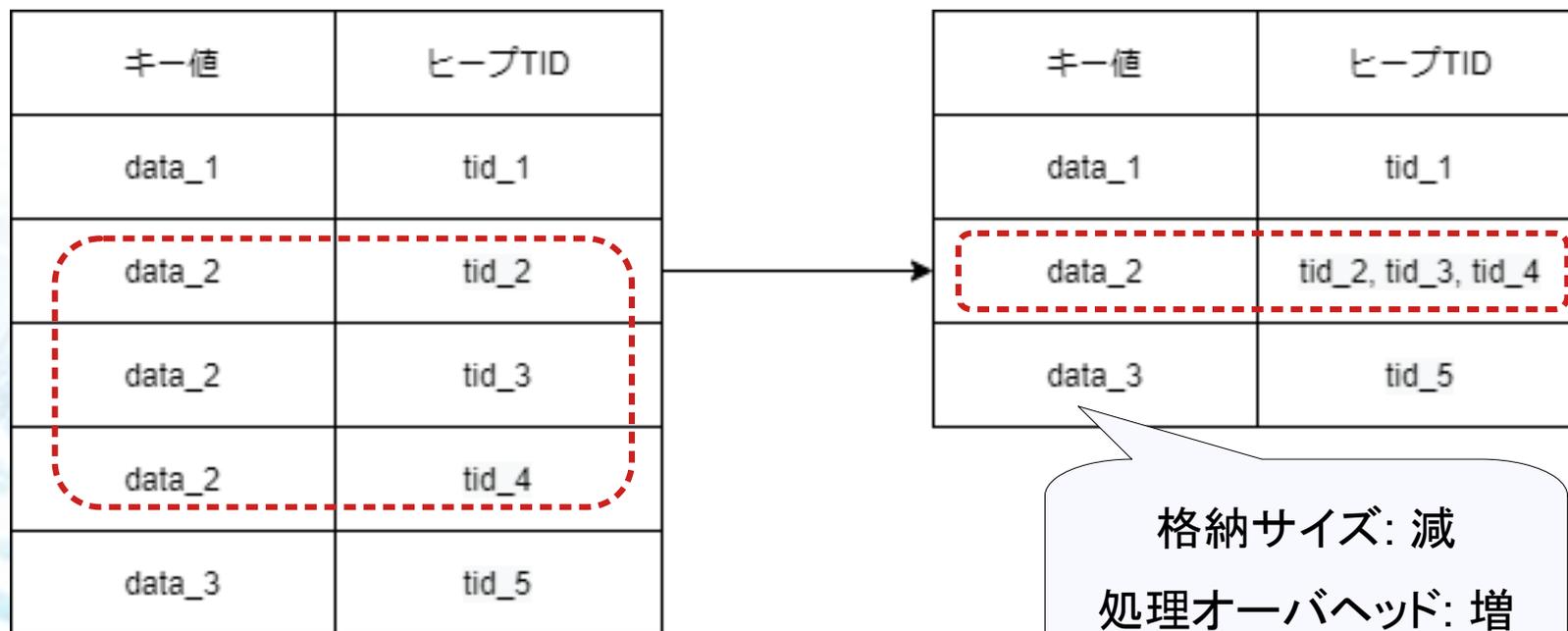
```
db1=# SELECT jsonb_path_query(  
    '{"timestamp": "2020-06-06 05:05:05 PM +09:00"}'::jsonb,  
    '$.timestamp.datetime("YYYY-MM-DD HH:MI:SS AM TZH:TZM")');
```

```
jsonb_path_query
```

```
-----  
"2020-06-06T17:05:05+09:00"
```

Btreeインデックスの性能向上(1)

- 重複排除 (deduplication)
 - 格納データ構造の変更

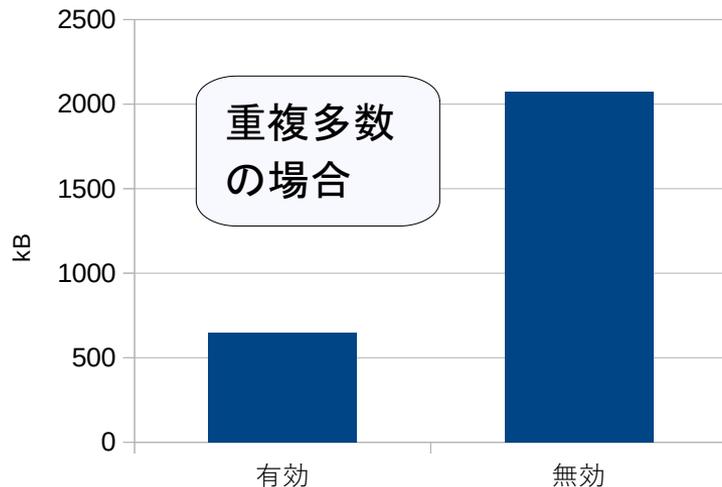


Btreeインデックスの性能向上(2)

