
PGECons
PostgreSQL Enterprise Consortium
パーティショニングした
テーブルの移行と運用
第1.0版
PostgreSQL Enterprise Consortium
WG2(移行WG)
Copyright © PostgreSQL Enterprise Consortium. All Rights Reserved.

はじめに

■ 本書の目的

移行元のOracleでパーティション表を活用しているデータベースをPostgreSQLに移行するとき、または新規にテーブルのパーティショニングを活用する際に有用な以下3点を理解できることを目的としております。

- 効果的なパーティショニング機能の活用方法を理解し設計できること。
- OracleからのPostgreSQLのパーティショニング機能への移行が適切にできること。
- 運用上、必要な保守作業を理解し実装できること。

なお、本書は宣言的パーティショニングを前提とします。

※本書執筆時点のPostgreSQLの最新バージョンは15になります。

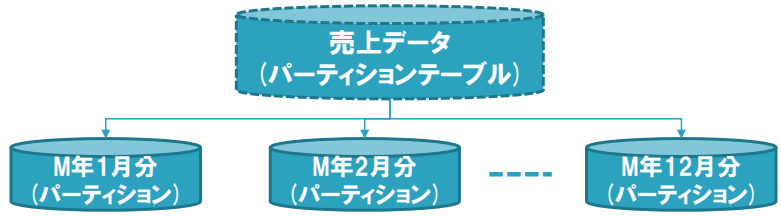
目次

1. 概要

1. 概要

1. 1 パーティショニングについて

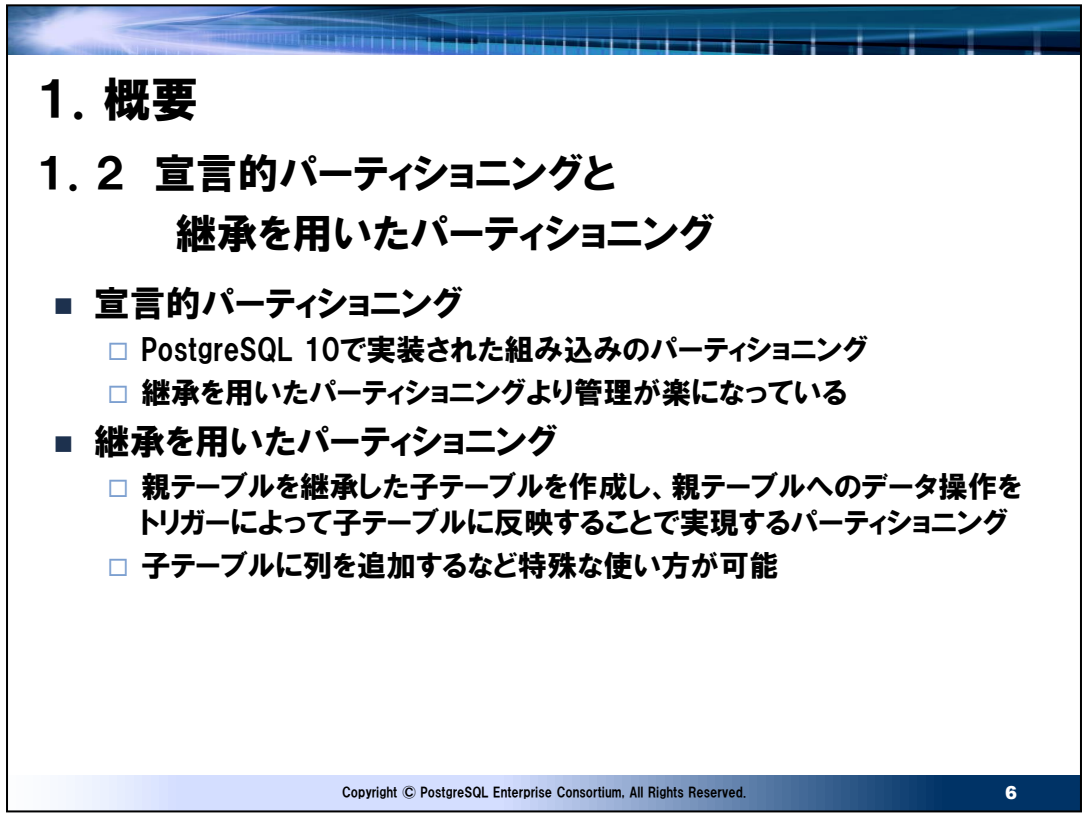
- テーブルに格納されるデータを、複数の領域に分けて格納することができる機能
  - テーブルに格納するデータ量の増大は性能劣化やデータベース運用を困難にする要因となる。
  - それを回避する機能として、データの格納先を複数の領域に分割するパーティショニング機能が備えられている。
  - PostgreSQLでは、宣言的パーティショニングと継承を用いたパーティショニングがサポートされている。



● パーティショニングについて

パーティショニングとは、テーブルに格納されるデータを複数の領域(パーティション)に分けて格納することができる機能です。例えば、日付を主キーとして持つデータを月ごとに分けてデータを格納することができます。

- テーブルに格納されるデータ量が増大した場合、データの検索には大量のデータを読み取る必要があり、性能が劣化することがあります。また、運用時に実行するバキュームやバックアップなどのメンテナンスにも時間がかかるようになり、運用に支障が出ることがあります。それらを回避するためにデータを複数のパーティションと呼ばれる領域に分割し、部分的に処理できるようにする機能がパーティショニングです。
- PostgreSQLでは、宣言的パーティショニングと継承を用いたパーティショニングの2つがサポートされています。



- 宣言的パーティショニングと継承を用いたパーティショニング
- 宣言的パーティショニングとは、対象となるテーブルをパーティションに分割することを宣言してテーブルを作成する方法です。

- 継承を用いたパーティショニングとは、対象となるテーブルを継承した子テーブル(パーティション)を作成し、対象テーブルにデータが挿入された際にトリガーによって子テーブルにデータを振り分けることでパーティショニングを実現する方法です。

- 継承を用いたパーティショニングはトリガーの設定など管理が複雑になりますが、PostgreSQL 10から組み込まれた宣言的パーティショニングはそのような手間が無く運用が楽になっています。継承を用いたパーティショニングでは、子テーブルに列を追加するなど宣言的パーティショニングではできない使い方を行うことも可能です。

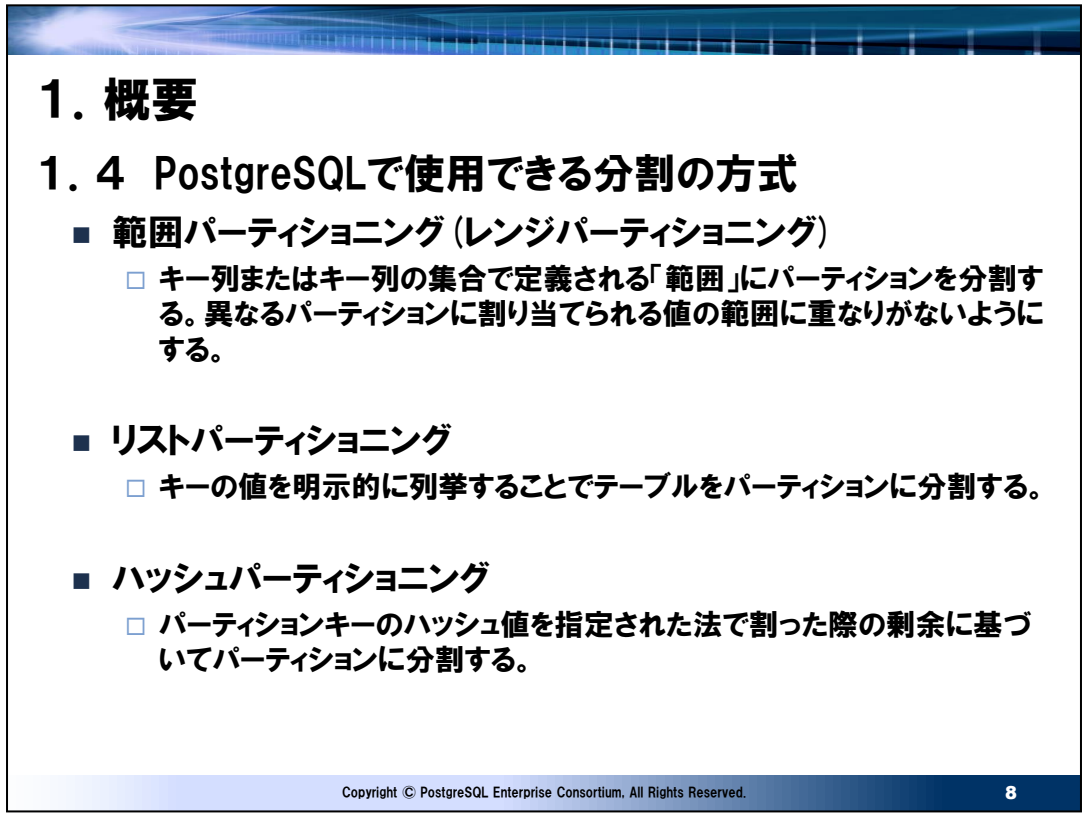
1. 概要

1. 3 PostgreSQLバージョンごとの主な変更

バージョン	機能
PostgreSQL 10	宣言的パーティショニングの追加 CREATE TABLE文のみでパーティションテーブルおよびそのパーティションを作成可能 以前は継承、トリガーなど定義する必要があった。
PostgreSQL 11	・ハッシュパーティションが可能 ・キーにマッチしないデフォルトパーティションが指定可能 ・パーティションキーを跨ぐ更新が可能（他のパーティションに移動） ・索引の自動生成（親テーブルで作成すれば子テーブルに自動で作成）
PostgreSQL 12	・パーティションテーブルを参照する外部キーに対応 ・パーティション毎に表領域の指定が可能
PostgreSQL 13	・ロジカルレプリケーションに対応 ・行単位Beforeトリガーに対応
PostgreSQL 14	・強いロックを伴わずにパーティションの切り離しが可能
PostgreSQL 15	・クエリの計画時間の改善など性能が向上

