OSS-DB

OSS-DB Exam Silver
技術解説無料セミナー
2020/10/17 開催

主題	開発/SQL　（出題範囲　32%）
副題	トランザクションの概念　【重要度：１】 SQLコマンド　【重要度：13】

本日の講師

NTT テクノクロス株式会社
上原 一樹

LPI-JAPAN

■OSS-DBとは

オープンソースのデータベースソフトウェア「PostgreSQL」を扱うことができる技術力の認定です。様々な分野でPostgreSQLの利用拡大が進む中でOSS-DBの認定を持つことは、自分のキャリアのアピールにもつながります。

✓OSS-DB Goldは設計やコンサルティングができる技術力の証明

PostgreSQLについての深い知識を持ち、データベースの設計や開発のほか、パフォーマンスチューニングやトラブルシューティングまで行えることが証明できます

✓OSS-DB Silverは導入や運用ができる技術力の証明

PostgreSQLについての基本的な知識を持ち、データベースの運用管理が行えるエンジニアとしての証明ができます

✓対象のバージョンはPostgreSQL 11

OSS-DB 技術者認定はなぜ必要か？

#OSS-DB

■一般論を学んだだけでは現場で活躍することは難しい

- 使う前に設定が必要

  デフォルトの状態では、業務要件に対応することは難しい。

  - ユーザ
  - アクセス権
  - テーブルの作成
  - プログラムの開発

- 重要な用途
  - 基幹業務での利用に向けた設計
  - バックアップ
  - セキュリティ

- 複雑な用途
  - パフォーマンスチューニング
  - トラブルシューティング

認定試験を通して、
実際の運用に則した技術、知識を習得できる

認定試験への傾向と対策

■公式ドキュメントは使用するバージョンのドキュメントを読むべき

- バージョンによって、機能の違いがあるため、採用するバージョンとあったものを読む。
- 本セミナー資料でも参考として、バージョン11のPostgreSQL文書のリンクを記載する。

■GUCパラメータやシステムテーブル・ビューは、単純に意味を覚えるのではなく、影響まで理解しなければならない

例 deadlock_timeout

意味: ロック状態になった時にデッドロック検出処理を開始するまでの待機時間

影響: 値を小さくすると、デッドロックの検出は早くなるが、 実際にはデッドロックが発生していないのに検出処理が 動くことが多くなるため、CPUに無駄な負荷がかかる可能性も高くなる

■実機での動作確認は極めて重要

- 自分が予想した通りに動作しなければ、何かしら理解不足があるということ。

本セミナーについて

■テーマ

- 本日は過去に実施されたオンラインセミナーでは扱っていない分野について解説する。
- 過去のオンラインセミナーの内容について
  - どちらも試験対策としてだけではなく、PostgreSQLを運用する上で重要な項目です。
  - 7月：「PostgreSQLのバックアップ方法」
    - https://oss-db.jp/__/download/5f0fb75b37471d78510554e4/20200719-silver-01.pdf
  - 9月：「VACUUM、ANALYZEの目的と使い方」と「自動バキュームの概念と動作」
    - https://oss-db.jp/__/download/5f3de5a37ea72b5fa4113218/20200905-silver-01.pdf

■本セミナーの進め方について

- 資料での説明に加え、デモを交えながら解説する。
- 実行するコマンドは資料の内容を参照。

本日のテーマ

データベースの導入手順や設定等の基本的な情報に加え、OSS-DB Silver(ver.2.0)で試験範囲として追加されたマテリアライズドビュー、宣言的パーティショニング等の機能を解説する。

■運用管理（52%）
- インストール方法　【重要度：2】
  - initdb
- 設定ファイル　【重要度：5】
  - postgresql.conf
  - pg_hba.conf
  - pg_settingsビュー

■開発/SQL　(32%)
- トランザクションの概念　【重要度：1】
  - トランザクション分離レベル
  - デッドロック
- SQL コマンド　【重要度：13】
  - マテリアライズドビュー
  - 宣言的パーティショニング

事前準備：環境構築

■VMを用意する

- CentOS 8

```
[root@ossdb ~]# cat /etc/redhat-release
CentOS Linux release 8.2.2004 (Core)
```

■PostgreSQLのインストール

- リポジトリ登録

```
[root@ossdb ~]# rpm -ivh https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-
x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

- PostgreSQL 11のインストール

```
[root@ossdb ~]# dnf -qy module disable postgresql
[root@ossdb ~]# dnf install -y postgresql11-server postgresql11-devel
```

※CentOS 7では、dnfコマンドの代わりにyumコマンドを使用する。なお、moduleはCentOS 8でのパッケージ管理の概念として新たに追加されたものである。

事前準備：環境構築

■OSユーザを切り替える

- postgresユーザはRPMを用いたインストールで自動で作成される。

```
[root@ossdb ~]# su - postgres
```

■環境変数の設定

- postgresユーザの .bash_profile は、~/.pgsql_profile を読み込む仕様になっている。

```
[postgres@ossdb ~]$ cat << EOF > ~/.pgsql_profile
PATH=/usr/pgsql-11/bin:$PATH
PGDATA=/var/lib/pgsql/local/11/data
export PATH PGDATA
EOF
```

- 再ログインまたは、sourceコマンドで環境変数を反映する。

```
[postgres@ossdb ~]$ source ~/.pgsql_profile
```

運用管理編

■運用管理（52%）
- インストール方法 【重要度：2】
  - initdb
- 設定ファイル 【重要度：5】
  - postgresql.conf
  - pg_hba.conf
  - pg_settingsビュー

■開発/SQL（32%）
- トランザクションの概念 【重要度：1】
  - トランザクション分離レベル
  - デッドロック
- SQL コマンド 【重要度：13】
  - マテリアライズドビュー
  - 宣言的パーティショニング

データベースクラスタの作成（initdb）

■データベースクラスタ（DBクラスタ）を作成する

