CO-Solutions

LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Silver 技術解説無料セミナー

2013/7/27

株式会社コーソル
プリンシパルエンジニア
渡部 亮太

© LPI-Japan 2013. All rights reserved.

自己紹介+所属会社紹介

CO-Solutions OSS-DB

- **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 MASTER Platinum 11g/10g, OSS-DB Gold, Oracle Certified Professional MySQL 5 Database Administrator **保持**
  - **著書「プロとしての**Oracle**アーキテクチャ入門」「プロとしての**Oracle**運用管理入門」**
  - **社内外における多数の講演実績あり**

- **株式会社コーソル**
  - 「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。**
  - **現在**PostgreSQL、MySQL**などの**OSS-DB**領域へも事業範囲を拡大中**
  - **社員数**: 107**名** (2013**年**7**月現在**)
  - **所在地**: **東京 千代田区（本社）、福岡**

© LPI-Japan 2013. All rights reserved.

Agenda

- 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

- 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

© LPI-Japan 2013. All rights reserved.

OSS-DB技術者認定試験の概要

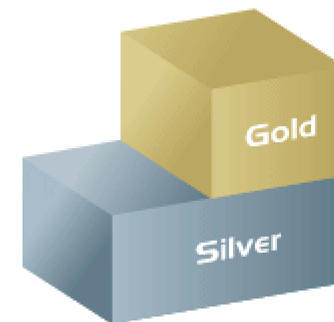
- OSS-DB技術者認定試験の概要
- 出題範囲とキーワード、本セミナーの説明箇所
- 注意事項

© LPI-Japan 2013. All rights reserved.

OSS-DB技術者認定試験の概要

CO-Solutions OSS-DB

- 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



© LPI-Japan 2013. All rights reserved.

OSS-DB Silverの出題範囲と説明する箇所

CO-Solutions OSS-DB

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

本資料に含む箇所（セミナー解説では取り上げない）

© LPI-Japan 2013. All rights reserved.

OSS-DB Silverの出題範囲と説明する箇所

CO-Solutions OSS-DB

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

© LPI-Japan 2013. All rights reserved.

OSS-DB Silverの出題範囲と説明する箇所

CO-Solutions OSS-DB

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、current_time、extract、to_char)
	トランザクション	構文（BEGIN、COMMIT、ROLLBACK、SAVEPOINTなど）、分離レベル、ロック

© LPI-Japan 2013. All rights reserved.

出題範囲、出題形式に関する注意事項

OSS-DB

- 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
  - PostgreSQLの最新バージョンは9.2
    （次期バージョン9.3は現在ベータステータス）
- 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

© LPI-Japan 2013. All rights reserved.

OSS-DB

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

© LPI-Japan 2013. All rights reserved.

インストール方法の分類

- 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 / Fodera / 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　Windows（オススメ）
  - 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

© LPI-Japan 2013. All rights reserved.

- PostgreSQL9.0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

© LPI-Japan 2013. All rights reserved.

yumを用いたインストール（Red Hat系Linux）

CO-Solutions OSS-DB

- 2. yumでPostgreSQL9.0をインストール
  - # yum install postgresql90-server
    - 1. でインストールしたリ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-9.0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-9.0/bin/initdb --no-locale -D /var/lib/pgsql/9.0/data
    - --no-localeオプション　：ロケールを使用しない（推奨）
  - ［注意］ # service postgres-9.0 initdb
    でもOKだが、ロケールを使用する設定となる

© LPI-Japan 2013. All rights reserved.

yumを用いたインストール（Red Hat系Linux）

- 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-9.0/bin/を追加
  - $ cat　~/.bash_profile

```
  :
PGDATA=/var/lib/pgsql/9.0/data
export PGDATA
PATH=/usr/pgsql-9.0/bin/:$PATH
export PATH
```

- 5.（オプション）PostgreSQL自動起動設定
  - # chkconfig postgresql-9.0 on

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

© LPI-Japan 2013. All rights reserved.

- ［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、PostgreSQL9.0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-9.0 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-9.0/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-9.0 start （起動）
    - # chkconfig postgresql-9.0 on （自動起動設定）

© LPI-Japan 2013. All rights reserved.

- 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

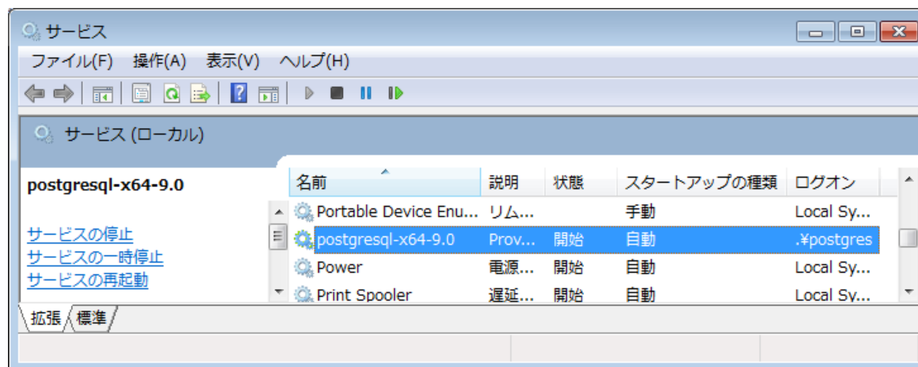
© LPI-Japan 2013. All rights reserved.