- PostgreSQLバージョンごとの主な変更

PostgreSQL 10にて、宣言的パーティショニングが実装されました。その後、徐々に機能が追加されています。また、性能の向上も図られています。



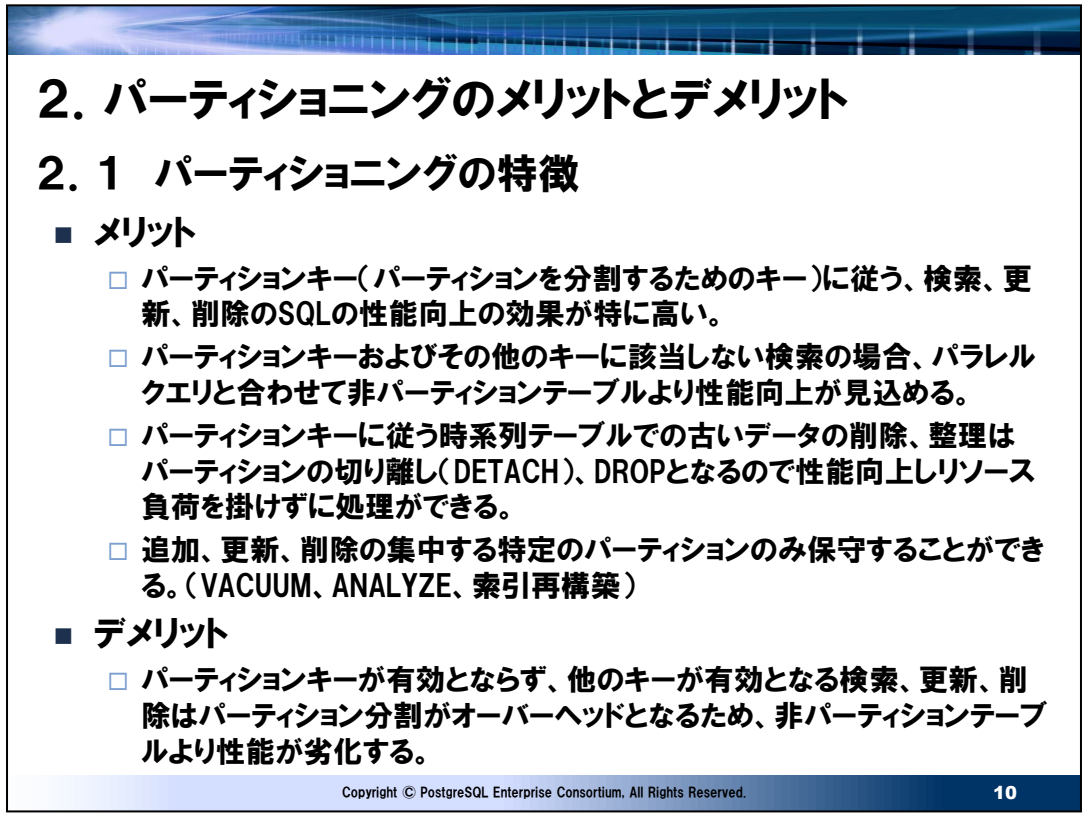
- PostgreSQLで使用できる分割の方式
宣言的パーティショニングを用いる際に、データをパーティションに分割する方式として以下のようなものがあります。

- 範囲パーティショニング(レンジパーティショニング)
テーブルを、キー列またはキー列の集合で定義される「範囲」に沿ってパーティションに分割します。異なるパーティションに割り当てられる値の範囲に重なりがないようにします。
例えば、日付の範囲によってパーティションに分割したり、特定のIDの範囲によって分割したりします。

- リストパーティショニング
各パーティションに現れるキーの値を明示的に列挙することでテーブルをパーティションに分割します。

- ハッシュパーティショニング
各パーティションに対して法と剰余を指定することでテーブルをパーティションに分割します。各パーティションは、パーティションキーのハッシュ値を指定された法で割った際の剰余が特定の値となる行を保持します。

2. パーティショニングの メリットとデメリット

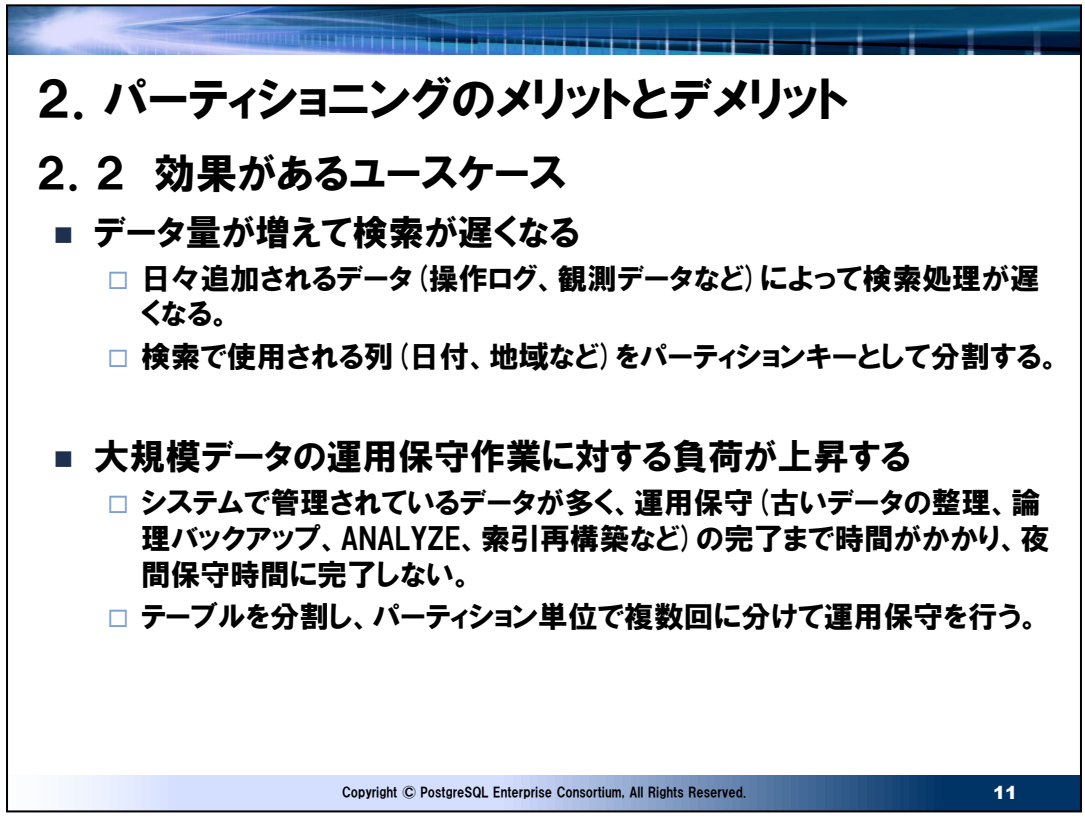


- パーティショニングの特徴

メリットとしては、データを分割して管理することで部分的な処理が可能となり、処理速度が向上することが挙げられます。

- パーティションキーに従って分割されたデータのみを処理する場合は、性能向上の効果が特に高くなります。
- 分割されたパーティションに対して並列処理をすることでパラレルクエリと合わせた性能向上が見込めることがあります。
- パーティション単位でのデータ削除や、パーティションごとの保守が可能となります。例えば、夜間保守など1回の作業時間が限られている場合に、1回の保守では特定のパーティションのみを対象とし、複数回に分けることで保守を時間内に完了させます。