```
[postgres@ossdb ~]$ initdb -D ~/local/11/data --no-locale --encoding=utf8
```

- DBクラスタとは
  - データベースの格納領域を指し、一つのインスタンスによって管理される。
  - データベースの集合体であり、デフォルトでは「postgres」、「template0」、「template1」という3つのデータベースが作成される。
  - ファイルシステムとしては、DBクラスタは全てのデータが格納される1つのディレクトリとなる。

■DBクラスタは、pg_ctlコマンドでも作成可能

```
[postgres@ossdb ~]$ pg_ctl initdb -D ~/local/11/data
```

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/creating-cluster.html
https://www.postgresql.jp/document/11/html/app-pg-ctl.html

データベースクラスタの作成（initdb）の補足

■-Dオプションは省略可能

- 環境変数PGDATAを定義すれば、-Dオプションを省略できる。

```
[postgres@ossdb ~]$ initdb --no-locale --encoding=utf8
```

- 環境変数が定義済みであれば、pg_ctlコマンド等、他でも同様に-Dオプションを省略可能となる。

■デフォルトで作成されるデータベースについて

- template0、template1は、テンプレートデータベースと呼ばれており、CREATE DATABASE実行時のベースとなる特別なデータベースである。
- CREATE DATABASE実行時、デフォルトでは、template1が新しいデータベースのコピー元となる。
  - データベース作成時にオプションで任意のデータベースをテンプレートとして指定することも可能。
- template0は削除不可、オブジェクトの追加もできない。

■initdbを実行したOSユーザがデータベースの管理者（スーパーユーザ）となる

- ただし、以下のように-Uオプションを指定した場合は指定したユーザ名が管理者となる。

```
[postgres@ossdb ~]$ initdb -D ~/local/11/data --no-locale --encoding=utf8 -U admin
```

pg_hba.confによる認証方式の設定

■PostgreSQLでは、クライアント認証を $PGDATA/pg_hba.conf で管理する

- フォーマットは以下のとおり。

```
local      database  user  auth-method  [auth-options]
host       database  user  address  auth-method  [auth-options]
hostssl    database  user  address  auth-method  [auth-options]
hostnossl  database  user  address  auth-method  [auth-options]
host       database  user  IP-address  IP-mask  auth-method  [auth-options]
hostssl    database  user  IP-address  IP-mask  auth-method  [auth-options]
hostnossl  database  user  IP-address  IP-mask  auth-method  [auth-options]
```

- 設定ファイルでは、接続先DB（どこに）、接続するDBMSユーザ（誰が）、接続元（どこから）に対してDBへの接続の許可/拒否を定義する。
  - 書かれていない内容の接続は全て拒否される。
  - テーブル単位での制御はできない。
    - データベース内の個別テーブルの制御はGRANT文、REVOKE文を使用する。
  - databaseに「replication」を指定した場合は物理レプリケーション接続の制御として解釈される。
- デフォルトではローカル接続が許可された状態となっているので、動作確認程度であれば変更不要。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/auth-pg-hba-conf.html

PostgreSQLの起動/停止（pg_ctl）

■起動

- 環境変数PGDATAが定義済みであれば、-Dオプションは省略可能。

```
[postgres@ossdb ~]$ pg_ctl -D ~/local/11/data start
```

■停止

```
[postgres@ossdb ~]$ pg_ctl -D ~/local/11/data stop
```

- 停止には以下の3つのモードが存在する。-mオプションで指定可能。

パラメータ名	説明
smart	全てのクライアントが切断し、オンラインバックアップ実行中の場合はそれが終了するまで待つ。
fast（デフォルト）	クライアントが切断するまで待たず、実行中のオンラインバックアップを終了させる。 全ての実行中のトランザクションをロールバックし、サーバを停止する。
immediate	クリーンアップ処理なしで、全てのサーバプロセスを即座に中断する。 次回起動時にクラッシュリカバリが行われる。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/app-pg-ctl.html

設定ファイル（postgresql.conf）の設定

■コメントアウトされている場合はデフォルト値が採用される
- 同じパラメータを複数設定した場合、ファイルの後ろに定義されたものが最終的に反映される。

■設定項目の詳細はpg_settingsで確認可能
- パラメータの値が反映の条件や変更可能なDBMSユーザはそれぞれ異なるので注意が必要。
  - 反映の条件（データベースサーバ起動時/reload実行時等）はcontextで確認可能。

```
postgres=# SELECT name,setting,context FROM pg_settings WHERE name = 'work_mem';
   name   | setting | context
----------+---------+---------
 work_mem | 4096    | user
(1 row)
```

contextの一部を紹介する。
- postmaster：起動時に反映
- sighup：pg_ctl reload時に反映
- user：任意のユーザが変更可能
- …等々、詳細はドキュメントを参照。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/view-pg-settings.html

■SET文を用いた設定方法
- postgresql.confに設定した場合、DBクラスタ全体に反映されてしまう。全体だと影響が大きくなりすぎるパラメータや、特定の処理にだけ適用させたい設定については、SET文を使用して、影響範囲を絞る。

```
postgres=# SET work_mem TO '64MB';
```

pg_settingsビューの補足

- pg_settingsビューはSHOW文、SET文の代わりとして使用可能
  - パラメータを参照することはもちろん、設定内容を更新することができる。
    - 更新の有効範囲はSET文と同様で、実行したセッションでのみ有効。
  - なお、INSERT、DELETEは実行できない。

- pg_settingsを用いた設定方法（SET文相当）

```
postgres=# UPDATE pg_settings SET setting = '64MB' WHERE name = 'work_mem';
 set_config
------------
 64MB
(1 row)
```

設定ファイル（postgresql.conf）の設定の補足

■動作確認等の環境で利用頻度の高いパラメータの一部をご紹介

- 対象のパラメータ及び、デフォルトでの設定内容は以下のとおり。

