PostgreSQLエンタープライズ・コンソーシアム 技術部会 WG#3

2018年度WG3活動報告書
パーティショニング調査編

目次

1. 改訂履歴

版	改訂日	変更内容
1.0	2019/03/13	2018年度版として新規作成 - PostgreSQL 10および11を対象バージョンとする

2. ライセンス

3. はじめに

3.1. PostgreSQLエンタープライズコンソーシアムとWG3について

PostgreSQLエンタープライズコンソーシアム(略称 PGECons)は、PostgreSQL本体および各種ツールの情報収集と提供、整備などの活動を通じて、ミッションクリティカル性の高いエンタープライズ領域へのPostgreSQLの普及を推進することを目的として設立された団体です。

PGECons 技術部会ではPostgreSQLの普及に資する課題を活動テーマとし、3つのワーキンググループで具体的な活動を行っています。

- WG1（新技術検証ワーキンググループ）
- WG2（移行ワーキンググループ）
- WG3（課題検討ワーキンググループ）

これら3つのワーキンググループのうち、WG1、WG3については 2015 年度まではそれぞれ、「性能ワーキンググループ」、「設計運用ワーキンググループ」という名称で活動してきました。2016年度は、従来の活動領域を広げる意図のもとでそれらを再定義し、上記のような名称に改めました。

これに伴い、WG3ではPostgreSQLの設計運用を中心としたさまざまな課題の解決のための調査検証を行い、PostgreSQLが広く活用される事を推進していくこととしました。

3.2. 本資料の概要と目的

爆発的なデータ増加が見込まれる昨今において、大規模データへの対応はますます重要性を増しています。大規模データへの対応策は、「データ分割」「並行処理」「データ圧縮」の3つが挙げられます。ここでは「データ分割」の1手法であるテーブル・パーティショニングを取り上げます。

本資料はWG3の2018年度の活動としてPostgreSQL10から実装された宣言的パーティション機能について、管理性や性能について調査検討した結果をまとめたものです。パーティション設計の参考になれば幸いです。

3.3. 本資料の構成

- PostgreSQLにおけるパーティション
- パーティションの管理
- パーティションの性能
- パーティションのTips
- まとめ

3.4. 想定読者

本資料の読者は以下のような知識を有していることを想定しています。

- DBMSを操作してデータベースの構築、保守、運用を行うDBAの知識
- PostgreSQLを利用する上での基礎的な知識

4. PostgreSQLにおけるパーティショニング

テーブル・パーティショニングとは、論理的には1つの大きなテーブルであるものを物理的に小さなテーブルに分割することです。PostgreSQL 9.6以前は、後述する継承を使用したパーティショニング、PostgreSQL 10以降では加えて宣言的パーティショニングを利用できます。なお、パーティション機能はOracleなど多くのデータベースにおいても実装されており、PostgreSQLにおけるパーティションも 目的やメリットにおいて多くの類似点があります。

この章では、パーティションの目的および継承を使用したパーティショニングと宣言的パーティショニングの概要および特性について簡単に紹介します。

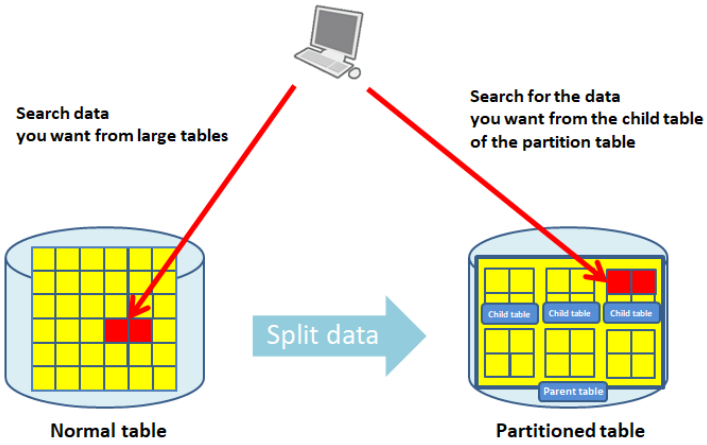
4.1. パーティショニングの目的

論理的に一つのテーブルを物理的に複数のテーブルに分割することで後述する様々な利点を得られます。

4.1.1. 性能の向上

頻繁に検索されるテーブル内のデータを一つのパーティションまたは一部のパーティションに格納することで、物理的に1つのテーブルで構成した場合と比較してパーティション毎のサイズや索引サイズが小さくなるため、パーティションやその索引内のよく使用される部分がメモリに乗りやすくなります。また条件指定を付けたSQLで全表スキャンが選択された場合にも、1つの大規模なテーブルをすべて読み込むのではなく、対象のデータが含まれる一部のパーティションだけを読み込むことができるため性能の向上が期待できます。

通常の大規模テーブルを検索する場合とパーティションテーブルを検索する場合の図



4.1.2. 運用性の向上

テーブル内のデータを分割して保存することで一括でのデータロードや削除が、パーティションの追加や削除で行える場合あります。例えば、1つの大規模なテーブルで大量のデータを削除する場合は、大量のDELETE文による負荷とそれに伴うVACUUMによる負荷が 発生します。パーティションテーブルであれば、DROP TABLEやTRUNCATEでパーティションやそのデータごと削除できる場合があるため 上述の負荷をかけず高速に大量データを削除することが可能になります。

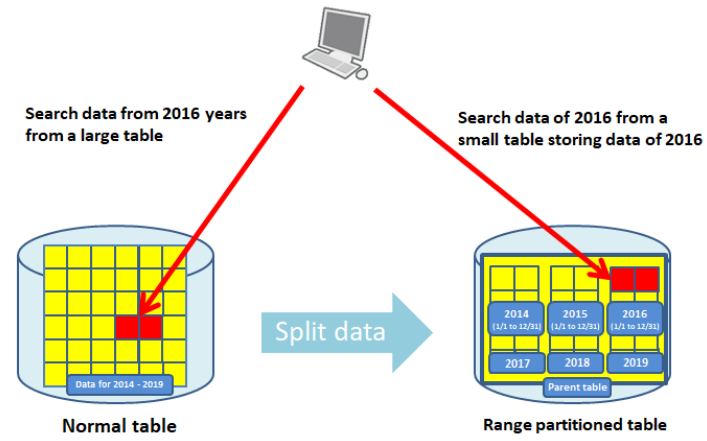
使用頻度の高いデータをSSDなど高価で高性能な記憶メディアに配置し、使用頻度の低いデータを安価で遅い記憶メディアに配置することも可能です。

なお、データ分割方法としては主に以下の3種類があります。

- 範囲（レンジ）パーティション

日付や特定のビジネスオブジェクトの識別子などの範囲（レンジ）でデータを分割する方法です。データの範囲を指定してSQLを実行する場合に有効です。

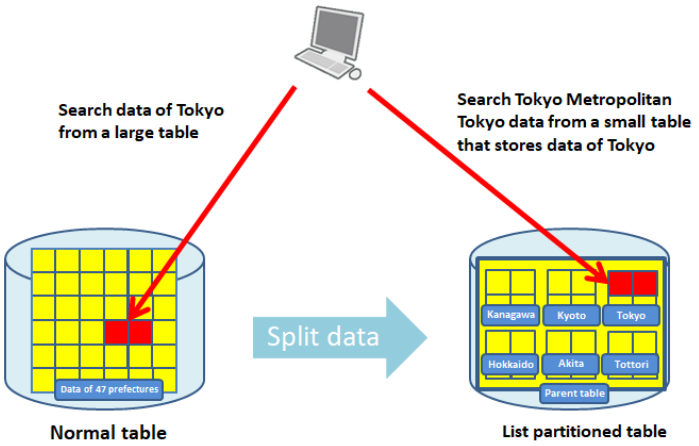
年毎の範囲でテーブルのデータを分割した範囲パーティションの図



- リストパーティション

会社の拠点や都道府県など不連続な値のリストを元にデータを分割する方法です。不連続な列の値を指定してSQLを実行する場合に有効です。

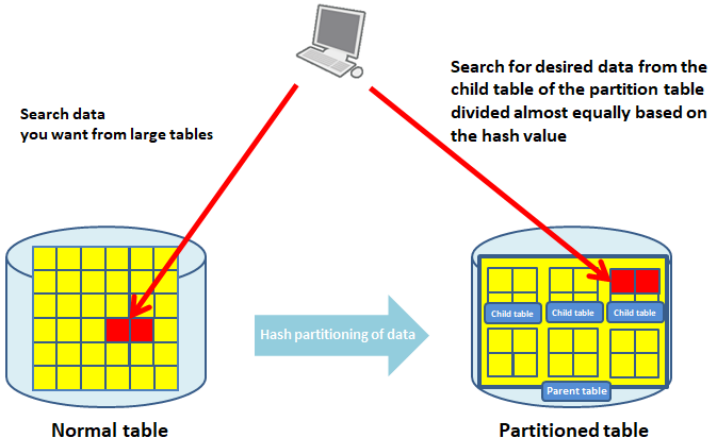
都道府県毎にテーブルのデータを分割したリストパーティションの図



- ハッシュパーティション

列のハッシュ値を元にデータを分割する方法です。テーブルのデータを均等に分割する場合に有効です。

テーブルのデータをバッシュ分割したハッシュパーティションの図



4.2. 継承を使用したパーティショニング

テーブルの「継承」と「CHECK制約」と「トリガー」の機能を使用してデータを分割するパーティションの実装方法です。PostgreSQL 9.6以前のバージョンでパーティショニングを行うには、この方法を採用する必要があります。後述する宣言的パーティショニングより柔軟な構成が行えるという点では優れていますが、データ挿入時の性能が劣っている点や パーティションの構築手順が煩雑である点から、PostgreSQL 10以降では宣言的パーティショニングを採用することが一般的です。

以下に、継承を使用したパーティションテーブルを作成する手順を紹介します。以下の例では、従業員の情報を管理するemployee表のデータを入社日時（HIREDATE列）の年度毎に分けています。

(1) パーティションテーブル（親テーブル）の作成

```
CREATE TABLE employee_range
(EMPLOYEE_ID    INT,
 NAME       VARCHAR(10),
 HIREDATE      DATE NOT NULL,
 LOCATION      VARCHAR(20));
```

(2) 各月のパーティション（子テーブル）作成
CHECK制約でHIREDATE列の期間を指定し、INHERIT（継承）で親テーブル名を指定します。

```
CREATE TABLE employee_range_201801 (
  CHECK ( HIREDATE >= DATE '2018-01-01' AND HIREDATE < DATE '2018-02-01')
  ) INHERITS (employee_range);

CREATE TABLE employee_range_201802 (
  CHECK ( HIREDATE >= DATE '2018-02-01' AND HIREDATE < DATE '2018-03-01')
  ) INHERITS (employee_range);

CREATE TABLE employee_range_201803 (
  CHECK ( HIREDATE >= DATE '2018-03-01' AND HIREDATE < DATE '2018-04-01')
  ) INHERITS (employee_range);

 ～ 中略 ～

CREATE TABLE employee_range_201812 (
  CHECK ( HIREDATE >= DATE '2018-12-01' AND HIREDATE < DATE '2019-01-01')
  ) INHERITS (employee_range);
```

(3) パーティションキーの索引作成

```
CREATE INDEX employee_range_ind_201801 ON employee_range_201801 (HIREDATE);
CREATE INDEX employee_range_ind_201802 ON employee_range_201802 (HIREDATE);
CREATE INDEX employee_range_ind_201803 ON employee_range_201803 (HIREDATE);
～ 中略 ～
CREATE INDEX employee_range_ind_201812 ON employee_range_201812 (HIREDATE);
```

(4) トリガーファンクションの作成

```
CREATE OR REPLACE FUNCTION employee_range_insert_trigger()
RETURNS TRIGGER AS
$$
DECLARE
  part text;
BEGIN
  part := 'employee_range_' || to_char(new.hiredate, 'YYYYMM');
  EXECUTE 'INSERT INTO ' || part || ' VALUES(($1).*)' USING new;
  RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

(5) トリガーの作成（上記トリガーファンクションを呼び出す）