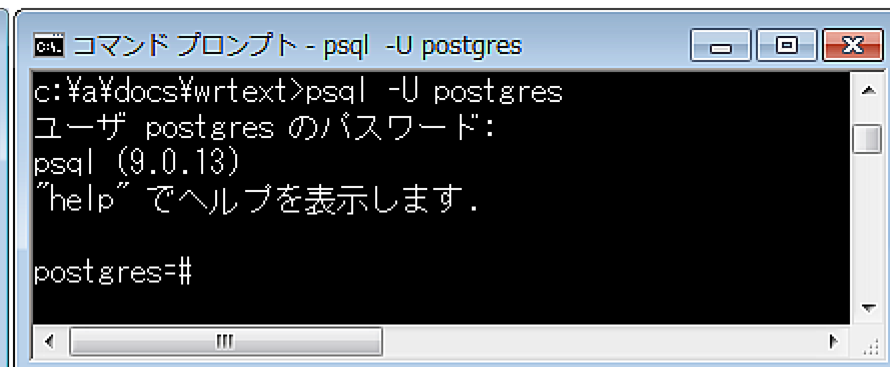
インストール後の設定

- （オプション）環境変数PATHの設定
  C:¥Program Files¥PostgreSQL¥9.0¥bin を追加する

[注意]

- データベースクラスタはC:¥Program Files¥PostgreSQL¥9.0¥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





© LPI-Japan 2013. All rights reserved.

ソースコードからのインストール

CO-Solutions　OSS-DB

- 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/9.0/html/の15章(Linux)、16章(Windows)に解説されている

© LPI-Japan 2013. All rights reserved.

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

© LPI-Japan 2013. All rights reserved.

PostgreSQLのアーキテクチャ

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

© LPI-Japan 2013. All rights reserved.

インスタンスを構成するプロセス

- PostgreSQLはマルチプロセスアーキテクチャ
  - ワーカプロセス、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

```
[postgres ~]$ ps -eo pid,ppid,command |grep postgres
 2664     1 /usr/pgsql-9.0/bin/postgres
 2666  2664 postgres: writer process
 2667  2664 postgres: wal writer process
 2668  2664 postgres: autovacuum launcher process
 2669  2664 postgres: stats collector process
 2679  2664 postgres: user1 db1 [local] idle
 2681  2664 postgres: user2 db1 [local] idle
```

マスタサーバプロセス

ワーカプロセス

サーバプロセス

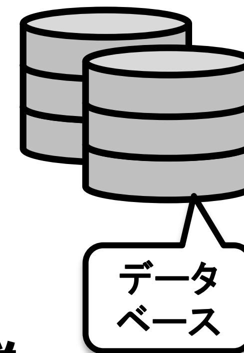
ワーカプロセスの名称

ワーカプロセス、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（親プロセスがマスタサーバプロセス）

サーバプロセスの接続情報
ユーザー名、データベース名等

© LPI-Japan 2013. All rights reserved.

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
```

© LPI-Japan 2013. All rights reserved.

initdbコマンド

- 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

© LPI-Japan 2013. All rights reserved.

initdbコマンドの実行例

OSS-DB

```
[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

© LPI-Japan 2013. All rights reserved.

データベース

- 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

© LPI-Japan 2013. All rights reserved.

初期状態で存在するデータベース

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

インスタンスの起動

OSS-DB

- 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で対応する
- $ 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
```

© LPI-Japan 2013. All rights reserved.

インスタンスの停止

OSS-DB

- $ 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
```

© LPI-Japan 2013. All rights reserved.

psqlユーティリティとインスタンスへの接続

- 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.

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

© LPI-Japan 2013. All rights reserved.

接続関連のpsqlコマンドオプション

CO-Solutions OSS-DB

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
```

© LPI-Japan 2013. All rights reserved.

psqlのプロンプト表示

- 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

© LPI-Japan 2013. All rights reserved.

psqlで実行できるコマンド

OSS-DB

- 1. SQL文
  - psqlよりサーバプロセスに送信され、サーバプロセスで処理される
  - 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で処理されるpsql独自のコマンド
  - 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

© LPI-Japan 2013. All rights reserved.

psqlでのSQL文の実行

OSS-DB

```
[postgres ~]$ psql
  :
postgres=# SELECT * FROM DEPT;
 deptno |    dname     |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)

postgres=# SELECT *
postgres-# FROM DEPT;
 deptno |    dname     |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

セミコロン';'でコマンド終了

コマンド途中で改行可能

© LPI-Japan 2013. All rights reserved.

psqlメタコマンド – オブジェクト情報表示

■ 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

© LPI-Japan 2013. All rights reserved.

■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)

db1=> ¥d+
                    List of relations
 Schema | Name | Type  | Owner |  Size   | Description
--------+------+-------+-------+---------+-------------
 public | dept | table | user1 | 0 bytes |
 public | emp  | table | user1 | 0 bytes |
(2 rows)
```

末尾にセミコロン';'は不要

'+'指定で追加情報を表示

© LPI-Japan 2013. All rights reserved.

- 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)
```

© LPI-Japan 2013. All rights reserved.

メタコマンド¥dS

OSS-DB

■ 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)
```

© LPI-Japan 2013. All rights reserved.

psqlのメタコマンド – そのほか

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]	
¥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

© LPI-Japan 2013. All rights reserved.

■ 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)
```

© LPI-Japan 2013. All rights reserved.

■ 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)

db1=> ¥x
Expanded display is on.
db1=> SELECT * FROM tbl1;
-[ RECORD 1 ]
id   | 1
col1 | AAA
-[ RECORD 2 ]
id   | 1
col1 | BBB