```
#listen_addresses = 'localhost'
#port = 5432
#shared_buffers = 128MB
logging_collector = on
#log_checkpoints = off
#log_lock_waits = off
#log_autovacuum_min_duration = -1
```

- パラメータ概要

パラメータ名	概要
listen_addresses	PostgreSQLが接続を待ち受けるIPアドレスを設定する。「*」で全てのIPアドレスを受け付ける。
port	PostgreSQLが使用するport番号を設定する。複数DBクラスタを起動する場合等に使用する。
shared_buffers	PostgreSQLの処理で主に使用されるメモリ量を設定する。
logging_collector	サーバログの出力を制御。10以前はデフォルトが無効となっていた。
log_checkpoints	チェックポイント処理をサーバログに出力する。解析に使用する。デフォルトでは無効。
log_lock_waits	ロック待ち情報をサーバログに出力する。解析に使用する。デフォルトでは無効。
log_autovacuum_min_duration	autovacuum処理をサーバログに出力する。解析に使用する。デフォルトでは「-1（無効）」。

サンプル問題（インストール方法）

■OSS-DB Silver　サンプル問題/例題解説　から本テーマに関するものをいくつか紹介します。

- 本セミナーでは1題のみ、時間を取って実施していただき、その後解説を行います。
  - 残りの問題は自習でご活用ください。
- 時間：2分間

サンプル問題　1.60

■**ホストdbserver(192.168.1.11)で稼働しているデータベースに、クライアントdbclient(192.168.1.12)上のユーザtest1がpsqlを使って接続する。pg_hba.confの内容が次の通りであるとして、以下の記述から適切なものを2つ選びなさい。**