デメリットとしては、パーティショニングの処理にオーバーヘッドがかかることです。そのため、パーティションキーが有効とならないデータアクセスなどでは、非パーティションテーブルより性能が劣化することがあります。

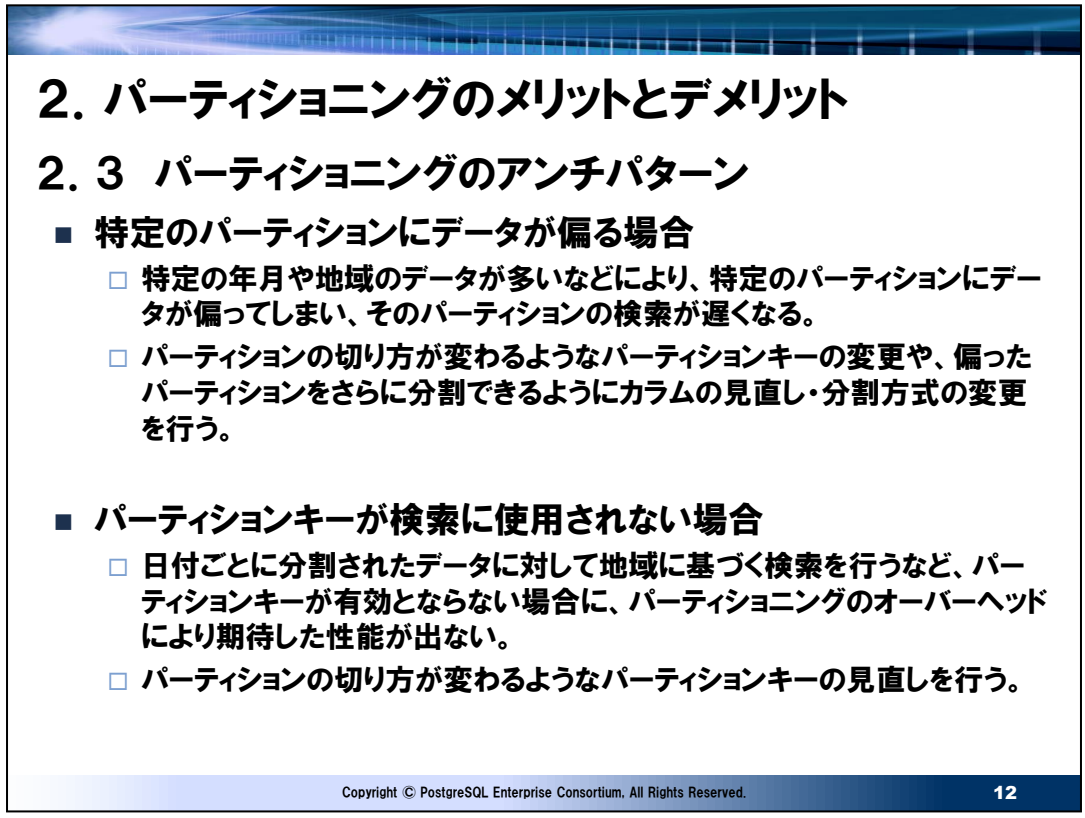


- 効果があるユースケース

パーティショニングを用いることで効果が見込めるユースケースとしては以下のようなものが考えられます。

- データ量が増えて検索が遅くなる
  ある処理の操作ログや気象の観測データなど日々追加されるデータがある場合、データの増加に伴って検索などの処理が遅くなることがあります。
  このような場合に、日付や地域などの検索で使用される列をパーティションキーとした分割を行うことで処理が遅くなることを緩和することが期待できます。

- 大規模データの運用保守作業に対する負荷が上昇する
  システムで管理されているデータが多く、バキュームやバックアップなどの運用保守に時間がかかり、決められた夜間保守時間内に完了しない問題が発生することがあります。
  そのような場合に、テーブルをパーティションに分割することでパーティション単位での運用保守が可能となります。パーティション単位で運用保守を複数回に分けて行うことで、一回の作業時間を決められた保守時間に収めるようにします。



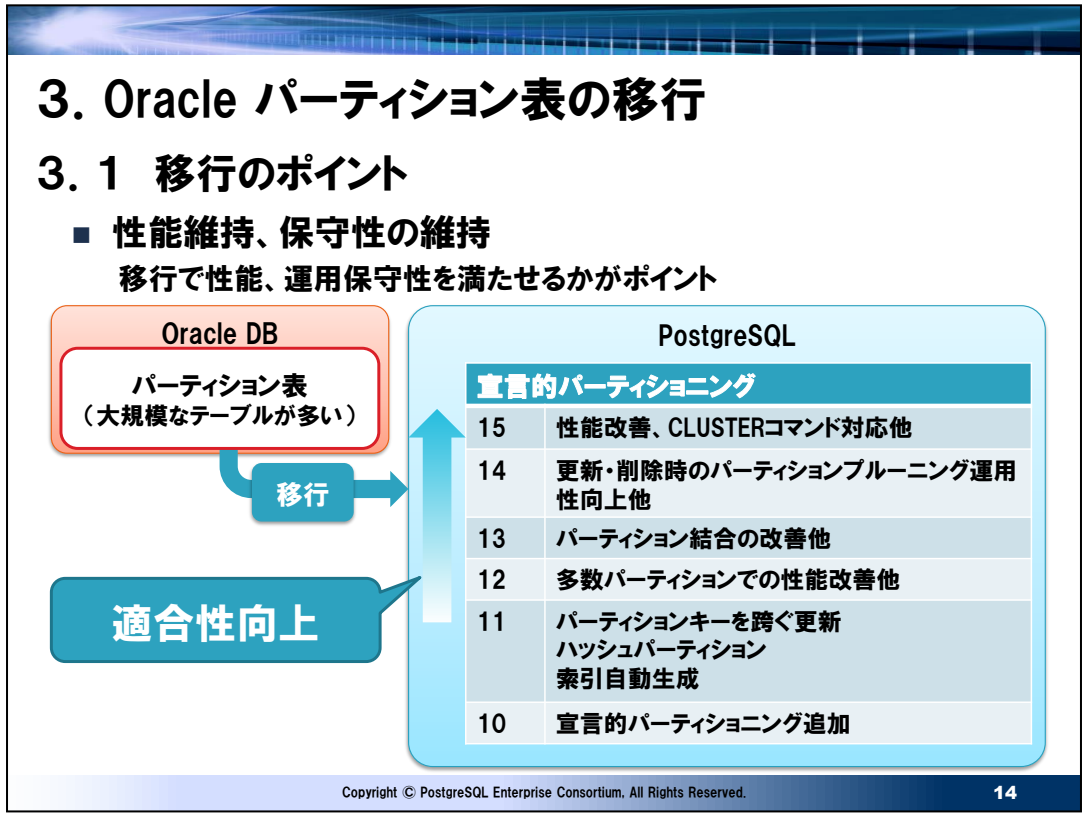
- パーティショニングのアンチパターン

パーティショニングを使用する際のアンチパターンとして以下のケースが考えられます。

- 特定のパーティションにデータが偏る場合
  パーティションキーに指定された年月や地域ごとのデータに偏りがあり、特定のパーティションにデータが偏って格納されてしまい、そのパーティションの検索が遅くなることがあります。
  このような場合は、パーティションに格納されるデータに偏りがなくなるようにパーティションキーの変更を検討します。または偏ったパーティションをさらに分割できるように、パーティションキーに用いられているカラムの定義の見直し(地域をさらに分割するなど)や、範囲パーティショニングやリストパーティショニングなどの分割方法の変更を検討します。

- パーティションキーが検索に使用されない場合
  デメリットで述べましたように、検索キーとパーティションキーが異なるときは、パーティショニング処理のオーバーヘッドにより期待した性能が出ない場合があります。
  そのような場合は、検索キーとパーティションキーが一致するように見直しを行います。

