



OSS-DB Exam Silver 技術解説無料セミナー

2016/03/20

株式会社コーソル
技術統括 渡部 亮太



■ 渡部 亮太（わたべ りょうた）

■ Oracle Database、PostgreSQL、MySQLなど様々なデータベースに関わる技術支援ならびに技術教育に従事

■ Oracle Databaseをはじめとする各種データベース関連製品のサポートに8年以上携わる

■ 保持資格

■ OSS-DB Gold

■ ORACLE MASTER Platinum
12c / 11g / 10g

■ LPIC 301 (Linux)

■ OCP MySQL 5.6 / 5 Database Administrator

■ Oracle Exadata X3 Administrator

■ Oracle ACE

■ など

■ 各種執筆活動

■ 書籍「プロとしてのOracleアーキテクチャ入門【第2版】」

■ 書籍「プロとしてのOracle運用管理入門」

■ 寄稿・記事掲載：@IT、DB Onlineなど多数

■ 社内外における多数の講演実績あり





■ 会社概要

- 「CO-Solutions = 共に解決する」の理念のもと、Oracle技術に特化した事業を展開。心あるサービスの提供とデータベースエンジニアの育成に注力している。
- Oracle Databaseのサポート、設計、構築、DBAを提供
- 現在PostgreSQL、MySQLなどのOSS-DB領域へも事業範囲を拡大中
- 従業員数：125名（2016年3月現在）
 - 男女比=約 7:3
- 所在地：東京 千代田区(本社)、福岡

くるみんマーク取得しました！
厚生労働大臣による
「子育てサポート企業」認定



DBプロ集団は、OSS-DBで飛躍。
(株)コーソルは、社内でのOSS-DBの資格取得を推進しています

年々高まるOSS-DBのニーズに応えるため、弊社でも本年度より本格的にOSS-DBへの取り組みを開始しました。社員は大家業向まで、「全員取得」に向けた4ヶ月間の資格取得推進社内キャンペーンの経験、目標を上回るGold5名、Silver11名の合格者を出すことができました。これにより個人個人が技術面における自信を持たせたり、社員の間にDB技術のプロフェッショナル集団としての一体感も生まれたように感じます。これからの社員インフラを支えるエンジニアを育てやすく、多くの方に資格取得をお薦めいたします。

株式会社コーソル 代表取締役 金山 俊明

知識を体系立てて整理する絶好のチャンス。受験を通じて、まだ理解できていない部分や疑問点を発見しやすくなりました。講習やライブ配信も、受講生同士の質疑応答も、非常に役に立ちました。

新しい挑戦が仕事の、自分の可能性を広げる。受講でOracleを勉強している中で、異なるDB製品の違いや、OSS-DB市場の動向なども勉強になりました。

自分自身の「詳しく知りたい」を満たすために、会社からの奨励で受講することになった。試験勉強だけでなく、試験当日の緊張感も、試験結果の発表も、非常に有意義な経験でした。

DBスペシャリストを認定する資格 / OSS-DB技術者認定試験

OSS-DB
Open Source Software Database Professional Certification

Silver Gold

株式会社コーソル
OSS/Linux技術者認定試験
エルピーアイジャパン (LPI-Japan)

TEL 03-3568-4482
Fax 03-3568-4483
http://www.oss-db.jp/
E-mail info@lpi.or.jp

OSS-DB 検索



- 1. OSS-DB技術者認定試験の概要
 - 2. PostgreSQLのインストール
 - 3. PostgreSQLのアーキテクチャと初期構成
 - 4. ポイント解説:運用管理
 - 5. ポイント解説:リレーショナルデータベースの概念とSQL (抜粋)
-
- 解説のなかで以下についても触れます
 - 1. 他のRDBMS経験者が初めてPostgreSQLを使うときに理解しにくいポイント
 - 2. OSS-DB Silverの例題解説



OSS-DB技術者認定 試験の概要

- OSS-DB技術者認定試験の概要
- 出題範囲とキーワード、本セミナーの説明箇所
- 注意事項



■ Silver認定の基準

- 合格すべき試験：OSS-DB Exam Silver
- データベースの導入、DBアプリケーションの開発、DBの運用管理ができること
- OSS-DBの各種機能やコマンドの目的、使い方を正しく理解していること

■ Gold認定の基準

- 合格すべき試験：
OSS-DB Exam Silver + OSS-DB Exam Gold
- トラブルシューティング、パフォーマンスチューニングなどOSS-DBに関する高度な技術を有すること
- コマンドの出力結果などから、必要な情報を読み取る知識やスキルがあること





分類	項目	キーワード/トピック
一般知識 (20%)	OSS-DB(PostgreSQL)の一般的特徴	バークレー校POSTGRESプロジェクト、標準SQLとの対応、8.0よりWindows対応
	ライセンス	BSDライセンス(著作権表示配布必須、商用利用可、無保証) GPLライセンスとの比較(ソースコード開示義務なし)
	コミュニティと情報収集	コミュニティベースの開発と情報展開
	リレーショナルデータモデルの基本概念	テーブル、行、列、リレーション、タプル、属性、ドメイン、関係演算(選択、射影、結合)
	RDBMSの一般知識、役割	トランザクション、ACID特性
	SQL一般知識、SQL分類(DDL、DML、DCL)	DDL(CREATE TABLEなど)、DML(SELECT、UPDATE、INSERT、DELETE)、DCL(BEGIN、COMMITなど)
データベース設計、正規化	正規化の目的、第一正規形、第二正規形、第三正規形、関数従属、候補キー、主キー、非候補キー	



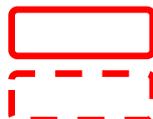
本セミナー解説で取り上げる箇所

別資料に記載(本セミナー解説では取り上げない)

→ http://www.oss-db.jp/news/event/2013/07/event_20130727_01.shtml からダウンロードできます



分類	項目	キーワード/トピック
運用管理 (50%)	インストール方法	ソースコードのコンパイル(makeコマンド)、パッケージ管理システム、データベースクラスタ、initdbコマンド、テンプレートデータベース
	標準付属ツールの使い方	psql、pg_ctl、createuser、dropuser、createdb、dropdb
	設定ファイル	postgresql.conf、pg_hba.conf、主要なパラメータ(接続関連パラメータ、ログ設定パラメータ)、パラメータ設定の確認と変更(SHOWコマンド、SETコマンド、pg_ctl reload、SIGHUP)
	バックアップ方法	pg_dump、pg_dumpall、COPYコマンド、¥copyメタコマンド、コールドバックアップ、ベースバックアップ、PITR、WALの構成
	基本的な運用管理作業	ユーザーの管理、テーブル単位の権限(GRANT、REVOKE)、インスタンスの起動・停止(pg_ctl start/stop)、バキューム、自動バキューム、プランナ統計の収集 情報スキーマ、システムカタログ



本セミナー解説で取り上げる箇所

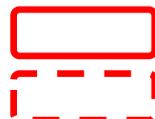


別資料に記載(本セミナー解説では取り上げない)

→ http://www.oss-db.jp/news/event/2013/07/event_20130727_01.shtml からダウンロードできます



分類	項目	キーワード/トピック
開発 /SQL (30%)	SELECT文	LIMIT、OFFSET、ORDER BY、DISTINCT、GROUP BY、HAVING、副問い合わせ、JOIN(外部結合含む)、EXIST、IN
	その他のDML	INSERT文、UPDATE文、DELETE文
	データ型	BOOLEAN、文字列、数値、時間 NULL、SERIAL、OID、配列
	テーブル定義	CREATE TABLE、制約、デフォルト値、ALTER TABLE、DROP TABLE
	その他のオブジェクト	インデックス、ビュー、ルール、トリガー、スキーマ、関数(定義、PL/pgSQL)
	組み込み関数	集約関数 (count、sum、avg、max、min)、算術関数、演算子、文字列関数 (char_length、lower、upper、substring、replace、trim) 文字列演算子 / 述語 (、~、LIKE、SIMILAR TO) 時間関数 (age、current_date、current_timestamp、等)
	トランザクショ ン	構文 (BEGIN、COMMIT、ROLLBACK、SAVEPOINTなど)、分離レベル、ロック



本セミナー解説で取り上げる箇所

別資料に記載(本セミナー解説では取り上げない)

→ http://www.oss-db.jp/news/event/2013/07/event_20130727_01.shtml からダウンロードできます



■ 最新の出題範囲

<http://www.oss-db.jp/outline/examarea.shtml>

- 出題範囲に関するFAQ → <http://www.oss-db.jp/faq/#n02>

■ 出題数、合格ライン：50問、64点

■ 前提とするRDBMSはPostgreSQL9.0以上

- PostgreSQLの最新バージョンは9.5

本資料の説明はPostgreSQL9.0をベースとします。
一部Postgres9.1以降のバージョンについて触れる箇所もあります。

■ OS固有の内容は出題されない

- ただし、OSに依存する記号や用語は、Linuxのものを使用

- OSのコマンドプロンプトには \$ を使う
- 「フォルダ」ではなく「ディレクトリ」と呼ぶ
- ディレクトリの区切り文字には / を使う

■ 基本的な出題形式

- 「適切なものを1つ(?つ)選びなさい」
- 「誤っているものを1つ(?つ)選びなさい」

問題文をちゃんと読み、
どちらのタイプの出題形式か、
まず把握しましょう



PostgreSQLの インストール

- インストール方法の分類
- yumを用いたインストール
- ワンクリックインストール
- ソースコードからのインストール



■ 1. パッケージ管理システムを使ってインストール

Linux

オススメ

■ パッケージ管理システムはディストリビューションによって異なる

- yum : Red Hat系Linux (Red Hat Linux / Fedora / Cent OS / Oracle Linux)
- apt : Ubuntu

■ インストールと管理が簡単

■ 2. ワンクリックインストーラを使ってインストール

Linux

オススメ

Windows

■ EnterpriseDB社が配布するインストーラを用いてインストール

■ インストールと管理が簡単

■ 3. ソースコードからビルドしてインストール

Linux

Windows

■ Cコンパイラとビルドツールを用いてソースコードをビルド

■ 細かいビルドオプションを設定可能

■ 慣れればさほど難しくないが、類似の作業を実施したことがない場合は敷居が高いかもしれない



- PostgreSQL9.5を想定したインストール手順を記載
 - OSユーザーpostgresで実行するコマンドは \$... として
 - OSユーザーrootで実行するコマンドは # ... として記載

- 0. インストール済みのPostgreSQLがあれば削除する
 - # yum list installed |grep postgres
 - # yum remove postgresql postgresql-libs postgresql-server
 - ※: 削除対象として指定するパッケージ名はインストール状況により異なる

- 1. yumリポジトリ設定をインストール
 - <http://yum.postgresql.org/repopackages.php> から、インストールするPostgreSQLのバージョン、Linuxディストリビューションのバージョンに対応するRPMファイルをダウンロード
 - ダウンロードしたRPMファイルをrpmコマンドでインストール
 - ⇒ /etc/yum.repos.d/にPostgreSQL用のyumリポジトリ設定がインストールされる



■ 2. yumでPostgreSQL9.5をインストール

- # yum install postgresql95-server

- 1. でインストールしたりポジトリ設定に従い、必要なファイルがインターネット上のサーバからダウンロードされ、インストールされる

- 関連ファイルは/usr/pgsql-9.5以下に配置される

- いくつかのプログラムについては/usr/binにシンボリックリンクが作成される

- OSユーザーpostgresが作成される

- .bash_profileに環境変数PGDATA設定済み

■ 3. データベースクラスタの初期化

- \$ /usr/pgsql-9.5/bin/initdb --no-locale -D /var/lib/pgsql/9.5/data

- --no-localeオプション :ロケールを使用しない(推奨)

- **[注意]** # service postgres-9.5 initdb
でもOKだが、ロケールを使用する設定となる



■ 4. (オプション) OSユーザーpostgresの環境変数設定

- 環境変数PGDATAが設定されていることを確認

- 環境変数PATHに/usr/pgsql-9.5/bin/を追加

- `$ cat ~/.bash_profile`

：

```
PGDATA=/var/lib/pgsql/9.5/data
```

```
export PGDATA
```

```
PATH=/usr/pgsql-9.5/bin/:$PATH
```

```
export PATH
```

■ 5. (オプション) PostgreSQL自動起動設定

- `# chkconfig postgresql-9.5 on`

- <http://www.postgresql.org/download/linux/redhat/>
に手順の説明(英語)がある



■ [注意]

- 1. ディストリビューション標準のPostgreSQLはバージョンが古い
 - # yum install postgresql-server
を実行すると古いバージョンのPostgreSQLがインストールされてしまう
 - 古いバージョンがインストールされていた場合、PostgreSQL9.5をインストールする前に削除する
- 2. OSユーザーpostgres が自動的に作成される
- 3. 関連ファイルは /usr/pgsql-9.5 にインストールされる
 - 主要なコマンドは /usr/binにシンボリックリンクが張られているため、コマンド名のみで起動可能
 - pg_ctlなどのコマンドはシンボリックリンクが張られていないため、コマンド名を絶対パスで指定するか、/usr/pgsql-9.5/binにPATHを設定する必要がある
 - 参考: http://lets.postgresql.jp/documents/tutorial/new_rpm
- 4. PostgreSQLサーバはOS標準のシステムサービスとして登録される
 - # service postgresql-9.5 start (起動)
 - # chkconfig postgresql-9.5 on (自動起動設定)



■ ワンクリックインストール

- インストーラをダウンロードしてインストールするだけで基本的にOK

<http://www.enterprisedb.com/products-services-training/pgdownload>

- Windows/Mac/Linuxいずれでも利用可能

- Windowsではワンクリックインストールの利用を推奨

- インストールガイド(英語)

<http://www.enterprisedb.com/resources-community/pginst-guide>

- GUIの管理ツール(pgAdmin III)も同時にインストールされる

■ インストーラの設定項目

- 基本的にデフォルト値のままで良い

- スーパーユーザー(postgres)のパスワード:任意の文字列を指定

- ロケール(Locale)設定:"Default locale"から"C"に変更することを推奨

- スタックビルダ(Stack Builder)の起動:任意

- ApacheやPHPなど、PostgreSQLと一緒に使われるソフトウェアを、インストールできる



■ インストール後の設定

■ (オプション) 環境変数PATHの設定

C:¥Program Files¥PostgreSQL¥9.5¥bin を追加する

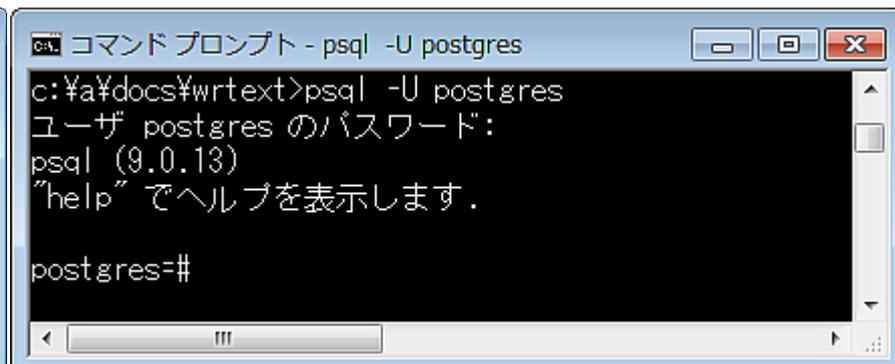
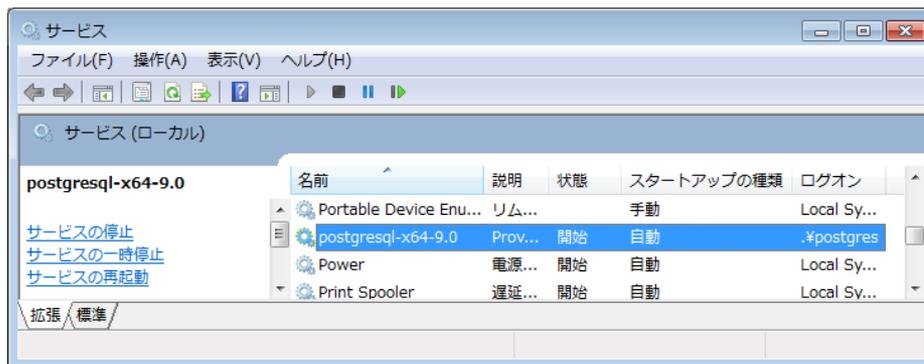
■ [注意]