```
CREATE TRIGGER employee_range_insert_trigger
BEFORE INSERT ON employee_range
FOR EACH ROW EXECUTE PROCEDURE employee_range_insert_trigger();
```

4.3. 宣言的パーティショニング

CREATE TABLE構文でPARTITION BY句にパーティション方針（範囲／リスト／ハッシュ）とパーティションキーを 指定してデータを分割するパーティションの実装方法です。PostgreSQL 10以降のバージョンで使用できます。上述した継承を使用したパーティショニングに比べると以下のような制限がありますが、データ挿入時の性能やCREATE TABLE構文で簡単にパーティションを構築できる点などから、PostgreSQL 10以降では宣言的パーティショニングが 採用されることが多くなっています。

- パーティションテーブルでは主キーがサポートされない
- 一意制約や排他制約は個々のパーティション上にのみ作成可能
- 別のパーティションをまたがるUPDATEはできない(PostgreSQL 10のみの制限)
- 行トリガーが必要であれば、パーティションテーブルではなく、個々のパーティションに定義する必要がある
- 一時リレーションと永続的リレーションを同じパーティションツリーに混合することはできない

以下に、継承を使用したパーティショニングで作成したパーティションテーブルと同じものを宣言的パーティショニングで構築する手順を紹介します。継承を使用したパーティションの構築手順よりも短い手順でパーティションテーブルを作成できます。

(1) 親テーブルの作成

```
CREATE TABLE employee_range
(EMPLOYEE_ID  INT,
 NAME   VARCHAR(10),
 HIREDATE  DATE NOT NULL,
 LOCATION  VARCHAR(20))
 PARTITION BY RANGE (HIREDATE);
```

(2) 各月それぞれのパーティションテーブル（子テーブル）を作成

```
 CREATE TABLE employee_range_201801 PARTITION OF employee_range
    FOR VALUES FROM ('2018-01-01') TO ('2018-02-01');

 CREATE TABLE employee_range_201802 PARTITION OF employee_range
    FOR VALUES FROM ('2018-02-01') TO ('2018-03-01');

 CREATE TABLE employee_range_201803 PARTITION OF employee_range
    FOR VALUES FROM ('2018-03-01') TO ('2018-04-01');

～ 中略 ～

 CREATE TABLE employee_range_201812 PARTITION OF employee_range
    FOR VALUES FROM ('2018-12-01') TO ('2019-01-01');
```

(3) パーティションキーの列に索引を作成

```
CREATE INDEX employee_range_ind_201801 ON employee_range_201801 (HIREDATE);
CREATE INDEX employee_range_ind_201802 ON employee_range_201802 (HIREDATE);
CREATE INDEX employee_range_ind_201803 ON employee_range_201803 (HIREDATE);
～ 中略 ～
CREATE INDEX employee_range_ind_201812 ON employee_range_201812 (HIREDATE);
```

4.4. PostgreSQL 11新機能

PostgreSQL 10で実装された宣言的パーティショニングはPostgreSQL 11で以下の新機能が実装されています。
今後も更なる発展が期待されます。

4.4.1. 機能系

- パーティションを跨る更新
- デフォルトパーティションの設定
- パーティションテーブルの主キー、一意キーの設定
- パーティションテーブルへの外部キー
- ハッシュパーティショニング
- パーティションテーブルへの索引作成がパーティションへカスケードする

4.4.2. 性能系

- パーティション選定の高速化
- パーティション・ワイズアグリゲート
- パーティション・ワイズジョイン
- 自動パーティションプルーニング　（本検証では未実施）

5. パーティションテーブルの管理

イメージを掴むため、Oracleの管理SQLと対比します。厳密なSQLリファレンスでありません。

5.1. テーブル

5.1.1. テーブル管理SQL

表記

- 便宜上、パーティションテーブルを「親」、パーティションを「子」と表記します。
- パーティション特有のキーワードは強調表示します。
- パーティションテーブル(親テーブル) : (例) pt1
- パーティション(子テーブル) : (例) p1,p2,...
- テーブルスペース : (例) ts1

表 5.1 テーブル管理SQL

操作	Oracle	PostgreSQL (宣言的パーティション)
パーティション表の作成	1CREATE文で親テーブル、子テーブルを同時に作成できる CREATE TABLE pt1 (列定義) PARTITION BY [RANGE | LIST | HASH] INTERVAL () --インターバルの指定 (RANGEの場合 : PARTITION p1 VALUES LESS THAN ..., LISTの場合 : PARTITION p2 VALUES ..., HASHの場合 : PARTITIONS n (	親テーブルと子テーブルの数だけCREATE文を実行する ・親: CREATE TABLE pt1 (列定義) **PARTITION** [**RANGE** | **LIST** | **HASH**] (part_key); ・子: CREATE TABLE p1 **PARTITION OF** pt1 **FOR VALUES** (範囲) RANGEの場合: **FROM** (xxx) **TO** (yyy) LISTの場合 : **IN** (xxx,yyy) HASHの場合 : **WITH** (**MODULUS** xxx, **REMAINDER** yyy) PostgreSQL 11では"DEFAULT"を指定可能
パーティションの追加	ALTER TABLE pt1 ADD PARTITION p1;	CREATE TABLE p1 **PARTITION OF** pt1 **FOR VALUES**...; (上記参照)
パーティションの削除	ALTER TABLE pt1 DROP PARTITION P1;	DROP TABLE p1;
パーティションの データ削除	ALTER TABLE pt1 TRUNCATE PARTITION p1;	TRUCNATE p1;
パーティション名の変更	ALTER TABLE pt1 RENAME PARTITION p1 TO p2;	ALTER TABLE p1 RENAME TO p2;
パーティションの分割	ALTER TABLE pt1 SPLIT PARTITION p1 AT (xx);	なし
パーティションのマージ	ALTER TABLE pt1 MERGE PARTITIONS p1,p2 INTO PARTITION p3;	なし
パーティションの 表領域移動	ALTER TABLE pt1 MOVE PARTITION p1 TABLESPACE ts1;	ALTER TABLE p1 SET TABLESPACE ts1;
パーティションの活性	ALTER TABLE pt1 MODIFY p1 ONLINE;	なし
パーティションの非活性	ALTER TABLE pt1 MODIFY p1 OFFINE;	なし
テーブルをパーティションに変換	ALTER TABLE pt1 ADD PARTITION p1; ALTER TABLE pt1 EXCHANGE PARTITION p1 WITH TABLE t2;	ALTER TABLE pt1 **ATTACH PARTITION** p1 **FOR VALUES** (..);
パーティションをテーブルに変換	CREATE TABLE t2 (..); ALTER TABLE pt1 EXCHANGE PARTITION p1 WITH TABLE t2;	ALTER TABLE pt1 **DETACH PARTITION** p1;
パーティション単位の参照	SELECT * FROM pt1 PARTITION (p1);	SELECT * FROM p1;
リストパーティションの値の追加	ALTER TABLE pt1 MODIFY PARTITION p1 ADD VALUES (val2);	なし
リストパーティションの値の削除	ALTER TABLE pt1 MODIFY PARTITION p1 DROP VALUES (val2);	なし

5.2. インデックス

5.2.1. Oracleのパーティションインデックス種類

インデックス種別	指定ワード	PostgreSQLで相当する索引の可否
ローカル・同一キー索引	LOCAL	可能
ローカル・非同一キー索引	LOCAL	可能
グローバル・パーティション索引	GLOBAL	不可能
グローバル・非パーティション索引	なし	不可能

5.2.2. インデックス管理SQL

表記

- 便宜上、パーティションテーブルを「親」、パーティションを「子」と表記します。
- パーティション特有のキーワードは強調表示します。
- パーティションテーブル（親）　:(例) pt1
- パーティション（子）　:(例) p1,p2,...
- テーブルスペース　:(例) ts1
- インデックス（親）　:(例) idx1
- インデックス（子）　:(例) idx1_p1,...

表 5.2 インデックス管理SQL

操作	Oracle	PostgreSQL (宣言的パーティション)
ローカル索引	CREATE INDEX idx1 ON pt1 (col1) LOCAL;	CREATE INDEX idx1 ON pt1 (col1); 親に実施した場合、親には索引定義のみ作成、 各子テーブルに索引を作成。 子に実施した場合、特定子テーブルに索引を作成
グローバル・パーティション索引	CREATE INDEX idx1 ON pt1 (col1) GLOBAL PARTITION BY RANGE (col1) (PARTITION p_100 VALUES LESS THAN (value1), PARTITION p_1000 VALUES LESS THAN (value2) ...)	なし
グローバル・非パーティション索引	CREATE INDEX idx1 ON pt1 (col1);	なし
親テーブルの索引定義	なし	CREATE INDEX idx1 ON **ONLY** pt1 (col1); 親の索引に実施、定義のみ作成される
親子の索引の紐づけ	なし	ALTER INDEX idx1 **ATTACH PARTITION** idx1_p1; 親の索引と子の索引を紐付ける
索引パーティションの属性の変更	なし	ALTER INDEX idx1 SET (ATTIBUTE=value); 子の索引に実施
索引パーティションのデフォルト属性の変更	ALTER INDEX idx1 MODIFY DEFAULT ATTRIBUTES	ALTER INDEX idx1 SET (ATTIBUTE=value); 親の索引に実施する事でデフォルト属性の変更に相当 子の既存索引には影響せず、 以後に追加されたパーティション索引作成時に有効
索引パーティションの表領域移動	ALTER INDEX idx1 REBUILD PARTITION pt1 PARAMETERS('TABLESPACE=ts2');	ALTER INDEX idx1 SET TABLESPACE ts2; 親の索引に実施した場合、親の索引の定義のみ変更。 子の既存索引には影響せず、以後に追加されたパーティション索引作成時に有効。 子の索引に実施した場合、特定索引の表領域を移動。
索引パーティション名の変更	ALTER INDEX idx1 RENAME PARTITION idx2;	ALTER INDEX idx1 RENAME TO idx2; 親の索引に実施した場合、 親の索引名のみ変更される。子の既存索引には影響せず、 子に自動作成される索引の命名規則にも影響しない。 <子テーブル>_<列名>_idx 子の索引に実施した場合 、特定索引の名前を変更
索引パーティションの分割	ALTER INDEX idx1 SPLIT PARTITION pi1 AT (value) INTO PARTITION pi2 ..., PARTITION p3 ...); ALTER INDEX idx1 REBUILD PARTITION pi2; ALTER INDEX idx2 REBUILD PARTITION pi3;	なし
索引パーティションの削除(グローバル索引)	ALTER INDEX idx1 DROP PARTITION p1; ALTER INDEX idx1 REBUILD PARTITION P1; (ローカル索引のパーティションは削除できない)	なし
索引パーティションの再構成	ALTER INDEX idx1 REBUILD PARTITION pt1;	REINDEX INDEX idx1; 単位は他に TABLE,SCHEMA,DATABASE,SYSTEMがある 子の索引に対して個別に実施。 親の索引にはREINDEXはできない(エラーが発生)。

5.2.3. 索引作成のカスケード

PostgreSQL 11の宣言的パーティションでは親テーブルに索引を作成する事で、子テーブルに作成されます。

(実行例)
lineorder_yyyyテーブルは年ごとのパーティションテーブル,

```
● PostgreSQL 10 : 索引作成のカスケード不可能
=# CREATE INDEX lineorder_yyyy_orderkey ON lineorder_yyyy (lo_orderkey);
ERROR:  cannot create index on partitioned table "lineorder_yyyy"

● PostgreSQL 11 : 索引作成のカスケード可能
=# CREATE INDEX lineorder_yyyy_ordatedate ON lineorder_yyyy (lo_orderdate);
CREATE INDEX
```

- PostgreSQL 11では子テーブルに索引が作成されていることをシステムカタログから確認

```
=# SELECT tablename,indexname FROM pg_indexes WHERE tablename LIKE 'lineorder_yyyy_%';

      tablename      |              indexname
---------------------+---------------------------------------
 lineorder_yyyy_1992 | lineorder_yyyy_1992_lo_orderdate_idx
 lineorder_yyyy_1993 | lineorder_yyyy_1993_lo_orderdate_idx
 lineorder_yyyy_1994 | lineorder_yyyy_1994_lo_orderdate_idx