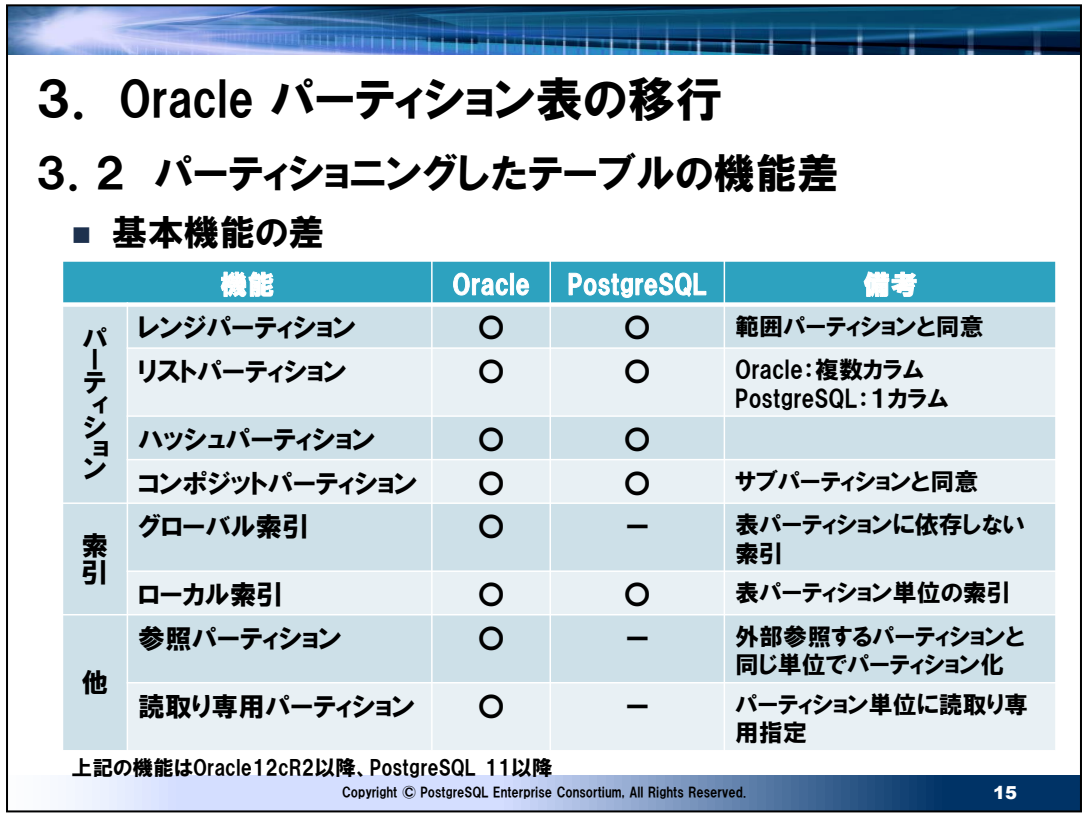
3. Oracle パーティション表の移行



- Oracle Enterprise Edition（以下Oracle EEとする）のパーティション表の移行

PostgreSQLのメジャーバージョンアップにより、機能、性能、運用面が向上することで、より大規模なDBを運用するケースも増加しています。この中ではOracle EEで大規模なDBも移行の対象に含まれるようになりました。

Oracle EEのDBでかつ大規模なDBであれば、パーティショニングを利用して大規模テーブルのアクセス性能を確保しているのが一般的です。DB移行では、この性能を確保できるか評価する必要があります。PostgreSQLでもバージョン10から宣言的パーティショニングがサポートされ、さらにバージョンを重ねる毎に性能向上、機能追加がなされることで商用運用でも十分利用可能なレベルに達しています。それでもOracleのパーティショニング機能を100%満たすものではありません。機能面と運用保守面からその違いを理解し、移行上どのようなリスクがあるか、リスク・課題がある場合はどのような対策を実施すべきか理解することが必要です。次節以降、３． ２で基本機能差、３． ３で保守の差について説明します。



- 基本機能差について

パーティショニングの基本機能としてレンジパーティション（PostgreSQLのマニュアルでは範囲パーティションと呼ばれている）、リストパーティション、ハッシュパーティションはPostgreSQLでもサポートしております。索引については性能上、重要な差異があります。ここではOracleからの移行のためOracleでの呼び方でローカル索引、グローバル索引と呼びます。

- ローカル索引

  表パーティション毎の索引。PostgreSQLのパーティションの索引もこの索引になります。

- グローバル索引

  表パーティションに依存しない索引。GLOBAL句を付けて索引を作成します。索引はパーティションテーブル全体を対象に作成されます。索引そのものはテーブルのパーティションとは無関係な切り口でパーティション化することもできます。メリットはパーティションキー以外でパーティション・プルーニングを効かせることができることです。

  例えば、販売日をパーティションキーとする売上テーブルがあるとき、伝票番号をレンジで範囲を指定しパーティション化した索引を作ることもできます。伝票番号がシリアルに増加する番号である場合、特定範囲の伝票番号は販売日の特定年月の範囲に存在するとします。この場合、伝票番号検索でも特定の索引パーティションを読み、テーブルもパーティション・プルーニングが効いて特定の表パーティションのみを検索対象に絞り込めるので検索を高速化できます。デメリットとして表パーティションのメンテナンスを行うとグローバル索引全体も保守が必要になり保守コストを増加させます。

  移行時の注意点として、この索引を利用して移行元DBで性能を確保している場合、PostgreSQLへの移行で性能を維持することは困難になります。

3. Oracle パーティション表の移行

3. 3 パーティショニングしたテーブルの運用保守の差

■ 保守の差

機能	Oracle	PostgreSQL	備考
パーティション追加	○	○	新規パーティションの追加
パーティション削除	○	○	パーティション単位の削除
パーティション切捨て	○	○	パーティション単位のTRUNCATE
パーティション交換	○	○	パーティション表から、非パーティション表への変更またはその逆
インターバルパーティション	○	△ pg_partman	時間隔のレンジパーティションの自動追加
自動リストパーティション	○	ー	リストパーティションの自動追加
パーティションマージ	○	ー	複数のパーティションの結合。レンジパーティションは隣接するパーティションが対象
パーティション分割	○	ー	1つのパーティションを複数に分割

● 運用保守の差について

パーティショニングしたテーブルの保守は、パーティションの追加、削除、またはパーティションを使い回すような用途であれば切り捨てが運用の中で実施されます。基本となる保守機能はPostgreSQLでもサポートされます。以下ではPostgreSQLでサポートされない機能について説明します。移行元のOracle DBでこれらを運用で実施している場合、対応のための作業コストが増加します。

- パーティションの自動追加

  時間隔のレンジパーティションでは新しいレンジの日時データが発生したとき自動でパーティションを作成したいケースがあります。PostgreSQLの場合は基本機能ではサポートされません。利用者が新しい日時向けのパーティションを手動もしくは運用バッチなどで事前に作成しておく必要があります。追加ソフトウェアのpg_partmanを利用することでバッチ処理として作成の自動化を図ることができます。ただし、Oracleのように対象データが発生した時点での自動パーティション追加はできません。

  また、リストパーティションでもOracle12cR2以降では、追加されるパーティションキーが存在しない場合、自動でパーティションを追加することが可能です。PostgreSQLの場合、自動でリストパーティションを作成する機能はありません。レンジ、リストを問わずパーティションに適合しないデータの挿入、または更新でエラーを発生させたくない場合はデフォルトパーティションを利用することになります。注意点は新たなパーティションを作成するとき、パーティションキーのデータがデフォルトパーティションに存在するとパーティションを作成できません。事前にデフォルトパーティションからデータを移動し、パーティションを作成後に移動したデータを戻す作業が必要になります。

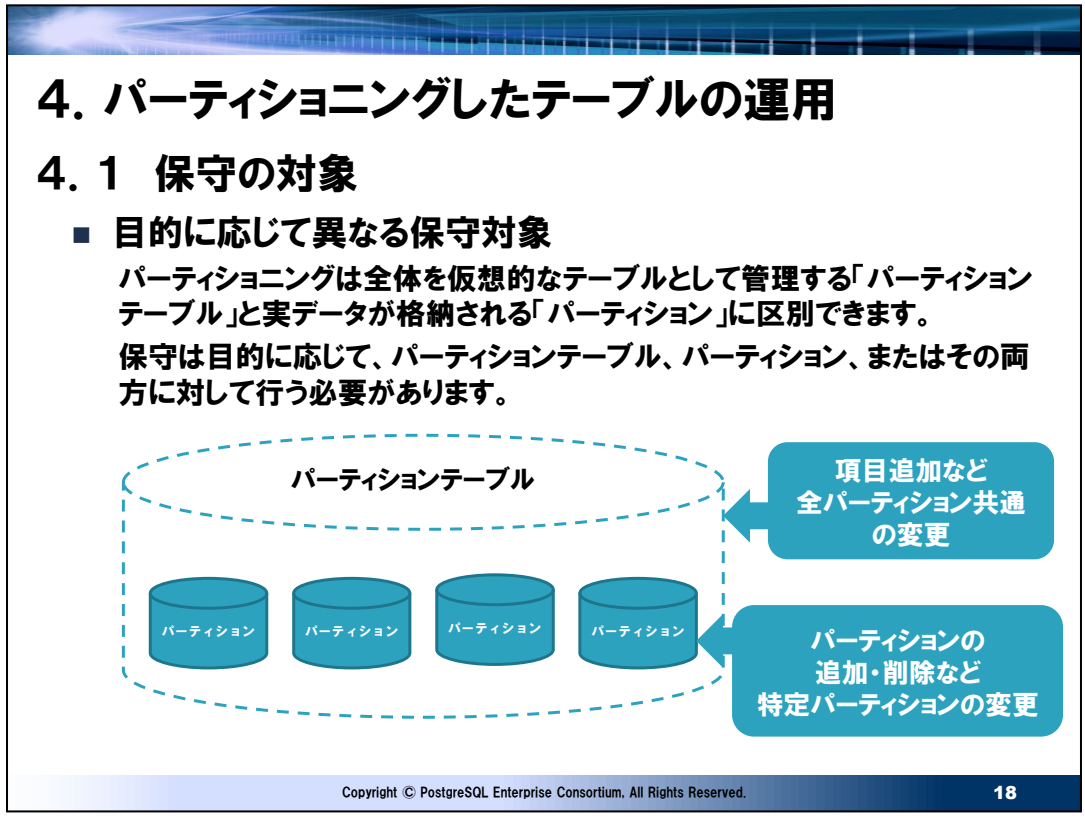
- パーティションのマージ、結合

  パーティショニングされたテーブルを長期間利用することでパーティション毎のデータ量のバラつきが発生することがあります。データ量が少なくなったパーティションのマージまたは結合をOracleは１つのコマンドで実施できますが、PostgreSQLでは手動での作業（パーティションを切り離し、一つのパーティションにデータを寄せて、再度パーティション範囲、またはリストの定義を行い組み込むなど）が必要になります。