db1=> ¥x
Expanded display is off.
db1=> SELECT * FROM tbl1;
 id | col1
----+------
  1 | AAA
  1 | BBB
(2 rows)
```

1行のデータ=1行表示

1列のデータ=1行表示

© LPI-Japan 2013. All rights reserved.

例題解説1

OSS-DB

■ 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：d

© LPI-Japan 2013. All rights reserved.

例題解説2

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: b, c

© LPI-Japan 2013. All rights reserved.

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

© LPI-Japan 2013. All rights reserved.

PostgreSQLのパラメータ

CO-Solutions OSS-DB

- PostgreSQLには数多くのパラメータが存在する
  - PostgreSQL9.0.13 では 195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

- 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)

© LPI-Japan 2013. All rights reserved.

パラメータ設定ファイル (postgresql.conf)

CO-Solutions OSS-DB

- 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
# -----------------------------
   :
listen_addresses = '*'
port = 5432
   :

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

コメント行
('#'から行末までがコメント)

"パラメータ名 = 値" という形式で
パラメータを設定

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

© LPI-Japan 2013. All rights reserved.

主要なパラメータ（接続と認証）

- 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

© LPI-Japan 2013. All rights reserved.

- 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

© LPI-Japan 2013. All rights reserved.

主要なパラメータ（クライアント動作制御）

- 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

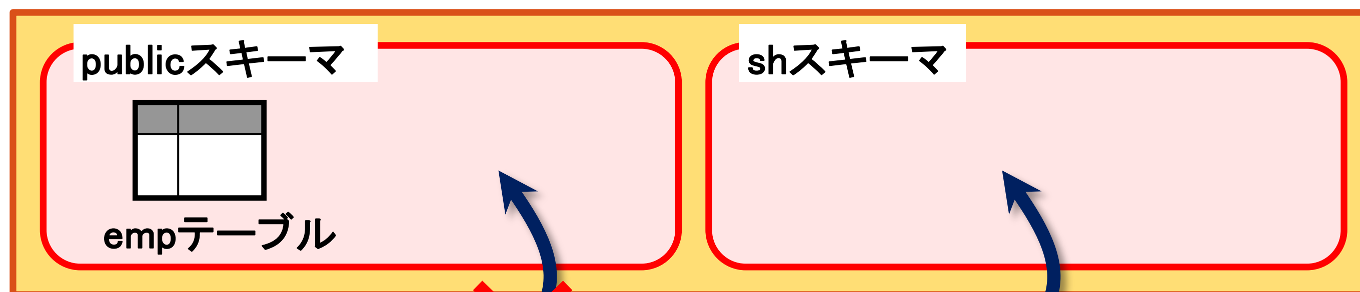
© LPI-Japan 2013. All rights reserved.

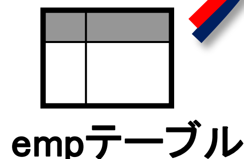
スキーマと名前空間

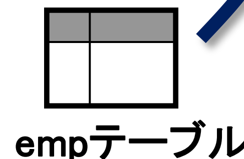
OSS-DB

- 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



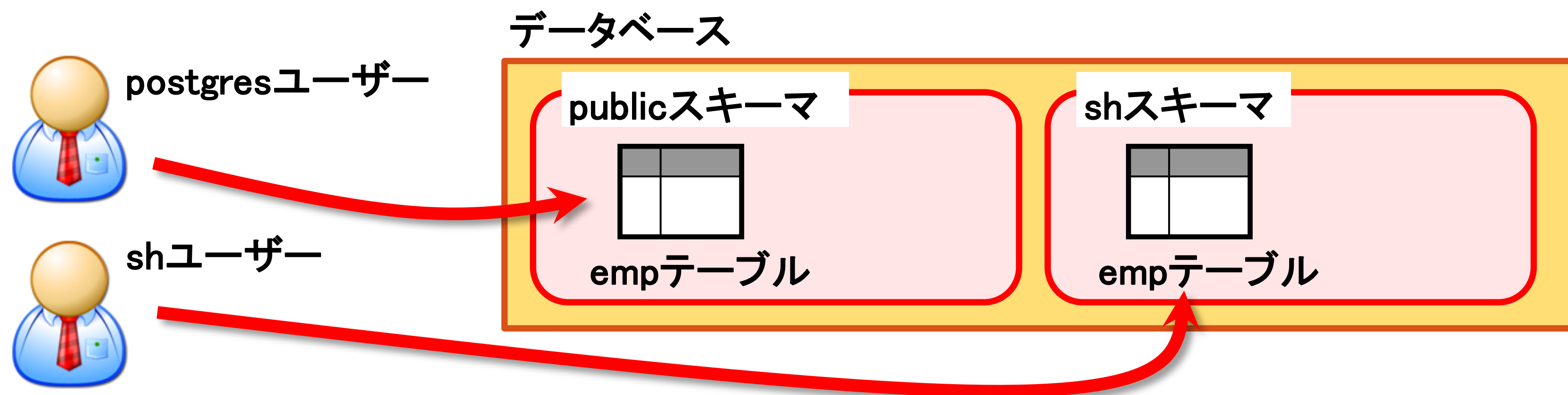




© LPI-Japan 2013. All rights reserved.

スキーマとsearch_path

- 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



© LPI-Japan 2013. All rights reserved.

(参考) データベース周辺のアーキテクチャ

データベース

スキーマ テーブル　スキーマ テーブル

接続　ユーザー

スキーマ テーブル　スキーマ テーブル

データベースクラスタ

ユーザー定義

- 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

© LPI-Japan 2013. All rights reserved.

（参考）Oracle Databaseのアーキテクチャ

- 複数のデータベースをまとめた概念は基本的にない
（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が保持

© LPI-Japan 2013. All rights reserved.

（参考）MySQLのアーキテクチャ

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

主要なパラメータ(クライアント動作制御)

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

主要なパラメータ(ログ関連)

- 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

© LPI-Japan 2013. All rights reserved.

- 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

© LPI-Japan 2013. All rights reserved.

■ 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

© LPI-Japan 2013. All rights reserved.

主要なパラメータ(ログ関連)

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

主要なパラメータ（参考：性能関連）

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

主要なパラメータ（参考：PITR関連）

- 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)

© LPI-Japan 2013. All rights reserved.

現行パラメータ設定の確認方法

- 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%';

© LPI-Japan 2013. All rights reserved.

起動中のパラメータ変更

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）**

© LPI-Japan 2013. All rights reserved.

クライアント認証の設定 (pg_hba.conf)

- **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**

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

記載した順序で評価される

```
# 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

© LPI-Japan 2013. All rights reserved.