・・・略・・・

(7 rows)
```

- パーティションテーブルの索引はrelkind = 'I'（大文字)として作成されている。サイズは0バイト。

```
=# SELECT relname FROM pg_class WHERE relkind='I' AND relname LIKE 'lineorder_yyyy%';

          relname
---------------------------
 lineorder_yyyy_ordatedate
(1 row)

=# SELECT pg_relation_size('lineorder_yyyy_ordatedate');

 pg_relation_size
------------------
0
(1 row)
```

注意点
索引の属性情報（テーブルスペースなど）も引き継ぐため、同一のテーブルスペースに作成される。
親テーブルに作成後にALTER INDEX でローカル索引を指定してテーブルスペースを変更することは可能。

5.2.4. 親テーブル管理の索引の削除

親テーブルで管理されている索引は個別には削除できません。

- （例）lineorder_yyyy_1998_lo_suppkey_idx索引はlineorder_yyyy_1998パーティションの索引

```
=# DROP INDEX lineorder_yyyy_1998_lo_suppkey_idx;
ERROR:  cannot drop index lineorder_yyyy_1998_lo_suppkey_idx because index
lineorder_yyyy_suppkey requires it
```

子テーブル特有の索引を作成することも可能です。
特定子テーブルに索引作成後、親テーブルにその列の索引を作成した場合、事前作成の索引がアタッチされます。
その場合、PostgreSQLの命名規則に則った索引と独自命名の索引が混在します。
特定パーティションに索引を作成後、親テーブルに同一列の索引を作成すると、自動的に親テーブルの索引にアタッチされます。
アタッチされると個別に削除ができません。

```
-- 子テーブルに索引作成
=# CREATE INDEX lineorder_yyyy_1999_partkey ON lineorder_yyyy_1999 (lo_partkey);
CREATE INDEX

-- 親テーブルに索引作成
=# CREATE INDEX lineorder_yyyy_partkey ON lineorder_yyyy (lo_partkey);
CREATE INDEX

-- 親テーブルの索引にアタッチ後、子テーブルの索引単独で削除できないことを確認
=# DROP INDEX lineorder_yyyy_1999_partkey;
ERROR:  cannot drop index lineorder_yyyy_1999_partkey because index
lineorder_yyyy_partkey requires it
```

アタッチ時にはインデックスの作成が行われるため時間を要します（その間、親テーブルはロックされた状態です）。
親テーブルのロック時間を短縮するには、事前に個別インデックスを作成しておくのが有効です。

索引作成文のONLYオプションについて * ONLYを付与することで、既存パーティションに対して索引作成処理が実行されることを回避します。* 親テーブルに索引定義のみ作成し、子テーブルに作成した索引をアタッチします。* なお以後に作成されたパーティションには索引が作成されます。

- （例）lineorder_yyyyパーティションテーブルのlo_partkey列に対する索引

```
-- 親テーブルに ONLYオプションを付与して索引を作成
=# CREATE INDEX lineorder_yyyy_partkey ON ONLY lineorder_yyyy (lo_partkey);
CREATE INDEX

-- 子テーブルに索引作成
=# CREATE INDEX lineorder_yyyy_1998_partkey ON lineorder_yyyy_1998 (lo_partkey);
CREATE INDEX

-- 親テーブルのインデックスに子テーブルのインデックスをアタッチ
=# ALTER INDEX lineorder_yyyy_partkey ATTACH PARTITION lineorder_yyyy_1998_partkey;
ALTER INDEX
```

5.2.5. プライマリキーの作成

PostgreSQL 11では全パーティションに跨るプライマリーキーやユニークインデックスを作成することが可能となりました。
これによりパーティションテーブルのデータの整合性が高まります。
ただし索引キーにはパーティションキーを含む必要があります。

●ケース1 lineorder非パーティションテーブルにlo_orderkey列及びlo_linenumber列でプライマリーキーに作成

```
=#  ALTER TABLE lineorder ADD CONSTRAINT lineorder_pk PRIMARY KEY
(lo_orderkey,lo_linenumeric);
ALTER TABLE
```

●ケース2 lineorder_yyyyパーティションテーブルで同様のインデックス作成を試みるとエラーが発生

```
=# ALTER TABLE lineorder_yyyy ADD CONSTRAINT lineorder_yyyy_pk PRIMARY KEY
   (lo_orderkey,lo_linenumber);
ERROR:  insufficient columns in PRIMARY KEY constraint definition
DETAIL:  PRIMARY KEY constraint on table "lineorder_yyyy" lacks column "lo_orderdate" which
is part of the partition key.
```

●ケース3 lineorder_yyyyパーティションテーブルでパーティションキーであるlo_orderdate列を列構成に含めると作成成功（配置場所は任意）

```
=# ALTER TABLE lineorder_yyyy ADD CONSTRAINT lineorder_yyyy_pk PRIMARY KEY
   (lo_orderkey,lo_linenumber,lo_orderdate);
ALTER TABLE
```

6. パーティションテーブルの性能

6.1. パフォーマンス検証の方針とテーマ

6.1.1. 検証方針

シンプルなSQLによる基礎的な性能検証を目的としています。そのためスタースキーマ構成などの複雑なSQLの性能検証は実施していません。

6.1.2. 検証テーマ

- 宣言的パーティショニングによる挿入処理(COPY FROM / INSERT)の改善
  - 継承を使用したパーティショニングの課題であった挿入処理が宣言的パーティショニングで改善されていることを確認します。
- パーティションテーブルと非パーティションテーブルの特性
  - 非パーティションテーブル/パーティションテーブルがそれぞれ優位となるケースを確認します。
- パーティションの粒度（パーティション数）
  - 年 / 年月 / 年月日　など粒度を細かくすることのメリットと注意点を確認します。
- パーティション選択の高速化
  - パーティション選択の性能がPostgreSQL 11で改善されていることを確認します。
- パラレルワーカによるパーティション走査の高速化
  - 並列度を増加することで高速化することを確認します。
  - PostgreSQL 11で改善されていることを確認します。
- 集計処理（パーティション・ワイズアグリゲート）
  - 集計処理の性能がPostgreSQL 11で改善されていることを確認します。
- 結合処理（パーティション・ワイズジョイン）
  - パーティションテーブル同士の結合処理がPostgreSQL 11で改善されていることを確認します。
- 削除処理（DELETE/TRUNCATE）
  - パーティションテーブルのパーティション単位のDELETEやTRUNCATEの効果を確認します。

6.2. 検証構成

6.2.1. 検証ハードウェア構成

検証環境のハードウェア構成の概略を示します。
本性能検証ではDBサーバ上でSQLを直接実行する形式としています。
ハードウエア構成

項目	仕様
CPU	Intel Xeon CPU E5-2640 v2 @ 2.00GHz * 32 Core
搭載メモリ	64GB
内蔵ストレージ	約 2TB

6.2.2. 検証ソフトウェア構成

検証環境のソフトウェア構成を示します。

ソフトウエア構成

項目	仕様
OS	Red Hat Enterprise Linux Server release 6.5 (Santiago)
PostgreSQL	PostgreSQL 10.1 PostgreSQL 11.1

6.2.3. PostgreSQLパラメータ

PostgreSQLのパラメータを示します。

- PostgreSQL 11で導入されたパラメータには(11)を付記しています。
- パーティショニング関連は太字表記としています。
- パラレル関連についてはパラレルの検証以外はデフォルト設定です。

- 共有バッファはテーブル(lineorder系)が格納できるサイズを確保し、pg_prewarmでメモリに配置しています。

PostgreSQL 主要パラメーター一覧

分類	パラメータ	設定値	備考
パラレル	max_parallel_workers_per_gather	2	デフォルト
	max_parallel_workers	8	デフォルト
チェックポイント	checkpoint_timeout	1d	暗黙的にチェックポイントを発生させない ことを目的とした設定
	max_wal_size	100GB	
共有メモリ	shared_buffers	5GB	lineorderテーブルが搭載できるサイズ 処理前にpg_prewarmでメモリに搭載
プランナ	effective_cache_size	10GB	shared_buffersの2倍
	enable_parallel_append (11)	on	パラレルクエリにてパラレルAppend有効
	enable_parallel_hash (11)	on	パラレルクエリにてパラレルHash有効
	enable_partition_pruning (11)	on	パーティションプルーニング有効
	enable_partitionwise_aggregate (11)	on	パーティションワイズアグリゲート有効
	enable_partitionwise_join (11)	on	パーティションワイズジョン有効
	parallel_leader_participation (11)	on	パラレルクエリにてリーダプロセスがワーカ プロセスと同様の処理を行わない
ワークメモリ	work_mem	1GB	ソート用メモリサイズ
	maintenance_work_mem	1GB	インデックス作成用メモリサイズ
トランザクション	max_locks_per_transaction	3000	共有ロックテーブルのエントリ数であり、パーティション数より多く設定する

6.3. 使用データ

6.3.1. lineorderテーブル

StarSchemaBenchmarkのlineorderを使用しました。
https://github.com/lemire/StarSchemaBenchmark

データはスケール4を指定して作成しています。

表 6.1 データ件数

件数	サイズ
約2,400万件	2.3GB

表 6.2 lineorderの年別の件数と割合

年	件数	割合
1992	3,642,219	15.2%
1993	3,647,031	15.2%
1994	3,641,723	15.2%
1995	3,635,511	15.2%
1996	3,654,372	15.2%
1997	3,639,076	15.2%
1998	2,136,738	8.9%

最終年以外はほぼ均等に分布しています。

6.3.2. 使用するパーティションテーブル

- lineorderテーブルを使用して、以下の各粒度で作成しています。
- 全てレンジパーティションによる時系列のパーティション表です。

- なお検証ケースによっては別の構成を使用する場合もあります。
  - コンポジットパーティション
  - パーティションテーブル同士の結合

表 6.3 パーティションテーブル

実装方式	粒度(パーティション数)	親テーブル名	子テーブル名(例)
継承によるパーティショニング	年 (7)	lineorder_9_yyyy	lineorder_9_yyyy_1992, lineorder_9_yyyy_1993, ... lineorder_9_yyyy_1997, lineorder_9_yyyy_1998
宣言的パーティショニング	年 (7)	lineorder_yyyy	lineorder_yyyy_1992, lineorder_yyyy_1993, ... lineorder_yyyy_1997, lineorder_yyyy_1998
宣言的パーティショニング	年月 (84)	lineorder_yyyymm	lineorder_yyyymm_199201, lineorder_yyyymm_199202, ... lineorder_yyyymm_199811, lineorder_yyyymm_199812
宣言的パーティショニング	年月日 (2556)	lineorder_yyyymmdd	lineorder_yyyymmdd_19920101, lineorder_yyyymmdd_19920102, ... lineorder_yyyymmdd_19981230, lineorder_yyyymmdd_19981231

6.3.3. 各方式で共通の手順

基本的に以下はすべての検証ケースでセットで実行しています。

- pg_prewarmで共有メモリに載せます。パーティションテーブルを指定できないため、パーティション毎に実施します。
- VACUUMを実行して、Index Only Scan が効くようにします。パーティションテーブルを指定できます。

6.4. COPY FROM

継承によるパーティショニングでは挿入処理のパフォーマンスが課題の一つでした。
挿入処理はトリガーベースとなるためオーバーヘッドが発生するためです。
ここではCOPY FROM による挿入処理を検証します。
主な確認ポイント

- 宣言的パーティショニングでは継承によるパーティショニングより改善していること

6.4.1. 検証方法(COPY FROM)

以下の構成のテーブルを使用します。

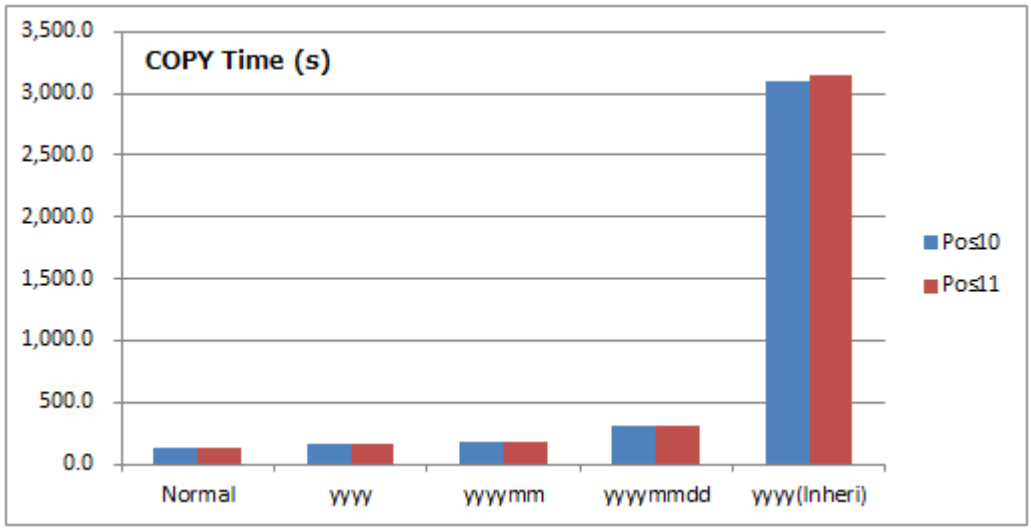
表 6.4 使用するテーブル

構成	グラフの凡例表記
非パーティションテーブル	Normal
宣言的パーティションテーブル(年)	yyyy
宣言的パーティションテーブル(年月)	yyyymm
宣言的パーティションテーブル(年月日)	yyyymmdd
継承によるパーティションテーブル(年)	yyyy(Inheri)

CSVファイルを単一のCOPY FROM文にて各構成テーブルへロードします。
(非パーティション表の場合)