- パーティションの分割

  マージ・結合同様に運用中に特定パーティションが大きくなり過ぎた場合、Oracleは１つのコマンドでパーティション分割が可能です。PostgreSQLでは手動での分割が必要になります。

４．パーティショニングしたテーブルの運用



- マニュアル

コマンドなどの詳細は以下マニュアルを参照してください。

https://www.postgresql.jp/document/XX/html/ddl-partitioning.html （XXはPostgreSQLのメジャーバージョン）

- 保守の対象

PostgreSQLのパーティショニングしたテーブルは、全体を仮想的なテーブルとして管理する「パーティションテーブル」と実データが格納される「パーティション」に区別できます。保守はその目的に応じてパーティションテーブルに対して行うもの、パーティションに対して行うもの、またその両方に対して行うものがあります。

以下は保守の例になります。

- パーティションテーブルとパーティションの両方に対して必要な保守：パーティショニングしたテーブルの作成
- パーティションテーブルに対して必要な保守：項目の追加、パーティショニングしたテーブル全体の削除（パーティションも同時に削除されます）、索引作成
- パーティションに対して必要な保守：パーティションの追加、削除、組み込み（ATTACH）、切り離し（DETACH）、FILLFACTORのような個別の物理テーブルのオプション変更
- パーティションテーブルとパーティションのどちらでも実施できる保守：TRUNCATE、VACUUM、ANALYZE、索引再構築。いずれもパーティションテーブルに対して実施すると管理されるすべてのパーティションを対象に処理され、パーティションに対して実行すれば対象のパーティションのみ処理が実施されます。

4.2章ではスライド部分に保守の一覧を表として整理し、ノート部分に作業実施例を含め運用上のポイントを説明します。同じ目的の保守であっても手段が複数あるものもあり、運用に最適な手段を選択する必要があります。大規模なテーブルであってもパーティショニングを活用することで、メンテナンス対象のパーティションを絞り込み、保守時間を短縮することが可能になります。

4. パーティショニングしたテーブルの運用

4. 2 保守一覧(1/3)

保守内容	保守対象		説明
	パーティションテーブル	パーティション	
パーティショニングしたテーブルの作成	○	○	①パーティションテーブル作成 ②①に対しパーティションを追加
パーティショニングしたテーブルの削除	○	ー	パーティションテーブル、パーティションの両方が削除される。
項目の追加、修正	○	ー	全てのパーティションに対して実施される。 パーティション個別には実行できない。
パーティションの追加　方法1 (CREATE TABLE)	ー	○	パーティションテーブルに対しパーティションを追加作成
パーティションの追加　方法2 (ATTACH)	○	○	①パーティションの元のテーブルを作成 ②パーティションテーブルに①をパーティションとして組み込みを行う。

- 保守一覧の説明
- パーティショニングしたテーブルの作成

　パーティショニングしたテーブルは、パーティションテーブルを作成した後、パーティションテーブルで管理されるパーティションを追加することで作成されます。列集合の範囲でパーティション化する場合は範囲パーティション（以降レンジパーティションとする）、列の値を明示的に列挙する場合はリストパーティションになります。パーティションテーブルは、このパーティションキーを指定します。

時間隔のレンジパーティションの作成例

パーティションテーブルを作成します。パーティションキーはTEST_YMDになります。

```
test01=# CREATE TABLE TEST_PART_TBL(
test01(#     TEST_NO numeric(10),
test01(#     TEST_YMD timestamp NOT NULL,
test01(#     TEST_ID  varchar(32) ,
test01(#     TEST_CHAR varchar(300),
test01(#     CONSTRAINT TEST_PART_TBL_PK PRIMARY KEY (
test01(#         TEST_NO,
test01(#         TEST_YMD
test01(#     )
test01(# )
test01-# PARTITION BY RANGE (
test01(#     TEST_YMD
test01(# );
CREATE TABLE
```

次にパーティションを追加します。レンジパーティションの指定は、パーティションキーの列集合の範囲を開始以上、終了未満で指定します。以下は2023年の１月度の日付を対象したパーティションになります。

```
test01=# CREATE TABLE TEST_PART_TBL_P2023_01 PARTITION OF TEST_PART_TBL
test01-# FOR VALUES FROM ('2023-01-01') to ('2023-02-01') with (FILLFACTOR = 80);
CREATE TABLE
```

同様にパーティションテーブルで管理される月度のパーティションを事前に作成しておきます。想定外のパーティションキーのデータが発生したとき、エラーを発生させたくない場合は以下のようにDEFAULT句を用いてデフォルトパーティションを用意しておきます。デフォルトパーティションで注意する点は、新規にパーティションを追加しようとしたとき、パーティションキーのデータがデフォルトパーティションに存在するとパーティション追加はエラーになります。事前に該当データを移動（デフォルトパーティションからは削除という意味）しておく手間が発生します。

```
test01=# CREATE TABLE TEST_PART_TBL_DEFAULT PARTITION OF TEST_PART_TBL DEFAULT with (FILLFACTOR = 80);
CREATE TABLE
```

作成したテーブルを確認します。下記では上記の1月分に加え2月分のパーティションも追加しています。

```
test01=# ¥d+ TEST_PART_TBL
                                        Partitioned table "public.test_part_tbl"
    Column     |            Type             | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
---------------+-----------------------------+-----------+----------+---------+----------+-------------+--------------+-------------
 test_no       | numeric(10,0)               |           | not null |         | main     |             |              |
 test_ymd      | timestamp without time zone |           | not null |         | plain    |             |              |
 test_id       | character varying(32)       |           |          |         | extended |             |              |
 test_char     | character varying(300)      |           |          |         | extended |             |              |
Partition key: RANGE (test_ymd)
Indexes:
    "test_part_tbl_pk" PRIMARY KEY, btree (test_no, test_ymd)
Partitions: test_part_tbl_p2023_01 FOR VALUES FROM ('2023-01-01 00:00:00') TO ('2023-02-01 00:00:00'),
            test_part_tbl_p2023_02 FOR VALUES FROM ('2023-02-01 00:00:00') TO ('2023-03-01 00:00:00'),
            test_part_tbl_default DEFAULT
```

- パーティショニングしたテーブルの削除

  パーティショニングしたテーブルの削除はパーティションテーブルを削除すれば関連するパーティションも共に削除されます。

- 項目の追加・修正

  項目の追加を行う際はパーティションテーブルに対して実施します。以下では項目：TEST_FILLERを追加しています。

```
test01=# ALTER TABLE TEST_PART_TBL ADD COLUMN TEST_FILLER TEXT;
ALTER TABLE
```

  以下のように関連するパーティションに対しても変更は反映されています。

```
test01=# ¥d test_part_tbl_p2023_01
                  Table "public.test_part_tbl_p2023_01"
    Column     |            Type             | Collation | Nullable | Default
---------------+-----------------------------+-----------+----------+---------
 test_no       | numeric(10,0)               |           | not null |
 test_ymd      | timestamp without time zone |           | not null |
 test_id       | character varying(32)       |           |          |
 test_upd_time | timestamp without time zone |           |          | now()
 test_char     | character varying(300)      |           |          |
 test_filler   | text                        |           |          |                    ←自動で反映された。
Partition of: test_part_tbl FOR VALUES FROM ('2023-01-01 00:00:00') TO ('2023-02-01 00:00:00')
Indexes:
    "test_part_tbl_p2023_01_pkey" PRIMARY KEY, btree (test_no, test_ymd)
```

- パーティションの追加

  時間隔のレンジパーティションでは新しい日時のデータを格納するために運用上で随時パーティション追加が発生します。追加手段は２つあり、運用上の制限などにより以下のいずれかで対応します。