■ データベースクラスタはC:¥Program Files¥PostgreSQL¥9.5¥data に作成済み

■ PostgreSQLサーバはWindowsサービスとして構成され、自動起動設定済

■ PostgreSQLユーザーpostgresのパスワードはインストーラで設定した値

■ 通常Windowsのログオンユーザー名 ≠ "postgres"であるため、postgresユーザーでPostgreSQLに接続するときは、ユーザー名を明示的に指定する必要がある





- 1. PostgreSQLの公式サイトからソースコードをダウンロード

<http://www.postgresql.org/ftp/source/>

- 2. ソースコードをビルドしてインストール

基本的には以下のコマンドを実行すればOK

```
$ ./configure
```

```
$ make(あるいは $ make world)
```

```
# make install (あるいは # make install-world)
```

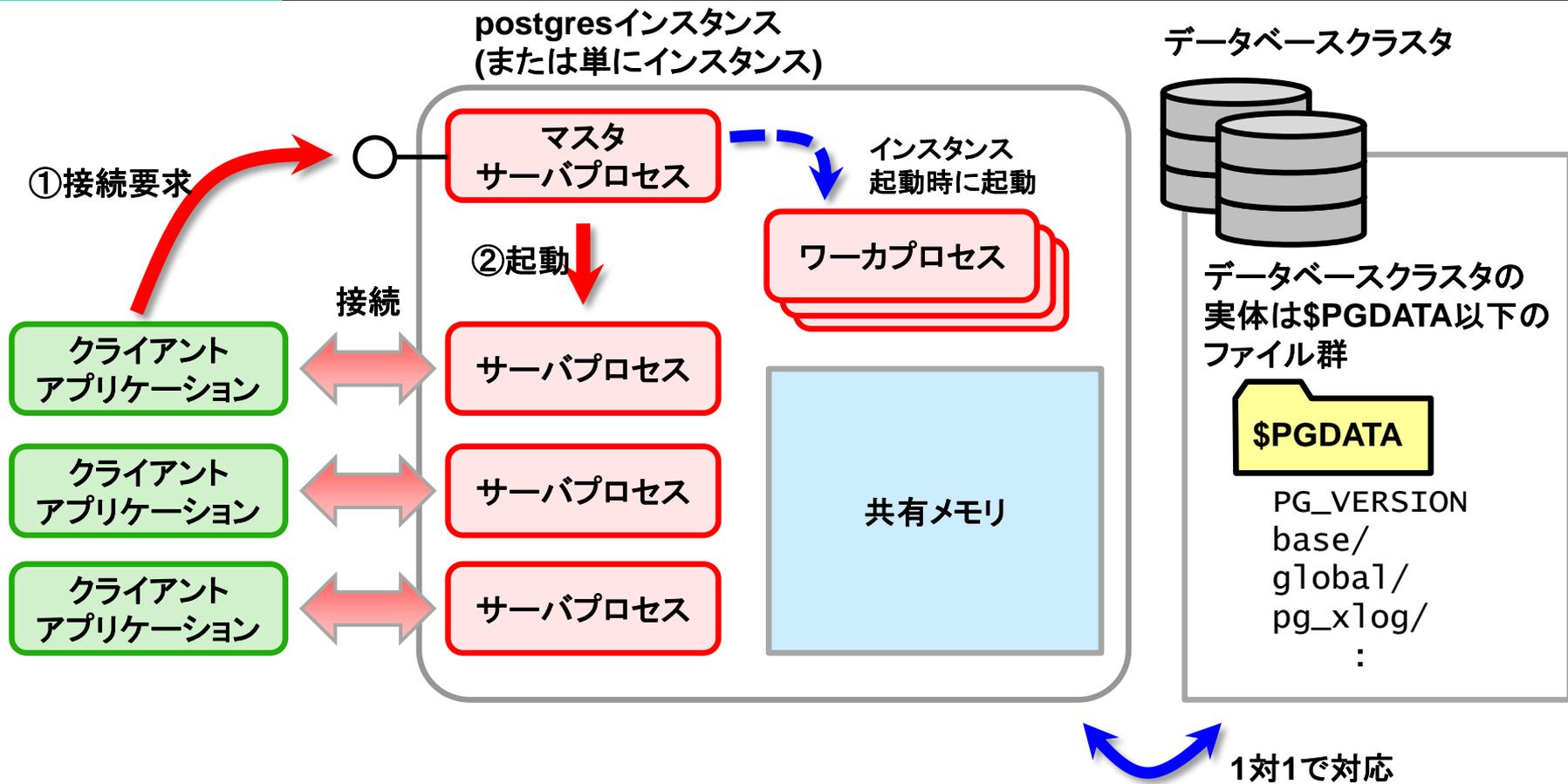
- インストールの手順は、オンラインマニュアル

<http://www.postgresql.jp/document/9.5/html/> の15章
(Linux)、16章(Windows)に解説されている



PostgreSQLの アーキテクチャと初期構成

- PostgreSQLのアーキテクチャ
- データベースクラスタの初期化
- インスタンスの起動と停止
- インスタンスへの接続
- psqlユーティリティ
- psqlのメタコマンド

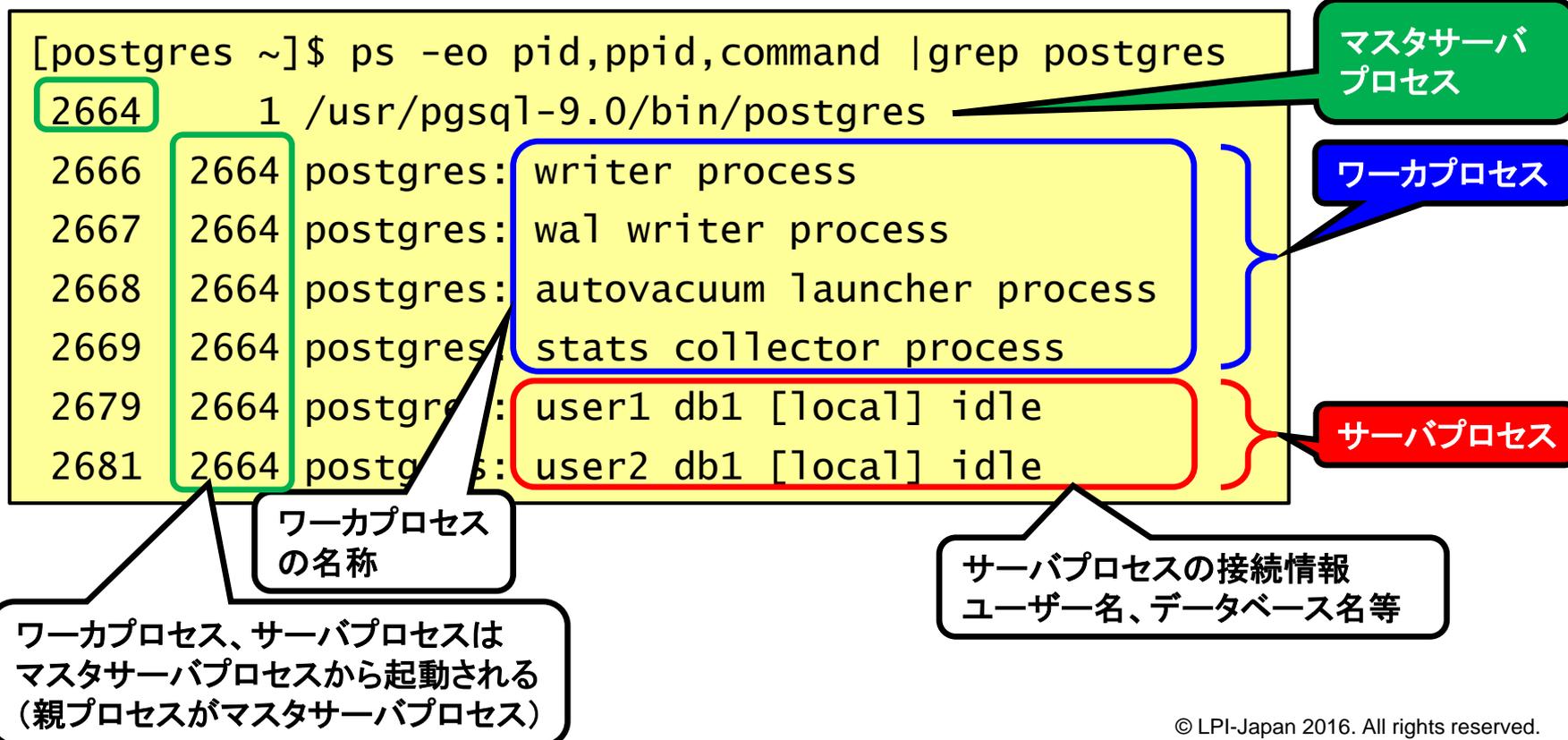


postgresインスタンス	特定のデータベースクラスタの処理を実現するプロセス群と共有メモリ 1つのpostgresインスタンスがある1つのデータベースクラスタに対応する
データベースクラスタ	データベースが配置される特定のディレクトリ以下の領域 通常、そのディレクトリパスを環境変数PGDATAに設定する



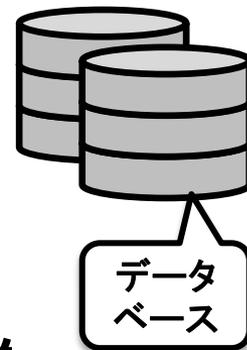
■ PostgreSQLはマルチプロセスアーキテクチャ

- ワーカープロセス、サーバプロセスはマスタサーバプロセスから起動される
- 多くのワーカープロセスは常時起動し、定期的にプロセス固有のタスクを実行する
- 1つの接続に対して1つのサーバプロセスが起動する





データベースクラスタ



■ データベースクラスタとは

- データベース「クラスタ」= 複数のデータベースの集合
- デフォルトでpostgres, template0, template1というデータベースが含まれる

■ データベースクラスタの実体とPGDATA環境変数

- データベースクラスタの実体は、あるディレクトリ以下のファイル群
- そのディレクトリパスを通常PGDATA環境変数に設定する
 - PGDATA環境変数を設定しておくと、多くのコマンドでデータベースクラスタの指定を省略できる

```
[postgres ~]$ echo $PGDATA
/var/lib/pgsql/9.0/data
[postgres ~]$ ls -F $PGDATA
base/          pg_log/       pg_tblspc/    postmaster.opts
global/       pg_multixact/ pg_twophase/  postmaster.pid
pg_clog/      pg_notify/    PG_VERSION
pg_hba.conf   pg_stat_tmp/  pg_xlog/
pg_ident.conf pg_subtrans/  postgresql.conf
```



- データベースクラスタを作成するOSコマンド
- 同時にスーパーユーザー権限を持つPostgreSQLユーザーを作成する
 - デフォルトでinitdbを実行したOSユーザーと同じユーザー名
 - 通常OSユーザー"postgres"でinitdbを実行するため、スーパーユーザー権限を持つPostgreSQLユーザー"postgres"が作成される
 - -U オプションを指定して、任意のユーザー名を指定できる
- 主なオプション
 - -D : データベースクラスタを作成するディレクトリ
 - -D オプションを指定しない場合は環境変数PGDATAを使用
 - initdb に限らず、他の多くのコマンドでも同様のルールが適用される
 - -E : デフォルトのエンコーディング(文字セット)
 - --locale: ロケール
 - --no-locale: ロケールを使用しない ← 一般的に推奨される
 - -U : スーパーユーザー権限を持つPostgreSQLユーザー名



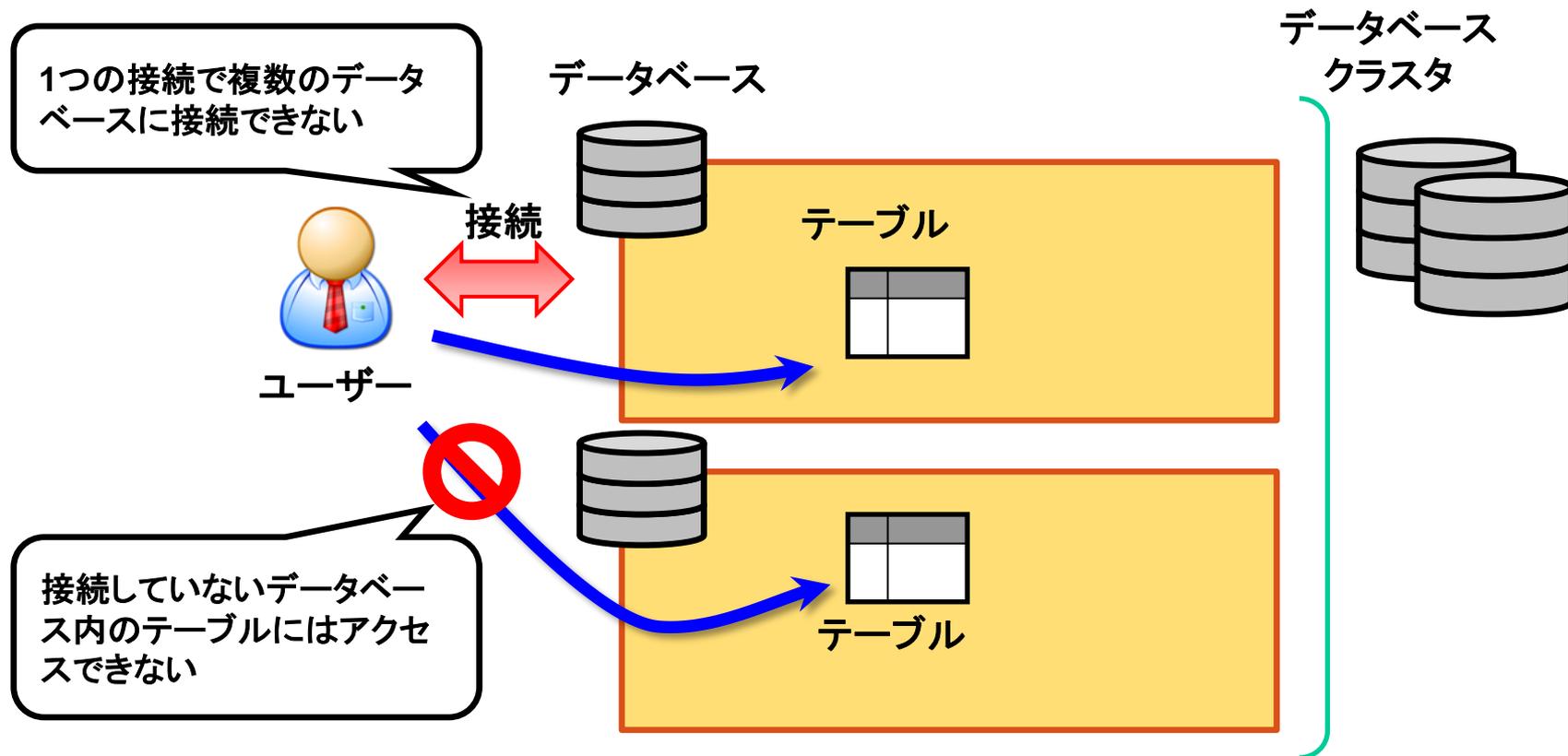
```
[postgres ~]$ echo $PGDATA
/var/lib/pgsql/9.0/data
[postgres ~]$ initdb --no-locale -E UTF8
:
The database cluster will be initialized with locale C.
:
Success. You can now start the database server using:

    postgres -D /var/lib/pgsql/9.0/data
or
    pg_ctl -D /var/lib/pgsql/9.0/data -l logfile start
```