```
local all  all peer
host  test all 192.168.1.12/32 trust
host  all  all 192.168.1.0/24  md5
```

A)　test1というデータベースにtest1というユーザで接続するときは、パスワードなしで認証される。
B)　testというデータベースにtest1というユーザで接続するときは、パスワードなしで認証される。
C)　test1というデータベースにtestというユーザで接続するときは、パスワードで認証する。
D)　testというデータベースにtestというユーザで接続するときは、パスワードで認証する。
E)　test2というデータベースにtest1というユーザで接続するときは、接続を拒絶される。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_management_03/60_160125

サンプル問題　1.73

■テンプレートデータベースに関する説明として誤っているものを１つ選びなさい。

A) CREATE DATABASEを実行すると、内部的にはテンプレートデータベースの1つがコピーされる。
B) 使用するテンプレートデータベースを指定しなければ、デフォルトではtemplate1が使用される。
C) デフォルトでは、データベースクラスタ内に2つのテンプレートデータベースが存在している。
D) データベースクラスタ内に作成できるテンプレートデータベースの数は最大で5個までである。
E) データベースがテンプレートデータベースかどうかを設定するフラグがあり、そのフラグが設定されているデータベースは削除(DROP DATABASE)できない。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_management_04/73_170522

サンプル問題　1.21

■**postgresql.conf ファイルの設定方法について、最も適切なものを2つ選びなさい。**

A) 論理値型のパラメータは true あるいは false と小文字で設定しなければならない
B) メモリを設定するパラメータでは kB, MB, GB, TB の単位が指定できる
C) 時間を設定するパラメータは、すべて秒単位で指定する
D) 行頭が # の行はコメント行である
E) パラメータの設定は
```
param1 = val1;
```
のようにセミコロンで終了し、セミコロンがなければ次行に継続する

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_management_02/21_121023

開発/SQL編（トランザクションの概念）

- 運用管理（52%）
  - インストール方法 【重要度：2】
    - initdb
  - 設定ファイル 【重要度：5】
    - postgresql.conf
    - pg_hba.conf
    - pg_settingsビュー
- 開発/SQL（32%）
  - トランザクションの概念 【重要度：1】
    - トランザクション分離レベル
    - デッドロック
  - SQL コマンド 【重要度：13】
    - マテリアライズドビュー
    - 宣言的パーティショニング

トランザクションの概念

■トランザクションとは

- ユーザから見た一連の処理のまとまりを指す。
- DBMSは、トランザクション管理機能によってデータの論理的な整合性を確保している。

■ここで言う一連の処理というのは、例として、銀行での振込処理が該当します。

- 口座001から口座002に1000円を振り込む

1. `UPDATE 預金口座 SET 預金残高 = 預金残高 - 1000 WHERE 口座番号 = 001`
2. `UPDATE 預金口座 SET 預金残高 = 預金残高 + 1000 WHERE 口座番号 = 002`

- 上記の処理2の実行中にデータベースがクラッシュした場合でも、トランザクション管理機能によって、処理1の実行結果が取り消される。
  - つまり、「全ての処理が実行される」か、「どちらの処理も実行されない」のいずれかの結果となる。

ACID（アシッド）特性について

■トランザクション処理に求められる4つの要素

ACID特性	意味	機能例
Atomicity （原子性）	トランザクションは、完全に実行されるか、完全に実行されないかのいずれかでなければならない	コミットメント制御機能
Consistency （整合性）	トランザクションの実行結果に関係なく、データは事前に与えられた制約を破った状態で残ってはならない	排他制御機能 （同時実行制御）
Isolation （独立性・分離性）	トランザクションは、同時実行中の他のトランザクションの影響を受けず、並列実行時であっても単独実行時と同じ結果を返さなければならない	
Durability （耐久性・持続性）	トランザクションの結果は、障害発生時でも失われてはならない	障害回復機能

PostgreSQLのトランザクション

■トランザクション管理

分類	SQL文
トランザクション開始	BEGIN; もしくは START TRANSACTION;
トランザクション確定	COMMIT; もしくは END;
更新の廃棄	ABORT; もしくは ROLLBACK [TO セーブポイント名];
セーブポイント作成	SAVEPOINT セーブポイント名;
セーブポイント破棄	RELEASE [SAVEPOINT] セーブポイント名;

- SAVEPOINTでは、現在のトランザクション内でセーブポイントを作成することができる。
  - セーブポイントを設定することで、それ以降に実行されたコマンドを全てロールバックし、セーブポイント時点の状態に戻すことができる。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/sql-savepoint.html

■PostgreSQLでは、基本的にAUTOCOMMITが有効となっている。

- トランザクションを明示的に開始しない場合は、ひとつのSQLがひとつのトランザクションとして扱われ、即時にコミットされる。

トランザクション分離レベル

■PostgreSQLで実装されているトランザクション分離レベルは以下のとおり。

- read committed（※PostgreSQLでは、read uncommittedを設定しても同等に扱われる）
  - 1つの問い合わせで見えるのは、その問い合わせが開始される前にコミットされた行のみ見える。デフォルト設定。
- repeatable read
  - 現在のトランザクションにおける全ての問い合わせは、トランザクションで最初の問い合わせまたは更新処理が実行される前にコミットされた行だけ見える。
- serializable
  - 現在のトランザクションにおける全ての問い合わせは、トランザクションで最初の問い合わせまたは更新処理が実行される前にコミットされた行だけ見える。
  - 同時実行中のトランザクションのパターンによって、直列に処理が行えない場合はエラーとなり、ロールバックされる。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/transaction-iso.html
https://www.postgresql.jp/document/11/html/sql-set-transaction.html

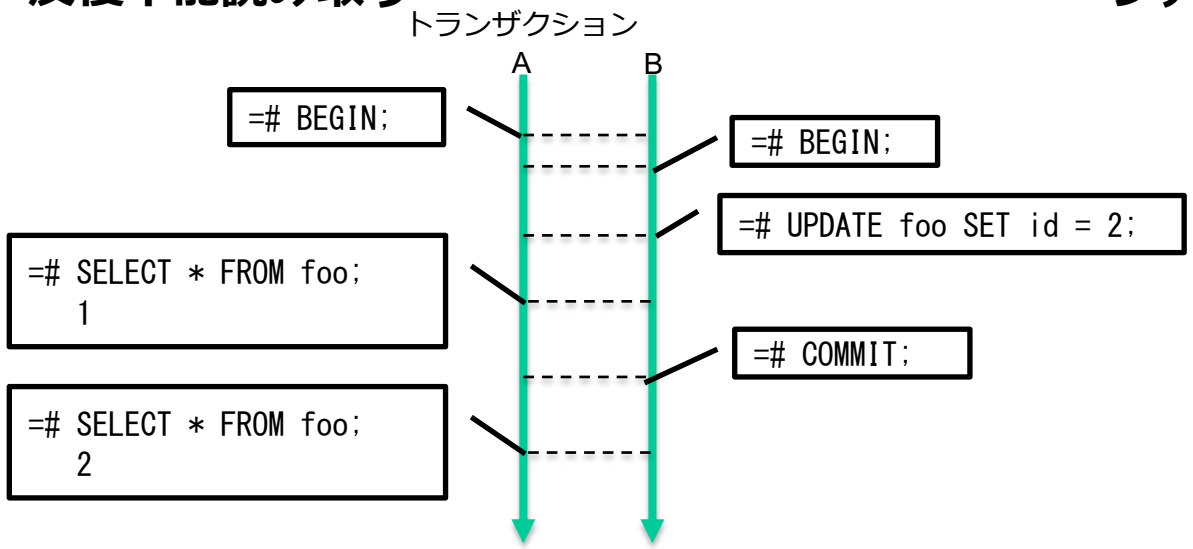
トランザクションの隔離性

■read committedで発生する可能性のある現象

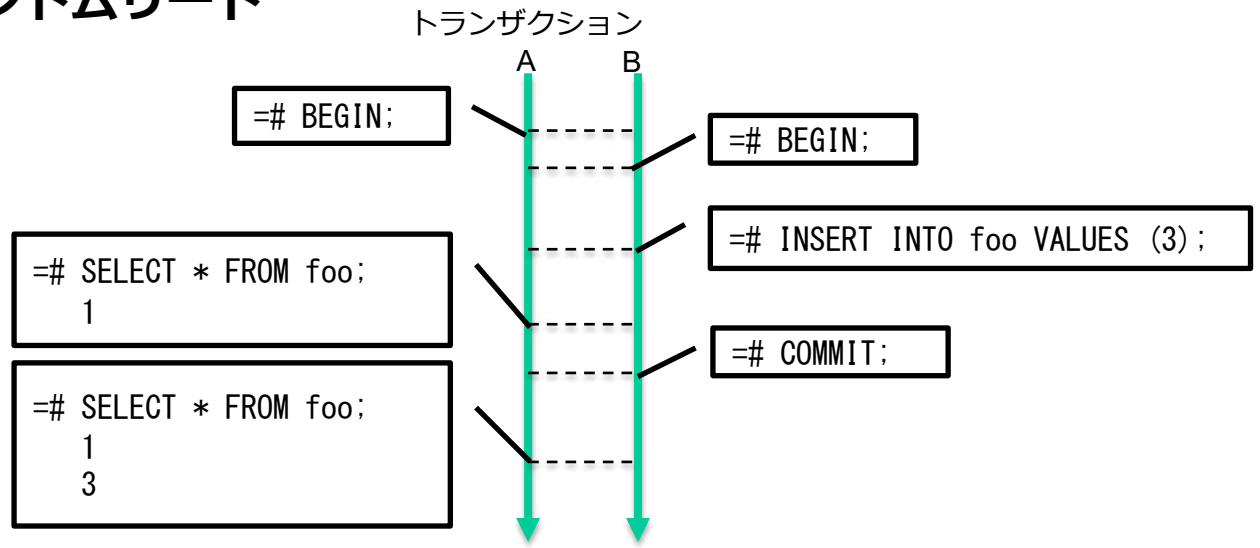
- 反復不能読み取り（ファジーリード）
  - トランザクションが反復して同じデータを読み込むとき、別のトランザクションがそのデータを更新してCOMMITしたために、同じデータを読んでいるにもかかわらず、異なる結果を得てしまうこと。
- ファントムリード
  - トランザクションがある検索条件で問合せを再実行したとき、別のトランザクションが挿入をしてCOMMITしたために、同じ検索条件にもかかわらず、異なる結果を得てしまうこと。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/transaction-iso.html

反復不能読み取り

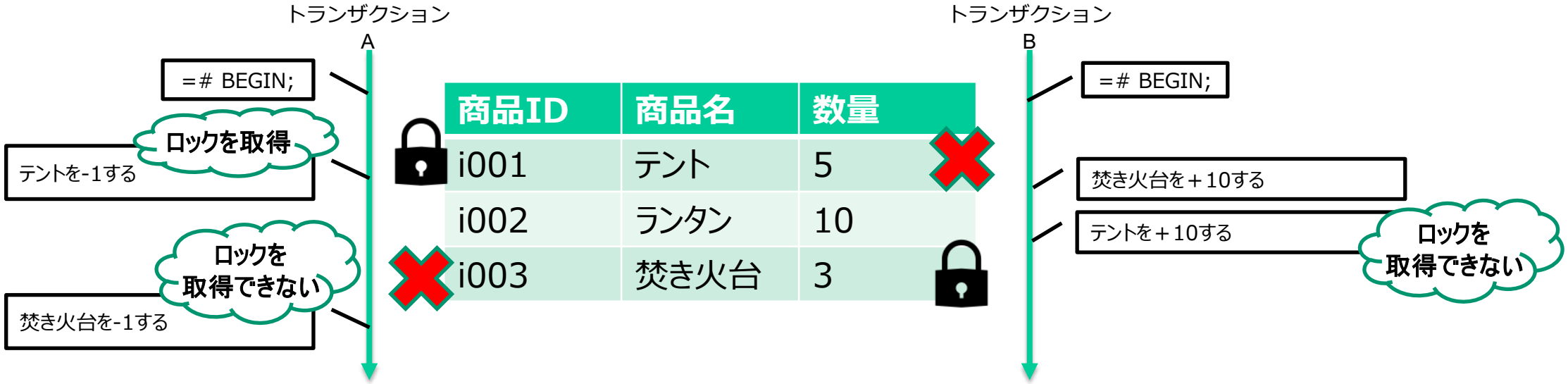


ファントムリード



デッドロック

■複数のトランザクションにおいて、お互いが相手の取得しているロック開放待ちとなり、どちらの処理も進まなくなっている状態を指す。



■PostgreSQLではデッドロックを検知する機構を有しており、「deadlock_timeout」というパラメータで制御される。

- デッドロックを検知した場合、問題となっているトランザクションのいずれかが自動でキャンセルされる。