  【方法１　CREATE TABLE】

  「パーティショニングしたテーブルの作成」のパーティション追加と同様です。**保守作業上で注意すべき点はパーティションテーブルに対するトランザクションが発生している場合、CREATE TABLE投入前のトランザクションの完了を待つためCREATE TABLE文が完了できません。この状態で別のトランザクションが発生すると、そのトランザクションがCREATE TABLEの完了を待つというロックエスカレーションが発生することです。**常時、大量のトランザクションが発生するシステムでは保守時間を設け、その時間内で実施することを考慮する必要があります。以下はロックエスカレーションを発生させる例になります。

  [トランザクション１]（CREATE TABLE投入前のトランザクション）

```
test01=# begin;
BEGIN
test01=*# select count(*) from test_part_tbl where test_ymd = '2023-01-10';
 count
-------
    38
(1 row)
```

  [トランザクション２]（CREATE TABLE）

```
test01=# CREATE TABLE TEST_PART_TBL_P2023_02 PARTITION OF TEST_PART_TBL
test01-# FOR VALUES FROM ('2023-02-01') to ('2023-03-01') with (FILLFACTOR = 80);
```

  応答せずトランザクション１の完了を待ちます。

  [トランザクション３]（ロックエスカレーションするトランザクション）

```
test01=# begin;
```

```
BEGIN
test01=*# select count(*) from test_part_tbl where test_ymd = '2023-01-12 ';
```

クエリは応答せずトランザクション２の完了を待機し、ロックエスカレーションが発生します。

[トランザクション１]

トランザクションを終了させます。

```
test01=*# end;
COMMIT
```

[トランザクション２]

トランザクション１の完了後、即時応答します。

```
CREATE TABLE
test01=#
```

[トランザクション３]

トランザクション２の完了後、即時応答します。

```
 count
-------
    36
(1 row)
```

【方法２　ATTACH】

CREATE TABLEによるパーティションの追加と比較して、手順は増えますが同時実行のトランザクションがあっても追加が可能です。常時、大量のトランザクションが発生するシステムでのパーティション追加はこの方法で行います。

追加例

まず、LIKE句を用いてパーティションテーブルを元にテーブルを作成します。

```
test01=# CREATE TABLE TEST_PART_TBL_P2023_03 (LIKE TEST_PART_TBL INCLUDING DEFAULTS INCLUDING CONSTRAINTS) with ( FILLFACTOR = 80);
CREATE TABLE
```

次に作成したテーブルをパーティションとしてパーティションテーブルにATTACHします。

```
test01=# ALTER TABLE TEST_PART_TBL ATTACH PARTITION TEST_PART_TBL_P2023_03
test01-# FOR VALUES FROM ('2023-03-01') to ('2023-04-01');
ALTER TABLE
```

4. パーティショニングしたテーブルの運用

4. 2 保守一覧(2/3)

保守内容	保守対象		説明
	パーティションテーブル	パーティション	
パーティションの削除（DROP）	ー	○	特定のパーティションを削除する。
パーティションの切り離し（DETACH）	○	○	特定のパーティションをパーティションテーブルから切り離す。
パーティションの切り捨て（TRUNCATE）	○	○	パーティションのデータを切り捨てる。パーティションテーブルに実施した場合は全てのパーティションが対象。
テーブルのオプション等の変更（FILLFACTOR,TABLESPACE）	△	○	FILLFACTORはパーティション単位に変更可能。表領域はパーティションテーブル、パーティションどちらも可能。パーティションテーブルに対して実施の場合は以後作成されるパーティションのデフォルトが変更される。

- 保守一覧の説明

• パーティションの削除（DROP）

時間隔のレンジパーティションでは、管理対象外となった過去のデータ削除として古いパーティションの削除を行います。**CREATE TABLEのパーティション追加同様にロックエスカレーションを発生させます**ので、常時、大量のトランザクションが発生するシステムでは保守時間を設け、そこで実施することを考慮する必要があります。運用影響を最小限にしたい場合、PostgreSQL 14以降であれば次の項で示すようにCONCURRENTLY句を付与してDETACHした後に削除する方法があります。

• パーティションの切り離し（DETACH）

切り離しはパーティションをパーティションテーブルの管理対象外にします。切り離されたパーティションは通常のテーブルになります。**DETACHはATTACHと異なり、CREATE TABLEのパーティション追加同様にロックエスカレーションを発生させます。**運用においてはバージョン13以前と14以降では取りうる対策は異なるため、以下に例を示します。

【バージョン13以前】

[トランザクション１]

```
test01=# begin;
BEGIN
test01=*# select count(*) from test_part_tbl where test_ymd = '2023-02-10';
 count
-------
     0
(1 row)
```

トランザクションは完了させないでおきます。クエリの検索対象データはDETACHするパーティションとは異なるパーティションです。

[トランザクション２ (DETACH)]

```
test01=# ALTER TABLE TEST_PART_TBL DETACH PARTITION TEST_PART_TBL_P2023_01 ;
```

応答しません。トランザクション１の完了を待機している状態。

[トランザクション３ (トランザクション２が完了しない状態で実行)]

```
test01=# begin;
BEGIN
test01=*# select count(*) from test_part_tbl where test_ymd = '2023-02-11' ;
```

クエリはDETACHするパーティションとは異なるパーティションですが上記クエリは応答せずDETACHの完了を待機し**ロックエスカレーションが発生します。**

[トランザクション１]

ここでトランザクションを完了させます。

```
test01=*# end;
COMMIT
```

[トランザクション２]

```
ALTER TABLE
```

即座に応答しDETACHは完了します。

[トランザクション３]

トランザクション３の待機中のクエリも即座に応答します。

```
 count
-------
     0
(1 row)
```

バージョン13以前で上記を回避する対策としてはDETACHは常時、大量のトランザクションが発生するシステムでは保守時間を設け、そこで実施することを考慮する必要があります。

バージョン14以降ではALTER TABLE DETACHはCONCURRENTLY句を付加することでロックエスカレーションを回避できます。ただし、デフォルトパーティションが存在するとCONCURRENTLY句はエラーとなり実行できません。以下、例はデフォルトパーティションが存在しない状態での実施例です。

【バージョン14以降（CONCURRENTLY句を付加した例）】

[トランザクション１]

```
test01=# begin;
BEGIN
test01=*# select count(*) from test_part_tbl where test_ymd = '2023-02-10';
 count
-------
     0
(1 row)
```

トランザクションは完了させないでおきます。

[トランザクション２（DETACH+CONCURRENTLY)]

```
test01=# ALTER TABLE TEST_PART_TBL DETACH PARTITION TEST_PART_TBL_P2023_01 CONCURRENTLY;
```

応答しません。トランザクション１の完了を待機するところまでは同じです。

[トランザクション３]

```
test01=# begin;
BEGIN
test01=*# select count(*) from test_part_tbl where test_ymd = '2023-02-11';
 count
-------
     0
(1 row)
```

ロックエスカレーションは発生せず、クエリは応答します。

[トランザクション１]

ここでトランザクションを完了させます。

```
test01=*# end;
COMMIT
```

[トランザクション２]

```
ALTER TABLE
```

即座に応答しDETACHは完了します。

なお、デフォルトパーティションが存在する状態でCONCYRRENTLY句を利用すると以下のエラーになります。

```
test01=# ALTER TABLE TEST_PART_TBL DETACH PARTITION TEST_PART_TBL_P2023_01 CONCURRENTLY;
ERROR:  cannot detach partitions concurrently when a default partition exists
```

- パーティションの切り捨て（TRUNCATE)

パーティショニングされたテーブル全体またはパーティション単位での切り捨てが可能です。パーティション単位の場合、切り捨て対象のパーティションに対するトランザクションが発生していれば通常のテーブルと同様にトランザクションの終了待ちとなります。切り捨て対象でない他のパーティションへのトランザクションであれば影響はありません。パーティションテーブルに対して切り捨てを行った場合はパーティショニングされたテーブル全体＝全てのパーティションに対して実施されます。

- テーブルオプションの変更

  パーティションに対して行う変更は通常のテーブルと同様です。パーティションテーブルに対しては指定できるもの、できないものがあります。

  - FILLFACTORの指定はパーティションに対してのみ可能です。
  - 表領域の指定はできますが、これは作成済みのパーティションの表領域が変更されるのではなく、以後CREATE TABLEで作成されるパーティションの格納場所が指定した表領域になります。