- **--no-locale**を指定しているため、ロケールを使用しません
- **-E UTF8**を指定しているため、デフォルトのエンコーディングはUTF8です
- **-D** オプションを指定していないため、環境変数PGDATAが示すディレクトリにデータベースクラスタを作成します
- **-U** オプションを指定していないため、**initdb**を実行したOSユーザーのユーザー名"postgres"でスーパーユーザー権限を持つPostgreSQLユーザーを作成します



- 特定のデータベースクラスタに含まれる
- ユーザーの接続先、オブジェクト(テーブル、インデックスなど)の格納先となる
- 各データベースは独立性が高い





- initdbコマンド(またはpg_ctl initdbコマンド)でデータベースクラスタを作成する
- 作成直後の状態で、以下の3つのデータベースが存在する

データベース名	接続可否	用途
template0	不可	新規データベースを作成するときのひな形データベースに使用できる
template1	可	新規データベースを作成するときのデフォルトのひな形データベース 新規データベースに作成しておきたい追加オブジェクトを作成して行くことができる
postgres	可	好きな用途に使用できる テーブル、インデックスなどを配置できる



- データベースクラスタ内のデータベースにアクセスするには、データベースクラスタに対応するインスタンスを起動する必要がある
 - データベースクラスタとインスタンスは1対1に対応する
- `$ pg_ctl start`
- **主なオプション**
 - `-D` : データベースクラスタがあるディレクトリ
 - `-D` オプションを指定しない場合は、環境変数PGDATAを使用
 - `-w` : **起動完了を待機**
- `$ pg_ctl status`で**起動状態を確認できる**

```
[postgres ~]$ pg_ctl start -w
waiting for server to start.... done
server started
[postgres ~]$ pg_ctl status
pg_ctl: server is running (PID: 2955)
/usr/pgsql-9.0/bin/postgres
```



■ \$ pg_ctl stop

■ 主なオプション

■ -D : データベースクラスタがあるディレクトリ

■ -w : 停止完了を待機

■ -m : 停止モード

■ smart : すべてのクライアント接続が切断されるまで待機してから終了
(デフォルト)

■ fast : 実行中の処理を中断し、すべてのクライアント接続を切断してから終了

■ immediate : 実行中の処理を強制終了し、すべてのクライアント接続を切断してから終了。次回起動時にクラッシュリカバリ処理が自動実行される

■ **[注意]** Oracle Databaseのshutdownコマンドと停止モードの意味が異なる

```
[postgres ~]$ pg_ctl stop -w
waiting for server to shut down.... done
server stopped
[postgres ~]$ pg_ctl status
pg_ctl: no server running
```



- psqlユーティリティ
 - 管理コマンドやSQLを対話的に実行できる
- psqlを用いたインスタンスへの接続
 - 引数またはオプションで接続情報を指定する
- 引数で接続情報を指定した例

接続先
データベース名

PostgreSQL
ユーザー名

```
[postgres ~]$ psql db1 user1
Password for user user1:
psql (9.0.13)
Type "help" for help.
```

パスワードを入力

```
db1=> SELECT current_user, current_database();
 current_user | current_database
-----+-----
 user1        | db1
(1 row)
```



コマンドオプション	説明
-U ユーザー名 --username=ユーザー名	接続するPostgreSQLユーザーを指定 [デフォルト] OSユーザーと同名のPostgreSQLユーザー
-d データベース名 --dbname=データベース名	接続先データベースを指定 [デフォルト] OSユーザーと同名のデータベース
-h ホスト名 --host=ホスト名	接続先サーバを実行しているマシンのホスト名を指定 [デフォルト] Unixドメインソケット接続方式で接続、 WindowsではlocalhostへのTCP/IP接続
-p ポート番号 --port=ポート番号	接続先サーバが接続を待ち受けるポート番号または Unixドメインソケットファイルの拡張子を指定 [デフォルト] 5432

■ 以下のコマンドは同じ接続処理(LinuxでOSユーザーpostgresで実行)

■ UNIXドメインソケット接続方式(ファイルの拡張子は5432)

■ postgresユーザーでデータベースpostgresに接続

```
$ psql -U postgres -d postgres
$ psql --username=postgres --dbname=postgres
$ psql
$ psql -p 5432
```



- デフォルトで接続先データベース名とスーパーユーザーかどうかが表示される

プロンプトの表示例	接続先データベース	スーパーユーザーかどうか
db1=>	db1	一般ユーザー
postgres=#	postgres	スーパーユーザー
db1=#	db1	スーパーユーザー

- (参考) プロンプト文字列のカスタマイズ

```
postgres=# \echo :PROMPT1
%/%R%#
postgres=# \set PROMPT1 '%n@%m %~%R%# '
postgres@[local] ~=#
```

- 詳細は

<http://www.postgresql.jp/document/9.0/html/app-psql.html#APP-PSQL-PROMPTING>



- 1. SQL文
 - RDBMSで汎用的に使える データ、オブジェクト操作のためのコマンド
 - DML文(SELECT、UPDATE、INSERT、DELETEなど)
 - DDL文(CREATE TABLE、ALTER TABLEなど)
 - DCL文(BEGIN、ENDなど)
 - 主要なSQLについては本セミナーの後半で説明
- 2. psqlメタコマンド
 - psql独自のコマンド。主に管理情報の参照に使用される
 - 半角のバックスラッシュで始まる
 - 環境によっては'¥' (円記号)で表示される場合あり
 - 多くのコマンドが存在
 - 詳細は
<http://www.postgresql.jp/document/9.0/html/app-psql.html#APP-PSQL-META-COMMANDS>



```
[postgres ~]$ psql
```

```
:
```

```
postgres=# SELECT * FROM DEPT;
```

セミicolon';'でコマンド終了

deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
(4 rows)
```

```
postgres=# SELECT *
```

コマンド途中で改行可能

```
postgres-# FROM DEPT;
```

deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
(4 rows)
```



■ ユーザーがアクセス可能なオブジェクトに関する情報を表示する

コマンド	表示対象
¥d[S+] [パターン]	テーブル、ビュー、シーケンス
¥du[+] [パターン]	ユーザー
¥dn[+] [パターン]	スキーマ
¥dt[S+] [パターン]	テーブル
¥dv[S+] [パターン]	ビュー
¥ds[S+] [パターン]	シーケンス
¥di[S+] [パターン]	インデックス
¥df[S+] [パターン]	関数

- メタコマンドに続けて"+"を指定 → 追加情報を表示
- メタコマンドに続けて"S"を指定 → システムオブジェクトも表示
- メタコマンドの引数にパターン文字列を指定 → パターンにマッチしたオブジェクトを表示



コマンド	説明
¥l	データベース一覧を表示
¥c	別のデータベースまたは別のユーザーでインスタンスに接続
¥x [on off]	問合せ結果の表示形式を変更する
¥i <ファイル名>	ファイルに記録されたコマンドを実行
¥o <ファイル名>	実行結果をファイルに出力
¥timing [on off]	SQLの実行時時間を表示
¥! <OSコマンド>	OSのコマンドを実行 コマンドを指定しなかった場合、シェルを実行
¥?	psqlメタコマンドのヘルプ
¥h <SQLコマンド>	SQLコマンドのヘルプ
¥q	psqlを終了する



■ テーブル、ビュー、シーケンスを表示する

db1=> ¥d

List of relations

Schema	Name	Type	Owner
public	dept	table	user1
public	emp	table	user1

(2 rows)

末尾にセミicolon';'は不要

db1=> ¥d+

'+'指定で追加情報を表示

List of relations

Schema	Name	Type	Owner	Size	Description
public	dept	table	user1	0 bytes	
public	emp	table	user1	0 bytes	

(2 rows)



- パターンに合致するオブジェクトの より詳細な情報を表示する
 - 列名、データ型、制約、インデックスなど

```
db1=> ¥d emp
                Table "public.emp"
column |          Type          | Modifiers
-----+-----+-----
empno  | numeric(4,0)           | not null
ename  | character varying(10) |
job    | character varying(9)  |
mgr    | numeric(4,0)           |
hiredate | timestamp without time zone |
sal    | numeric(7,2)           |
comm   | numeric(7,2)           |
deptno | numeric(2,0)           |
Indexes:
    "pk_emp" PRIMARY KEY, btree (empno)
Foreign-key constraints:
    "fk_deptno" FOREIGN KEY (deptno) REFERENCES dept(deptno)
```



■ システムカタログのオブジェクトを含めて表示する

```
db1=> ¥dS
```

```
List of relations
```

Schema	Name	Type	Owner
pg_catalog	pg_aggregate	table	postgres
pg_catalog	pg_am	table	postgres
pg_catalog	pg_amop	table	postgres
:			
pg_catalog	pg_user_mappings	view	postgres
pg_catalog	pg_views	view	postgres
public	dept	table	user1
public	emp	table	user1

(82 rows)



■ データベースの一覧を表示する

```
postgres=# \l
```

List of databases						
Name	Owner	Encoding	Collation	Ctype	Access privileges	
db1	postgres	UTF8	C	C	=Tc/postgres	+
					postgres=CTc/postgres+	
					user1=CTc/postgres	
postgres	postgres	UTF8	C	C		
template0	postgres	UTF8	C	C	=c/postgres	+
					postgres=CTc/postgres	
template1	postgres	UTF8	C	C	=c/postgres	+
					postgres=CTc/postgres	

(4 rows)



■ 問合せ結果の表示形式を変更する

```
db1=> SELECT * FROM tbl1;
```

id	col1
1	AAA
1	BBB

(2 rows)

1行のデータ=1行表示

```
db1=> ¥x
```

Expanded display is on.

```
db1=> SELECT * FROM tbl1;
```

```
-[ RECORD 1 ]
id      | 1
col1    | AAA
-[ RECORD 2 ]
id      | 1
col1    | BBB
```

1列のデータ=1行表示

```
db1=> ¥x
```

Expanded display is off.

```
db1=> SELECT * FROM tbl1;
```

id	col1
1	AAA
1	BBB

(2 rows)



操作および処理	PostgreSQL	Oracle Database
インストール方法	パッケージ管理システム or ワンクリックインストーラ or ソースコードよりビルド	Oracle Universal Installer
データベースクラスタの 作成 (データベースの作成)	initdbコマンド	DBCA (DataBase Configuration Assistant)
コマンドラインユーティリ ティ	psql	sqlplus (SQL*Plus)
インスタンスの起動	pg_ctl start	SQL*Plus の startup コマンド
インスタンスの停止	pg_ctl stop	SQL*Plus の shutdown コマ ンド



■ 運用管理 - インストール方法

以下の説明のうち、適切でないものを1つ選びなさい

- a. **initdbコマンドを実行すると、自動的にスーパーユーザー権限を持つ PostgreSQLユーザーが作成される**
- b. **initdbコマンドを実行すると、postgres、template0、template1という3つのデータベースが作成される**
- c. **template1データベースは、新規データベース作成時のデフォルトのひな形データベースになる**
- d. **postgresデータベースには管理用の特殊なテーブルが格納されるため、アプリケーション固有のテーブルやインデックスを格納してはいけない**

回答：？



■ 運用管理 - 標準付属ツールの使い方

user1ユーザーでdb1データベースに接続するコマンドとして、適切なものを2つ選びなさい。

- a. `psql user1 db1`
- b. `psql db1 user1`
- c. `psql -U user1 -d db1`
- d. `psql -u user1 -d db1`
- e. `psql -user user1 -dbname db1`

回答: ?, ?