```
=# CHECKPOINT;
=# BEGIN;
=# TRUNCATE lineorder;
=# COPY lineorder FROM '/tmp/lineorder.dat' DELIMITER '|';
=# COMMIT;
```

6.4.2. 結果(COPY FROM)



COPY FROM の所要時間比較

- 青色がPostgreSQL 10、茶色がPostgreSQL 11を表しています。縦軸の目盛りを合わせています。
- 縦軸は時間（秒）であり、小さい方が優れています。

6.4.3. 考察（COPY FROM）

- 宣言的パーティションは継承によるパーティションの5%程度であり、圧倒的に高速化しています。
- PostgreSQL 10とPostgreSQL 11では、有意な違いはありません。

6.5. INSERT

ここではINSERTによる挿入処理の処理時間を検証します。
またPostgreSQL 11では、パーティション選定処理が高速化されています。
多数のINSERT文を発行することで確認を試みます。
主な確認ポイント

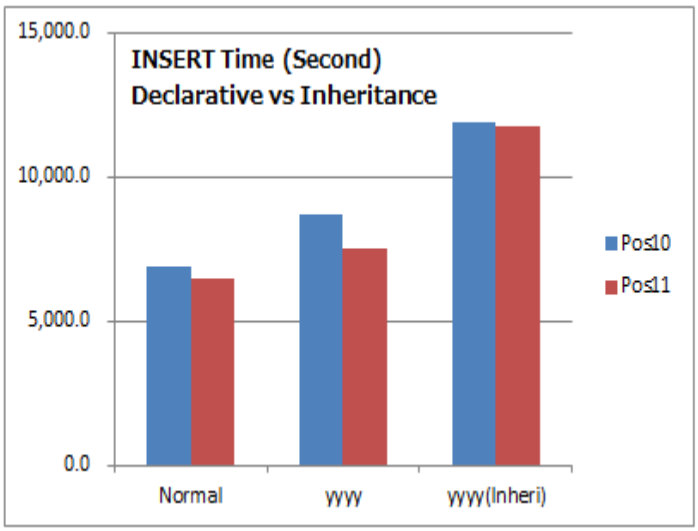
- 宣言的パーティショニングでは継承によるパーティショニングより改善していること
- 宣言的パーティショニングにおいて、PostgreSQL 11 はPostgreSQL 10より改善していること

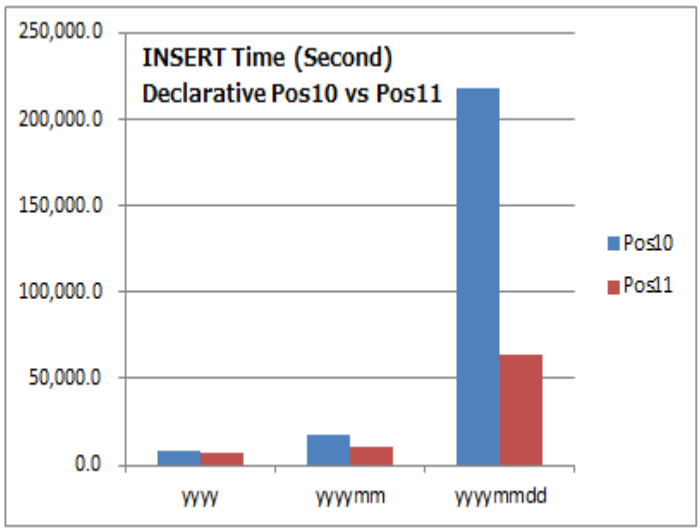
6.5.1. 検証方法（INSERT）

テーブル構成はCOPY FROM と同じです。
件数分(約2,400万件）のINSERT文を実行します。
(非パーティション表の場合)

```
=# CHECKPOINT;
=# BEGIN;
=# TRUNCATE lineorder;
=# INSERT INTO lineorder VALUES (...); --1件目
=# INSERT INTO lineorder VALUES (...); --2件目
=# INSERT INTO lineorder VALUES (...); --3件目
...
=# INSERT INTO lineorder VALUES (...); --2,400万件目
=# COMMIT;
```

6.5.2. 結果（INSERT）





INSERT の所要時間比較

- 青色がPostgreSQL 10、茶色がPostgreSQL 11を表しています。縦軸の目盛りが異なることにご注意ください。
- 縦軸は時間（秒）であり、小さい方が優れています。
- 左は、「非パーティション表」、「宣言的パーティション(年)」、「継承によるパーティション(年)」のパーティション方式による比較です。
- 右は、「宣言的パーティション」に限定し、PostgreSQL 10とPostgreSQL 11の粒度毎の比較です。

6.5.3. 考察（INSERT）

- 宣言的パーティションは継承によるパーティションより36%程度時間が短縮しています（左図）。
- PostgreSQL 11の宣言的パーティションはPostgreSQL 10より70%程度時間が短縮されています（右図）。パーティション選定処理の高速化によるものと考えられます。PostgreSQL 11のリリースノート より
  - E.3.3.1.1. Partitioning
  - Allow faster partition elimination during query processing
- PostgreSQL 11の宣言的パーティションでもパーティション数が増えるほど時間を要するする傾向があります（右図）。

6.6. アナライズ

アナライズはプランナ統計情報を更新する処理であり、メンテナンス作業として重要です。パーティション数が増えることの影響度を確認します。

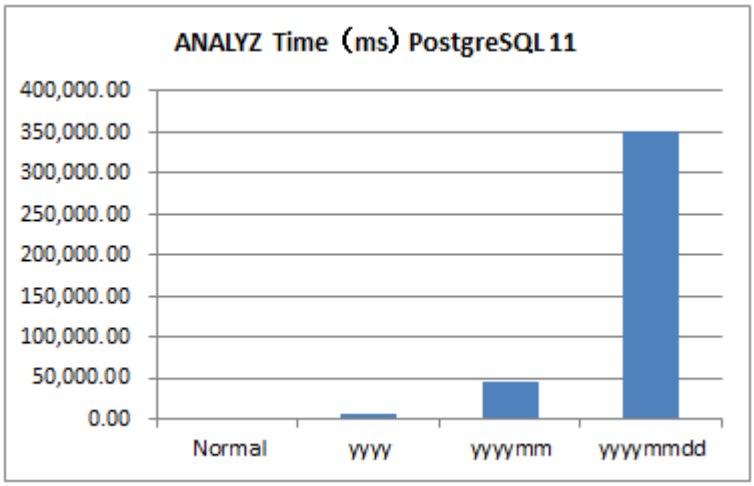
6.6.1. 検証方法（アナライズ）

パーティション表では親テーブルに対してアナライズを実行します。
アナライズは親テーブルに実行すると子テーブルにカスケードして実行されます。
テーブル構成はインデックス作成と同一です。

非パーティションの場合のSQL

```
=# ANALYZE VERBOSE lineorder;
```

6.6.2. 結果（アナライズ）



アナライズの所要時間比較

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。

6.6.3. 考察（アナライズ）

- パーティション表ではパーティション数に応じて所要時間が長くなる傾向にあります。
- （子）テーブル毎の走査ページおよび件数にはしきい値があり、それを上回っているテーブルではサンプリングで実施します。
- しきい値を下回るとサンプリングではなく全ページあるいは全件が対象となります。

表 6.5 アナライズにおけるサンプル走査ページ数および件数

構成	テーブル数	（子）テーブル毎の走査ページ数	（子）テーブル毎の対象件数
非パーティションテーブル	1	30,000ページ	30,000件
宣言的パーティションテーブル（年）	7	30,000ページ	30,000件
宣言的パーティションテーブル（年月）	84	約 5,500ページ（全ページ）	30,000件
宣言的パーティションテーブル（年月日）	2,556	約 180ページ（全ページ）	約 10,000件（全件）

```
●非パーティションテーブルの場合
=# ANALYZE VERBOSE lineorder;
...
INFO:  "lineorder": scanned 30000 of 436302 pages, ...; 30000 rows in
sample, ...

●パーティションテーブル（年）の場合
=# ANALYZE VERBOSE lineorder_yyyy;
...
INFO:  "lineorder_yyyy_1992": scanned 30000 of 66222 pages, ...; 30000 rows
in sample, ...
INFO:  "lineorder_yyyy_1993": scanned 30000 of 66310 pages, ...; 30000 rows
in sample, ...
...

●パーティションテーブル（年月）の場合
=# ANALYZE VERBOSE lineorder_yyyymm;
...
INFO:  "lineorder_yyyymm_199201": scanned 5645 of 5645 pages, ...; 30000
rows in sample, ...
INFO:  "lineorder_yyyymm_199202": scanned 5259 of 5259 pages, ...; 30000
rows in sample, ...
...

●パーティションテーブル（年月日）の場合
=# ANALYZE VERBOSE lineorder_yyyymmdd;
...
INFO:  "lineorder_yyyymmdd_19920101": scanned 179 of 179 pages, ...; 9837
rows in sample, ...
INFO:  "lineorder_yyyymmdd_19920102": scanned 182 of 182 pages, ...; 9998
rows in sample, ...
...
```

- 自動アナライズにおいては更新されたパーティションのみ実施されるため、問題はないと考えられます。

6.7. テーブル走査（索引がない場合）

集計処理を含まないシンプルな走査におけるパーティションの優位性を確認します。
ここでは非パーティションおよびパーティションともに条件に指定した列に索引がない場合です。

6.7.1. 検証方法（索引がない場合）

- バージョン
  - PostgreSQL 11
- テーブル構成

表 6.6 テーブル構成（索引がない場合）

テーブル構成	グラフの凡例表記
非パーティションテーブル	Normal
宣言的パーティションテーブル（年）	yyyy
宣言的パーティションテーブル（年月）	yyyymm
宣言的パーティションテーブル（年月日）	yyyymmdd

- 検索範囲

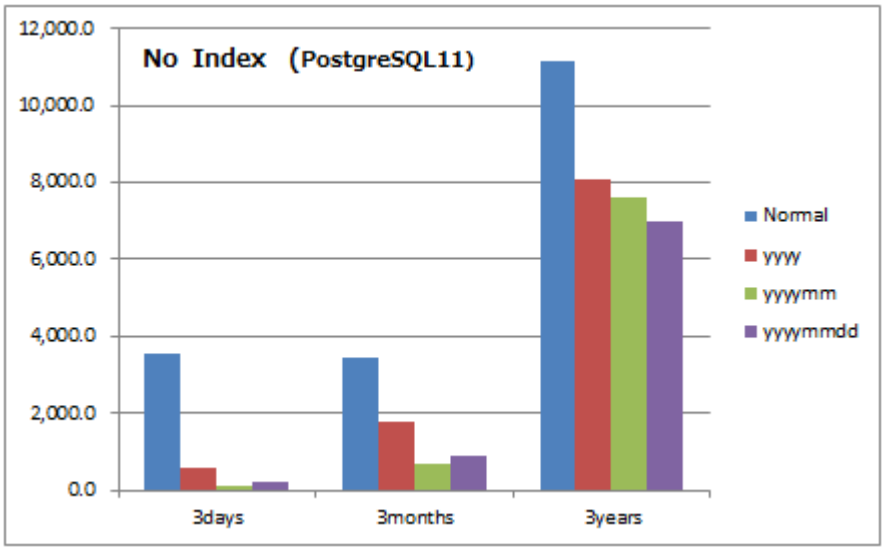
表 6.7 検索範囲（索引がない場合）

検索範囲	条件指定	グラフの凡例表記
3日	lo_orderdate BETWEEN 19920101 AND 19920103	3days
3月	lo_orderdate BETWEEN 19920101 AND 19920331	3months
3年	lo_orderdate BETWEEN 19920101 AND 19941231	3years

- 非パーティションで検索範囲3日の場合のSQL