```
ERROR:  deadlock detected
```

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/explicit-locking.html
参考：LPI-JAPAN OSS-DB
https://oss-db.jp/dojo/dojo_12

サンプル問題（トランザクションの概念）

■OSS-DB Silver　サンプル問題/例題解説　から本テーマに関するものをいくつか紹介します。

- 本セミナーでは1題のみ、時間を取って実施していただき、その後解説を行います。
- 時間：2分間

サンプル問題　3.72

■以下の一連のSQL文を実行した後、テーブルtestには何行のデータがあるか。行数を答えよ。

```
CREATE TABLE test (id INTEGER, val TEXT);
INSERT INTO test(id, val) VALUES (1,'aaa'),(2,'bbb');
BEGIN;
INSERT INTO test(id, val) VALUES (3,'ccc'),(4,'ddd'),(5,'eee');
SAVEPOINT S1;
UPDATE test SET val = 'CCC' WHERE id = 3;
SAVEPOINT S2;
DELETE FROM test WHERE id IN (2,4);
ROLLBACK TO S2;
INSERT INTO test(id, val) VALUES(6,'fff'),(7,'ggg');
COMMIT;
```

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_development_04/72_170228

サンプル問題　3.73

■**PostgreSQLのトランザクション分離レベルの説明として、適切なものを2つ選びなさい。**

A) Read Uncommittedでは、他のトランザクションが更新した後、まだcommitしていないデータを読めてしまう。

B) Read Committedでは、トランザクション内で同じSELECT文を2回続けて実行しても、異なる結果が返されることがある。

C) Repeatable Readでは、トランザクション内で同じSELECT文を2回続けて実行したら、必ず同じ結果が返される。

D) Serializableは読み取りトランザクション専用の分離レベルである。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_development_04/73_170403

サンプル問題　3.111

■以下の一連の操作を行った。最後のSELECT文がcurrvalとして返す値は何か。

```
CREATE TABLE test1(id integer, val varchar(10));
CREATE TABLE test2(id integer, val varchar(10));
CREATE SEQUENCE seq1;
INSERT INTO test1(id, val) VALUES (1, 'abc'), (2, 'def');
INSERT INTO test2(id, val) VALUES (nextval('seq1'), 'xyz');
BEGIN;
INSERT INTO test2(id, val) SELECT nextval('seq1'), val FROM test1;
ROLLBACK;
INSERT INTO test2(id, val) VALUES (nextval('seq1'), 'pqr');
SELECT currval('seq1');
```

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_development_06/111_200129

開発/SQL（SQLコマンド編）

■運用管理（52%）
- インストール方法 【重要度：2】
  - initdb
- 設定ファイル 【重要度：5】
  - postgresql.conf
  - pg_hba.conf
  - pg_settingsビュー

■開発/SQL（32%）
- トランザクションの概念 【重要度：1】
  - トランザクション分離レベル
  - デッドロック
- SQL コマンド 【重要度：13】
  - マテリアライズドビュー
  - 宣言的パーティショニング

マテリアライズドビュー

■マテリアライズドビューは、PostgreSQL 9.3に実装された。

- PostgreSQL 9.4以降、リフレッシュ中の参照処理のブロック緩和、処理のパラレル化等の機能強化が行われている。

■ビューとは

- データベースには予め定義したSQL文をテーブルのように扱うビューという機能がある。
- ビューは実際のデータを持っているわけではなく、問い合わせでビューが参照される度に、指定された問い合わせが行われ、あたかも通常のテーブルのように扱うことができる。
- ビューは以下のようなSQL文で定義することができる。