ポイント解説： 運用管理

- パラメータの設定 (postgresql.conf)
- 設定の確認
- クライアント認証の設定 (pg_hba.conf)
- ユーザー管理
- ロール属性としての権限
- データベースの作成と削除
- バックアップ



- PostgreSQLには数多くのパラメータが存在する
 - PostgreSQL9.0.13 では 195、
PostgreSQL9.5.0 では 247のパラメータが存在
- パラメータの設定値を変更することで、インスタンスの動作特性を調整できる
- 設定値は `$PGDATA/postgresql.conf` に記載する
- [補足] 機能拡張
 - 9.3以降では`include_dir`パラメータによる設定ファイルの分割が可能
 - 9.4以降ではALTER SYSTEM文を用いて変更した設定値を`postgresql.auto.conf`に保管する仕組みが導入



- PostgreSQLの動作を調整するパラメータを設定するファイル
 - \$PGDATA/postgresql.conf
- postgresql.confの例

```
[postgres ~]$ cat $PGDATA/postgresql.conf
```

```
# -----  
# PostgreSQL configuration file  
# -----
```

コメント行
('#'から行末までがコメント)

```
    :  
listen_addresses = '*'  
port = 5432  
    :
```

"パラメータ名 = 値" という形式で
パラメータを設定

```
log_rotation_age = 1d  
log_rotation_size = 10MB  
    :
```

単位を示す文字列を使用できる
サイズ: kB, MB, GB
時間: ms, s, min, h, d



- OSS-DB Silverの試験対策として
 - 試験で問われるのは、以下の4つ
(数字はバージョン9.0のマニュアルの節番号)
 - 記述方法(18.1)
 - 接続と認証(18.3)
 - クライアント接続デフォルト(18.10)
 - エラー報告とログ取得(18.7)



■ マニュアルの18.3節(接続と認証)を参照

■ listen_addresses

- クライアントからのTCP/IP接続を待ち受けるIPアドレス(またはホスト名)を指定
- '*' を指定すると、全IPアドレスから接続可能となる
- 複数のIPアドレスを指定可能 (カンマ区切りで記載、複数NIC構成で使用)

パラメータ型	文字列
デフォルト値	'localhost'
設定変更の反映タイミング	インスタンス起動時

■ port

- クライアントからのTCP/IP接続を待ち受けるTCPポート番号を指定

パラメータ型	整数
デフォルト値	5432
設定変更の反映タイミング	インスタンス起動時



■ max_connections

■ 同時接続数の上限を指定

パラメータ型	整数
デフォルト値	100
設定変更の反映タイミング	インスタンス起動時

■ superuser_reserved_connections

■ max_connections のうち、スーパーユーザー用に予約する接続数を指定

■ 一般ユーザーの最大同時接続数 = max_connections - superuser_reserved_connections

パラメータ型	整数
デフォルト値	3
設定変更の反映タイミング	インスタンス起動時



- マニュアルの18.10節(クライアント接続デフォルト)を参照
- search_path
 - スキーマ検索パスを設定

パラメータ型	文字列
デフォルト値	'"\$user", public'
設定変更の反映タイミング	任意



■ スキーマとは

- オブジェクト(テーブル、インデックスなど)の論理的なコンテナ
 - すべてのオブジェクトはいずれか1つのスキーマに含まれる

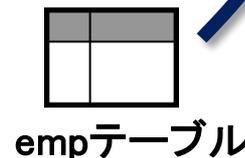
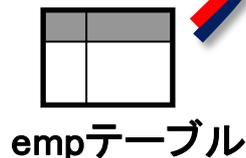
■ データベースとスキーマ

- すべてのデータベースにpublicというスキーマが初期状態で作成済み
- 1つのデータベースに複数のスキーマを作成できる

■ スキーマはオブジェクト名の名前空間として機能する

- 同じスキーマに、同じ名前のオブジェクトを作成できない
- 「スキーマ名.オブジェクト名」がオブジェクトの完全修飾名

データベース





■ search_pathパラメータの役割

■ 1. オブジェクトの検索順序

- スキーマを指定せずにオブジェクトを参照した場合、どのスキーマから検索するか

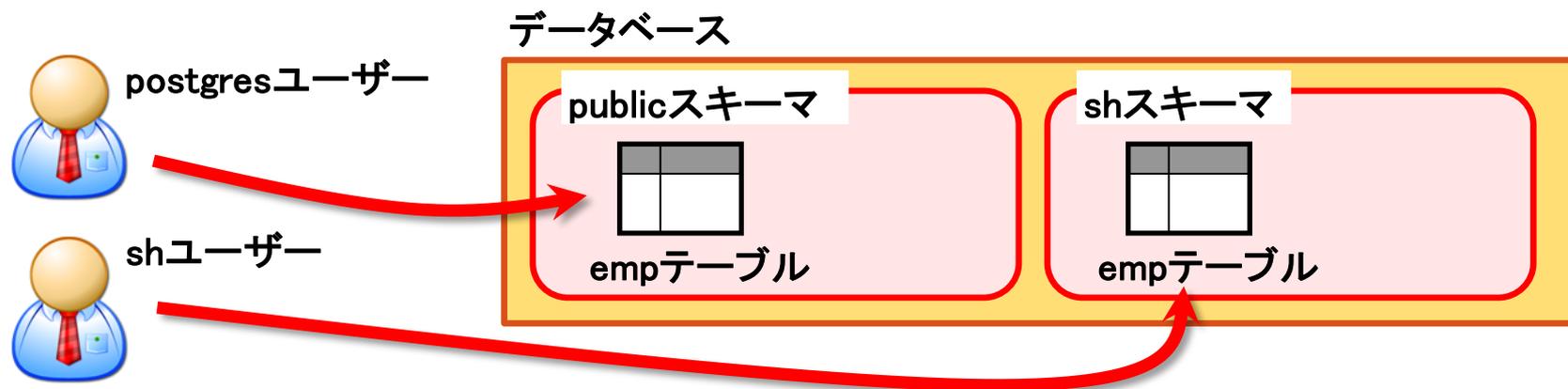
■ 2. デフォルトのオブジェクト格納先スキーマ

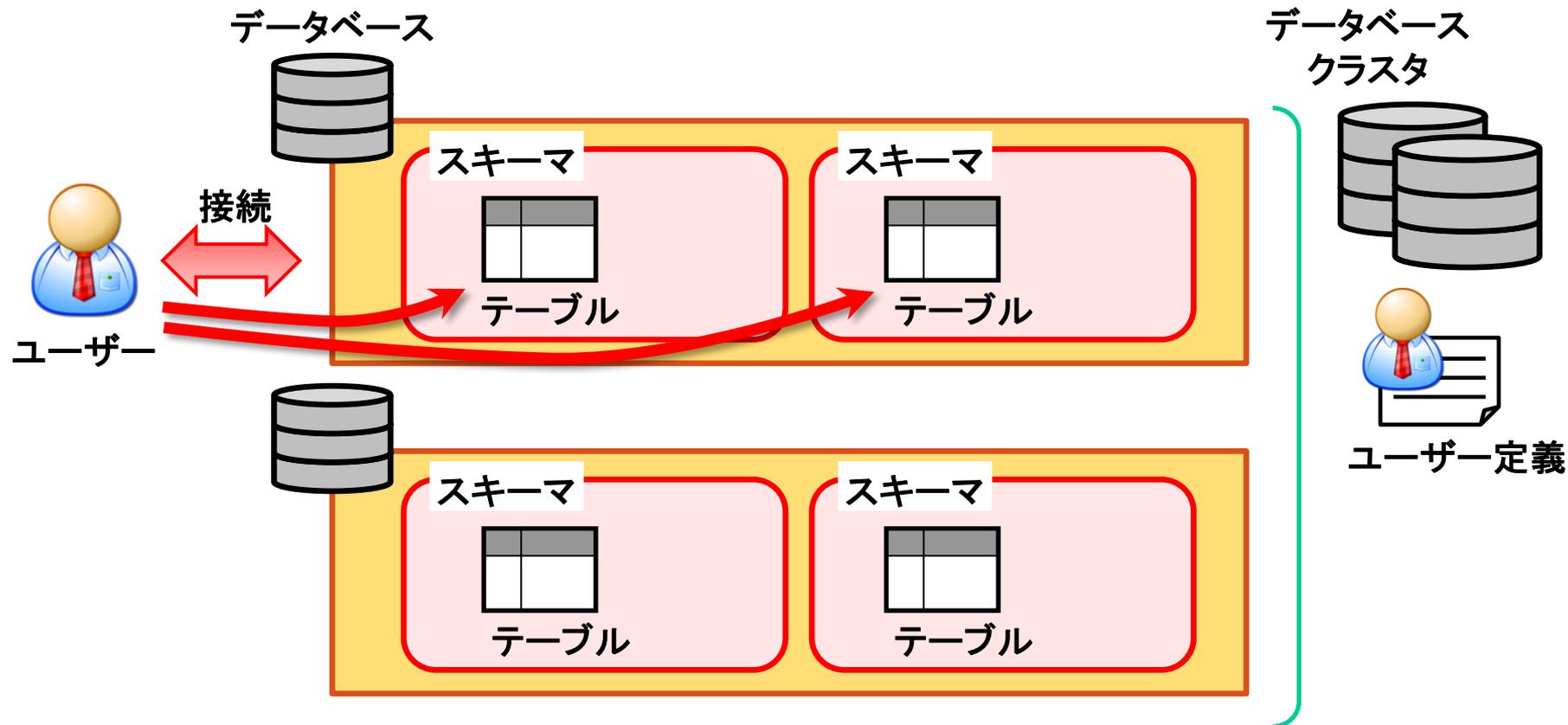
- スキーマを指定せずにオブジェクトを作成した場合、どのスキーマに格納するか

■ search_pathパラメータのデフォルト値「"\$user", public」の意味

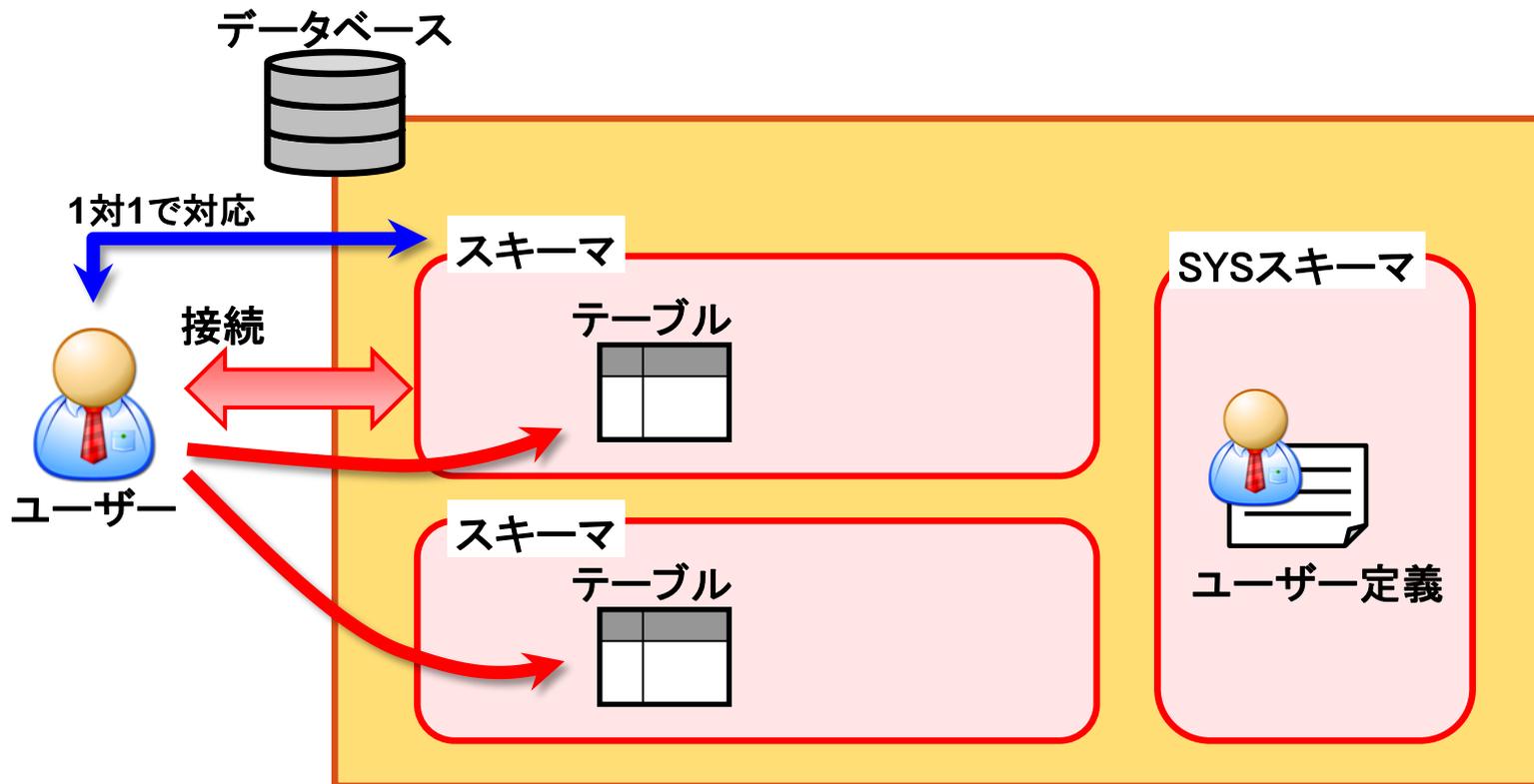
- 接続ユーザー名と同名のスキーマを検索し、なければpublicスキーマから検索する

■ 例) SELECT * FROM emp; 実行時

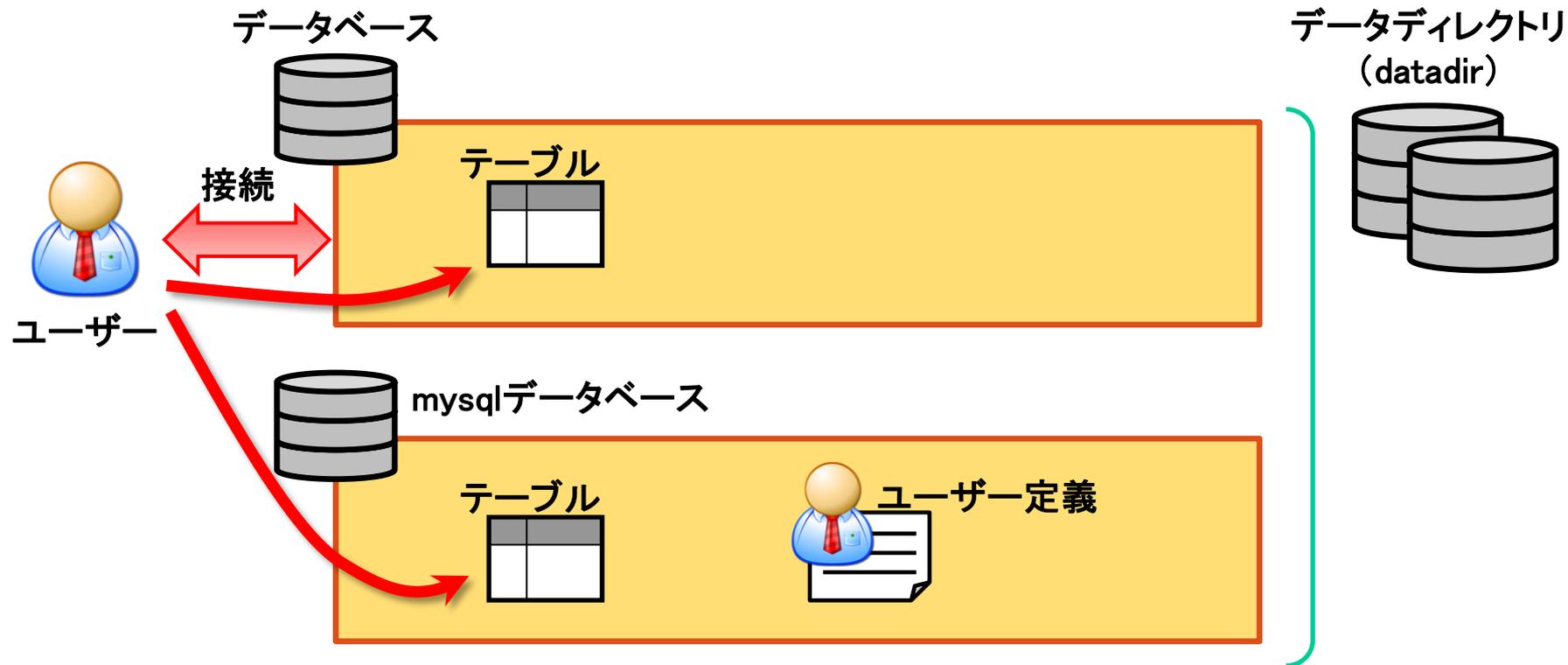




- データベースクラスタ内に1つ以上のデータベースが存在
- データベース内に0つ以上のスキーマが存在
- オブジェクトはある特定のスキーマに存在
- ある接続の接続先はある特定のデータベース
- ユーザー定義はデータベースクラスタが保持



- 複数のデータベースをまとめた概念は基本的でない
(強いて言えば 12c新機能 マルチテナントアーキテクチャが相当)
- ユーザーと同名のスキーマ名が必ず存在し、そのスキーマに接続する
- オブジェクトは所有者ユーザーのスキーマに存在
- ユーザー定義はデータベースのSYSスキーマに格納される