- 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ユーザーとPostgreSQLユーザーが同じであれば、無条件に接続を許可
  - peer　　：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。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

© LPI-Japan 2013. All rights reserved.

pg_hba.confの記載例と解釈


```
# TYPE    DATABASE   USER        CIDR-ADDRESS     METHOD
local     all        postgres                     md5      ①
local     all        all                          ident    ②
host      all        all         127.0.0.1/32     trust    ③
host      db1        all         192.168.0.0/24   reject   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)

© LPI-Japan 2013. All rights reserved.

（参考）デフォルトのpg_hba.confの留意点

- 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

© LPI-Japan 2013. All rights reserved.

システムカタログと情報スキーマ

OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

- 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

© LPI-Japan 2013. All rights reserved.

ユーザー削除

- 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

© LPI-Japan 2013. All rights reserved.

- 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

© LPI-Japan 2013. All rights reserved.

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
  - 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
  - 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] [, ...];

© LPI-Japan 2013. All rights reserved.

主要なロール属性としての権限

OSS-DB

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

© LPI-Japan 2013. All rights reserved.

ロール属性としての権限の確認

■ ¥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                          | {}
```

© LPI-Japan 2013. All rights reserved.

ユーザー (USER) とロール (ROLE)

- 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 → LOGIN
      CREATE ROLE → NOLOGIN

© LPI-Japan 2013. All rights reserved.

データベースの作成・削除

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

- 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
  - バックアップ取得方法
    - 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
    - バックアップ方法だけでなく、復旧方法も理解しておくこと
    - 復旧にはどの程度の時間が必要か
      - 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
- 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こと

© LPI-Japan 2013. All rights reserved.

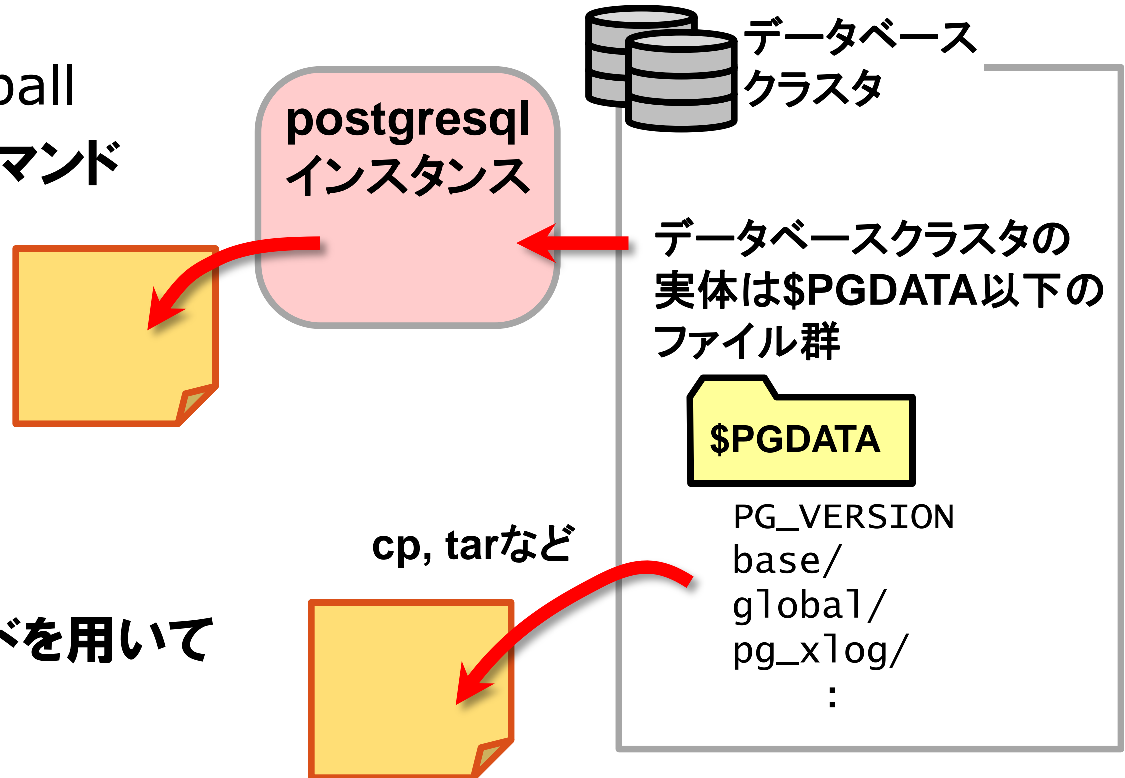
PostgreSQLのバックアップ方法

- 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
  - 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

© LPI-Japan 2013. All rights reserved.

バックアップ方法の整理

- 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

© LPI-Japan 2013. All rights reserved.

バックアップ方法の整理

バックアップ方法	取得方法	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**

© LPI-Japan 2013. All rights reserved.

pg_dumpによるバックアップと復旧

- 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）

© LPI-Japan 2013. All rights reserved.

pg_dumpの出力ファイル形式

- -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

© LPI-Japan 2013. All rights reserved.

pg_dumpall によるバックアップと復旧

- **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**

© LPI-Japan 2013. All rights reserved.

pg_dumpとpg_dumpallの比較

OSS-DB

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

© LPI-Japan 2013. All rights reserved.

COPY文（CSVファイル入出力）

- 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**

© LPI-Japan 2013. All rights reserved.

¥copyメタコマンド（ファイル入出力）

- ¥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

© LPI-Japan 2013. All rights reserved.

コールドバックアップ

- 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

© LPI-Japan 2013. All rights reserved.