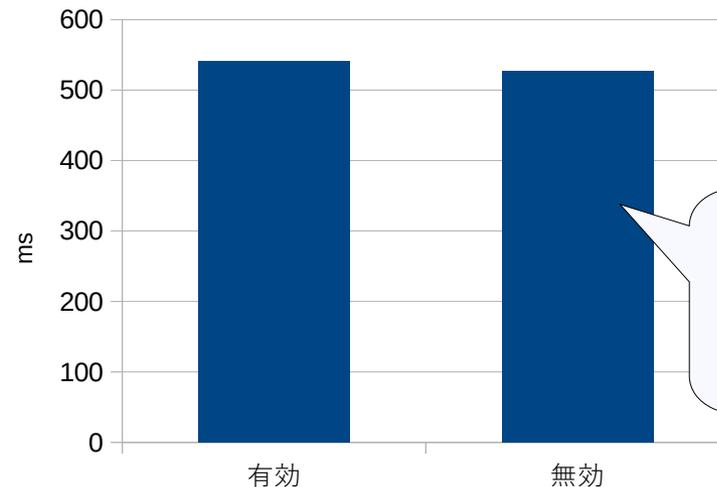
```
db1=# CREATE TABLE btree_test (f1 text COLLATE "C");
db1=# CREATE INDEX ON btree_test (f1) WITH (deduplicate_items=on);
db1=# INSERT INTO btree_text
      SELECT 'foo' FROM generate_series(1,100000);
```

onが
デフォルト

deduplicate によるインデックスサイズ差異



deduplicate による INSERT 処理時間の差異



Btreeインデックスの性能向上(3)

- 遅延処理方式
 - ページ分割のときに重複排除を行う
- 重複排除が効かない場合
 - データ型：
 - 非決定的な COLLATE を使った文字列型
 - real、double precision
 - numeric
 - コンテナ型(配列、複合型、範囲型、JSONB)
 - INCLUDE を伴うインデックス

新たな実行プラン要素(1)

- インクリメンタルソート

```
db1=# SELECT * FROM t_log;
```

dt	severity	message
2020-05-25 14:09:51	ERROR	syntax error at or near "XX"
2020-05-25 14:09:52	FATAL	role "pgsql" does not exist
2020-05-25 14:09:52	LOG	received SIGHUP,
2020-05-25 14:09:53	ERROR	cannot convert va

: (後略)

```
db1=# CREATE INDEX ON t_log (dt);
```

```
db1=# SELECT * FROM t_log WHERE dt > ' 2020-05-25 14:10'
      ORDER BY dt, severity, message LIMIT 1000;
```

ソート項目のうち、
dt列はインデックスで
既にソート済み

新たな実行プラン要素(2)

```

Limit (cost=0.38..74.06 rows=1000 width=46)
      (actual time=0.031..0.599 rows=1000 loops=1)
-> Incremental Sort (cost=0.38..735.25 rows=9974 width=46)
      (actual time=0.030..0.534 rows=1000 loops=1)

Sort Key: dt, severity, message
Presorted Key: dt
Full-sort Groups: 32  Sort Method: quicksort  Average
Memory: 27kB  Peak Memory: 27kB
-> Index Scan using t_log_dt_idx on t_log
      (cost=0.29..379.83 rows=9974 width=46)
      (actual time=0.011..0.333 rows=1001 loops=1)
      Index Cond:
        (dt > '2020-05-25 14:10:00'::timestamp without time zone)
Planning Time: 0.107 ms
Execution Time: 0.649 ms