- 複数のデータベースが存在するが、データベースクラスターに相当する用語はない（強いて言えば「データディレクトリ」「datadir」が対応する）
- スキーマやオブジェクトの所有者という概念はない
- ある接続の接続先はある特定のデータベースになるが、別のデータベースのオブジェクトにもアクセス可能（デフォルトのスキーマを選択している感覚に近い）
- ユーザー定義はmysqlデータベースという特殊なデータベースが保持



■ timezone

■ タイムスタンプ解釈用の時間帯

パラメータ型	文字列
デフォルト値	'unknown' (システム環境を参照)
設定変更の反映タイミング	任意

■ client_encoding

■ クライアント用の符号化方式(文字エンコーディング)を指定

パラメータ型	文字列
デフォルト値	データベースの符号化方式と同じ
設定変更の反映タイミング	任意



- マニュアルの18.7節(エラー報告とログ取得)を参照
- log_destination
 - ログの出力形式と出力先を指定
 - 複数指定可能(カンマ区切りで指定)

パラメータ型	文字列
デフォルト値	'stderr'
設定変更の反映タイミング	postgresql.confの再読み込み

■ 設定値と出力形式、出力先

設定値	出力形式	出力先
stderr	テキスト形式	標準エラー出力
csvlog (要logging_collector=on)	CSV形式	標準エラー出力
syslog	テキスト形式	syslog
eventlog (Windows環境のみ)	イベントログ形式	Windowsイベントログ



■ logging_collector

- on に設定すると log_destination=stderr/csvlog のログ出力をファイルへリダイレクトする
 - ワーカープロセス logger processが起動し、ログ収集処理を実行する
- log_destination=csvlogを指定した場合はonにする必要がある

パラメータ型	論理値
デフォルト値	off
設定変更の反映タイミング	インスタンス起動時

■ log_filename

- logging_collectorによりログ出力をリダイレクトするファイル名を指定

パラメータ型	文字列
デフォルト値	postgresql-%Y-%m-%d_%H%M%S.log (log_destination=csvlogの場合拡張子がcsvとなる)
設定変更の反映タイミング	postgresql.confの再読み込み



■ log_line_prefix

■ ログ各行の先頭に付加する文字列を書式文字列を使って指定

パラメータ型	文字列
デフォルト値	'(空文字)' ←何も付加しない
設定変更の反映タイミング	postgresql.confの再読み込み

書式指定文字列	出力内容
%t	タイムスタンプ
%u	ユーザー名
%d	データベース名
%p	プロセス名
%c	セッションID
%x	トランザクションID

■ 原則的に%t(時刻)、%p(プロセスID)は必須

■ **[注意]** 設定値の末尾に空白文字をいれないと、付加する文字列とログ本体がくっついて読みにくい



■ log_rotation_age

- ログローテーション間隔を指定
- 0を指定すると間隔ベースのログローテーションが無効化される

パラメータ型	時間を示す数値(デフォルトは分単位)
デフォルト値	1d(1日)
設定変更の反映タイミング	postgresql.confの再読み込み

■ log_rotation_size

- ログファイルの最大サイズを指定
- ファイルサイズが最大サイズに達するとログローテーションが実行される
- 0を指定すると最大サイズベースのログローテーションが無効化される

パラメータ型	サイズを示す数値(デフォルトはkB単位)
デフォルト値	10MB
設定変更の反映タイミング	postgresql.confの再読み込み



- `shared_buffers`
 - 共有メモリバッファのサイズ、デフォルトは32MB
 - RAMが1GB以上あるシステムでの推奨サイズはシステムメモリの25%
- `checkpoint_segments`
 - このパラメータで指定した個数のWALファイル(トランザクションログ、16MB)が書き出されると、自動的にチェックポイントが発生する
 - デフォルトは3
 - 10以上が推奨、更新が多いシステムでは大きめ(32以上)にする
- `wal_buffers`
 - WAL出力に使われるバッファのメモリサイズ
 - デフォルトは64kB (PostgreSQL 9.0まで)
 - PostgreSQL 9.1ではデフォルトが変更、`shared_buffers`の1/32とWALファイルのサイズ(16MB)の小さい方



- wal_level
 - WALに書き出す情報の種類を指定
 - 値は、minimal(default), archive, hot_standby
 - ログアーカイブ(PITR)を使うには archive または hot_standby に設定
- archive_mode
 - ログアーカイブを使うには on に設定
- archive_command
 - WALファイルの退避に使うシェルコマンド
 - 例:archive_command = 'cp %p /mnt/pg-arch/%f'
- archive_timeout
 - WALファイルが一杯にならなくても(16MBに達しなくても)強制的にアーカイブさせる(次のWALファイルに切り替える)までの時間を秒数で指定
 - デフォルトは0(強制切り替えしない)



■ 1. SHOW コマンド

■ 特定のパラメータの設定値を確認

=> SHOW パラメータ名;

■ 全パラメータの設定値を確認

=> SHOW ALL;

■ 2. pg_settingsビュー

■ => SELECT * FROM pg_settings;

■ pg_settingsビューは、システムカタログと呼ばれる内部情報にアクセスできる特殊なオブジェクトの1つ

■ 通常のテーブルと同様にSELECT文でアクセスできる

■ WHERE句で表示項目を限定できる

=> SELECT name, setting FROM pg_settings
WHERE name LIKE '%vacuum%';



変更方法	変更対象	実行例
SETコマンド	現行セッション (または現行トランザクション)	=> SET client_encoding TO 'UTF8';
postgresql.confの変更 + 再読み込み または SIGHUP送信	インスタンス全体	\$ vi \$PGDATA/postgresql.conf \$ pg_ctl reload または \$ kill -HUP <マスタサーバプロセスのpid>

- **[注意]** すべてのパラメータが起動中に変更できるわけではない
 - パラメータによっては、変更タイミングや変更可能ユーザーが制限される
 - listen_addresses : 起動中の変更不可(要インスタンス再起動)
 - log_destination : セッション単位での変更不可
 - log_statement : 一般ユーザーの変更不可(スーパーユーザーのみ)



■ クライアント接続に使用する認証方式に関する設定ファイル

■ \$PGDATA/pg_hba.conf

■ 接続種別、接続先データベース、PostgreSQLユーザー、接続元IPアドレス毎に使用される認証方式を設定する

■ 編集後に \$ pg_ctl reload で反映させる

■ pg_hba.confの例と記述形式

記載した順序で
評価される

接続種別	接続先データベース	PostgreSQLユーザー	接続元クライアントのIPアドレス範囲	使用される認証方式
# TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
local	all	postgres		md5
local	all	all		ident
host	all	all	127.0.0.1/32	trust
host	db1	all	192.168.0.0/24	reject

#以後は
コメント

どの行もマッチしない場合は接続拒否

```
[postgres ~]$ psql -U user1 -d db1
psql: FATAL: no pg_hba.conf entry for host "[local]", user "user1",
database "db1", SSL off
```



■ 主な認証方法(METHOD)

- md5 : パスワード認証、パスワードはハッシュ化され送信される
(安全)
- password : パスワード認証、パスワードは平文で送信される(危険)
- ident : OSユーザー名とPostgreSQLユーザー名が同じであれば、
無条件に接続を許可
- peer : OSユーザー名とPostgreSQLユーザー名が同じであれば、
無条件に接続を許可。9.1より導入。
ローカル接続時に使用される
- trust : 無条件に接続を許可
- reject : 常に接続を拒否



接続種別	接続先データベース	PostgreSQLユーザー	接続元クライアントのIPアドレス範囲	使用される認証方式
# TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
local	all	postgres		md5
local	all	all		ident
host	all	all	127.0.0.1/32	trust
host	db1	all	192.168.0.0/24	reject

記載した
順序で
評価される

- ①
- ②
- ③
- ④

- ① 接続種別がUNIXドメインソケット接続の場合(TYPE=local)で、PostgreSQLユーザーが"postgres"の場合、パスワード認証(METHOD=md5)で認証する
- ② 接続種別がUNIXドメインソケット接続の場合(TYPE=local)で、(PostgreSQLユーザーが"postgres"以外の場合、)OSユーザーとPostgreSQLユーザーが同じであれば無条件に接続を許可する(METHOD=ident)
- ③ 接続種別がTCP/IP接続の場合(TYPE=host)で、接続元クライアントのIPアドレスが127.0.0.1の場合、無条件に接続を許可する(METHOD=trust)
- ④ 接続種別がTCP/IP接続の場合(TYPE=host)で、接続先データベースがdb1かつ接続元クライアントのIPアドレス範囲が192.168.0.1～192.168.0.255の場合、接続を拒否する(METHOD=reject)



- 多くのパッケージのデフォルト設定で、認証方式 = ident または peer の接続設定のみ記載されている
 - OSユーザー名 = PostgreSQLのユーザー名であればデータベースに接続可
 - 通常、"postgres" PostgreSQLユーザーが作成済みであるため、OSユーザー"postgres"から、パスワードなしで接続できる

```
[postgres ~]$ id
uid=26(postgres) gid=26(postgres) groups=26(postgres)
[postgres ~]$ psql -U postgres
psql (9.0.13)
Type "help" for help.

postgres=#
```

- **[注意]** PostgreSQLユーザーを追加した場合、pg_hba.confにそのユーザー向けの接続設定を追加する必要がある
 - PostgreSQLのユーザー名 = OSユーザー名 となるOSユーザー名を追加する対処も技術的には可能



■ データベースの内部情報を格納するテーブル(およびビュー)の集合

分類	移植性	説明と例
システムカタログ (pg_???)	なし	PostgreSQL独自の情報を含む、各種情報を取得できる SELECT * FROM pg_tables;
情報スキーマ (information_schema.???)	あり	標準SQLに準拠しており、移植性が高い情報の取得方法だが、PostgreSQL独自の情報が含まれない SELECT * FROM information_schema.tables;

- 特に強い理由がない限り、システムカタログを中心に情報を取得する
 - 内部情報の収集処理には一般に移植性が求められないため



■ createuser コマンド (OSコマンド)

- `$ createuser [オプション] [ユーザー名]`

- オプションを指定しなかった場合、以下を対話的に入力する

- 新規ユーザー名

- 新規ユーザーをスーパーユーザーとするかどうか

- 新規ユーザーにデータベース作成の権限を与えるかどうか

- 新規ユーザーにユーザー作成の権限を与えるかどうか

- **[注意]** PostgreSQL 9.2では仕様が変更になり、`--interactive` オプションを指定しなければ、対話的入力を行わない

■ CREATE USER文 (SQL)

- `CREATEROLE` 権限が必要

- `=# CREATE USER ユーザー名 [オプション];`

- 対話的な入力による権限設定はできない



- dropuser コマンド(OSコマンド)
 - \$ dropuser [接続オプション] [ユーザー名]
 - 接続情報以外にオプションを指定する必要は一般的にない
- DROP USER 文 (SQL)
 - =# DROP USER [ユーザー名];
- **[注意]**
 - 当該ユーザーがテーブルなど何らかのオブジェクトを所有している場合、それらをすべて削除しなければユーザーを削除できない



■ 権限

- 特定のアクションを実行するために必要な権利
- アクションごとに多くの権限が存在

■ PostgreSQLにおける権限の分類

権限	説明	関連コマンド
ロール属性としての権限	データベースクラスタで管理される対象に対する権限 (例: データベースの作成、ログイン可否)	ALTER USER
アクセス権限	データベースに存在するある特定のオブジェクトに対する権限 (例: テーブルtbl1の参照、シーケンスseq1の使用)	GRANT REVOKE

■ 本章ではロール属性としての権限を説明

- アクセス権限については後半で説明



■ データベースクラスタで管理される対象に対する権限

- データベースの作成可否、ロール/ユーザーの作成可否など

- 権限に対応するロール属性が存在する

- 権限を持たないことを示す属性(NO...) が存在

- 例) CREATEDB : データベースを作成できる
 ⇔ NOCREATEDB : データベースを作成できない

- (参考) 権限以外のロール属性もある

- 例) PASSWORD属性、CONNECTION LIMIT属性

■ ロール属性の設定

- 既存ユーザーのロール属性を変更

=# ALTER USER ユーザー名[権限関連のロール属性名] [, ...];

- 例) =# ALTER USER scott CREATEDB NOCREATEROLE;

- ユーザー新規作成時にロール属性を指定

\$ createuser [権限関連のオプション] ユーザー名

または

=# CREATE USER ユーザー名[権限関連のロール属性名] [, ...];



ロール属性名 (⇔ 反対の意味の属性)	説明	createuserの オプション(*1)
SUPERUSER ⇔ NOSUPERUSER [デ]	スーパーユーザー権限を持つ ⇔ 一般ユーザー権限を持つ	-s ⇔ -S
CREATEDB ⇔ NOCREATEDB [デ]	データベースを作成可能 ⇔ 作成不可	-d ⇔ -D
CREATEROLE ⇔ NOCREATEROLE [デ]	ロールを作成可能 ⇔ 作成不可	-r ⇔ -R
CREATEUSER ⇔ NOCREATEUSER [デ]	SUPERUSERとNOSUPERUSERの別名 (廃止予定)	なし
LOGIN [デ*] ⇔ NOLOGIN	インスタンスにログイン可能(*2) ⇔ ログイン不可	-l ⇔ -L

[デ] : CREATE USER or CREATE ROLE実行時のデフォルト

[デ*] : CREATE USER実行時のデフォルト

CREATE ROLE実行時はNOLOGINがデフォルト

(*1) createuser実行時に属性を指定するときのオプション指定

(*2) インスタンスにログインするためには別途pg_hba.confの構成も必要



■ ¥duメタコマンド

```
postgres=# ¥du
```

List of roles

Role name	Attributes	Member of
postgres	Superuser, Create role, Create DB	{}
user1	Create DB	{}
user2	Create DB	{}



- PostgreSQLにおいて、**ユーザーとロールはまったく同じもの**
 - CREATE USER ≡ CREATE ROLE
 - ログイン関連のデフォルト属性のみ異なる
 - CREATE USER : [デ] LOGIN (ログイン可)
 - CREATE ROLE : [デ] NOLOGIN (ログイン不可)
- **[注意]** Oracle Databaseではユーザーとロールは異なる役割を持つ
 - ユーザー = いわゆる一般的な「ユーザー」
 - ロール = ユーザーをグループ化したもの



■ データベースの作成方法

- `$ createdb [オプション] データベース名 [コメント]`
- `=> CREATE DATABASE データベース名 [オプション];`
- **CREATEDB 権限が必要**
- **template1データベースをひな形にして、データベースが作成される**
 - **template1データベースは初期状態で作成済み**
 - **共通に使用するオブジェクトや関数がある場合は、事前にtemplate1に作成しておく**と、後で作成する手間が省ける
 - **別のデータベースをひな形にすることも可能 (-Tオプション)**

■ データベースの削除方法

- `$ dropdb データベース名`
- `=> DROP DATABASE データベース名`
- **データベースの所有者、またはスーパーユーザーのみが実行可能**