ポイント・イン・タイム・リカバリ (PITR) の概要

CO-Solutions OSS-DB

- **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**

© LPI-Japan 2013. All rights reserved.

PITRの概念と障害回復

- 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

© LPI-Japan 2013. All rights reserved.

（参考）コールドバックアップの障害回復

- 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

© LPI-Japan 2013. All rights reserved.

PITRの構成

■ postgresql.confの設定例

```
wal_level = archive または hotstanby
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

© LPI-Japan 2013. All rights reserved.

ベースバックアップの取得手順

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

ポイントインタイムリカバリの手順

- 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

© LPI-Japan 2013. All rights reserved.

復旧コマンドファイルと復旧モード

OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

他RDBMSとの用語の差異

- 用語の差異が、「理解しにくしさ」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

© LPI-Japan 2013. All rights reserved.

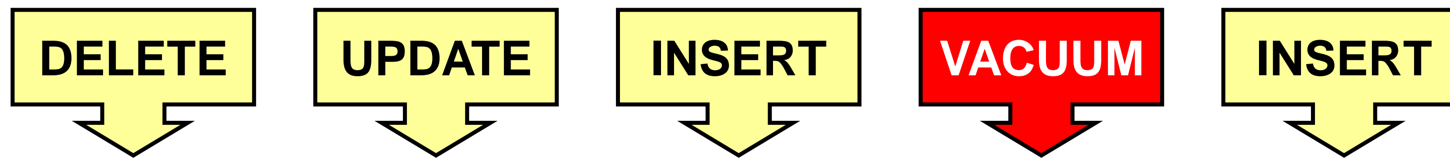
メンテナンス作業

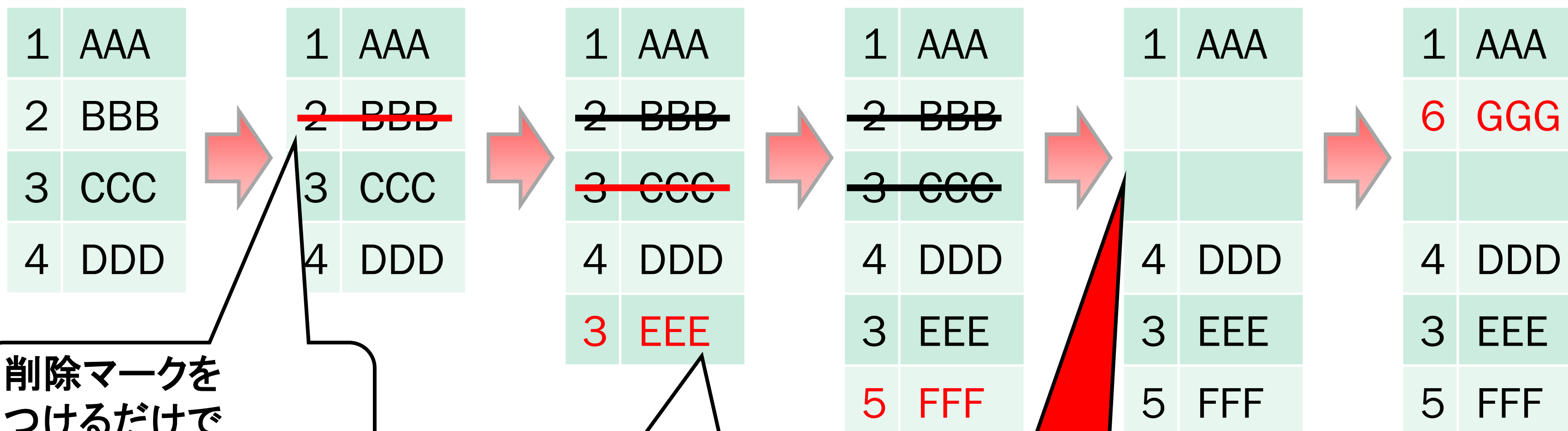
- 不要領域の回収　- いわゆるバキューム処理
  - 更新処理(UPDATE、DELETE)により生じた不要領域を回収する
  - INSERTによるデータ追加に使用できるように
  - データファイルの肥大化を抑制
- プランナ統計情報の収集
  - プランナ統計情報(＝行数、行サイズ、列値の偏りなど)を最新に更新する
  - プランナ統計情報が古いと、SQLの処理パフォーマンスが適切にならない場合がある
    - SQLの処理方法(プラン)は統計情報をもとに決定されるため

© LPI-Japan 2013. All rights reserved.

バキューム処理のイメージ

- 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**





© LPI-Japan 2013. All rights reserved.

バキュームの手動実行

- 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**

© LPI-Japan 2013. All rights reserved.

- 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

© LPI-Japan 2013. All rights reserved.

プランナ統計の手動収集

OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

OSS-DB

■ 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: b, d

© LPI-Japan 2013. All rights reserved.

例題解説4

CO-Solutions OSS-DB

■ 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: c, d

© LPI-Japan 2013. All rights reserved.

CO-Solutions OSS-DB

■ 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：d

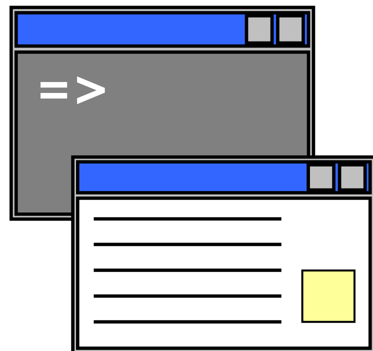
© LPI-Japan 2013. All rights reserved.

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

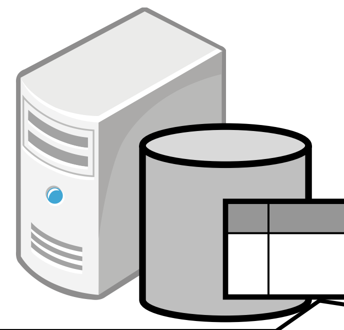
© LPI-Japan 2013. All rights reserved.

SQLとテーブル



クライアント
アプリケーション

[SQL]
SELECT * FROM emp;



テーブルemp

empno	ename	deptno
7369	SMITH	20
7499	ALLEN	30
7521	WARD	30
7566	JONES	20

- SQL: リレーショナルデータベースのデータの操作を行うときに使用される標準的な言語
- テーブル:リレーショナルデータベースにおいてデータの入れ物となる基本構造
  1つ以上の列から構成され、格納したデータが行に対応する

© LPI-Japan 2013. All rights reserved.

SQLとは

- リレーショナルデータベースにおける業界標準のデータ操作言語
  - ANSI/ISOで標準化されている
  - 基本的な部分はRDBMS製品で原則的に共通
    - 細かい部分ではRDBMS製品ごとに差異がある
- SQLの分類
  - DML(Data Manipulation Language)
    - データの検索と更新を行う
    - SELECT,INSERT, UPDATE, DELETEなど
  - DDL(Data Definition Language)
    - 表、インデックス、ビューなどの作成、変更、削除などを行う
    - CREATE TABLE, ALTER TABLE, CREATE INDEX, CREATE VIEWなど
  - DCL(Data Control Language)
    - 権限設定・トランザクション制御などを行う
    - GRANT, REVOKE, BEGIN など

© LPI-Japan 2013. All rights reserved.

（参考）SQLを基礎から学ぶためには

CO-Solutions OSS-DB

- **オープンソースデータベース標準教科書**
- http://www.oss-db.jp/ossdbtext/text.shtml
- SQL**について何も知らない人を対象に基礎から解説**
- PDF**版と**EPUB**版(スマートフォンなどで利用可能)を無料でダウンロード可能**