```

全列ソートの
とき 127 kB

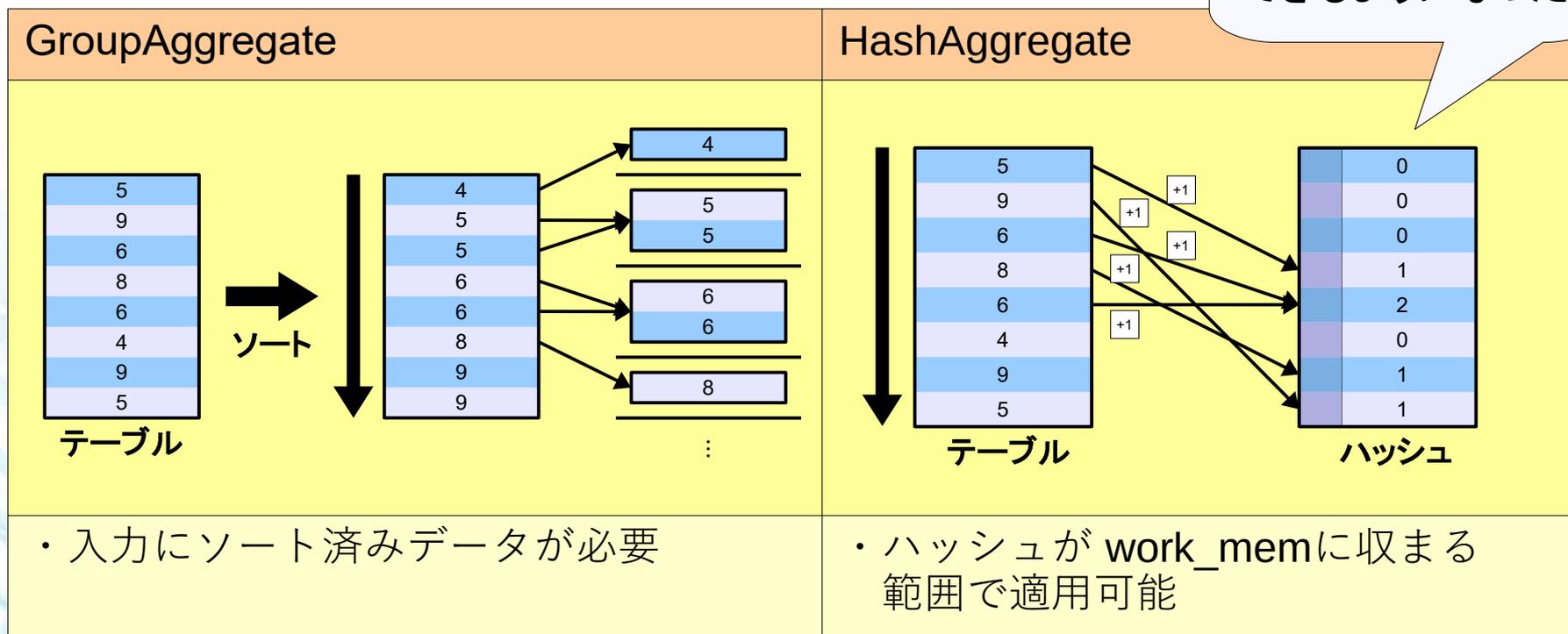
全列ソートの
とき 3.3 ms

- 高速化、使用メモリの節約
- `enable_incremental_sort` (デフォルト on)

新たな実行プラン要素(3)

- ストレージ上のハッシュ集約
 - 集約問い合わせのプラン要素:

新たに、ハッシュを
ストレージ書き出し
できるようになった



新たな実行プラン要素(4)

```

db1=# EXPLAIN (ANALYZE) SELECT uid, count(1) FROM t_login GROUP BY uid;
          QUERY PLAN
-----
HashAggregate  (cost=36995.00..41912.19 rows=101094 width=12)
                (actual time=198.841..295.113 rows=100000 loops=1)
  Group Key: uid
  Planned Partitions: 4
  Peak Memory Usage: 4145 kB
  Disk Usage: 30376 kB
  HashAgg Batches: 4
    -> Seq Scan on t_login (cost=0.00..16370.00 rows=1000000 width=4)
        (actual time=0.013..47.859 rows=1000000 loops=1)

Planning Time: 0.092 ms
Execution Time: 301.152 ms

```

work_mem
'4MB' の場合

- GroupAggregate より速く、メモリ HashAggregate より遅い
- ~~hashagg_avoid_disk_plan (デフォルト off)~~
 - つまり、ディスク上のハッシュ集約はデフォルトで有効

beta3 で
設定無しに

パーティションテーブルの改良 (1)