- データベースにおけるバックアップの重要性
 - 一般にデータベースでは重要なデータを管理している。ディスクの故障などによるデータの損失に備え、バックアップを取得することが**極めて重要**
- バックアップ方法の検討で考えておくべきこと
 - 正しいバックアップ取得方法を用いているかどうか
 - ファイルを単にコピーするだけでは一般にNG
 - 復旧ターゲットとシステム要件
 - データベースが破損した場合に、いつの時点に戻ればOKとするか
 - バックアップ取得時点？ 障害発生直前？
 - 復旧方法と所要時間
 - 障害が発生してから復旧完了までどの程度時間がかかるか
 - 当然ながら、復旧作業を円滑に実行できないと時間が余計にかかる
 - 復旧手順を理解および手順化しておくこと
- これから説明するバックアップ取得方法から、要件に合致する方法を使用する

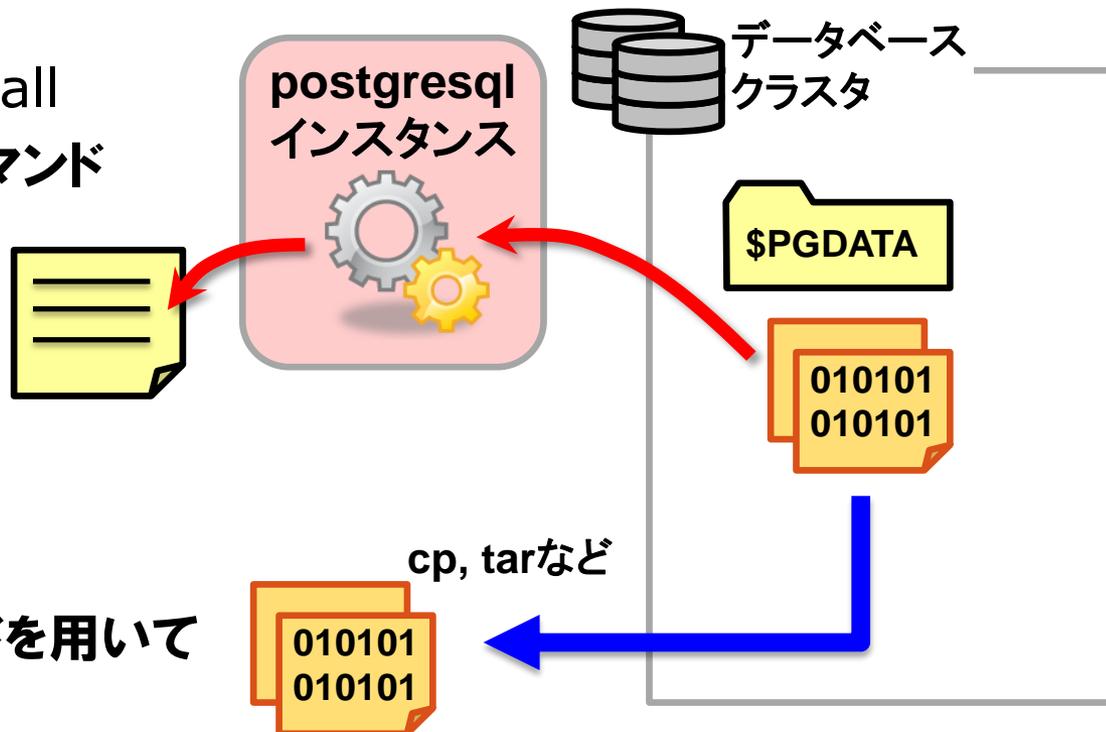


- **pg_dump コマンド**
 - データベース単位でバックアップを作成
 - psql または pg_restore コマンドを使ってリストア
- **pg_dumpall コマンド**
 - データベースクラスタ全体のバックアップを作成
 - psql コマンドを使ってリストア
- **COPY 文、¥copy メタコマンド**
 - テーブル単位でCSV形式ファイルの入出力
- **コールドバックアップ(ディレクトリコピー)**
 - OS付属のコピー、アーカイブ用コマンドを使ってバックアップを作成
 - 簡単で確実な方法だが、インスタンスを停止する必要がある
- **ポイント・イン・タイム・リカバリ(PITR)のベースバックアップ**
 - インスタンスを停止する必要がないが、使い方がやや複雑
 - WAL(Write Ahead Logging)機能と組み合わせて、任意の時点にリカバリ可能



■ 論理バックアップ

- PostgreSQLの機能を用いてデータをエクスポートする
- pg_dump / pg_dumpall
- COPY文 / ¥copyメタコマンド



■ 物理バックアップ

- 構成ファイルをOSコマンドを用いてコピー
- コールドバックアップ
- PITRのベースバックアップ

データベースクラスタの実体は\$PGDATA以下のファイル群



バックアップ方法	分類	取得タイミング	取得単位	復旧ターゲット
pg_dumpコマンド	論理 バック アップ	起動中	データベース、 スキーマ、 テーブル	バックアップ 取得時点
pg_dumpallコマンド			データベース クラスタ	バックアップ 取得時点
COPY文 ¥copyメタコマンド			テーブル	バックアップ 取得時点
コールド バックアップ	物理 バック アップ	停止中	データベース クラスタ	バックアップ 取得時点
ポイント・イン・タイム・ リカバリ(PITR)の ベースバックアップ		起動中		障害発生 直前

■ **[注意] 障害発生直前に復旧できるのはポイントイン・タイム・リカバリのベースバックアップのみ**

■ **更新の欠落が許されない環境では、ポイント・イン・タイム・リカバリの使用を推奨**



- インスタンス起動中に、バックアップを取得
 - 様々な単位でバックアップを取得できる
 - データベースdb1をバックアップ

```
$ pg_dump db1 > db1.sql
```
 - データベースdb1のテーブルtbl0をバックアップ

```
$ pg_dump -t tbl0 db1 > tbl0.sql
```
 - データベースdb1のスキーマpublicをバックアップ

```
$ pg_dump -n public db1 > public.sql
```
- いくつかのファイル形式をサポート
 - テキスト形式(SQL)、カスタム形式、tar形式
 - ファイル形式により、復旧方法が異なる
 - (詳細は次スライド)



■ -F または --format= オプションで出力ファイル形式を指定

■ カスタム形式でのデータベースのバックアップ例

```
$ pg_dump -F c db1 > db1.custom
```

■ [注意] 出力形式により復旧方法(インポート方法)が異なる

出力ファイル形式	オプション	説明
テキスト形式 (SQL) [デフォルト]	-F p または --format=plain	SQL文が記録されたテキストファイル インポート前にデータを編集可能 psqlコマンドを使用してインポートできる
カスタム形式	-F c または --format=custom	独自形式のバイナリファイル 自動的に圧縮される pg_restoreコマンドを使用してインポートできる インポートを並列で実行できる
tar形式	-F t または --format=tar	リストア用の SQL スクリプトと、各テーブルごとの データファイルがTAR形式で1つのファイルに アーカイブされている pg_restoreコマンドを使用してインポートできる



- インスタンス起動中に、データベースクラスタ全体のバックアップを取得
 - `$ pg_dumpall [各種オプション] -f ファイル名`
または
`$ pg_dumpall [各種オプション] >ファイル名`
- ユーザー情報などのグローバルオブジェクトもバックアップ可能
(pg_dumpでは取得できない)
 - `-g` オプションを指定すると、グローバルオブジェクトのみバックアップする
- `psql` コマンドで復旧(インポート)する
 - `$ psql -f ファイル名 postgres`
または
`$ psql postgres <ファイル名`



バックアップ コマンド	バックアップ単位	出力形式	復旧時に使用する コマンド
pg_dump	データベース、 スキーマ、 テーブル	テキスト形式(SQL)	psql
		カスタム形式	pg_restore
		tar形式	
pg_dumpall	データベース クラスタ全体 (グローバルオブジェク ト含む)	テキスト形式(SQL)	psql

■ [注意] pg_dumpの注意点

- データベースクラスタ全体のバックアップが取得できない
- pg_dumpを全データベースに対して実行しても、グローバルオブジェクトがバックアップできない

■ [注意] pg_dumpallの注意点

- 出力形式がテキスト形式(SQL)に限定される



- COPY TO (データをCSVファイルとしてエクスポート)
 - =# COPY テーブル名 TO 'ファイル名' [その他オプション];
- COPY FROM (CSVファイルをテーブルにインポート)
 - =# COPY テーブル名 FROM 'ファイル名' [その他オプション];
- データベースサーバマシン上のファイルシステムにファイル出力(入力)する
- **[注意]** 原則的に、スーパーユーザーのみ実行可能
 - データベースサーバマシン上のファイルシステムにアクセスするため
 - ただし、ファイル名としてSTDOUT (STDIN)を指定すると、標準出力(標準入力)とのデータのやり取りになる。この場合は、一般ユーザーでも実行可能



- ¥copy to （データをファイルにエクスポート）
 - => ¥copy テーブル名 to ファイル名 [その他オプション]
- ¥copy from（ファイルをテーブルにインポート）
 - => ¥copy テーブル名 from ファイル名 [その他オプション]
- クライアントマシン(psqlを実行しているマシン)上のファイルシステムにファイル出力(入力)する
- デフォルトのファイル形式はタブ区切りのテキストファイル
- オプションに"csv"と指定すれば、カンマ区切りのCSVファイル



■ インスタンスを停止してPGDATA以下の全ファイルをバックアップ

■ バックアップ用のファイルコピー方法は自由

- `$ cp -r $PGDATA <別のディレクトリ>`
- `$ tar czf $PGDATA <tar.gzファイル名>`
- ストレージ機器のスナップショット
- など

■ コピー先をPGDATAがあるディスクとは別のディスクにすることを推奨

- PGDATAがあるディスクが破損した場合に、バックアップも一緒に失われないように
- (参考) テーブル空間をつかっている場合はそのディレクトリ以下のファイルも一緒にバックアップする