```
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT * FROM lineorder
   WHERE  lo_orderdate BETWEEN 19920101 AND 19920103;
```

6.7.2. 結果（索引がない場合）



テーブルスキャンの所要時間（索引がある場合）

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。

6.7.3. 考察（索引がない場合）

- 非パーティションテーブルではSeq scanとなるため、プルーニングが効くパーティションテーブルが明確に優位です。
- 非パーティションテーブルと年月パーティションの比較では、3日で **96%** 、3月で **80%** 、3年で **32%** 改善されています。

6.8. テーブル走査（索引がある場合）

集計処理を含まないシンプルな走査におけるパーティションの優位性を確認します。
ここでは非パーティションテーブルおよびパーティションテーブルともに条件に指定した列に索引がある場合です。

索引は通常の **B-Treeインデックス** を使用しています。
意思決定支援システム用途としては **BRINインデックス** も有力ですが、以下の理由から取り止めました。

- 通常のB*Tree索引で高速であったため
- データ挿入が頻繁に行われるシステムを想定しているため

6.8.1. 検証方法（索引がある場合）

- バージョン
  - PostgreSQL 11
- テーブル構成

表 6.8 テーブル構成（索引がない場合）

構成	グラフの凡例表記
非パーティションテーブル	Normal
宣言的パーティションテーブル（年）	yyyy
宣言的パーティションテーブル（年月）	yyyymm
宣言的パーティションテーブル（年月日）	yyyymmdd

- 検索範囲

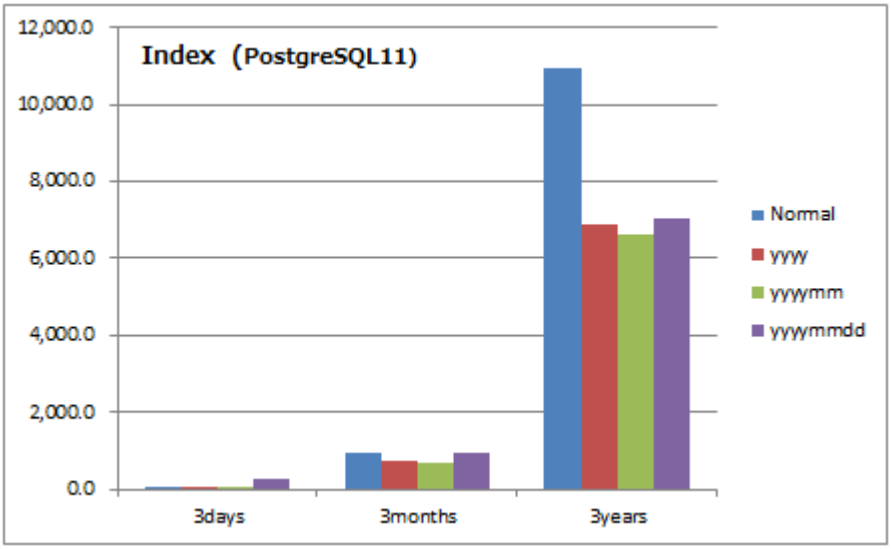
表 6.9 検索範囲（索引がある場合）

検索範囲	条件指定	グラフの凡例表記
3日	lo_orderdate BETWEEN 19920101 AND 19920103	3days
3月	lo_orderdate BETWEEN 19920101 AND 19920331	3months
3年	lo_orderdate BETWEEN 19920101 AND 19941231	3years

- 非パーティションで検索範囲3日の場合のSQL

```
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT * FROM lineorder
   WHERE  lo_orderdate BETWEEN 19920101 AND 19920103;
```

6.8.2. 結果（索引がある場合）



テーブルスキャンの所要時間（索引あり）

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。
- 左がPostgreSQL 10、右がPostgreSQL 11です。縦軸の目盛りを合わせています。

6.8.3. 考察（索引がある場合）

- 非パーティション表も索引があると限定的な範囲ではパーティション表と遜色ありません。
- ただし索引走査は検索範囲が一定割合を超えると **全表走査に切り替わる** ため、パーティションより遅くなります。
- パーティションプルーニングは安定して動作します。

6.9. パラレル走査

パーティションテーブルにおけるパラレルクエリの影響度を検証します。

- PostgreSQL 11においてパーティション走査が改善されていること
- 並列度（ワーカー数）を増加することで高速化すること

なお日付条件は指定しないこととします。

6.9.1. 検証方法（パラレル）

重要度列(lo_orderpriority)に索引を作成し、索引が使用される条件を付与します。
パラレル度を 2,4,6と増加させて構成毎の効果の有無を確認します。
日付条件を指定しないため、パーティションテーブルではプルーニングは行いません。

- バージョン
  - PostgreSQL 10 および PostgreSQL 11
- テーブル構成

表 6.10 テーブル構成

構成	グラフの凡例表記
非パーティションテーブル	Normal
宣言的パーティションテーブル（年）	yyyy
宣言的パーティションテーブル（年月）	yyyymm
宣言的パーティションテーブル（年月日）	yyyymmdd

- 並列度

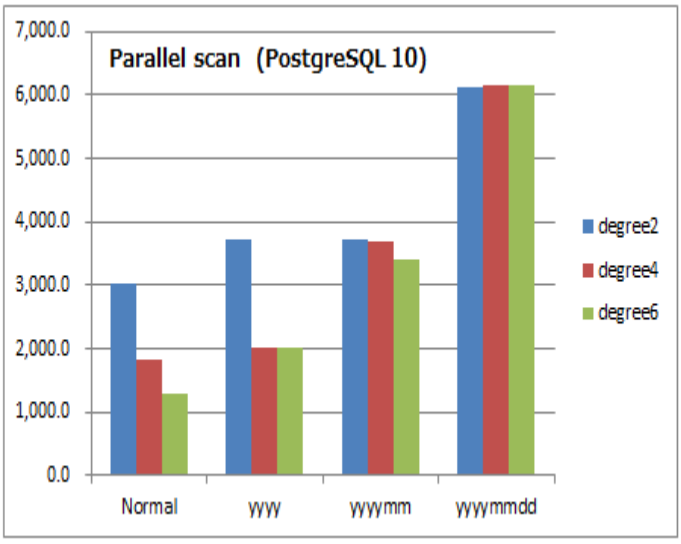
表 6.11 並列度

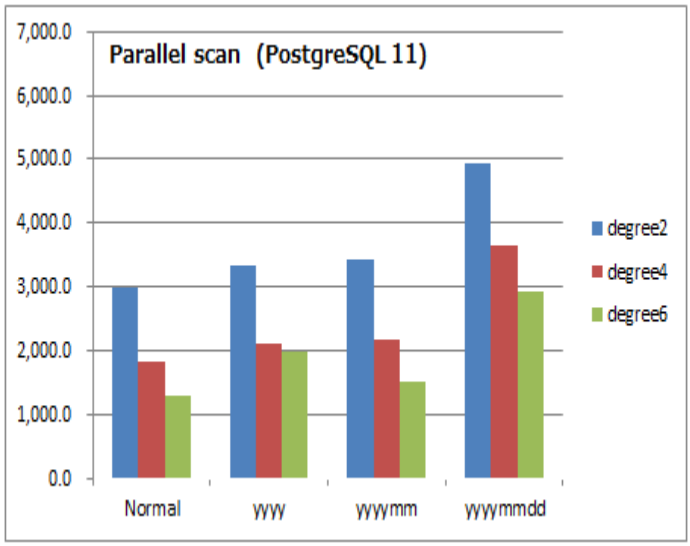
並列度	パラメータ設定	グラフの凡例表記
並列度2	SET max_parallel_workers_per_gather=2;	degree2
並列度4	SET max_parallel_workers_per_gather=4;	degree4
並列度6	SET max_parallel_workers_per_gather=6;	degree6

- 非パーティションの場合のSQL

```
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT count(*) FROM lineorder
   WHERE  lo_orderpriority IN ('1-URGENT','2-HIGH');
```

6.9.2. 結果（パラレル）





パラレルスキャンの所要時間

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。
- 左がPostgreSQL 10、右がPostgreSQL 11です。縦軸の目盛りを合わせています。

6.9.3. 考察（パラレル）

- PostgreSQL 11ではパラレル走査が高速化されています。特に年月日パーティションテーブルでの改善が顕著です。
- パーティションにおけるパラレル走査の効率化によるものと考えられます。PostgreSQL 11のリリースノート より
  - E.3.3.1.2. Parallel Queries
  - Allow partition scans to more efficiently use parallel workers
- PostgreSQL11では並列度（ワーカー数）を増加することで高速化する余地があります。
- プルーニングを行わない状況では、非パーティションテーブルが最も高速です。

6.10. 集計処理

集計処理を含む走査におけるパーティションの優位性を確認します。
非パーティション表およびパーティション表ともに条件に指定した列に索引がある場合で特性を確認します。
PostgreSQL 11では集計合処理が効率化されました(**パーティション・ワイズアグリゲート**）。
PostgreSQL 10では各パーティションをAppendで統合した後に集計します。
PostgreSQL 11ではAppend前にパーティション毎に集計することで効率化しています。

本機能を有効化するためには、以下のパラメータを設定する必要があります。

表 6.12 パラメータ

パラメータ	設定値	備考
enable_partitionwise_aggregate	on (有効)	デフォルトはoff(無効)

6.10.1. 検証方法（集計）

- バージョン
  - PostgreSQL 10 および PostgreSQL 11
- テーブル構成

表 6.13 使用するテーブル

構成	グラフの凡例表記
非パーティションテーブル	Normal
宣言的パーティションテーブル（年）	yyyy
宣言的パーティションテーブル（年月）	yyyymm
宣言的パーティションテーブル（年月日）	yyyymmdd

- 検索範囲

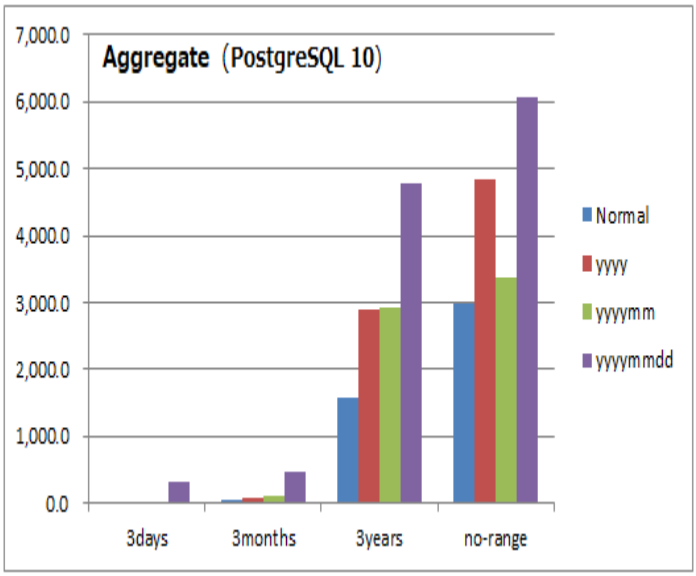
表 6.14 検索範囲（索引がある場合）

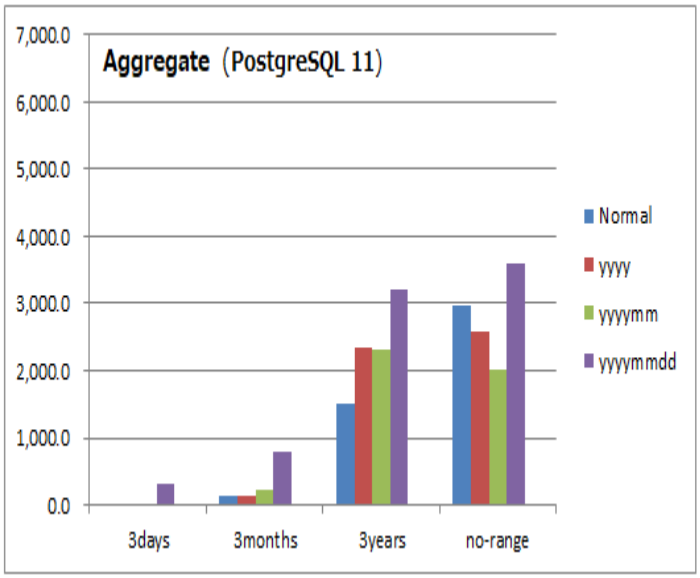
検索範囲	条件指定	グラフの凡例表記
3日	lo_orderdate BETWEEN 19920101 AND 19920103	3days
3月	lo_orderdate BETWEEN 19920101 AND 19920331	3months
3年	lo_orderdate BETWEEN 19920101 AND 19941231	3years
範囲指定なし		no-range

- 非パーティションで検索範囲3日の場合のSQL

