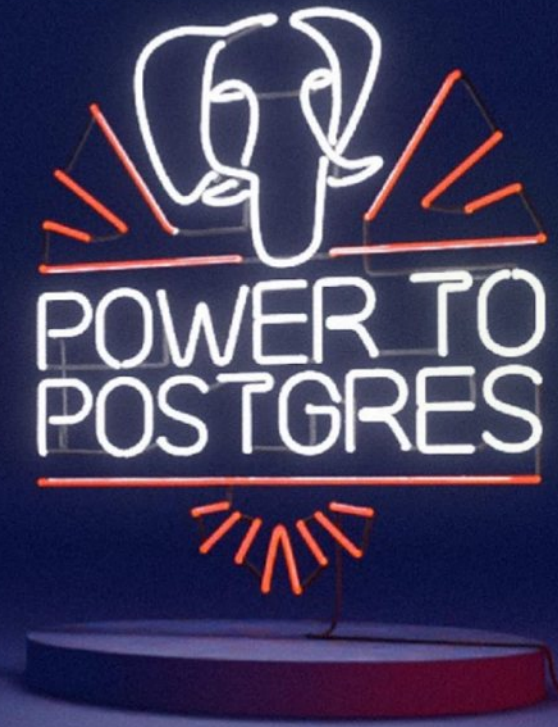
新バージョンで更に進化する EDB Postgres Advanced Server 13

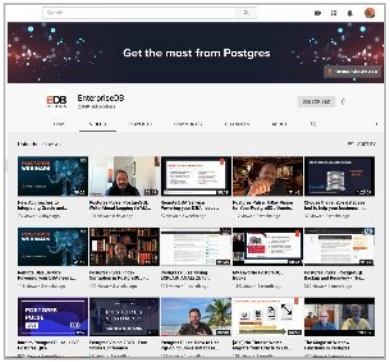
Rushabh Lathia
Marc Linster

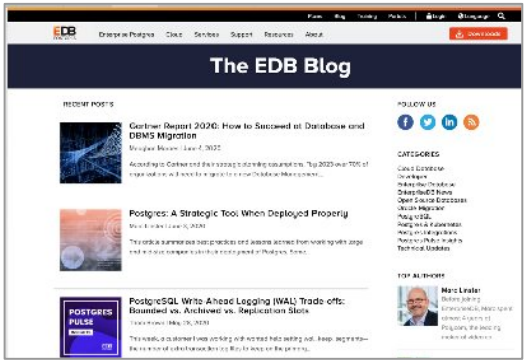


EDB

アジェンダ

- PG/EPAS v13 の主な新機能
- パーティショニングの深堀り
- 質疑応答
- その他のリソース
  - EDB Postgres Youtube Channel
  - EDB Postgres Pulse
  - EDB Postgres Blogs