- ロジカルレプリケーション対応
 - 親パーティションテーブルで一括指定

親を指定すれば、
子パーティション群が
登録される。

```
db1=# CREATE PUBLICATION pub1 FOR TABLE pt_member;
```

```
db1=# SELECT * FROM pg_publication_tables;
```

pubname	schemaname	tablename
pub1	public	p_member_1
pub1	public	p_member_10001
pub1	public	p_member_20001

(3 rows)

オプション指定で、
単一テーブル同様の
パブリケーションも可能

```
db1=# CREATE PUBLICATION pub2 FOR TABLE pt_member
      WITH (publish_via_partition_root = true);
```

《→ pg_publication_tables には 1 エントリだけ追加》

パーティションテーブルの改良 (2)

- パーティション毎の結合の改良

```
SELECT lo.mes, ld.detail FROM t_log_parted lo
INNER JOIN t_log_detail_parted ld ON (lo.id = ld.id);
```

LEFT JOIN の
場合には、先に
Appendされる

従来は、
これがあると
機能しなかった

パーティションテーブル
t_log_parted

パーティション1
(id: 100000 ~ 199999)

パーティション2
(id: 200000 ~ 299999)

パーティション3
(id: 300000 ~ 399999)

パーティション4
(id: 400000 ~ 499999)

結合

結合

結合

パーティションテーブル
t_log_detail_parted

パーティション1
(id: 100000 ~ 199999)

パーティション2
(id: 200000 ~ 299999)

パーティション3
(id: 300000 ~ 399999)

パーティションテーブルの改良 (3)

- 行単位BEFOREトリガ対応

```
db1=# CREATE FUNCTION tf_1() RETURNS TRIGGER LANGUAGE plpgsql AS  
$$ BEGIN NEW.mes := substring(NEW.mes from 1 for 1000);  
RETURN NEW; END; $$;
```

```
db1=# CREATE TRIGGER trg1 BEFORE INSERT OR UPDATE  
ON pt_log_parted FOR ROW EXECUTE FUNCTION tf_1();
```

- パーティションテーブル(親テーブル)にトリガ定義が可能に
- パーティション間の行移動が生じる場合は、トリガ実行時にエラーになる

本例なら id を
変更する場合

並列VACUUM

- テーブルの各インデックスを並列で処理

- デフォルトで有効

- インデックスの数だけ並列処理を試みる

無効化するために
オプション指定する

- 関連設定

- `max_parallel_maintenance_workers = 2`
(`max_worker_processes`、`max_parallel_workers`)
- `min_parallel_index_scan_size = 512kB`

```
VACUUM (PARALLEL 《並列ワーカ数》 [, ...]) [《テーブル指定》];
```

```
vacuumdb --parallel=《並列度》 ...
```

```
vacuumdb --jobs=《ジョブ数》 ...
```

ジョブ数指定は本機能とは別、
複数テーブル間の並列処理

進捗報告ビューの拡張(1)

対象コマンド	進捗確認用ビュー
(FULLでない) VACUUM	pg_stat_progress_vacuum
CLUSTER、VACUUM FULL	pg_stat_progress_cluster
CREATE INDEX、REINDEX	pg_stat_progress_create_index
pg_basebackup	pg_stat_progress_basebackup
ANALYZE	pg_stat_progress_analyze

今回追加の
2項目

- 処理の phase ごとに各種指標値を報告
 - phase の理解が必要
- 実行完了すると進捗報告のエントリが無くなる
- ANALYZE等で複数テーブルが対象のコマンドだと、対象全体の何割まで完了したかの判断が難しい。

進捗報告ビューの拡張(2)

- pg_basebackup 進捗の phase:

- ↓ initializing

- ↓ waiting for checkpoint to finish

- ↓ estimating backup size

- ↓ streaming database files

- ↓ waiting for wal archiving to finish

- ↓ transferring wal files

pg_basebackup コマンドの -Pv オプションでも各 phase を確認できる。

進捗を参照する SQL 例

```
SELECT phase, backup_streamed / backup_total::float * 100
       AS "progress_ratio(%)" FROM pg_stat_progress_basebackup;
```

```
[ RECORD 1 ]-----+-----
phase           | streaming database files
progress_ratio(%) | 32.19553708609419
```

接続認証の拡張

- クライアントでチャンネルバインディング必須指定
 - `channel_binding=require` (prefer, disable)
 - バージョン11.x 以降サーバの `scram-sha-256` 認証で
- TLSバージョン指定
 - `ssl_min_protocol_version`
 - `ssl_max_protocol_version`
- クライアント証明書のパスワードの指定
 - `sslpassword`