  例

```
test01=# ALTER TABLE test_part_tbl SET TABLESPACE testspace;
ALTER TABLE
test01=# ¥d+ test_part_tbl
                                        Partitioned table "public.test_part_tbl"
    Column     |            Type             | Collation | Nullable | Default | Storage  | Compression | Stats target | Description

---------------+-----------------------------+-----------+----------+---------+----------+-------------+--------------+-------------
 test_no       | numeric(10,0)               |           | not null |         | main     |             |              |
 test_ymd      | timestamp without time zone |           | not null |         | plain    |             |              |
 test_id       | character varying(32)       |           |          |         | extended |             |              |
 test_upd_time | timestamp without time zone |           |          | now()   | plain    |             |              |
 test_char     | character varying(300)      |           |          |         | extended |             |              |
 test_filler   | text                        |           |          |         | extended |             |              |
Partition key: RANGE (test_ymd)
Indexes:
    "test_part_tbl_pk" PRIMARY KEY, btree (test_no, test_ymd)
Partitions: test_part_tbl_p2023_01 FOR VALUES FROM ('2023-01-01 00:00:00') TO ('2023-02-01 00:00:00'),
            test_part_tbl_p2023_02 FOR VALUES FROM ('2023-02-01 00:00:00') TO ('2023-03-01 00:00:00'),
            test_part_tbl_p2023_03 FOR VALUES FROM ('2023-03-01 00:00:00') TO ('2023-04-01 00:00:00')
Tablespace: “testspace”       ←変更した表領域になっています

test01=# ¥d+ test_part_tbl_p2023_03
                                          Table "public.test_part_tbl_p2023_03"
    Column     |            Type             | Collation | Nullable | Default | Storage  | Compression | Stats target | Description

---------------+-----------------------------+-----------+----------+---------+----------+-------------+--------------+-------------
 test_no       | numeric(10,0)               |           | not null |         | main     |             |              |
 test_ymd      | timestamp without time zone |           | not null |         | plain    |             |              |
 test_id       | character varying(32)       |           |          |         | extended |             |              |
 test_upd_time | timestamp without time zone |           |          | now()   | plain    |             |              |
 test_char     | character varying(300)      |           |          |         | extended |             |              |
 test_filler   | text                        |           |          |         | extended |             |              |
Partition of: test_part_tbl FOR VALUES FROM ('2023-03-01 00:00:00') TO ('2023-04-01 00:00:00')
Partition constraint: ((test_ymd IS NOT NULL) AND (test_ymd >= '2023-03-01 00:00:00'::timestamp without time zone) AND (test_ymd <
'2023-04-01 00:00:00'::timestamp without time zone))
Indexes:
    "test_part_tbl_p2023_03_pkey" PRIMARY KEY, btree (test_no, test_ymd)
Access method: heap
Options: fillfactor=80
```

  既に存在するパーティションは変更されていません。変更する場合はパーティションに対してALTER TABLEする必要があります。

4. パーティショニングしたテーブルの運用

4. 2 保守一覧(3/3)

保守内容	保守対象		説明
	パーティションテーブル	パーティション	
索引の追加 方法1（業務影響あり）	○	ー	パーティションテーブルに対し索引追加する場合、追加処理中はパーティションテーブルに対し書き込みロックがかかる。
索引の追加 方法2（業務影響小）	○	○	パーティションテーブルに対し索引をONLY句で追加後、各パーティションで索引追加しATTACHする。
VACUUM FULL	○	○	パーティションテーブルに対し実行すれば管理される全てのパーティションが対象。 パーティションに対して実行すれば指定したパーティションのみが対象。
VACUUM			
ANALYZE			

- 保守一覧の説明
- 索引の追加

通常のテーブルでは索引を作成する際、CONCURRENTLY句を付加することで対象のテーブルに挿入、更新、削除を防止するようなロックを獲得せずに索引を作成できますが、パーティションテーブルに索引を作成する際はCONCURRENTLY句を付加できません。実施すると以下のようにエラーになります。

```
test01=# create index concurrently test_part_tbl_test_id_idx on test_part_tbl (test_id);
ERROR:  cannot create index on partitioned table "test_part_tbl" concurrently
```

そのため、保守時間を確保できるかどうかで対応は異なります。

【方法１　保守時間を確保できるとき】

パーティションテーブルに対してCREATE INDEXで索引を追加します。全てのパーティションに対して索引が作成されます。一つのコマンドで実施可能ですが、追加処理中は書き込みロックがかかるため、保守時間内に実施する必要があります。

【方法２　保守時間を確保できないとき】

作業手順は増えますが、業務への影響を最小限にすることができます。

①ONLY句を付与してパーティションテーブルに索引作成

```
test01=# create index test_part_tbl_test_id_idx on only test_part_tbl (test_id);
CREATE INDEX
```

ただし、この時点ではパーティションに索引は作成されておらず使用不可能です。パーティションテーブルの索引はINVALIDとなっています。

```
test01=# ¥di
                                List of relations
 Schema |            Name              |       Type        |  Owner   |         Table
--------+------------------------------+-------------------+----------+------------------------
 public | test_part_tbl_default_pkey   | index             | postgres | test_part_tbl_default
 public | test_part_tbl_p2023_01_pkey  | index             | postgres | test_part_tbl_p2023_01
 public | test_part_tbl_p2023_02_pkey  | index             | postgres | test_part_tbl_p2023_02
 public | test_part_tbl_pk             | partitioned index | postgres | test_part_tbl
 public | test_part_tbl_test_id_idx    | partitioned index | postgres | test_part_tbl
(5 rows)

test01=# ¥d test_part_tbl
                  Partitioned table "public.test_part_tbl"
    Column     |            Type             | Collation | Nullable | Default
---------------+-----------------------------+-----------+----------+---------
```

```
 test_no       | numeric(10,0)               |           | not null |
 test_ymd      | timestamp without time zone |           | not null |
 test_id       | character varying(32)       |           |          |
 test_upd_time | timestamp without time zone |           |          | now()
 test_char     | character varying(300)      |           |          |
 test_filler   | text                        |           |          |
Partition key: RANGE (test_ymd)
Indexes:
    "test_part_tbl_pk" PRIMARY KEY, btree (test_no, test_ymd)
    "test_part_tbl_test_id_idx" btree (test_id) INVALID
Number of partitions: 2 (Use ¥d+ to list them.)
Tablespace: "testspace"
```

②各パーティションにCONCURRENTLY句を用いて索引作成

```
test01=# create index concurrently test_part_tbl_p2023_01_test_id_idx on test_part_tbl_p2023_01 (test_id);
CREATE INDEX
```

③②で作成した索引を①で作成した索引にATTACH

```
test01=# alter index  test_part_tbl_test_id_idx attach partition test_part_tbl_p2023_01_test_id_idx;
ALTER INDEX
```

②、③を繰り返し全てのパーティションにATTACHが完了するとパーティションテーブルの索引が有効になり、INVALIDは消えます。

```
test01=# ¥d test_part_tbl
                Partitioned table "public.test_part_tbl"
    Column     |            Type             | Collation | Nullable | Default
---------------+-----------------------------+-----------+----------+---------
 test_no       | numeric(10,0)               |           | not null |
 test_ymd      | timestamp without time zone |           | not null |
 test_id       | character varying(32)       |           |          |
 test_upd_time | timestamp without time zone |           |          | now()
 test_char     | character varying(300)      |           |          |
 test_filler   | text                        |           |          |
Partition key: RANGE (test_ymd)
Indexes:
    "test_part_tbl_pk" PRIMARY KEY, btree (test_no, test_ymd)
    "test_part_tbl_test_id_idx" btree (test_id)
Number of partitions: 2 (Use ¥d+ to list them.)
Tablespace: "testspace "
```

注意点

②のCREATE INDEX後、ATTACHする前にこの索引が利用されているトランザクションが発生すると、③のATTACHは完了できません。その状態で同索引が利用されるような新規クエリは待たされます。この動作はパーティションの追加の方法１やパーティションの削除と同様であり、限定的ですが業務影響を発生させる可能性があるため、追加される索引が利用されない時間帯で、かつ②で索引を作成したら即③のATTACHを実施することを推奨します。

- VACUUM FULL

パーティションテーブルに対して実施した場合、すべてのパーティションにVACUUM FULLを実施します。パーティションテーブルに対しトランザクションが発生している場合はトランザクションが完了するまで待機します。パーティションに対して実施した場合は、そのパーティションのみVACUUM FULLが実施されます。パーティションテーブルに対しトランザクションが発生している場合でもパーティション・プルーニングにより除外されたパーティションであれば影響ありません。

- VACUUM

パーティションテーブルに対して実施した場合、すべてのパーティションにVACUUMを実施します。パーティションに対して実施した場合は、そのパーティションのみVACUUMが実施されます。

- ANALYZE

パーティションテーブルに対して実施した場合、パーティションテーブルは全てのパーティションから統計情報を収集しパーティションテーブルの統計情報を更新し、さらに各パーティションに対しても統計情報を更新します。パーティションテーブルのみの統計情報の更新はできません。パーティションに対して実施した場合は、そのパーティションのみ統計情報を更新します。また、自動バキューム処理ではパーティションの統計情報は更新されますが、パーティションテーブルについては更新されません。