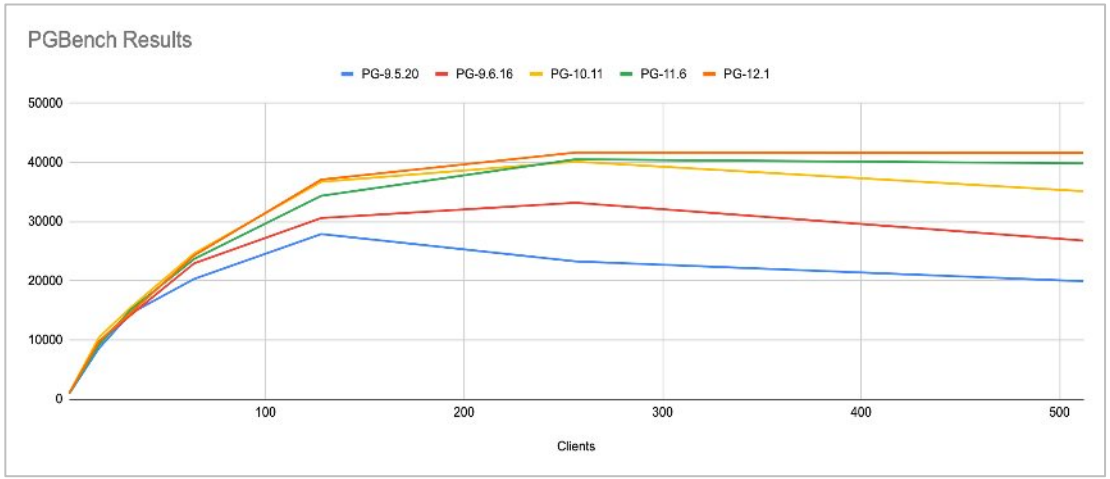




スピード：Postgres の進化の鍵を握る

2020年1月：4年間でTPS50%改善

https://www.enterprisedb.com/postgres-tutorials/benchmarking-postgresql-aws-m5metal-instance



スピードは方程式の一部

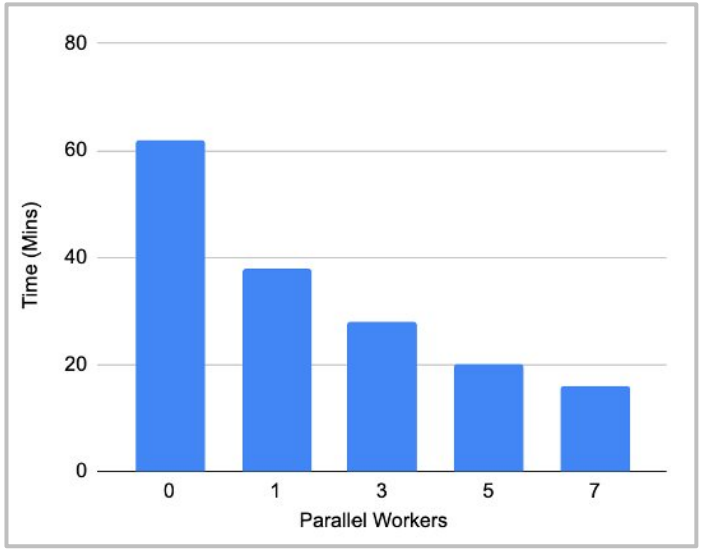
EDB Postgres の主な新機能

- 新しい機能と強化された主要機能リスト
  - Vacuum
  - セキュリティと一貫性
  - データの読み込み
  - パーティショニング
- 完全なリストは、リリースノート参照

バキューム（Vacuum）の改善

インデックスのパラレルバキュームと追加専用テーブルのバキューム

- インデックスのパラレルバキュームの性能
  - 5000万回のインプレースアップデートを 実行した後のバキューム性能 - マルチプロセスベンチマークで4倍高速化
- 追加専用処理に対応したオートバキューム
  - 統計を再計算!
  - IOTテーブルに重要



https://www.enterprisedb.com/postgres-tutorials/what-parallel-vacuum-postgresql-13

セキュリティと一貫性

libpq チャネル結合と pg_catcheck の改善

- libpq チャネル結合
  - 中間者攻撃を止める
- pg_catcheckの新機能
  - システムカタログ破損診断ツール
  - 参照：https://github.com/EnterpriseDB/pg_catcheck
  - 新しい機能：すべてのリレーション（テーブル）に対して初期ファイルが利用可能かどうかをチェック
  - ‘could not open file xxx’ 問題に対応

pg_catchack の新しい機能

"could not open file xxx"を検索

- 新しいオプション --select-from-relations
- このオプションは、エラー発生理由を教えてくれない
- データベース内のテーブル、TOASTテーブル、マテリアライズドビューに対応
- このオプションは、インデックスや、最初のもの以外のリレーションセグメントにはまだ役に立たない
- >>> リレーションはアクセス可能 <<< “Could not open file”
- EDB Postgres Advanced ServerとPostgreSQLに対応

サンプル：

```
rushabh@rushabh:pg_catcheck$
./pg_catcheck edb --select-from-
relations
notice: unable to query relation
"public"."emp": ERROR:  could not
open file "base/16198/16394":
Permission denied
notice: unable to query relation
"public"."jobhist": ERROR:  could
not open file "base/16198/16405":
No such file or directory
progress: done (2
inconsistencies, 0 warnings, 0
errors)
```