■ 復旧方法

■ バックアップを戻して(リストア)、インスタンスを再起動

■ [注意] バックアップ取得時点の状態にしか戻れない

- バックアップ取得時点→障害発生時点までに実行された更新はすべて失われる

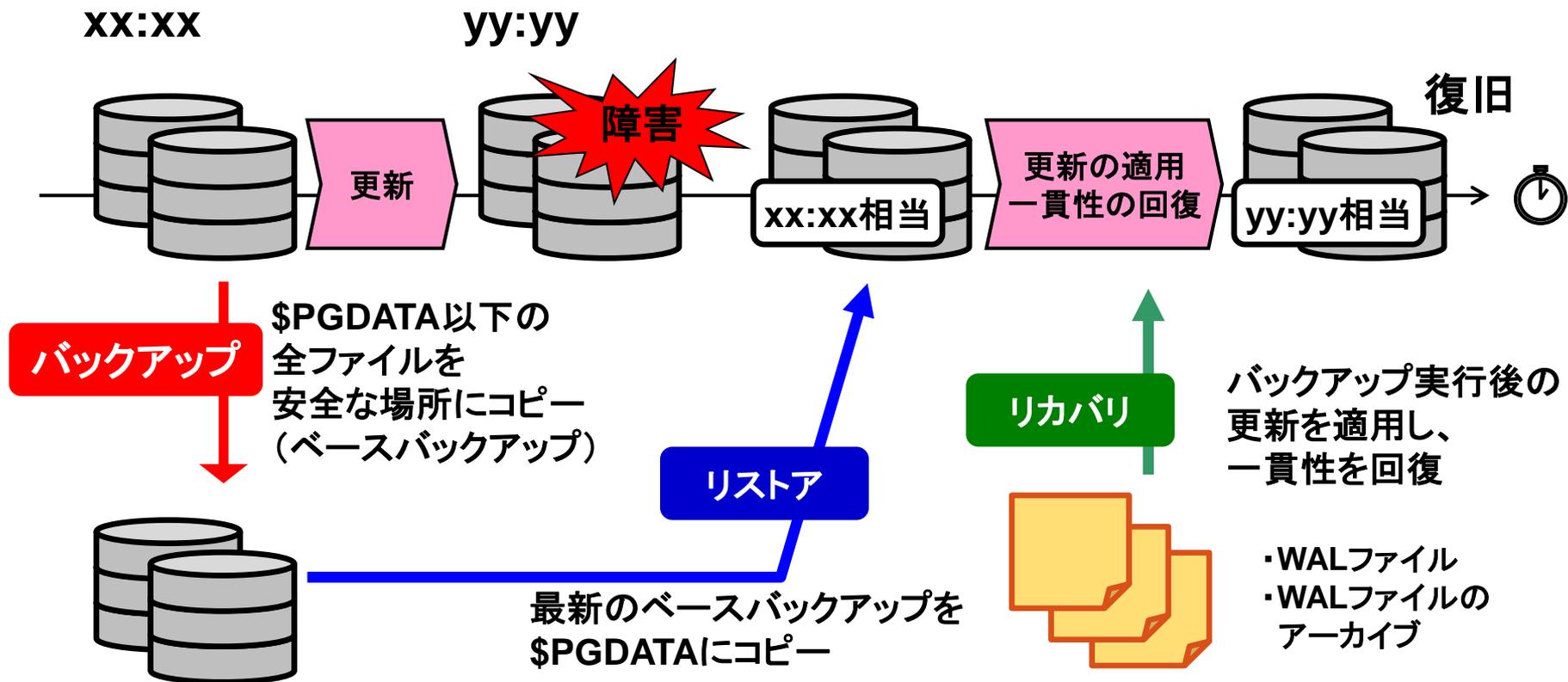


■ 従来のバックアップ方法の重大な問題点

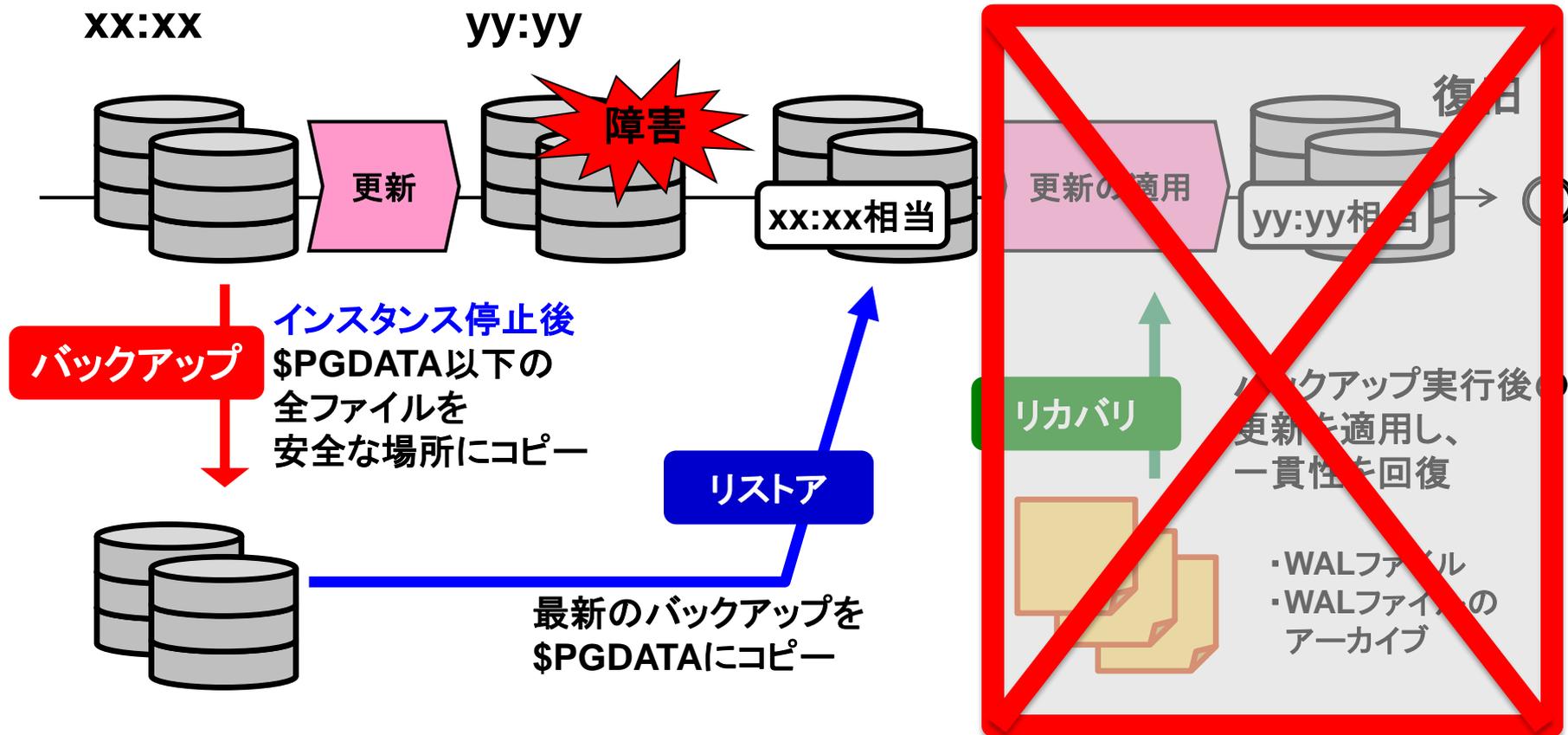
- 障害発生時の**復旧ターゲットがバックアップ取得時点に限定**される
- バックアップ取得～障害発生時点 の間に適用された**更新が失われる**
- 失われた更新を何らかの方法で復元する必要があるが、**システムの的に解決することは一般に困難であり、人手による作業が不可欠**
 - 例)
実行した更新内容をユーザーやアプリケーションのログから特定し、SQLを作成してpsqlから実行する、など

■ PITR (Point In Time Recovery)の利点

- **障害発生の直前の状態までデータを復旧(リカバリ)できる**
 - ただし、すべてのWALファイルが失われていない場合
- **過去のある時点に復旧することも可能**
 - データを誤って削除した場合などに有効



- リカバリ(WALファイルの適用)により**障害発生直前の状態に復旧できる**
- ただし、WALの構成と定期的なバックアップ、アーカイブを含めたWALファイルの保管が必要



- インスタンス停止後にバックアップを取得する必要がある
- 障害発生時はバックアップ取得時点の状態に戻る
- リカバリを実行できないため、**障害発生直前の状態に復旧できない**



■ postgresql.confの設定例

```
wal_level = archive または hotstandby  
archive_mode=on  
archive_command = 'cp -i %p /somedir/%f </dev/null'# Unix  
archive_command = 'copy "%p" "X:¥¥somedir¥¥%f"' # Windows
```

■ **[注意]** デフォルト設定ではPITR使用不可

■ archive_commandの指定

- WAL ファイルを安全な場所にコピーするため、OSのファイルコピーコマンドを指定する
- コピー元ファイル名に %p (アーカイブするファイルのパス名)を指定
- コピー先ファイル名に%f (%pのファイル名部分のみ)を含める
- WALのコピー先は、データベースクラスタ(\$PGDATA)とは別のディスク装置上のディレクトリにすることを推奨



- PITRに使用できるバックアップをベースバックアップと呼ぶ
- ベースバックアップの取得手順
 - 1. スーパーユーザーで接続し、バックアップ開始をサーバに通知
 - =# SELECT pg_start_backup('label');
 - 2. tar, cpio などのOSコマンドでバックアップを取得
(インスタンスは起動したままでよい)
 - 3. 再度、スーパーユーザーで接続し、バックアップ終了をサーバに通知
 - =# SELECT pg_stop_backup();
- (参考)PostgreSQL 9.1では pg_basebackup コマンドにより、上記の手順をまとめて実行できる



- ベースバックアップを用いて障害発生直前の状態まで復旧する

- **ポイントインタイムリカバリの手順**

- 1. インスタンスを停止する（起動している場合）
- 2. `$PGDATA/pg_xlog`内のファイルをコピーする
（またはデータベースクラスタ内の全ファイルをコピーする）
- 3. `$PGDATA`以下の全ファイルを削除する
- 4. ベースバックアップを`$PGDATA`にリストアする
- 5. `$PGDATA/pg_xlog`内のファイルをすべて削除する
- 6. 2. でコピーしておいた`$PGDATA/pg_xlog`内のファイルを、
現在の `$PGDATA/pg_xlog`にコピーする
- 7. `$PGDATA/recovery.conf`（復旧コマンドファイル）を作成する
- 8. インスタンスを起動する

`$PGDATA/pg_xlog`にある最新のWALファイルを一旦退避してから元に戻す
→ 直近の更新処理がWALファイルに記録されているため



- インスタンス起動時に `$PGDATA/recovery.conf`がある場合、インスタンスは復旧モードに入る(リカバリを実行する)

- `$PGDATA/recovery.conf`の例

```
restore_command = 'cp /somedir/%f "%p"' # Unix  
restore_command = 'copy "C:¥¥somedir¥¥%f" "%p"' # Windows
```

- `restore_command`に指定されたファイルコピーコマンドを実行して、WALファイルのアーカイブをコピーする
- リカバリ処理を実行し、WALファイルに記録された更新処理をデータベースクラスタに適用する
- 復旧完了後、`recovery.conf` は `recovery.done`にリネームされる
 - 再び復旧モードを入らないように



- 用語の差異が、「理解しにくさ」につながるケースがあります
- 「ホットバックアップ」
 - 一般的に「コールドバックアップ」と対比させる形で「インスタンス起動中の物理バックアップ」を指す
 - PostgreSQLでは「論理バックアップ」を指す場合が多い
「インスタンス起動中の物理バックアップ」は「ベースバックアップ」と呼ぶ
- 「リストア」
 - 「物理バックアップしたファイルの戻し(コピー)」に限定して使用することもある
 - PostgreSQLでは「論理バックアップを含めたバックアップしたデータの復旧処理全般」を指す
- 「ポイントインタイムリカバリ (PITR)」
 - 一般的に、「過去のある時点をターゲットにした復旧」を指し、「最新時点をターゲットにした復旧」と明確に区別する
 - PostgreSQLでは「WALを用いたリカバリを伴う復旧全般」を指す



	PostgreSQL	Oracle Database
論理バックアップ	pg_dump, pg_dumpall	DataPump (expdp), Export (exp)
コールドバックアップ (インスタンスを停止し ての物理バックアップ)	cp, tarなど	cp, tarなど または RMANでBACKUP DATABASE
ホットバックアップ (インスタンス稼働中の 物理バックアップ)	1. pg_start_backup() 2. cp, tarなど 3. pg_stop_backup() または pg_basebackup (PostgreSQL 9.1以降)	1. BEGIN BACKUP 2. cp, tarなど 3. END BACKUP または RMANでBACKUP DATABASE

参考) Oracle DatabaseのRMANに類似した機能を持つpg_rmanという非標準ツールがある
<http://sourceforge.net/projects/pg-rman/>

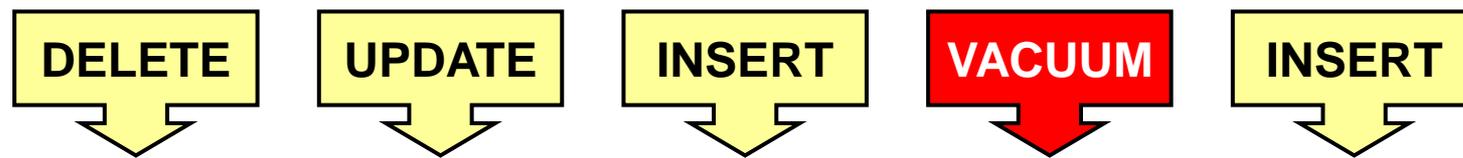


- **不要領域の回収 - いわゆるバキューム処理**
 - 更新処理(UPDATE、DELETE)により生じた不要領域を回収する
 - INSERTによるデータ追加に使用できるように
 - データファイルの肥大化を抑制
- **プランナ統計情報の収集**
 - プランナ統計情報(=行数、行サイズ、列値の偏りなど)を最新に更新する
 - プランナ統計情報が古いと、SQLの処理パフォーマンスが適切にならない場合がある
 - SQLの処理方法(プラン)は統計情報をもとに決定されるため



■ PostgreSQLは追記型アーキテクチャ

- DELETEしても、削除マークをつけるだけで、領域を占有し続ける
- UPDATEはDELETE+INSERTとして処理される
- バキュームを実行すると、削除マークをつけた領域が再利用可能になる



1	AAA
2	BBB
3	CCC
4	DDD

1	AAA
2	BBB
3	CCC
4	DDD

1	AAA
2	BBB
3	CCC
4	DDD
3	EEE

1	AAA
2	BBB
3	CCC
4	DDD
3	EEE
5	FFF

1	AAA
4	DDD
3	EEE
5	FFF

1	AAA
6	GGG
4	DDD
3	EEE
5	FFF

削除マークをつけるだけで領域は占有したまま

更新前のデータに削除マークをつけ、更新後のデータを追加

削除マークをつけていた領域が再利用可能に



■ 1. VACUUM文

■ =# VACUUM オプション テーブル名

■ オプション

■ FULL : 不要領域回収後、データファイルを縮小する

■ ANALYZE : 不要領域回収後、プランナ統計情報を収集する

■ 2. vacuumdbコマンド

■ \$ vacuumdb 接続オプション オプション データベース名

■ オプション

■ -f / --full : 不要領域回収後、データファイルを縮小する

■ -z / --analyze : 不要領域回収後、プランナ統計情報を収集する

■ [注意]

■ フルバキューム(VACUUME FULL / vacuumdb -full)はテーブルを排他的にロックする

■ 8.3以降は自動バキュームがデフォルトでONになっているため、手動実行の必要性は薄い



- 不要領域の回収(バキューム)とプランナ統計情報の収集が自動的に実行される
 - テーブルごとにデータの変更量が設定値を超えると実行される
 - (参考) track_count=true の設定が必要(デフォルトtrue)
 - **[注意]** VACUUME FULLは実行しない
- 8.3以降からデフォルトでON
 - 一般に自動バキュームONが推奨される
 - OFFにしたい場合は autovacuum=off と設定する



- **プランナ統計はANALYZE文で手動収集できる**
 - =# ANALYZE テーブル名;
 - 指定したテーブルのプランナ統計情報を収集する
 - テーブル名を省略すると、データベース内の全テーブルのプランナ統計情報を収集する
- **[注意]**
 - ANALYZE文を手動実行する必要性は薄い
 - 8.3以降は自動バキュームがデフォルトでONになっており、プランナ統計情報も自動的に収集されるため
 - ただし、データを大量に更新した直後に、そのテーブルに対してSQLを実行する場合は、ANALYZE文を手動実行することが推奨される
 - 自動バキュームが実行されるまで、プランナ統計情報が古い状態になり、SQLの処理パフォーマンスが意図せず悪化する可能性があるため



処理	PostgreSQL	Oracle Database
バキューム	手動バキューム または autovacuum	不要
プランナ統計の収集 (オプティマイザ統計の 収集)	ANALYZEの手動実行 または autovacuum	DBMS_STATS.GATHER*の手 動実行 または オプティマイザ統計自動収集



■ 運用管理 - 設定ファイル

PostgreSQLのパラメータ設定に関する説明で、適切なものを2つ選びなさい。

- a. パラメータは/etc/postgresql.confに記載する
- b. SET文によるパラメータ設定変更は、SET文を実行したセッションでのみ有効である
- c. SHOW PARAMETERコマンドで、現在のパラメータ設定を確認できる
- d. パラメータにはインスタンス起動中に設定変更できるものとできないものがある。

回答: ?, ?



■ 運用管理 - 基本的な運用管理作業

PostgreSQLのバキューム処理に関する説明で、適切なものを2つ選びなさい。

- a. PostgreSQL9.0では、自動バキュームはデフォルトOFFである
- b. データベースのデータの状態によっては、自動バキュームでVACUUM FULLが実行されることがある
- c. 自動バキュームではプランナ統計情報を収集する
- d. VACUUM FULLはテーブルの排他ロックを取得するため、稼働中の実行は一般に避けるべきである

回答: ?, ?



■ 運用管理 - バックアップ方法

PostgreSQLのバックアップに関する以下の記述から、誤っているものを1つ選びなさい

- a. デフォルトのパラメータ設定では、ポイント・イン・タイム・リカバリ(PITR)を使用することができない
- b. テーブル内のデータをCSV形式でバックアップするため、COPY文または`¥copy`メタコマンドが使用できる
- c. `pg_dump`ではユーザー定義をバックアップできない
- d. `pg_restore`コマンドは、`pg_dump`と`pg_dumpall`両方のバックアップのリストアに使用できる

回答：？



ポイント解説：SQL

- SQLの基本
- 表(テーブル)の作成とデータ型
- SELECT文 / UPDATE文 / INSERT文 / DELETE文
- 結合 / 集約 / 副問い合わせ
- その他のオブジェクト
- アクセス権限
- トランザクション



- あるオブジェクトに対して特定のアクションを実行する権利
 - 単に「権限」とも呼ばれる
 - 例) テーブルtbl0へのSELECT権限、UPDATE権限 など
- アクセス権限の付与：GRANT文
 - GRANT 権限 ON オブジェクト名 TO ユーザー名
[WITH GRANT OPTION];
 - WITH GRANT OPTION を指定すると、そのユーザーは他のユーザーに権限を付与できる
- アクセス権限の剥奪：REVOKE文
 - REVOKE 権限 ON オブジェクト名 FROM ユーザー名;
- 暗黙的に保持するアクセス権限
 - オブジェクトの所有者は、所有するオブジェクトに対するすべての操作が可能
 - スーパーユーザーは全てのオブジェクトに対するすべての操作が可能
 - など（詳細は後述）



- $\forall dp$ メタコマンド
 - $\forall z$ メタコマンドは $\forall dp$ メタコマンド の別名
- 他のオブジェクト
 - $\forall dn+$: スキーマ一覧にアクセス権限を表示
 - $\forall l$: データベース一覧にアクセス権限を表示
- デフォルトのアクセス権限は $\forall ddp$ メタコマンドで確認できる



■ アクセス権限表示の書式

■ ユーザー名 = 与えられた権限/権限を付与したユーザー

```
db1=> \dp tbl0
```

			Access privileges	
Schema	Name	Type	Access privileges	Column access privileges
public	tbl0	table	user1=arwdDxt/user1+	
			user2=arwd/user1	+
			=D/user1	

(1 row)

■ user1=arwdDxt/user1

- user1が全ての権限を持つ(自分が所有するオブジェクトへの権限)