チャンネルバインディング:
認証用のダイジェストやり取りに
セッション固有の情報を含めて、
Man-In-the-middle攻撃を防ぐ

pgbenchの拡張

- パーティションテーブルの生成

```
$ pgbench -i -s 50 --partitions 5 db1
```

pgbench_account が
100万行ずつ、
5パーティションの
パーティションテーブルに

- サーバ側データ生成

- これまでは pgbench側でデータ生成
- -I オプションで g の代わりに G を指定

```
$ pgbench -i -I dtGvp -s 50 -h remote_host db1
```

DBサーバが別ホスト
の場合に、初期化を
高速化

- データ生成時の画面出力が改善

- フェーズごとに所要時間報告

pg_rewindの改良

- 自動クラッシュリカバリ
 - 「対象データディレクトリはPostgreSQLを正常に停止したものでなければならない」の考慮が不要に
 - 「最初にクラッシュリカバリ起動して、停止」を自動化
- リカバリ設定を自動作成 (--write-recovery-conf)
 - ソースサーバに追従するスタンバイの設定を生成
- WALアーカイブ適用 (--restore-target-wal)
 - restore_command 設定を使う
 - ターゲットのWALアーカイブを書き戻すコマンドを指定

pg_verifybackup

- 取得済ベースバックアップの健全性を検査するコマンド
- pg_basebackup で backup_manifest が生成される

```
{ "PostgreSQL-Backup-Manifest-Version": 1,  
  "Files": [  
    { "Path": "backup_label", "Size": 224, "Last-Modified": "2020-08-06 04:47:55 GMT",  
      "Checksum-Algorithm": "CRC32C", "Checksum": "fd0ec8cb" },  
    { "Path": "global/1262", "Size": 8192, "Last-Modified": "2020-08-01 23:36:31 GMT",  
      "Checksum-Algorithm": "CRC32C", "Checksum": "8b5cc6da" },  
    { "Path": "global/2964", "Size": 0, "Last-Modified": "2020-08-01 23:23:16 GMT",  
      "Checksum-Algorithm": "CRC32C", "Checksum": "00000000" },  
    : 後略
```

- マニフェストに基づき、各ファイルのチェックサムを検証
- プレーン形式のみ対応対応
 - 展開後に検査するなら tar形式でも可

その他の運用コマンド拡張

- reindexdb が --jobs 対応
- pg_dump が 外部テーブル内容をダンプ可能に
 - --include-foreign-data
- trusted な EXTENSION
 - データベースに CREATE権限があれば
一般ユーザでも導入できる、という属性が追加

非互換の仕様変更(1)

- SIMILAR TO ... ESCAPE NULL
 - SQL標準通り、式が必ず NULL を返すように
 - 従来は ESCAPE指定を省略したのと同じ扱い

デフォルトの
エスケープ文字は
バックスラッシュ

- effective_io_concurrency
 - 設定値の意味が変更
 - ストレージI/O並列数ではなく、
プリフェッチするページ数を直接指定

$$NEW = round \left(\sum_{i=1}^{OLD} \frac{OLD}{i} \right)$$

```
db1=# SELECT round(sum(OLD / i::float))  
      FROM generate_series(1, OLD) s(i);
```

非互換の仕様変更(2)

- pg_stat_ssl、pg_stat_gssapi ビュー
 - 補助プロセスは出力されなくなる
- 待機イベントの名前変更
 - RecoveryWalAll → RecoveryWalStream
 - RecoveryWalStream → RecoveryRetrieveRetryInterval
- opaque疑似型の廃止
- 外部キー制約の 7.2以前の古い構文の廃止
- 演算子クラスの 7.4以前の古い構文を廃止

PostgreSQL 13 リリース予定

- 2020年 5月 21日 beta1 リリース
- 2020年 6月 25日 beta2 リリース
- 2020年 9月末くらい? 13.0 リリースの見込み
 - 2020年 3Q と計画されていたが、既に 8月
 - 8/13 beta3 リリース
 - 通常 rc 版のリリースがある

近年の実績:

version	release date
9.6.0	2016/9/29
10.0	2017/10/5
11.0	2018/10/18
12.0	2019/10/3

まとめ

- PostgreSQL は安定リリースを継続
- バージョン13 は変更が少なめのメジャーリリース
 - 非互換が少なく、アップグレードしやすい
 - 次バージョンに持ち越されていて様々な開発が継続