EDB*LOADER

edb*loader 重複行のアボート

- 要望の多い機能
- edbldrは 、一意性制約違反によりレコードの挿入に失敗した場合に操作全体を中止
- 推論的挿入を使用して行を挿入することで、これを修正
- 'handle_conflicts' が true でインデックスが存在する場合にのみ実行
- "ON CONFLICT … DO NOTHING"の方法に似ているが、レコードはBADファイルに保存

EDB Postgres のパーティショニング

- PostgreSQLはv10で宣言的パーティショニングを導入
  - 新リリースごとにパーティショニングの新機能を追加
  - パフォーマンスが向上
- EPASには9.4からパーティショニング機能があるが、これは継承ベースの機能
- v10から、EPASはPostgreSQLの宣言的パーティショニングを活用するために、継承ベースのパーティショニング実装に移行
  - EPASは、かなり役立ついくつかのパーティション機能を追加
  - パーティショニングのためのRedwood互換構文
  - エクスチェンジ パーティション
  - スプリット パーティションなど

パーティションの自動化

必要に応じて自動的に新しいパーティションを作成

- パーティションスキームを簡単に管理する方法
- 実行時のパーティション作成を気にする必要がない
- 古いノンパーティションからパーティションテーブルへのデータの簡単なLOADまたはCOPY
- 自動分割の種類
    - INTERVAL Partition（RANGE）
    - AUTOMATIC Partition（LIST）
    - PARTITIONとSUB-PARTITION番号（HASH）

インターバルパーティション

- INTERVAL PartitionはRANGEパーティションの拡張
- パーティションに INTERVAL式を提供する必要がある
- 与えられたタプルが既存のパーティションに合わない場合、自動的に新しいパーティションを作成
- INTERVAL句は、新しいパーティションの範囲サイズを決定
- 既存のパーティションをALTERしてINTERVALパーティションに変換することも可能

インターバルパーティションの例

サンプル（１）

```
edb@61349=#CREATE TABLE orders (id int, country_code varchar(5), order_date DATE )
PARTITION BY RANGE (order_date) INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
(PARTITION P1 values LESS THAN (TO_DATE('01-MAY-2020','DD-MON-YYYY')));
CREATE TABLE
edb@61349=#INSERT INTO orders VALUES (1, 'IND', '24-FEB-2020');
INSERT 0 1
edb@61349=#SELECT tableoid::regclass, * FROM orders;
 tableoid  | id | country_code |     order_date
-----------+----+--------------+--------------------
 orders_p1 |  1 | IND          | 24-FEB-20 00:00:00
(1 row)
```

インターバルパーティションの例

サンプル（２）

```
edb@61349=#INSERT INTO orders VALUES (1, 'IND', '23-JUN-2020');
INSERT 0 1
edb@61349=#SELECT tableoid::regclass, * FROM orders;
      tableoid       | id | country_code |     order_date
---------------------+----+--------------+---------------------
 orders_p1           |  1 | IND          | 24-FEB-20 00:00:00
 orders_sys613490102 |  1 | IND          | 23-JUN-20 00:00:00
(2 rows)
```

Other syntax options:

```
ALTER TABLE orders SET INTERVAL();
ALTER TABLE orders SET INTERVAL(NUMTOYMINTERVAL(1,'MONTH'));
```

自動パーティション

LISTパーティションの新規パーティションを自動的に作成する

- 自動リスト・パーティショニングは、リスト・パーティショニング・キーの新しい異なる値に対してパーティションを作成
- AUTOMATIC partitionはリストパーティションの拡張で、新しいタプルが既存のパーティションに合わない場合、システムが自動的にパーティションを作成
- ALTER TABLEコマンドを使用して、既存のテーブルで自動リスト分割を有効にすることも可能
- `ALTER TABLE <tablename> SET [MANUAL|AUTOMATIC]`

自動パーティションの例

サンプル：

```
CREATE TABLE orders
(id int, country_code varchar(5))
PARTITION BY LIST (country_code)
AUTOMATIC
(PARTITION p1 values ('IND'),
 PARTITION p2 values ('USA'));
```

Describe Table:

```
Partition key: LIST (country_code) AUTOMATIC
Partitions: orders_p1 FOR VALUES IN ('IND'),
            orders_p2 FOR VALUES IN ('USA')
```