```
postgres=# CREATE VIEW comedies AS SELECT * FROM films WHERE kind = 'Comedy';
```

  - comediesビューに対してSELECTを実行した場合、AS以降のSELECT文が実行される。

- クエリを擬似的なテーブルとして扱うことができるので、複雑なSQL等を毎回書く必要がない。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/rules-views.html

マテリアライズドビュー

■マテリアライズドビューとは
- 実体（データ）を持ったビュー。
- ビューとは異なり、毎回SQLが実行されることがないので、高速に結果を返すことが可能。

■利用方法
- マテリアライズドビューの作成

```
postgres=# CREATE MATERIALIZED VIEW comedies AS SELECT * FROM films WHERE kind = 'Comedy';
```

- リフレッシュ

```
postgres=# REFRESH MATERIALIZED VIEW comedies;
```

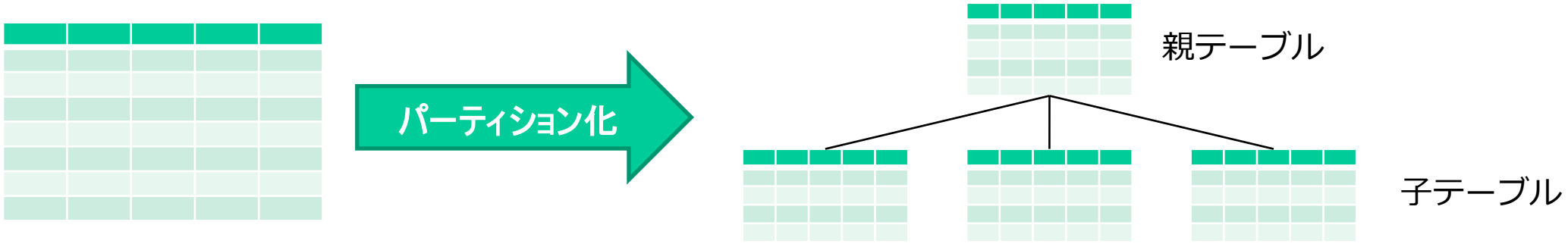
■利用する上で注意が必要な点
- 取得元とは別にデータを保有することになるので、データが陳腐化する場合がある。
- リフレッシュ処理には時間が掛かる。
  - 古くなった部分のデータだけをリフレッシュする機能はなく、全データがリフレッシュされるため、データの規模によっては多くの時間が掛かる。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/rules-materializedviews.html

宣言的パーティショニング

■パーティショニングとは

- 大規模なテーブルを小さくて管理しやすいテーブルに分割して格納する手法。



- パーティショニングすることで、性能改善やテーブルのメンテナンス等にメリットがある。
  - SQL処理でアクセスするデータ量を抑えられる。（この効果により性能が改善するケースもある。）
  - TRUNCATEやDROP TABLEによりパーティションデータを高速に削除することが可能。

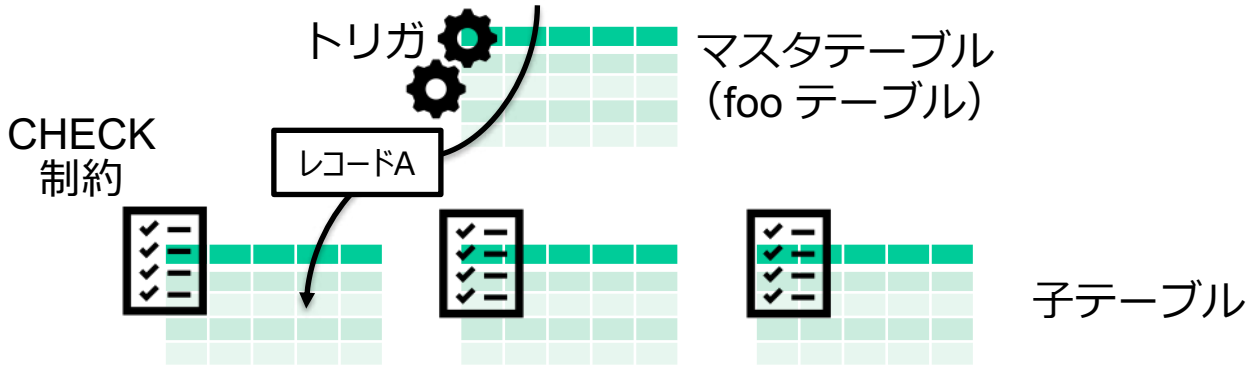
■宣言的パーティショニングとは

- PostgreSQL 10で実装された機能。以前は継承、トリガ機能を用いたパーティショニング方式のみだった。
- 宣言的になったことで、トリガや関数の作成が不要になり、CREATE TABLE文だけでパーティション構成を作成することが可能になった。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/ddl-partitioning.html

補足：従来型パーティション方式（継承、トリガ方式）

- PostgreSQL 9.6まではパーティショニングはトリガ関数やトリガ定義で挿入方式を定義し、テーブルのCHECK制約定義を組み合わせて実現していた。（PostgreSQL 10以降でも利用可能）
  - 子テーブルの作成は継承機能を利用する。
  - 親テーブルからの振り分けは自身で作り込む必要がある。（トリガ機能、CHECK制約を使用する）



トリガの定義例