■ user2=arwd/user1

- user2がINSERT(a),SELECT(r), UPDATE(w),DELETE(d)権限を持つ
- 権限を付与したのはuser1

■ =D/user1

- publicがTRUNCATE(D)権限を持つ
- 権限を付与したのはuser1

'+'は改行 (psqlの表示による)



権限	略号	対象オブジェクト	説明
SELECT	r	t, c, v, s	テーブル、ビューの参照 シーケンスへのSELECT
INSERT	a	t, c, v, s	テーブルへのデータ挿入
UPDATE	w	t, c, v, s	テーブルの更新 シーケンスの使用
DELETE	d	t	テーブルのデータ削除
TRUNCATE	D	t	テーブルのTRUNCATE
RULE	R	t, v, s	テーブル、ビューへのルール定義
REFERENCES	x	t	外部キー制約を持つテーブルの作成
TRIGGER	t	t	トリガ定義

対象オブジェクト: d=データベース, t=テーブル, c=カラム, v=ビュー, s=シーケンス,
S=スキーマ, f=関数, l=手続き言語



権限	略号	対象オブジェクト	説明
CREATE	C	d, t, v, S	オブジェクトの作成
CONNECT	c	d	データベースに接続可能
TEMP, TEMPORAY	T	d	一時テーブルの作成
EXECUTE	X	f	関数や演算子の使用
USAGE	U	l, s, S	手続き言語での関数作成 スキーマ内のオブジェクトへのアクセス シーケンスの使用許可(currvalおよび nextval関数)
ALL ALL PRIVILEGES	なし	d, t, v, s, S, f, l	すべての権限

対象オブジェクト: d=データベース, t=テーブル, c=カラム, v=ビュー, s=シーケンス,
S=スキーマ, f=関数, l=手続き言語



- 作成直後のオブジェクトに対するアクセス権限
 - 所有者(一般に作成ユーザー)は全権限を持つ
 - スーパーユーザーは全てのオブジェクトに対する全権限を持つ
 - 他のユーザーは一切の権限を持たない
 - ただし、publicロールに付与された権限は、全ユーザーに対して有効となる

- (参考) デフォルトのアクセス権限のカスタマイズ
 - ALTER DEFAULT PRIVILEGES文で作成直後のオブジェクトに対するデフォルトのアクセス権限を変更可能
 - **[注意]** すでに作成済みのオブジェクトに対するアクセス権限は変更されない
 - ALTER DEFAULT PRIVILEGES FOR USER 作成ユーザー名 GRANT 権限 ON オブジェクト種別 TO 付与ユーザー名;
 - ALTER DEFAULT PRIVILEGES FOR USER 作成ユーザー名 REVOKE 権限 ON オブジェクト種別 FROM 剥奪ユーザー名;



■ publicにアクセス権限を付与

- GRANT文の権限付与対象ユーザーに「public」を指定すると、全てのユーザーに権限を一括付与できる
- public経由で付与した権限は、ユーザー指定では剥奪できず、public指定で剥奪する必要がある
 - ユーザーA以外にアクセスを許可したい、という場合に(1)publicに付与 (2)ユーザーAから剥奪、という手順をとってもユーザーAはpublicの権限を経由してアクセス可能なままになる

■ ロールにアクセス権限を付与

- GRANT文で権限をロールに付与し、そのロールをユーザーに付与できる
 - PostgreSQLではユーザーとロールは同一のもので区別されない
- GRANT **ロール名** TO **ユーザー名** [WITH GRANT OPTION];
- **[注意]** ロール属性としての権限(例: LOGIN権限やSUPERUSER権限)も同時に付与される



■ Oracle Databaseの権限

- **オブジェクト権限** : 個々のオブジェクトに対する権限
→ (PostgreSQL) **アクセス権限にほぼ相当**
- **システム権限** : 特定の操作に対する権限
→ (PostgreSQL) **ロール属性としての権限にほぼ相当**
一部をデータベースクラスタ関連オブジェクトへのアクセス権限で実現
- **2つの権限をともにGRANT / REVOKEで管理**
→ (PostgreSQL) **アクセス権限をGRANT / REVOKEで管理**
ロール属性としての権限をALTER USERで管理

■ MySQLの権限

- **権限タイプに応じて、Global, Database, Tableなど様々なレベルで指定できる**
- **一部の権限を管理者用の権限と呼ぶ(RELOAD、SHUTDOWN、PROCESS、SUPER)**
- **管理者用の権限を含めたすべての権限をGRANT / REVOKEで管理**



■ トランザクションとは

■ 複数のSQL処理をまとめた作業単位

■ 例) 銀行口座間の資金移動

```
UPDATE 口座 SET 残高 = 残高 - 10000 WHERE 口座番号 = A;
```

```
UPDATE 口座 SET 残高 = 残高 + 10000 WHERE 口座番号 = B;
```

■ もし、トランザクションがないと...

■ UPDATE 口座 SET 残高 = 残高 - 10000 WHERE 口座番号 = A;

だけが実行され、

```
UPDATE 口座 SET 残高 = 残高 + 10000 WHERE 口座番号 = B;
```

が実行されない状況が発生しうる

■ トランザクションがあれば

■ UPDATE 口座 SET 残高 = 残高 - 10000 WHERE 口座番号 = A;

```
UPDATE 口座 SET 残高 = 残高 + 10000 WHERE 口座番号 = B;
```

の両方が正常に実行されたか、両方とも実行されなかったかのいずれか

■ ALL or NOTHINGを実現 → 中途半端な状態がないことが保証される

■ アプリケーションのエラー処理が圧倒的にシンプルに

■ エラーが発生したら再実行すればよい



■ ACID特性

■ トランザクションが持つ特性をまとめたもの

特性	意味
Atomicity (原子性)	一連の処理が完全に実行されるか、全く実行されないかのいずれかである
Consistency (一貫性または整合性)	トランザクション実行前の時点でデータベースのデータが整合性を保持していれば、トランザクションの実行後もデータの整合性を維持し続ける
Isolation (分離性または隔離性)	同時に実行されたトランザクション同士が相互に干渉しない、隔離された状態にある
Durability (持続性または永続性、耐久性)	完了したトランザクションは適切に記録され、容易に失われることはない

■ いずれも重要な特性だが、アプリケーションを開発するにあたっては「Atomicity(原子性)」の理解が最も重要

- 意味的に関連する複数のSQL処理をトランザクションとしてグループ化すること



■ トランザクションの開始と確定

```
BEGIN; -- トランザクション開始 (START TRANSACTION; も使用可)
SQL1;
  :
SQLn;
COMMIT; -- トランザクション終了
```

トランザクションとしてグループ化したいSQL

- SQL1～nがエラーなく正常に終了し、COMMITを実行すると、トランザクション内のすべての変更が確定される
 - 一旦確定した変更は取り消すことができない

■ トランザクションの取り消し

- COMMIT実行前にROLLBACKを実行した場合、トランザクション内のすべての変更が破棄される
- SQL1～nのいずれかでエラーが発生した場合、トランザクション内のすべての変更が破棄される
 - 正確にはトランザクションが無効になり、ROLLBACK以外のコマンドが実行不可になる



```
db1=> SELECT * FROM tab0;
```

```
n
```

```
---
```

```
(0 rows)
```

```
db1=> BEGIN;
```

```
BEGIN
```

```
db1=> INSERT INTO tab0 VALUES(1);
```

```
INSERT 0 1
```

```
db1=> INSERT INTO tab0 VALUES(2);
```

```
INSERT 0 1
```

```
db1=> COMMIT;
```

```
COMMIT
```

```
db1=> SELECT * FROM tab0;
```

```
n
```

```
---
```

```
1
```

```
2
```

```
(2 rows)
```

トランザクションとして
グループ化したいSQL



```
db1=> SELECT * FROM tab0;  
n  
---  
(0 rows)
```

```
db1=> BEGIN;  
BEGIN  
db1=> INSERT INTO tab0 VALUES(1);  
INSERT 0 1  
db1=> SELECT * FROM tab0;  
n  
---  
1  
(1 row)
```

```
db1=> ROLLBACK;  
ROLLBACK
```

```
db1=> SELECT * FROM tab0;  
n  
---  
(0 rows)
```

トランザクション実行前の
状態に戻る



■ トランザクション内の処理のうち、**一部だけ**をロールバックしたい

■ SAVEPOINT **セーブポイント名**; でセーブポイントを宣言

■ ROLLBACK TO **セーブポイント名**; でセーブポイントへロールバック

```
db1=> BEGIN;
BEGIN
db1=> INSERT INTO tab0 VALUES(1);
INSERT 0 1
db1=> SELECT * FROM tab0;
n
---
```

```
1
(1 row)

db1=> SAVEPOINT A;
SAVEPOINT
db1=> INSERT INTO tab0 VALUES(2);
INSERT 0 1
db1=> SAVEPOINT B;
SAVEPOINT
db1=> INSERT INTO tab0 VALUES(3);
INSERT 0 1
```

```
db1=> SELECT * FROM tab0;
n
---
```

セーブポイントAの
状態に戻る

```
1
2
3
(3 rows)

db1=> ROLLBACK TO A;
ROLLBACK
```

```
db1=> SELECT * FROM tab0;
n
---
```



■ 理想的なトランザクションの分離

- 同時に実行された複数のトランザクションが、完全に隔離された状態にある
- 他のトランザクションが実行した更新の影響をまったく受けない
- 反面、一般に同時実行性を損なう可能性

■ 4つのトランザクション分離レベル

- 分離性が異なる分離レベルを用意し、要件に応じて使い分けられるように
- SERIALIZABLE, REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED

■ phenomena（発生する現象 / 不都合な現象）

- ダーティリード(Dirty Read)
 - 他のトランザクションによる未コミットの挿入/更新/削除結果が見える
- ファジーリード(Fuzzy Read)
 - ノンリピータブルリードともいう
 - 他のトランザクションによるコミット済みの更新/削除結果が見える
- ファントムリード(Phantom Read)
 - 他のトランザクションによるコミット済みの挿入結果が見える



phenomena (発生する現象)

分離性

高

低

分離レベル	ダーティリード	ファジーリード	ファントムリード	PostgreSQLサポート
SERIALIZABLE	発生しない	発生しない	発生しない	○ (*1)
REPEATABLE READ	発生しない	発生しない	発生する	△ (*1)
READ COMMITTED	発生しない	発生する	発生する	○
READ UNCOMMITTED	発生する	発生する	発生する	× (*2)

■ [注意]

- (*1) 9.0以前では、REPEATABLE READを要求してもSERIALIZABLEとして動作する
9.1以降では、REPEATABLE READを要求すると従来のSERIALIZABLE相当の分離レベルとして動作し、SERIALIZABLEを要求すると従来のSERIALIZABLE相当の動作に監視機能が追加された動作となる
- (*2) PostgreSQLではREAD UNCOMMITTEDを指定してもREAD COMMITTEDとして動作する



■ 1. セッション単位で指定

- `SET default_transaction_isolation = '分離レベル';`
または
`SET default_transaction_isolation TO '分離レベル';`

■ 2. トランザクション単位で指定

- トランザクション開始時に
`BEGIN ISOLATION LEVEL 分離レベル;`
または
`START TRANSACTION ISOLATION LEVEL 分離レベル;`
- トランザクション開始直後に
`SET TRANSACTION ISOLATION LEVEL 分離レベル;`

■ デフォルトの分離レベルはREAD COMMITTED



- 1つのSQLを実行すると自動的にコミット処理が実行される
- すなわち、明示的にトランザクションを開始しない限り、ロールバックできない

```
db1=> INSERT INTO tab0 values(1);
```

```
INSERT 0 1
```

```
db1=> SELECT * FROM tab0;
```

```
n
```

```
---
```

```
1
```

```
(1 row)
```

INSERT文実行完了時点で自動的にコミットされている(オートコミット)

```
db1=> ROLLBACK;
```

```
NOTICE: there is no transaction in progress
```

```
ROLLBACK
```

```
db1=> SELECT * FROM tab0;
```

```
n
```

```
---
```

```
1
```

```
(1 row)
```

すでにコミットされているので、この時点で実行中のトランザクションはない
→ ロールバックを実行できない



- トランザクション内でエラーが発生すると、トランザクション全体が無効化される
 - 一旦無効化されるとロールバックするしかない
 - Oracle Databaseではエラー発生後もトランザクション処理を継続できる

```
db1=> BEGIN;
BEGIN
db1=> INSERT into TAB0 VALUES(1);
INSERT 0 1
db1=> INSERT into TAB0 VALUES('A');
ERROR: invalid input syntax for integer: "A"
LINE 1: INSERT into TAB0 VALUES('A');
                                   ^
```

トランザクションが無効化されたため、SQLを実行できない

```
db1=> INSERT into TAB0 VALUES(2);
ERROR: current transaction is aborted, commands ignored until end of transaction block
db1=> ROLLBACK;
ROLLBACK
```

ロールバックするしかない



- 一部のDDLをトランザクションに含めることができる
 - CREATE TABLE, DROP TABLE, TRUNCATE TABLEなど
 - DMLと同様にロールバックで処理が取り消される

```
db1=> BEGIN;
BEGIN
db1=> CREATE TABLE tab0 (n int);
CREATE TABLE
db1=> INSERT INTO tab0 values(0);
INSERT 0 1
db1=> SELECT * FROM tab0;
 n
 ---
 0
(1 row)

db1=> ROLLBACK;
ROLLBACK
db1=> SELECT * FROM tab0;
ERROR:  relation "tab0" does not exist
LINE 1: SELECT * FROM tab0;
                        ^
```

CREATE TABLEもロール
バックされた



■ 開発/SQL - 権限

権限に関する以下の記述から、正しいものを2つ選びなさい

- a. オブジェクトを所有するユーザーは、所有するオブジェクトに対するすべての権限を持つ
- b. スーパーユーザーにアクセス権限を付与することは、すべてのユーザーに対してアクセス権限を付与するのと同様である
- c. あるテーブルのアクセス権限は`dp`メタコマンドで確認できる
- d. スーパーユーザー権限を持つユーザーは"postgres"ユーザーのみである

回答: ?, ?



■ 開発/SQL - トランザクション

トランザクションに関する以下の記述から、誤っているものを2つ選びなさい

- a. トランザクションの原子性とは、他のトランザクションによる影響を受けない特性である
- b. SERIALIZABLE分離レベルは、REPEATABLE READ分離レベルより高い分離性を持つ
- c. トランザクションはSTART; で開始し、END;で終了する
- d. トランザクション内でエラーが発生すると、トランザクション全体が無効になるため、ロールバックするまでSQLを実行できない

回答: ?, ?



- **多くのデータベース関連試験の中でも、OSS-DB試験は非常に質が高く、かつ受験費用が安いお勧めの資格です**
- **日常業務でPostgreSQLに触れる機会があるエンジニアの方にはぜひ受験をお勧めします**
- **現時点でPostgreSQLに触れる機会がないエンジニアの方でも、データベースに関するスキルを高めたい方は、受験をお勧めします。**



ありがとうございました

■お問い合わせ■

株式会社コーソル 渡部 亮太

mail: ryota.watabe@cosol.jp

TEL: 03-3264-8800