どのパーティションにも収まらないレコードを挿入

```
INSERT INTO orders VALUES ( 1 , 'UK');
```

Describe table:

```
Partition key: LIST (country_code) AUTOMATIC
Partitions: orders_p1 FOR VALUES IN ('IND'),
  orders_p2 FOR VALUES IN ('USA'),
  orders_sys15960103 FOR VALUES IN ('UK')
```

自動ハッシュパーティショニング

パート1

- PARTITIONS < num> STORE IN句で、CREATE TABLEコマンドの間に指定した数のHASHパーティションを自動的に追加することが可能
- STORE IN句は、パーティションが分散されるべきテーブルスペースを指定するために使用

サンプル：

```
edb@72970=#CREATE TABLE hash_tab (
    col1            NUMBER,
    col2            NUMBER)
 PARTITION BY HASH (col1, col2)
 PARTITIONS 2 STORE IN (tablespace1, tablespace2);
 edb@72970=#¥d hash_tab
       Partitioned table "public.hash_tab"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 col1   | numeric |           |          |
 col2   | numeric |           |          |
Partition key: HASH (col1, col2)
Number of partitions: 2 (Use ¥d+ to list them.)
```

自動ハッシュパーティショニング

パート1

```
edb@89398=#SELECT table_name, partition_name, tablespace_name FROM user_tab_partitions
WHERE table_name = 'HASH_TBL';
 table_name | partition_name | tablespace_name
------------+----------------+-----------------
 HASH_TBL   | SYS0101        | TABLESPACE1
 HASH_TBL   | SYS0102        | TABLESPACE2
(2 rows)
```

自動ハッシュパーティショニング

パート2

- SUBPARTITIONS < num > は、 各パーティションに指定された数のサブパーティションを自動作成するための定義済みテンプレートを作成

```
CREATE TABLE ORDERS  (id int, country_code varchar(5),
order_date DATE)
PARTITION BY LIST (country_code) AUTOMATIC
SUBPARTITION BY HASH(order_date) SUBPARTITIONS 3
( PARTITION p1 VALUES ('USA', 'IND') );


edb@89398=#SELECT table_name, partition_name, subpartition_name
FROM user_tab_subpartitions WHERE table_name = 'ORDERS';
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 ORDERS     | P1             | SYS0101
 ORDERS     | P1             | SYS0102
 ORDERS     | P1             | SYS0103
(3 rows)
```

自動ハッシュパーティショニング

パート2

```
edb@89398=#INSERT INTO orders VALUES ( 2, 'UK', sysdate );
INSERT 0 1
edb@89398=#SELECT table_name, partition_name, subpartition_name FROM user_tab_subpartitions WHERE table_name = 'ORDERS';
 table_name | partition_name | subpartition_name
------------+----------------+-------------------
 ORDERS     | P1             | SYS0101
 ORDERS     | P1             | SYS0102
 ORDERS     | P1             | SYS0103
 ORDERS     | SYS893980105   | SYS893980106
 ORDERS     | SYS893980105   | SYS893980107
 ORDERS     | SYS893980105   | SYS893980108
(6 rows)
```

自動ハッシュパーティショニング

パート3

- サブパーティションの番号は以下のコマンドで変更することができ、新しい番号は後に作成されたパーティションでサブパーティションを 設定するために使用される。

```
ALTER TABLE tbl1 SET SUBPARTITION
TEMPLATE 100;
-- The following will reset the subpartition count to 1
ALTER TABLE tbl1 SET SUBPARTITION
TEMPLATE ( );
```

- このテンプレートは、新しいパーティションが作成される ADD PARTITION, SPLIT PARTITIONのようなすべてのパーティションコマンドで使用される。明示的なSUBPARTITIONの説明やSUBPARTITIONS <num>によって上書きすることができる。

```
 ALTER TABLE ORDERS ADD PARTITION p2 VALUES ('AUS') SUBPARTITIONS
10;
¥d orders_p2
 ...
Partition of: orders FOR VALUES IN ('AUS')
Partition key: HASH (col2)
Number of partitions: 10 (Use ¥d+ to list them.)
-- Similar also works when do the SPLIT PARTITION
ALTER TABLE tbl1 SPLIT PARTITION p1 VALUES (10, 20) INTO
(PARTITION p1_a, PARTITION p1_b);
```