```
CREATE TRIGGER insert_trigger
    BEFORE INSERT ON foo
    FOR EACH ROW EXECUTE FUNCTION foo_insert_trigger();
※トリガ関数foo_insert_trigger()は事前に定義しておく。
```

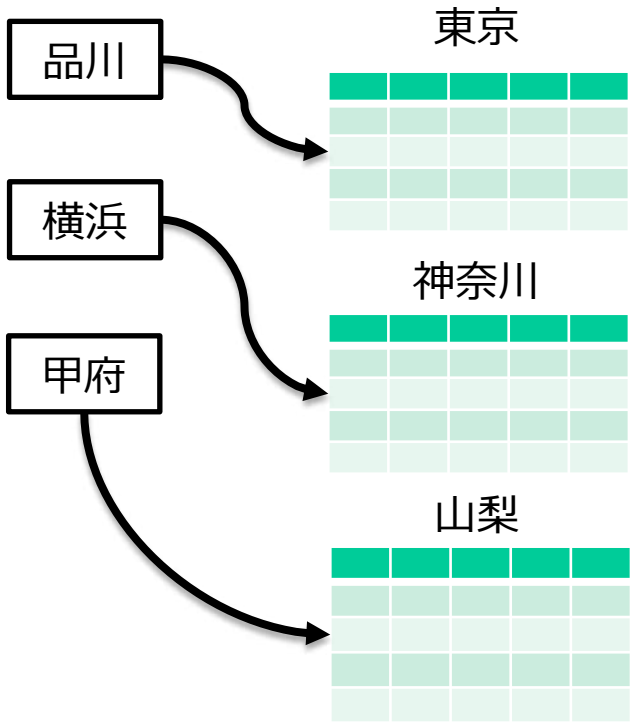
CHECK制約の定義例

```
CHECK (
    create_date >= DATE '2020-01-01'
    AND
    create_date <  DATE '2021-01-01'
)
```

利用可能なパーティション方式

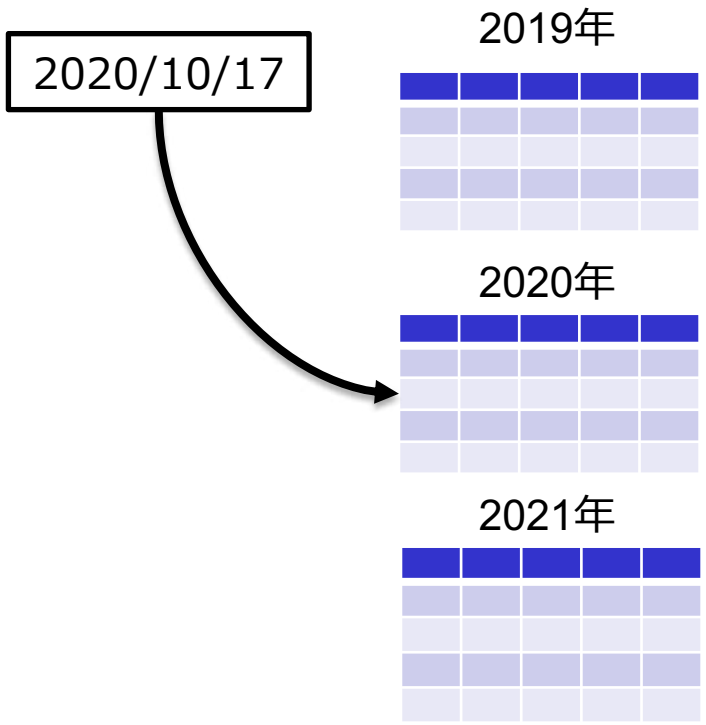
■リスト

- キー項目の値を使い、事前に指定した複数の値（リスト）から該当するパーティションに分割する。



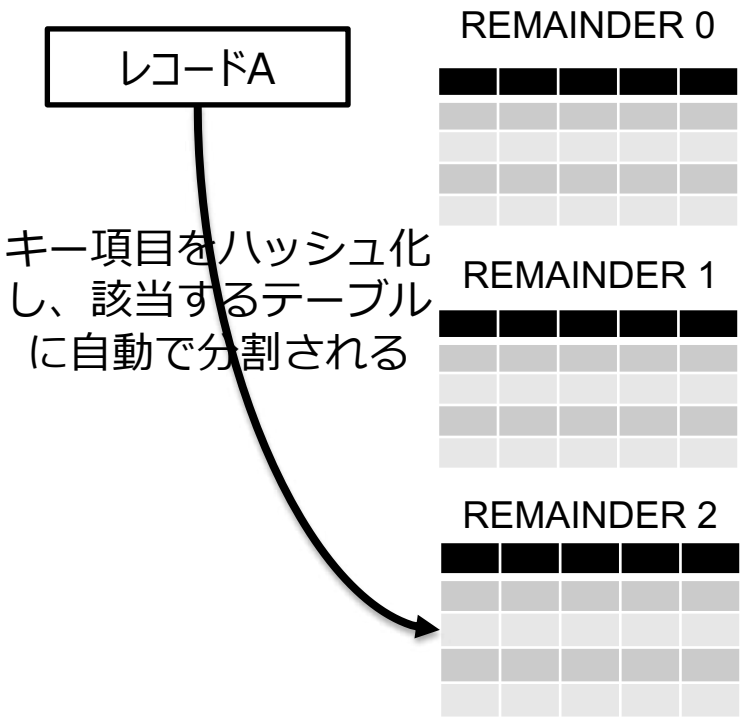
■レンジ

- キー項目の値の範囲でパーティション分割する。



■ハッシュ

- キー項目の値を用いて、指定した数のパーティションに自動で分割する。
- 概ね均等に分割される。



宣言的パーティショニングの作成方法１

■リストパーティションのサンプル

- 親テーブル作成

```
CREATE TABLE
    japan(
        pref text not null,
        population int
    )
PARTITION BY LIST (pref);
```

- 子テーブル作成

```
CREATE TABLE japan_tokyo PARTITION OF japan FOR VALUES IN ('tokyo');
CREATE TABLE japan_kanagawa PARTITION OF japan FOR VALUES IN ('kanagawa');
CREATE TABLE japan_yamanashi PARTITION OF japan FOR VALUES IN ('yamanashi');
```

■宣言的パーティショニングを使用すると、子テーブルのカラムは親テーブルの内容が引き継がれて、自動で作成される。

- 親テーブルと全く同じ構成である必要があるため、子テーブルだけ個別にカラムを追加することはできない。

参考：PostgreSQL文書
https://www.postgresql.jp/document/11/html/sql-createtable.html

宣言的パーティショニングの作成方法２

■レンジパーティションのサンプル

- 親テーブル作成