```
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT count(*) FROM lineorder
   WHERE  lo_orderdate BETWEEN 19920101 AND 19920103;
```

6.10.2. 結果（集計）





集計処理の所要時間

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。
- 左がPostgreSQL 10、右がPostgreSQL 11です。縦軸の目盛りを合わせています。

6.10.3. 考察（集計）

- PostgreSQL 11にて改善されていることが確認できました。
- だだしPostgreSQL11においても非パーティションが最速となるケースが多くなっています。
- パーティションの粒度としては、年月がバランスが良い結果となっています。

- 実行計画では以下の違いがあります。
  - PostgreSQL 10 : Parallel Append → 一括Partial Aggregate
  - PostgreSQL 11 : パーティション毎のPartial Aggregate → Parallel Append

```
●PostgreSQL 10
=# EXPLAIN SELECT COUNT(*) FROM lineorder_yyyy;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=559238.45..559238.46 rows=1 width=8)
   ->  Gather  (cost=559238.24..559238.45 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=558238.24..558238.25 rows=1 width=8)
               ->  Parallel Append  (cost=0.43..533241.69 rows=9998619 width=0)
                     ->  Parallel Index Only Scan using lineorder_yyyy_1996_lo_partkey_idx...
                     ->  Parallel Index Only Scan using lineorder_yyyy_1993_lo_partkey_idx...
                     ->  Parallel Index Only Scan using lineorder_yyyy_1992_lo_partkey_idx...
             ～ 中略 ～
                     ->  Parallel Index Only Scan using lineorder_yyyy_1998_lo_partkey_idx...
```

```
●PostgreSQL 11
=# EXPLAIN SELECT COUNT(*) FROM lineorder_yyyy;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=509246.68..509246.69 rows=1 width=8)
   ->  Gather  (cost=78242.82..509246.65 rows=14 width=8)
         Workers Planned: 2
         ->  Parallel Append  (cost=77242.82..508245.25 rows=7 width=8)
               ->  Partial Aggregate  (cost=77397.50..77397.51 rows=1 width=8)
                     ->  Parallel Index Only Scan using lineorder_yyyy_1996_lo_partkey_idx...
               ->  Partial Aggregate  (cost=77242.82..77242.83 rows=1 width=8)
                     ->  Parallel Index Only Scan using lineorder_yyyy_1993_lo_partkey_idx...
               ->  Partial Aggregate  (cost=77141.25..77141.26 rows=1 width=8)
                     ->  Parallel Index Only Scan using lineorder_yyyy_1992_lo_partkey_idx...
            ～ 中略 ～
               ->  Partial Aggregate  (cost=45260.90..45260.91 rows=1 width=8)
                     ->  Parallel Index Only Scan using lineorder_yyyy_1998_lo_partkey_idx...
```

6.11. コンポジット構成

サブパーティショニングと呼ばれる手法を使用して、パーティションのパーティションとして多段に定義することができます。
パーティション表（**親**）- パーティション（**子**）- サブパーティション（**孫**）
この構成では、サブパーティションにのみデータが挿入されます。
このような多段構成のパーティションはコンポジットパーティションと呼ばれます。

今回は以下のレンジ-リスト構成にて検証しました。
構成：lo_orderdate列（**レンジ**）- lo_orderpriority列（**リスト**）

lo_orderpriority列には重要度がセットされており、以下の値がほぼ均等に配分されています。

- '1-URGENT'
- '2-HIGH'
- '3-MEDIUM'
- '4-NOT SPECIFIED'
- '5-LOW'

6.11.1. 検証方法（コンポジット）

以下の構成のテーブルを使用します。

表 6.15 使用するテーブル

テーブル構成	グラフの凡例表記
非パーティションテーブル	Normal
宣言的パーティションテーブル（年）	yyyy
宣言的コンポジットパーティションテーブル（年-重要度）	yyyy-pri

PostgreSQL11のみを対象としています。

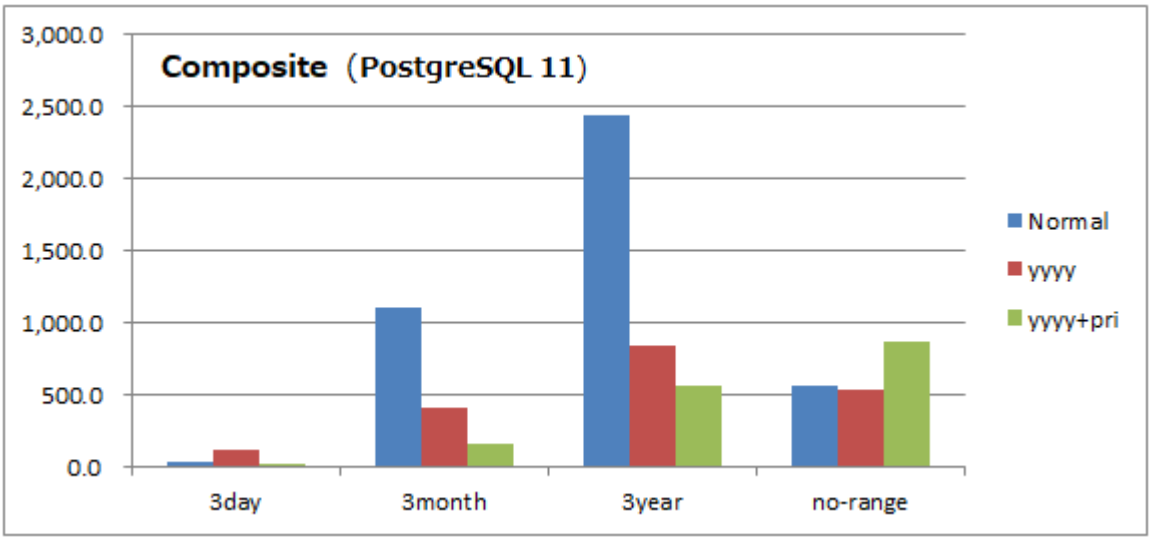
表 6.16 検索範囲

検索範囲	条件指定	グラフの凡例表記
3日	lo_orderdate BETWEEN 19920101 AND 19920103	3day
3月	lo_orderdate BETWEEN 19920101 AND 19920331	3month
3年	lo_orderdate BETWEEN 19920101 AND 19941231	3year
全範囲	（日付条件なし）	no-range

非パーティションで範囲3日の検索の場合

```
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT count(*) FROM lineorder
   WHERE  lo_orderdate BETWEEN 19920101 AND 19920103
   AND    lo_orderpriority IN ('1-URGENT');
```

6.11.2. 結果（コンポジット）



集計処理の所要時間

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。

6.11.3. 考察（コンポジット）

- コンポジット構成に沿った日付と重要度の2つの条件がある場合は、コンポジット構成は高い性能を発揮します。
- ただし日付条件がない場合はコンポジット構成が最も時間を要しています。カスタマイズされた構成であるため汎用性は低下することにご注意ください。

6.12. パーティションテーブル同士の結合

パーティションキーが同一のパーティションテーブル同士の結合処理の性能を検証します。
PostgreSQL 11では同一構成のパーティションテーブルの結合処理が効率化されました(**パーティション・ワイズジョイン**）。
PostgreSQL 10では各パーティションをAppendで統合した後に結合します。
PostgreSQL 11ではAppend前にパーティション毎に少ない件数でハッシュ結合することで効率化しています。

本機能を有効化するためには、以下のパラメータを設定する必要があります。

表 6.17 パラメータ

パラメータ	設定値	備考
enable_partitionwise_join	on (有効)	デフォルトはoff(無効)

6.12.1. 検証方法（結合）

それぞれ同じ粒度のパーティションテーブルを用意して結合します。

- バージョン
  - PostgreSQL 10 および PostgreSQL 11
- テーブル構成

表 6.18 テーブル構成（結合）

構成	グラフの凡例表記
非パーティションテーブル	Normal
宣言的パーティションテーブル（年）	yyyy
宣言的パーティションテーブル（年月）	yyyymm

- 検索範囲

表 6.19 検索範囲（結合）

検索範囲	条件指定	グラフの凡例表記
3日	lo_orderdate BETWEEN 19920101 AND 19920103	3days
3月	lo_orderdate BETWEEN 19920101 AND 19920331	3months

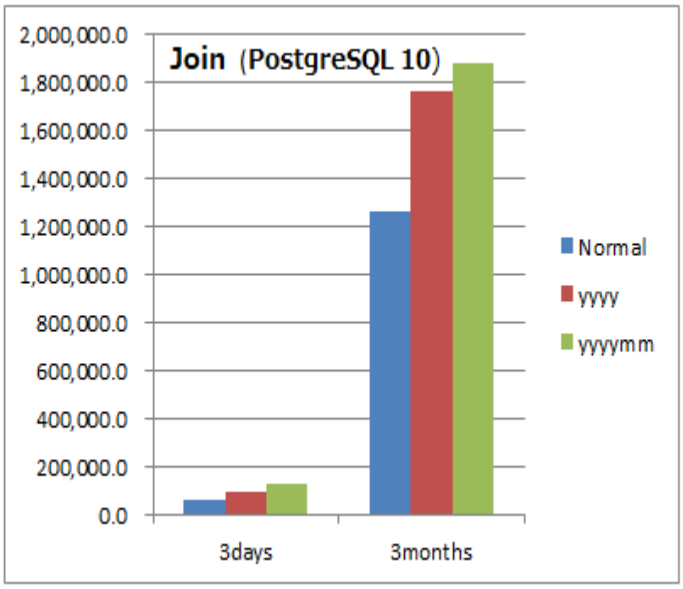
- 各構成の検索範囲3日の場合のSQL

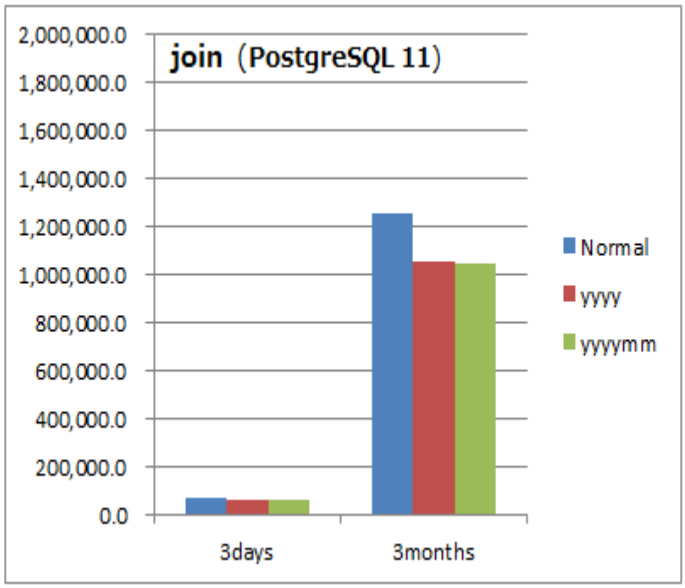
```
●非パーティションの場合
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT count(*)
   FROM   lineorder  l1,
          lineorder2 l2
   WHERE  l1.lo_orderdate = l2.lo_orderdate
   AND    l1.lo_orderdate between 19920101 AND 19920103;