パーティションワイズ結合

大幅な既存機能の強化

- PG 11、パーティションワイズ結合を導入
- SET enable_partitionwise_join デフォルトはoff
- 結合はパーティションキー上で行われ、両方のパーティションの境界は同じでなければならない
- PG 13、パーティションに同じ境界を持つことの制限を取り除くことで、同様の機能を強化
- 宣言的パーティショニングの強化、最適化、実装までの道のりの詳細については、私の同僚であるAmit Langote氏の「Postgresでのテーブルパーティショニング、どこまで来たか」という講演を参照

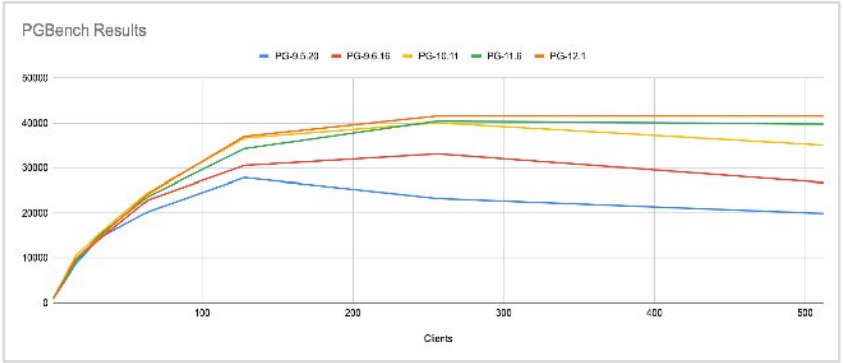
EPAS v13の特徴

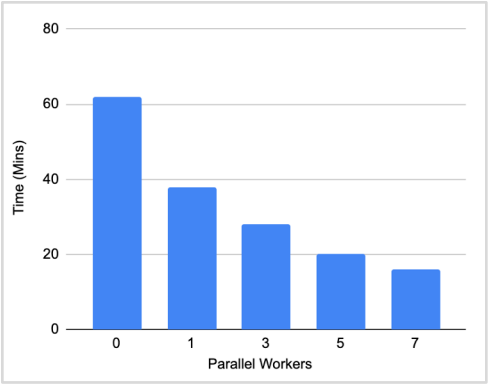
- CREATE TABLEとALTER TABLEの”USING INDEX”句。
- STATS_MODE 集計関数
- MEDIAN: smallint、 int、 bigint、および数値のバリアントを追加
- DBMS_SQLパッケージのDEFINE_COLUMN_LONG、COLUMN_VALUE_LONG、LAST_ERROR_POSITIONの追加関数/プロシージャをサポート
- “utl_http.end_of_body”例外をサポート
- EPASの関数 to_timestamp_tz()をサポート。
- CREATE TABLEとINDEXのPARALLEL/NOPARALLELオプション
- インデックスの作成構文は、カラム名と番号を含む、(col_name,1)
- 一括実行のために処理された行数をログに記録
- CSVとXMLの監査ログ間での一貫性
- psql や他のクライアントのようなすべての接続パラメータをサポートするための Edbldr
- パッケージ本体内での関数/手順の順送り宣言をサポート
- DBMS_CRYPTOパッケージにAES192とAES256のサポートを追加
- レッドウッド互換性表示を強化
- NEW/OLD変数とSTATEMENTレベルのトリガイベントを持つWHEN句を持つ複合トリガを作成可能
- 分割パーティションの動作をよりレッドウッドに近いものに修正
- edbldrとマルチインサート周りの強化
- EDB-SPL言語のさらなる強化
- to_number() 関数でFMフォーマットをサポート
- SYSDATEは、よりOracle（STABLE）のように

まとめ

Postgresはより速く、より有能になっている

- 4年間で50%以上のTPS改善
- インデックスのパラレルバキューム：4倍速
- 新機能と強化された機能
  - テーブルのみを追加するためのバキューム
  - セキュリティと一貫性
  - データの読み込み
  - パーティショニング





質疑応答

EDB

Thank You