© LPI-Japan 2013. All rights reserved.

SQL標準とRDBMS製品

- 「標準」に完全準拠したRDBMS製品はないと考えてよい
  - 各RDBMS製品は対応可能な範囲で「標準」を取り入れている
  - RDBMS製品間で極端な違いはないが、完全な互換性はない
  - 互換性が損なわれがちなポイント
    - DML：関数、データ型の違いに起因する差異、結合の文法など
    - DDL：使用可能文字や最大数などの細かい制限、データ型など
    - DCL：権限の考え方や種類の違い、トランザクション制御の考え方の違いなど
- PostgreSQLとSQL標準
  - PostgreSQLは標準への準拠度が比較的高いが、独自拡張部分もある
  - マニュアルの多くの箇所で、ANSI標準に準拠しているかどうか、PostgreSQLの独自拡張かどうかが明記されている
- ANSI標準の策定前から存在していたRDBMSとSQL標準
  - Oracle Databaseなど
  - 製品固有の仕様が多く残っている
  - 最近のバージョンでは標準の仕様の多くが取り入れられており、標準に準拠したSQLも使用できる

© LPI-Japan 2013. All rights reserved.

SQLにおける一般的な留意点

- 大文字と小文字の区別 - コマンドやテーブル名、列名
  - コマンドやテーブル名、列名などで大文字と小文字は区別されない
    - すなわち、SELECT * FROM emp
SELECT * FROM EMPの動作は同じ
    - 理解しやすさのため、本資料内ではSQLの予約語を大文字、表名、列名などを小文字で記述している
  - 大文字・小文字を区別したい場合は、ダブルクォートで囲む
    - 「abc」と「ABC」と「Abc」は同じ扱い
    - 「"Abc"」と「Abc」は別の扱い
  - 文字列データとしては大文字と小文字を区別する
    - すなわち、SELECT * FROM emp WHERE ename = 'SCOTT'と
SELECT * FROM emp WHERE ename = 'Scott'と動作が異なる
- テーブル名、列名における日本語の使用
  - テーブル名、列名には英数字のみを使うことを推奨する
  - テーブルや列の名前に日本語(漢字)を使用しても問題なく動作することが多いが、一般的には望ましくないため

© LPI-Japan 2013. All rights reserved.

SQLにおける一般的な留意点

- 文字列リテラル（データとしての文字列）
  - 文字列リテラルはシングルクォートで囲む
    - 文字列にシングルクォートを含めたい場合は、シングルクォートを2つ続けて書く
    - 'abc''123' は abc'123 として解釈される
    - 大文字と小文字は区別される（コマンド、テーブル名や列名の扱いを混同しないこと！）
- コメント
  - 「--」（ハイフン二つ）以降は改行までコメント
  - 「/*」から「*/」までは改行を含めてコメント

© LPI-Japan 2013. All rights reserved.

表(テーブル)の作成 - 基礎編

- 表は CREATE TABLE 文で作成する
  - CREATE TABLE テーブル名 (
    列名 データ型,
    列名 データ型,
    :
    );
  - 例:
    - CREATE TABLE emp(
      empno INTEGER,
      ename VARCHAR(20)
      deptno INTEGER
      );

empno	ename	deptno
7369	SMITH	20
7499	ALLEN	30
7521	WARD	30
7566	JONES	20

- [注意]
  - テーブル作成直後では、データ(行)は入っていない
  - データ(行)の順序という概念はない(順序は保証されない)
  - 表名、列名には英数字のみを使うことを推奨する

© LPI-Japan 2013. All rights reserved.

[補足] 一般的な用語とPostgreSQLの用語

- PostgreSQLで用いられる用語と一般的に用いられる用語との対応関係を理解しておくことをお勧めします

PostgreSQLの用語 (アカデミックなRDBMS用語)	一般的に用いられる用語
リレーション (relation)	テーブル
タプル (tuple)	行
属性 (attribute)	列

- マニュアル、エラーメッセージなどでこれらの用語が使用されます

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

© LPI-Japan 2013. All rights reserved.

■数値データ型

データ型	サイズ(byte)	説明
smallint	2	16bit符号付整数
integer / int	4	32bit符号付き整数
bigint	8	64bit符号付き整数
real / float	4	単精度浮動小数点数
double precision	8	倍精度浮動小数点数
numeric/decimal	可変長	任意精度の10進実数
serial	4	自動採番のinteger
bigserial	8	自動採番のinteger

■論理値データ型

データ型	サイズ(byte)	説明
boolean / bool	1	真(true)or 偽(false)

© LPI-Japan 2013. All rights reserved.

データ型(２)

■ 文字列データ型

データ型	説明
character varying(n)、varchar(n)	文字数制限付き可変長文字列
character(n)、char(n)	文字数制限付き固定長文字列 （末尾部分は空白文字で埋められる）
char	1文字
text	制限なし可変長文字列

© LPI-Japan 2013. All rights reserved.

■ 日付・時刻データ型

データ型	サイズ(byte)	説明
timestamp timestamp without time zone	8	日付時刻(タイムゾーンなし)
date	4	日付
time time without timezone	8	時刻(タイムゾーンなし)
timestamp with time zone	12	日付時刻(タイムゾーンあり)
time with time zone	12	時刻(タイムゾーンあり)
interval	12	時間間隔

© LPI-Japan 2013. All rights reserved.

NULL値

- NULL値とは
  - 「値不定」を表す特殊な値で、空文字列とは区別される
    - Oracle Databaseでは空文字列＝NULL
  - 「NULLであるか？」の判断には専用の演算子を用いる
    - 式 IS NULL ：式がNULLであればtrue
    - 式 IS NOT NULL ：式がNULLでなければtrue
    - ［注意］ 式 = NULL では、「NULLであるか？」は判断できない
- psqlにおけるNULL値の表示
  - デフォルトでは空白で表示
  - ¥pset null でNULL値の表示方法を変更可