●年パーティションの場合
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT count(*)
   FROM   lineorder_yyyy  l1,
          lineorder2_yyyy l2
   WHERE  l1.lo_orderdate = l2.lo_orderdate
   AND    l1.lo_orderdate between 19920101 AND 19920103;

●年月パーティションの場合
=# EXPLAIN (ANALYZE,BUFFERS)
   SELECT count(*)
   FROM   lineorder_yyyymm  l1,
          lineorder2_yyyymm l2
   WHERE  l1.lo_orderdate = l2.lo_orderdate
   AND    l1.lo_orderdate between 19920101 AND 19920103;
```

6.12.2. 結果（結合）





パーティション表同士の結合処理の所要時間

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。
- 左がPostgreSQL 10、右がPostgreSQL 11です。縦軸の目盛りを合わせています。

6.12.3. 考察（結合）

- PostgreSQL 11ではパーティションワイズジョイン機能により明確に改善されています。
- PostgreSQL 10では、パーティション表同士の結合は非パーティション表同士の結合より遅くなっていましたが、PostgreSQL 11では非パーティ

ション表同士の結合より速くなっています。

- 実行計画では以下の違いがあります。
    - PostgreSQL 10 : Appned → Merge Join → 一括Partial Aggregate
      日付条件が指定されていないテーブルはプルーニングが効きません。
    - PostgreSQL 11 : パーティション毎のParallel Hash Join → パーティション毎のPartial Aggregate → Parallel Append
      一つのテーブルに日付条件が指定されていれば、他のテーブルもプルーニングが効きます。

```
●PostgreSQL 10
=# EXPLAIN
   SELECT count(*)
   FROM   lineorder_yyyymm  l1,
          lineorder2_yyyymm l2
   WHERE  l1.lo_orderdate = l2.lo_orderdate
   AND    l1.lo_orderdate between 19920101 AND 19920331;
                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
Finalize Aggregate  (cost=794708399.23..794708399.24 rows=1 width=8)
  ->  Gather  (cost=794708399.02..794708399.23 rows=2 width=8)
        Workers Planned: 2
        ->  Partial Aggregate  (cost=794707399.02..794707399.03 rows=1 width=8)
              ->  Merge Join  (cost=1767840.77..681471503.59 rows=45294358172 width=0)
                    Merge Cond: (l2.lo_orderdate = l1.lo_orderdate)
                    ->  Sort  (cost=1648107.56..1673108.58 rows=10000407 width=7)
                          Sort Key: l2.lo_orderdate
                          ->  Append  (cost=0.42..485382.47 rows=10000407 width=7)
                                ->  Parallel Index Only Scan using lineorder2_yyyymm_199201_...
                                ->  Parallel Index Only Scan using lineorder2_yyyymm_199202_...
                                ->  Parallel Index Only Scan using lineorder2_yyyymm_199203_...
                                    ～ 中略 ～
                                ->  Parallel Index Only Scan using lineorder2_yyyymm_199810_...
                                ->  Parallel Index Only Scan using lineorder2_yyyymm_199811_...
                                ->  Parallel Index Only Scan using lineorder2_yyyymm_199812_...
```

```
●PostgreSQL 11
=# EXPLAIN
   SELECT count(*)
   FROM   lineorder_yyyymm  l1,
          lineorder2_yyyymm l2
   WHERE  l1.lo_orderdate = l2.lo_orderdate
   AND    l1.lo_orderdate between 19920101 AND 19920331;
                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
Finalize Aggregate  (cost=27222261.36..27222261.37 rows=1 width=8)
   ->  Gather  (cost=9140766.02..27222261.34 rows=6 width=8)
         Workers Planned: 2
         ->  Parallel Append  (cost=9139766.02..27221260.74 rows=3 width=8)
               ->  Partial Aggregate  (cost=9396752.33..9396752.34 rows=1 width=8)
                     ->  Parallel Hash Join  (cost=7879.52..6156318.15 rows=1296173675 ...
                         ～ 中略 ～
               ->  Partial Aggregate  (cost=9139766.02..9139766.03 rows=1 width=8)
                     ->  Parallel Hash Join  (cost=9084.77..5983755.36 rows=1262404264 ...
                         ～ 中略 ～
               ->  Partial Aggregate  (cost=8684742.34..8684742.35 rows=1 width=8)
                     ->  Parallel Hash Join  (cost=7340.90..5678423.17 rows=1202527670 ...
```

6.13. DELETEとTRUNCATE

DELETEおよびTRUNCATEについてパーティション表の特性を確認します。
時系列データの保存期間切れデータはTRUNCATEできることがパーティション表の大きなメリットです。

TRUNCATEはVACUUM負荷も気にする必要がないため、管理性に大きく寄与します。
以下の構成のテーブルを使用します。

表 6.20 使用するテーブル

構成	用途
非パーティションテーブル	DELETE
宣言的パーティション（年月）	対象パーティションのDELETEとTRUNCATE

PostgreSQL11のみを対象としています。

6.13.1. 検証方法（DELETEとTRUNCATE）

- 以下の3ケースを実施

表 6.21 DELETEとTRUNCATE

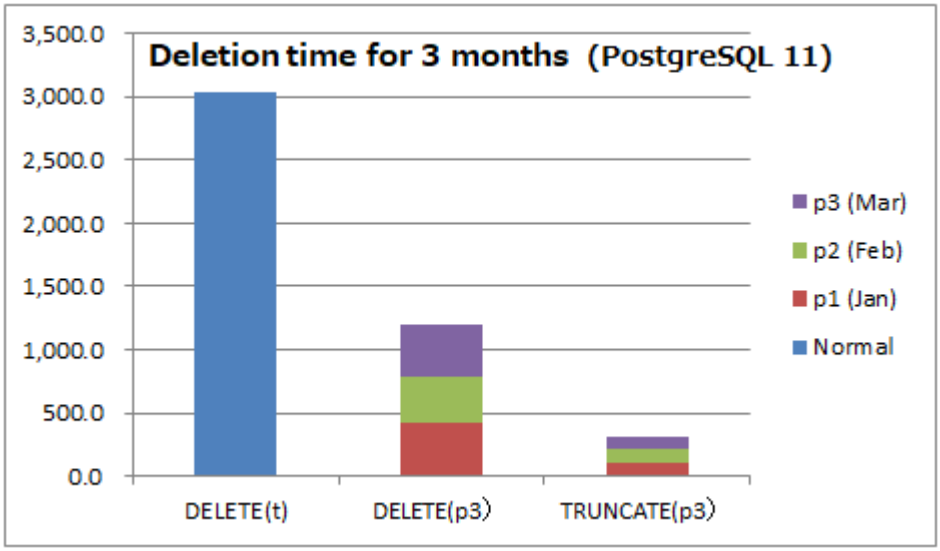
ケース	条件指定	グラフの凡例表記
ケース1	非パーティションテーブルに対して3ヶ月分のDELETE	DELETE(t)
ケース2	年月パーティションテーブルに対して3ヶ月分のDELETE	DELETE(p3)
ケース3	年月パーティションテーブルに対して3ヶ月分のTRUNCATE	TRUNCATE(p3)

```
●ケース1 （非パーティションテーブルに対して3ヶ月分のDELETE）
=# DELETE FROM lineorder WHERE lo_orderdate BETWEEN 19920101 AND 19920331

●ケース2 （年月パーティションテーブルに対して3ヶ月分のDELETE）
=# DELETE FROM lineorder_yyyymm_199201;
=# DELETE FROM lineorder_yyyymm_199202;
=# DELETE FROM lineorder_yyyymm_199203;

●ケース3 （年月パーティション表に対して3ヶ月分のTRUNCATE）
=# TRUNCATE lineorder_yyyymm_199201;
=# TRUNCATE lineorder_yyyymm_199202;
=# TRUNCATE lineorder_yyyymm_199203;
```

6.13.2. 結果（DELETEとTRUNCATE）



3ヶ月データにおけるDELETEおよびTRUNCATEの所要時間

- 縦軸は時間（ミリ秒）であり、小さい方が優れています。
- DELETE(p3)およびTRUNCATE(p3)は3パーティションに対してシリアルに実行した時間の合計です。

6.13.3. 考察（DELETEとTRUNCATE）

- TRUNCATEは当然ながら圧倒的に高速です。

- DELETEはこの後にVACCUMが発生することを考慮するとより大きな差と言えます。
- TRUNCATEができない場合でも、パーティションテーブルでは子テーブルを指定してのDELETEで改善が見込めます。
- 本検証では実施していませんが、INSERTとDELETEが同時に実行されるケースでも、それぞれ別パーティションとなることで改善が見込めます。

6.14. パーティショニング性能検証のまとめ

本性能検証では、主に以下の内容が確認できました。

6.14.1. 挿入処理

宣言的パーティションは継承によるパーティションの課題であった挿入処理(COPY FROM/INSERT)で大きな改善が確認できました。
特にCOPY FROM では宣言的パーティションの優位が顕著です。

6.14.2. 索引走査とプルーニング

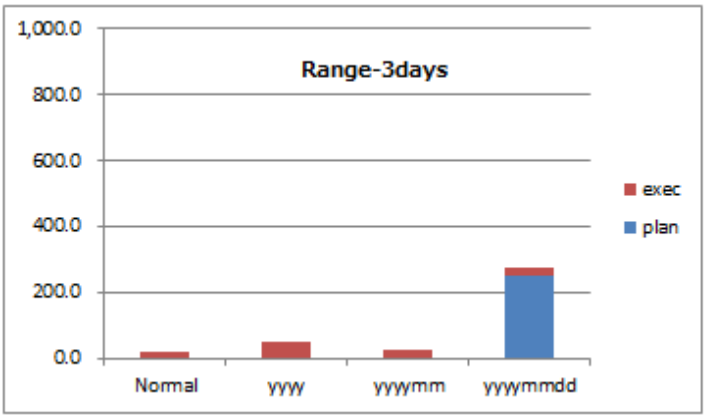
非パーティション表でも検索範囲が狭い範囲では、索引走査によりパーティション表と遜色ない結果となっています。
ただし検索範囲が一定割合を超える場合は全表走査に切り替わるため、パーティション表が優位となります。
パーティション表のプルーニングは検索範囲に依らず、安定的に機能します。

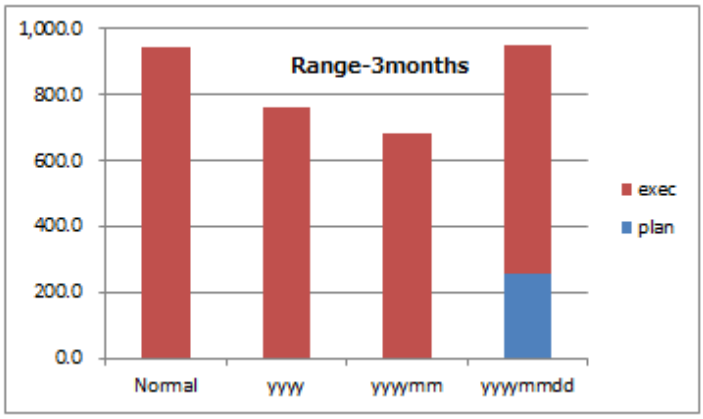
6.14.3. 適切なパーティション数

本検証におけるパーティション粒度（数）は年月(84)が最も効率的でした。
ただしパーティション数が数百のケースを検証できていないため、参考程度に留めてください。

6.14.4. パーティション数と実行計画

パーティション数が多い場合、パーティション選定の処理時間が増えますが、実行計画を立てる時間も増えます。
参考までに、全体の所要時間における実行計画を立てる時間の割合の例を示します。





所要時間における実行計画を立てる時間の割合

- 左が検索範囲3日、右が検索範囲3月です。
- 縦軸の単位はミリ秒です。
- 青色が実行計画を立てる時間、茶色が実行時間です。
- 3日（左図）の年月日構成では、所要時間の殆どが実行計画を立てる時間です(**91.2%**)。
- 3月（右図）の年月日構成では、実行計画を立てる時間は3日と同じですが、実行時間が長くなるため割合は減少します(**27.0%**)。

6.14.5. PostgreSQL 11新機能

パラレルワーカによるパーティション走査の高速化

- PostgreSQL 11ではパラレルワーカによるパーティション走査が改善されて高速化しています。
- PostgreSQL 11では並列度を増加することでより高速化する余地があります。

パーティション選定の高速化

- 挿入処理(INSERT)では2,400万件のINSERT文を実施したため、パーティション選定の高速化の成果が顕著に表れています。

パーティション・ワイズアグリゲート

- PostgreSQL 11の新機能であるパーティションワイズアグリゲートは、集計処理を高速化します。
- 集計処理は日常的に実施されるため大いに活用が見込めます。
- パラメータenable_partitionwise_aggregate=onを設定することで機能します(デフォルトはoff)。

パーティション・ワイズジョイン

- PostgreSQL 11の新機能であるパーティションワイズジョインは、同一レベルのパーティション表の結合処理を高速化します。
- 同一レベルのパーティション表の結合であるため、活用はある程度限られるかもしれません。
- パラメータenable_partitionwise_join=onを設定することで機能します(デフォルトはoff)。

6.14.6. 削除処理

- 削除処理(DELETE/TRUNCATE)ではパーティション表の優位が確認できました。
- パーティションの構成次第で、不要データをTRUNCATEで処理できます。

7. パーティションテーブルのTips

7.1. パーティションが変更になるUPDATE

PostgreSQL 11では、パーティションが変更になるUPDATEが可能になりました。
PostgreSQL 10までは、DELETEおよびINSERTが必要でした。
(実行例) 年月日パーティションの日付を変更