4. パーティショニングしたテーブルの運用

4. 3 パーティショニング管理ツール

- **pg_partman**

  pg_partmanは時間ベースとシリアルベースのレンジパーティションを管理するための周辺ソフトウェアです。運用におけるパーティションの追加、削除または切り離しを自動化することで、運用者の管理を軽減できます。また pg_partmanはAWS、Azureのデータベースサービスでもサポートされております。

  運用管理手順
  ① パーティションテーブルを作成
  ② create_parent関数でパーティションテーブルを管理テーブル登録、および初期パーティションを作成
  ③ パーティションの保存期間、削除・切り離しの指定
  ④ メンテナンスバッチ＊により自動で③の指定に従いパーティション追加、削除または切り離しを実行

  ＊：バッチの起動はpg_partmanで用意されているpg_partman_bgwを使用するか、pg_cronやその他のスケジューラツールから起動してもよい。

- マニュアル
  - インストールマニュアル

    https://github.com/pgpartman/pg_partman#installation
  - ドキュメント

    https://github.com/pgpartman/pg_partman/blob/master/doc/pg_partman.md
- 使用例

以下に使用例を示します。

①パーティションテーブル作成

```
test01=# CREATE TABLE TEST_PART_TBL(
test01(#     TEST_NO numeric(10),
test01(#     TEST_KBN numeric(2),
test01(#     TEST_YMD timestamp NOT NULL,
test01(#     TSET_UP_TIME timestamp without time zone Default now(),
test01(#     TEST_CHAR varchar(1000),
test01(#     CONSTRAINT TEST_PART_TBL_PK PRIMARY KEY (
test01(#         TEST_NO,
test01(#         TEST_YMD
test01(#     )
test01(# )
test01-# PARTITION BY RANGE (
test01(#     TEST_YMD
test01(# );
CREATE TABLE
```

②create_parent関数の実行

create_parent関数で管理テーブルであるpart_configに管理用レコード、初期パーティション、テンプレートテーブルが作成されます。関数の各パラメタの指定はドキュメントを確認してください。p_parent_table はパーティションテーブルを指定しますが、“スキーマ名.テーブル名”で指定する必要があります。

実施日が2022年12月の場合

```
test01=# SELECT partman.create_parent( p_parent_table => 'public.test_part_tbl',
test01(#  p_control => 'test_ymd',
test01(#  p_type => 'native',
test01(#  p_interval=> 'monthly',
test01(#  p_premake => 2);
```

```
 create_parent
---------------
 t
(1 row)
```

作成された管理テーブル（partman.part_config）を確認します。

```
test01=# select * from partman.part_config where parent_table = 'public.test_part_tbl';
-[ RECORD 1 ]--------------+---------------------------------------
parent_table               | public.test_part_tbl ←create_parentで指定したパーティションテーブル
control                    | test_ymd ←create_parentで指定したパーティションキー
partition_type             | native   ←create_parentで指定したパーティショニングの種類
partition_interval         | 1 mon    ←create_parentで指定したインターバル
constraint_cols            |
premake                    | 2        ←create_parentで指定したインターバル数
optimize_trigger           | 4
optimize_constraint        | 30
epoch                      | none
inherit_fk                 | t
retention                  |          ←保存期間：初期値は無制限。削除、切り離しなし。
retention_schema           |
retention_keep_table       | t        ←初期値は削除ではなく切り離し
retention_keep_index       | t
infinite_time_partitions   | f        ←初期値はfalse。パーティションにレコードが存在しないと自動でパーティション
は作成されません。
datetime_string            | YYYY_MM  ←parttion_intervalに応じて決定されパーティション名の区別に使われます。
automatic_maintenance      | on       ←初期値は自動メンテナンス対象
jobmon                     | t
sub_partition_set_full     | f
undo_in_progress           | f
trigger_exception_handling | f
upsert                     |
trigger_return_null        | t
template_table             | partman.template_public_test_part_tbl  ←テンプレートテーブルも作成されます。
publications               |
inherit_privileges         | f
constraint_valid           | t
subscription_refresh       |
drop_cascade_fk            | f
ignore_default_data        | f
```

作成されたパーティションを確認します。

```
test01=# ¥d+ TEST_PART_TBL
                                      Partitioned table "public.test_part_tbl"
    Column    |            Type             | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
--------------+-----------------------------+-----------+----------+---------+----------+-------------+--------------+-------------
 test_no      | numeric(10,0)               |           | not null |         | main     |             |              |
 test_kbn     | numeric(2,0)                |           |          |         | main     |             |              |
 test_ymd     | timestamp without time zone |           | not null |         | plain    |             |              |
 tset_up_time | timestamp without time zone |           |          | now()   | plain    |             |              |
 test_char    | character varying(1000)     |           |          |         | extended |             |              |
Partition key: RANGE (test_ymd)
Indexes:
    "test_part_tbl_pk" PRIMARY KEY, btree (test_no, test_ymd)
Partitions: test_part_tbl_p2022_10 FOR VALUES FROM ('2022-10-01 00:00:00') TO ('2022-11-01 00:00:00'),
            test_part_tbl_p2022_11 FOR VALUES FROM ('2022-11-01 00:00:00') TO ('2022-12-01 00:00:00'),
            test_part_tbl_p2022_12 FOR VALUES FROM ('2022-12-01 00:00:00') TO ('2023-01-01 00:00:00'),  ←実施日の月
            test_part_tbl_p2023_01 FOR VALUES FROM ('2023-01-01 00:00:00') TO ('2023-02-01 00:00:00'),
            test_part_tbl_p2023_02 FOR VALUES FROM ('2023-02-01 00:00:00') TO ('2023-03-01 00:00:00'),
            test_part_tbl_default DEFAULT
```

パーティションは以下のように作成されます。

- 実施日が2022年12月のため、 p_premake分のパーティションとして2ヵ月先の2023年2月までのパーティションが作成されます。
- create_parent関数のパラメタであるp_start_partitionが未指定の場合、p_premakeと同インターバルの数の過去のパーティションが作成されます。

- デフォルトパーティションは必ず作成されます。（抑止できないので不要であれば削除する）

③運用調整

パーティショニングしたテーブルの過去パーティションの保存期間や保存期間外となったパーティションの切り離し、または削除などの指定をpart_configの対象パーティションテーブルのレコードを更新することで調整します。以後はpg_partman_bgwまたは任意のスケジューラツールなどによりrun_maintenance（ファンクション）またはrun_maintenance_proc（run_maitenanceを起動するプロシージャ）が自動実行されることによってパーティションの追加、整理が行われます。

- 注意点

- パーティション作成済みのパーティションテーブルを登録する場合、パーティション名のフォーマットは"パーティションテーブル名+_p+part_config.datetime_string"であることが必要です。そうでない場合はパーティションの整理でエラーになります。既存のパーティション名をALTER TABLE パーティション名 RENAME TO 新パーティション名で変更してください。
- デフォルトパーティションにデータが挿入され、かつそのデータのパーティションを自動作成しようとしたときエラーになります。挿入される可能性のある未来日のデータがある場合はそのインターバル分をpart_config.premakeで指定して作成しておくか、デフォルトパーティションを削除（データ挿入時にエラーにはなるが、パーティション自動作成時にはエラーになることはない）しておきます。デフォルトパーティションを運用として利用する、しないはメリット、デメリットがあるので方針として決めておく必要があります。エラーが発生した場合の対応は以下のように実施します。
    ① デフォルトパーティションの対象のデータをバックアップした後に対象データを削除
    ② パーティションを作成
    ③ バックアップしたデータを作成されたパーティションに復元する
- create_parent関数は初期パーティション作成時にfillfactorを指定できません。初期作成されたパーティションは必要に応じてALTER TABLE パーティション名　SET (fillfactor = XX)で変更してください。また、以後自動で作成されるパーティションについてはテンプレートテーブルのfillfactorを変更しておくことでその値が利用されます。

5. 備考

ライセンス

本作品はCC-BYライセンスによって許諾されています。ライセンスの内容を知りたい方はこちらでご確認ください。文書の内容、表記に関する誤り、ご要望、感想等につきましては、PGEConsのサイトを通じてお寄せいただきますようお願いいたします。

- Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
- Microsoft、Windows Server、SQL Server、米国Microsoft Corporationの米国及びその他の国における登録商標または商標です。
- Oracleは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- PostgreSQLは、PostgreSQL Community Association of Canadaのカナダにおける登録商標およびその他の国における商標です。
- その他、本資料に記載されている社名及び商品名はそれぞれ各社が商標または登録商標として使用している場合があります。

著者

版	所属企業・団体名	部署名	氏名
第1．0版	富士通Japan株式会社	ソリューション開発本部 キャッシュレスソリューション 事業部	多田　明弘
	三菱電機株式会社	情報技術総合研究所 データマネジメント システム技術部	田中　覚

PGECons
PostgreSQL Enterprise Consortium