```
db1=> SELECT NULL;
 ?column?
----------

(1 row)
```

```
db1=> ¥pset null (NULL)
Null display is "(NULL)".
db1=> SELECT NULL;
 ?column?
----------
 (NULL)
(1 row)
```

© LPI-Japan 2013. All rights reserved.

他のRDBMSとの比較 – データ型

- 大枠で類似しているが、微妙に仕様が異なる
  - 他のRDBMSを想定して設計されたテーブル定義をそのまま流用することはできない
  - (参考) http://www.oss-db.jp/measures/dojo_05.shtml
- [注意] Oracle Databaseと意味が異なるデータ型

データ型	PostgreSQL	Oracle
integer	4バイトの整数	Oracleのデータ型としてintegerは存在しない （データ型としてintegerが指定された場合、自動的にnumber(38)に変換される）
varchar(n)	最大n文字の文字列 (n<=4096)	最大nバイト(デフォルト)の文字列 (n<=4000)
date	日付のみ	日付と時刻
time	時刻のみ	対応するデータ型なし

© LPI-Japan 2013. All rights reserved.

DML

- SELECT文
  - データを検索する
- INSERT文
  - データを挿入(追加)する
- UPDATE文
  - データを更新(修正)する
- DELETE文
  - データを削除する

- SELECT文以外の更新系SQLだけをDMLと呼ぶ場合もあるが、PostgreSQLのマニュアルでは、上記すべてをまとめてDMLと呼んでいる

© LPI-Japan 2013. All rights reserved.

SELECT 文(データ検索)

- データを検索(問い合わせ)して表示するには SELECT 文を使う
- SELECT * FROM テーブル名;
  - テーブル内のすべての行の、すべての列が返される

```
db1=> SELECT * FROM emp;
 empno | ename | deptno
-------+-------+--------
  7369 | SMITH |     20
  7499 | ALLEN |     30
  7521 | WARD  |     30
  7566 | JONES |     20
(4 rows)
```

© LPI-Japan 2013. All rights reserved.

SELECT 文(データ検索) - 列の指定

- SELECT **列名**, **列名**, ... FROM **テーブル名**;
  - **テーブル内のすべての行の、指定した列が返される**
  - **射影**(Projection)**と呼ぶ**

```
db1=> SELECT empno, ename FROM emp;
 empno | ename
-------+--------
  7369 | SMITH
  7499 | ALLEN
  7521 | WARD
  7566 | JONES
(4 rows)
```

© LPI-Japan 2013. All rights reserved.

SELECT 文(データ検索) - 列の別名指定

- SELECT **列名** AS **別名**, **列名** AS **別名**, … FROM **テーブル名**;
  - **テーブル内のすべての行の、指定した列が返される**
  - **列名の別名を指定できる**
  - **別名に予約語を含む場合は"…"で囲む**
  - **同様に最初の文字にいわゆる文字とアンダースコア以外、続く文字にいわゆる文字と数字、アンダースコア、$以外を使用する場合も"…"で囲む**

```
db1=> SELECT empno "no." , ename "Name" FROM emp;
 no.  | Name
------+-------
 7369 | SMITH
 7499 | ALLEN
 7521 | WARD
 7566 | JONES
(4 rows)
```

© LPI-Japan 2013. All rights reserved.

- SELECT **列名, 列名**, ... FROM **テーブル名** WHERE **条件**;
  - **条件(検索条件)に合致した行の、指定した列が返される**
  - **関係演算では、この操作を選択(**Selection**)と呼ぶ**