```
●PostgreSQL 10 ではエラーが発生
=# UPDATE lineorder_yyyymmdd
   SET    lo_orderdate = '19980102'
   WHERE  lo_orderdate = '19920101';
ERROR:  new row for relation "lineorder_yyyymmdd_19920101" violates partition constraint

●PostgreSQL 11 では正常終了
=# UPDATE lineorder_yyyymmdd
   SET    lo_orderdate = '19980102'
   WHERE  lo_orderdate = '19920101';
UPDATE 10228
```

7.2. プルーニング適用条件

パーティションプルーニングを行うには、WHERE句の条件でパーティションキーが指定されている必要があります。
ただし以下のようにWHERE句の指定の仕方によってパーティションプルーニングが効かない場合があるため、注意が必要です。
インデックスが使用される指定方法と類似していますが、前方一致が効かないなど相違点もあります。
以下はレンジおよびリストの場合であり、ハッシュは完全一致のみプルーニングされます。

表 7.1 パーティションプルーニング可能な条件指定

指定方法	指定例
完全一致	partition-key = 20190101
範囲指定	partition-key **BETWEEN** 20190101 **AND** 20190201 partition-key **IN** (20190101, 20190102) partition-key > 20190101 AND partition-key < 20190131

表 7.2 パーティションプルーニング不可能な条件指定

指定方法	指定例	備考
前方一致 中間一致 後方一致	partition-key LIKE 'hoge%' partition-key LIKE '%hoge%' partition-key LIKE '%hoge'	LIKE演算子ではプルーニングできない
関数指定	TO_CHAR(partition-key),'YYYYMMDD') = '20190101'	パーティションキー側に関数を指定している場合

7.3. パーティションテーブルおよびパーティションの一覧表示

パーティションテーブルの一覧を表示できるシステムカタログには以下があります。

- pg_classテーブル
- pg_partitioned_tableテーブル
- pg_inheritsテーブル

表 7.3 パーティション表の表示

システムカタログ	内容
pg_class	パーティションテーブルはrelkind = 'p'（宣言的パーティションテーブル） 子テーブルはrelkind = 'r' パーティションテーブルの索引は relkind='I'(大文字） 子テーブルの索引は relkind='i'(普通）
pg_partitioned_table	宣言的パーティションニングのパーティションテーブルが対象 PARTSTRAT列でRANGE(r)/LIST(l)/HASH(h)が判別できる。
pg_inherits	親子（継承）関係を示す。 パーティションテーブルとパーティションの一覧を表示できる。 継承を使用したパーティションニングおよび宣言的パーティションニングの両方に対応。

7.3.1. パーティションテーブルの一覧表示

（実行例）

```
-- パーティションテーブルの表示
=# SELECT relname FROM pg_class WHERE relkind='p';
        relname
-------------------------
 lineorder_yyyy
 lineorder_yyyymm
 lineorder_yyyymmdd
 ～以下略～

-- パーティションテーブルおよび種類の表示
=# SELECT partrelid::regclass,partstrat FROM  pg_partitioned_table;
        partrelid        | partstrat
-------------------------+-----------
 lineorder_yyyy          | r              -- レンジ
 lineorder_yyyymm        | r
 lineorder_yyyymmdd      | r
 ～以下略～
```

7.3.2. パーティションの一覧表示

（実行例）

```
-- パーティションテーブルおよびパーティションの表示
=# SELECT inhparent::regclass::name AS partition_table,
          inhrelid::regclass::name  AS partition
   FROM   pg_inherits;
    partition_table         |            partition
----------------------------+-----------------------------------
 lineorder_yyyy             | lineorder_yyyy_1992
 lineorder_yyyy             | lineorder_yyyy_1993
 lineorder_yyyy             | lineorder_yyyy_1994
 ～以下略～
```

7.3.3. パーティション範囲の表示

パーティション毎の範囲やリストを表示するには **pg_get_expr関数** を使用します。
（実行例）

```
=# SELECT i.inhparent::regclass::name AS partition_table,
          i.inhrelid::regclass::name  AS partition,
          pg_get_expr(c.relpartbound, c.oid) AS bound
   FROM   pg_class    c,
          pg_inherits i
   WHERE  c.oid = i.inhrelid
   AND    pg_get_expr(c.relpartbound, c.oid) IS NOT NULL;

 partition_table |        partition    |                bound
-----------------+---------------------+---------------------------------------------
～レンジの例～
lineorder_yyyy   | lineorder_yyyy_1992 | FOR VALUES FROM ('19920101') TO ('19930101')
lineorder_yyyy   | lineorder_yyyy_1993 | FOR VALUES FROM ('19930101') TO ('19940101')
lineorder_yyyy   | lineorder_yyyy_1994 | FOR VALUES FROM ('19940101') TO ('19950101')

～リストの例～
lineorder_yyyy_1992_sub | lineorder_yyyy_1992_sub1 | FOR VALUES IN ('1-URGENT       ')
lineorder_yyyy_1992_sub | lineorder_yyyy_1992_sub2 | FOR VALUES IN ('2-HIGH         ')
lineorder_yyyy_1992_sub | lineorder_yyyy_1992_sub3 | FOR VALUES IN ('3-MEDIUM       ')
```

7.4. パーティションテーブルの統計情報

以下の統計情報のシステムカタログではパーティション単位の情報が取得できます。
またパーティションテーブル全体の情報を取得するにはパーティションテーブル毎に集計します。

- pg_stat_<all/user/sys>_tables
- pg_stat_<all/user/sys>_indexes
- pg_statio_<all/user/sys>_tables
- pg_statio_<all/user/sys>_indexes
- pg_stat_xact_<all/user/sys>_tables
- pg_indexes

パーティションテーブルlineorder_yyyymmddの統計情報の取得例を示します。
n_live_tup列およびn_dead_tup列のパーティションテーブル全体の値を取得します。
ANALYZEはパーティションにカスケードして実施されます。

```
=# ANALYZE lineorder_yyyymmdd;
ANALYZE

=# SELECT sum(n_live_tup) AS n_live_tup,
          sum(n_dead_tup) AS n_dead_tup
   FROM   pg_stat_all_tables
   WHERE  relname LIKE 'lineorder_yyyymmdd_%';
   n_live_tup | n_dead_tup
  ------------+------------
     23996670 |          0
```

パーティション名がLIKEが指定できる場合は上記で構いませんが、以下のようにpg_inheritsを組み合わせるのが確実です。
パーティションテーブル名を完全一致で指定できるためです。

```
=# SELECT sum(n_live_tup) AS n_live_tup,
          sum(n_dead_tup) AS n_dead_tup
   FROM   pg_stat_all_tables
   WHERE  relname IN (SELECT inhrelid::regclass::name
                      FROM   pg_inherits
                      WHERE  inhparent::regclass::name = 'lineorder_yyyymmdd');
   n_live_tup | n_dead_tup
  ------------+------------
     23996670 |          0
```

7.5. パーティションテーブルを指定できないコマンド

PostgreSQL 11におけるコマンドまたは関数には、パーティションテーブル（親）を指定できないものがあります。
基本的にはメンテナンス系のコマンドまたは関数であるため、大きな問題はないものと考えられます。
以下に代表的なコマンドまたは関数を示します。

- COPY TO (ファイルに出力)
  - 下記のように直接パーティションテーブルを指定するとエラーとなります。
  - HINTにあるように、"COPY (SELECT * FROM <パーティションテーブル>) TO .." と指定することでご対応ください。
  - なおCOPY FROM はパーティションテーブルを直接指摘可能です。本検証でも実施しています。

```
--パーティションテーブル名がemployee_rangeのパーティション表に対してCOPY TO文を実行
postgres=# COPY employee_range TO '/tmp/employee.dat' WITH CSV;
ERROR:  cannot copy from partitioned table "employee_range"
HINT:  Try the COPY (SELECT ...) TO variant.
```

- pg_prewarmエクステンション
  - 本検証では以下の様にPL/pgSQLにてlineorder_yyyymmdd_<YYYYMMDD>パーティションのpg_prewarmを7年分実行しています。
  - 頻繁なアクセスが予想される直近のパーティションのみを対象にしても一定の効果が期待できます。

```
DO $$
DECLARE
    part      text := 'lineorder_yyyymmdd_';
    s_day     timestamp(0) := '19920101';
    e_day     timestamp(0) := '19981231';
    i_day     timestamp(0) := s_day;
BEGIN
    WHILE  i_day <= e_day LOOP
        PERFORM pg_prewarm(part || TO_CHAR(i_day,'YYYYMMDD'));
        i_day := i_day + interval '1 day';
    END LOOP;
END;
$$ LANGUAGE PLPGSQL;
```

- REINDEX
  - REINDEXはインデックスを再構築します。INDEX/TABLE/SCHEMA/DATABASE/SYSTEMの各単位で指定できます。
  - INDEX単位でパーティションテーブルのインデックスを指定した場合は、エラーが発生します。
  - TABLE単位でパーティションテーブルを指定した場合は、警告が発生して処理をスキップします。
- CLUSTER
  - CLUSTERはインデックスに従ってテーブルをクラスタ化します。
  - 親テーブルを指定するとエラーが発生します。
- pageinspectエクステンション
  - pageinspectは低レベルなデータベースページの内容を調べることができる関数を提供します。
  - 親テーブルを指定するとエラーが発生します。

8. まとめ

8.1. 宣言的パーティショニングの実装

PostgreSQL 10で実装された宣言的パーティショニングでは、従来の継承を使用したパーティショニングに比較し、
管理性および性能が大幅に向上し、導入のしきいが下がっています。

8.2. 宣言的パーティショニングの進化

PostgreSQL 11ではさらに進化しています。
性能検証では、以下の性能に関わるPostgreSQL 11の新機能について検証を実施し、確認ができました。

- パラレルワーカによるパーティション走査の高速化
- パーティション選定の高速化
- パーティション・ワイズアグリゲート
- パーティション・ワイズジョイン

宣言的パーティショニングは今後も進化を続けます。

9. 著者

（企業・団体名順）

版	所属企業・団体名	部署名	氏名
第1.0版 (2018年度WG3)	株式会社アシスト	データベース技術本部 技術統括部	柘植 丈彦
	株式会社アシスト	サービス事業部 サポートセンター	家島 拓也
	株式会社アシスト北海道	技術部	沼田 梨佐
	NTTテクノクロス株式会社	IoTイノベーション事業部	原田 登志