```
CREATE TABLE
    log(
        id int ,
        create_date date not null,
        num int
    )
PARTITION BY RANGE (create_date);
```

- 子テーブル作成

```
CREATE TABLE log_2019 PARTITION OF log FOR VALUES FROM ('2019-01-01') TO ('2020-01-01');
CREATE TABLE log_2020 PARTITION OF log FOR VALUES FROM ('2020-01-01') TO ('2021-01-01');
CREATE TABLE log_2021 PARTITION OF log FOR VALUES FROM ('2021-01-01') TO ('2022-01-01');
```

■リスト、レンジパーティションで範囲外のデータを挿入した場合はエラーとなる

- ただし、PostgreSQL11以降では、デフォルトパーティションを定義することができる。

```
CREATE TABLE log_other PARTITION OF log DEFAULT;
```

宣言的パーティショニングの作成方法３

■ハッシュパーティションのサンプル

- 親テーブル作成

```
CREATE TABLE
    staff(
        staff_id int not null,
        name text not null,
        age int
    )
PARTITION BY HASH (staff_id);
```

- 子テーブル作成

```
CREATE TABLE staff_a PARTITION OF staff FOR VALUES WITH (MODULUS 3, REMAINDER 0);
CREATE TABLE staff_b PARTITION OF staff FOR VALUES WITH (MODULUS 3, REMAINDER 1);
CREATE TABLE staff_c PARTITION OF staff FOR VALUES WITH (MODULUS 3, REMAINDER 2);
```

■キー項目をUPDATEすることはできるか？実行したら、どうなる？

- PostgreSQL11以降では、実行可能。更新後、適切なパーティションに移動される。
- PostgreSQL10ではエラーとなる。

パーティショニング利用に関して

■パーティション化しても、性能が改善するとは限らない

- 条件によっては、データ量は多くても元の巨大なテーブルのままでインデックスを使用した処理のほうが高速な場合もある。
- 使用する場合、想定する処理内容に加え、運用方法についても検討して決定する。

■バージョンによって制約内容が異なるので、なるべく最新のバージョンを使用する

- 基本的に最新のPostgreSQLを利用することで制限を最小限に利用することができる。
- 最新のPostgreSQLでは多くの改善が行われており、性能に関しても向上している。
  - パーティションプルーニング
    - 必要な子テーブルのみスキャンを行い、他の子テーブルへのアクセスをスキップする機能
  - Partitionwise Join
    - パーティション化された子テーブル同士を結合する機能

サンプル問題（SQLコマンド）

■OSS-DB Silver　サンプル問題/例題解説　から本テーマに関するものをいくつか紹介します。

- 本セミナーでは1題のみ、時間を取って実施していただき、その後解説を行います。
- 時間：2分間

サンプル問題　3.105

■ **PostgreSQLのバージョン10からサポートされた宣言的パーティショニングと、従来のバージョンからサポートされている継承を利用したパーティショニングの説明として、適切なものを３つ選びなさい。なお、親テーブルの名前は parent、パーティションの名前は partition1, partition2, partition3… だとします。**

A)　どちらの方法でも、親テーブルの作成は通常の CREATE TABLE parent… で行い、パーティションを使うための特別なオプション指定は必要ない。

B)　パーティションを作成するときのコマンドは、宣言的パーティショニングでは
　CREATE TABLE partition1 PARTITION OF parent…
継承を利用する場合は
　CREATE TABLE partition1 () INHERITS (parent)
で、どちらもCREATE TABLEを使用する。

C)　どちらの方法でも、INSERT文の実行時にデータの格納先となるパーティションが自動的に作成することはなく、事前にパーティションを作成しておく必要がある。

D)　どちらの方法でも、親テーブルとパーティションを適切なCREATEコマンドで作成しておけば、
　INSERT INTO parent…
で挿入するデータは自動的に適切なパーティションに格納される。

E)　宣言的パーティショニングではすべてのパーティションは同じ列を持っていて、異なる列を追加できないが、継承を利用したパーティショニングでは、パーティションごとに異なる列を追加できる。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■ 引用元
- https://oss-db.jp/sample/silver_development_06/105_190731

サンプル問題　3.98

■**マテリアライズド・ビューと通常のビューに関する説明として、正しいものを２つ選びなさい。**

A) どちらも CREATE VIEW コマンドで作成する。
B) ビューの定義が同じなら、一般的に、マテリアライズド・ビューからの検索の方が高速である。
C) ビューの定義が同じなら、どちらからSELECTしても、必ず同じ結果が返される。
D) マテリアライズド・ビューは、それに含まれるデータの量に従ってディスク容量を必要とするが、通常のビューはディスク容量を必要としない。
E) ビューを定義する問い合わせを変更する場合、通常のビューは CREATE OR REPLACE VIEW コマンドを使うのに対し、マテリアライズド・ビューでは REFRESH MATERIALIZED VIEW という特殊なコマンドを使用する。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_development_05/98_190409

サンプル問題　3.100

■**テーブルのパーティショニングを使うことで得られる可能性のある利点として、適切なものを選びなさい。**

A) データが占めるディスク容量を削減する。
B) 問い合わせの性能を大幅に向上させる。
C) 頻繁にアクセスする行と、そうでない行を、別のディスク領域に配置する。
D) 頻繁にアクセスする列と、そうでない列を、別のディスク領域に配置する。
E) DELETEの代わりにDROP TABLEを実行して、大量データの一括削除を高速化する。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

■引用元

- https://oss-db.jp/sample/silver_development_05/100_190515

■お問い合わせ■

NTTテクノクロス株式会社　ソフト道場

https://www.ntt-tx.co.jp/products/soft_dojyo/