```
db1=> SELECT empno, ename FROM emp WHERE empno = 7499;
 empno | ename
-------+-------
  7499 | ALLEN
(1 row)
```

- WHERE**句に条件(検索条件)を指定する**
- **条件(検索条件)に演算子を使う**
  - **例) 比較演算子、パターンマッチ演算子**

© LPI-Japan 2013. All rights reserved.

［補足］関係演算の用語

- SELECT 文には3つの基本機能を組み合わせて、様々な問合せを実現している
  - 選択(selection)
    - 行の抽出
    - WHERE句をもちいて検索条件を指定し、条件に合う行だけを取り出す
  - 射影(projection)
    - 列の抽出
    - SELECTの列リストに指定した列だけを取り出す
  - 結合(join)
    - 複数の表を結合して1つの表として扱う
- 複雑な問い合わせも、これらの基本機能の組み合わせ

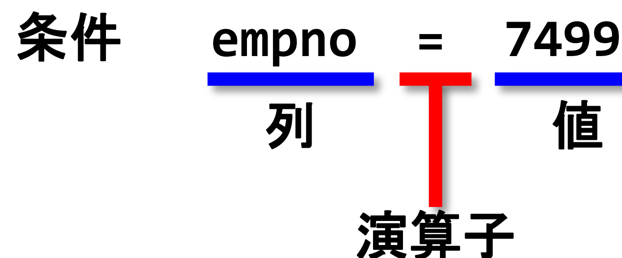
- 実務上は必ずしもこれらの用語を覚える必要はないが、試験対策のために押さえておくこと

© LPI-Japan 2013. All rights reserved.

WHERE句と条件

- SELECT ... FROM ... WHERE **条件** **でデータを絞り込む**
- **条件は、論理値(trueかfalse)を返す式**
  - true**が返された場合、データが取得される**
  - false**が返された場合、データは取得されない**
  - **条件は、論理値を返す演算子と式(列や値など)で構成される**
  - SELECT**文における条件**

```
SELECT empno, ename FROM emp WHERE empno = 7499;
```



- **複雑な条件**
  - **複数の条件を**AND**や**OR**で組み合わせてより複雑な条件をつくれる**
    - AND**でつないだ条件は、全てが**true**の場合に**true
    - OR**でつないだ条件は、いずれかが**true**の場合に**true

© LPI-Japan 2013. All rights reserved.

比較演算子

- **式(列や値など)を比較して、論理値(trueまたはfalse)を返す**

演算子	trueを返す場合
式 = 式	両辺が一致した場合
式 != 式、式 <> 式	両辺が一致しない
式 > 式	左辺が右辺より大きい（逆は「<」）
式 => 式	左辺が右辺以上（逆は「<=」）
式 BETWEEN 最少 AND 最大	式の値が「最少」以上「最大」以下の場合 「最少 <= 式 AND 式 <= 最大」と同じ
式 IN (値1, 値2, ...)	式が値1, 値2, ...のいずれかと一致した場合

© LPI-Japan 2013. All rights reserved.

- **パターンマッチ**
  - **特定のパターンにマッチしたデータを検索する方法**
  - **比較演算子を用いた条件では、単に「等しい」「大きい」「小さい」という条件しか検索に使用できない**

- **文字列** LIKE **パターン**
  - **パターンが文字列全体にマッチしたら**true（**大文字小文字区別あり**）
- **文字列** ILIKE **パターン**
  - **パターンが文字列全体にマッチしたら**true（**大文字小文字区別なし**）
- LIKE**で使用できるメタ文字**

メタ文字	マッチする文字列
%	任意の文字列
_	任意の一文字
￥	直後のメタ文字をエスケープ

© LPI-Japan 2013. All rights reserved.

SIMILAR TO演算子 – パターンマッチ

CO-Solutions OSS-DB

- 文字列 SIMILAR TO パターン
  - パターンが文字列全体にマッチしたらtrue
- SIMILAR TOで使用できるメタ文字

メタ文字	マッチする文字列
%	任意の文字列
_	任意の一文字
¥	直後のメタ文字をエスケープ
*	直前の項目の0回以上の繰り返し
+	直前の項目の1回以上の繰り返し
?	直前の項目の0回または1回の繰り返し
{m}	直前の項目のm回の繰り返し
{m,n}	直前の項目のm回からn回の繰り返し（nを省略すると上限なし）
(パターン)	パターンをグループ化

© LPI-Japan 2013. All rights reserved.

正規表現を使用してパターンマッチ

- LIKEやSIMILAR TOのメタ文字を強化したものと考えればよい
- 正規表現の詳細については http://www.postgresql.jp/document/9.0/html/functions-matching.html#POSIX-SYNTAX-DETAILS を参照

正規表現マッチ演算子

演算子	説明	trueとなる例
~	正規表現に一致する場合true (大文字小文字の区別あり)	'abc' ~ 'b.'
~*	正規表現に一致する場合true (大文字小文字の区別なし)	'abc' ~* 'B.'
!~	正規表現に一致しない場合true (大文字小文字の区別あり)	'abc' !~ 'B.'
!~*	正規表現に一致しない場合true (大文字小文字の区別なし)	'abc' !~* 'x.'

※: ピリオド '.'は任意の1文字にマッチする正規表現

© LPI-Japan 2013. All rights reserved.

検索結果のソート

- ORDER BY **列 または 式 または 位置番号のリスト**
  - **カンマ区切りで複数指定可能**
    - ORDER BY empno　　　　　　　:empno**列でソート**
    - ORDER BY empno, ename　　　:empno**列でソート、**
      empno**列が同じ場合は**ename**でソート**
  - **ソート基準の各項目ごとに、昇順・降順を指定可能**
    - **ソート項目** ASC　　　:**昇順（デフォルト）**
    - **ソート項目** DESC　　:**降順**
  - **基本的に**ORDER BY**句を一番最後に指定する**
- **[注意]** ORDER BY**を指定しないと、**SELECT**結果の並び順は不定**

© LPI-Japan 2013. All rights reserved.

- LIMIT句：行数を制限
- OFFSET 句：表示しない行数を指定

```
db1=> SELECT * FROM emp ORDER BY empno LIMIT 3;
 empno | ename | deptno
-------+-------+--------
  7369 | SMITH |     20
  7499 | ALLEN |     30
  7521 | WARD  |     30
(3 rows)

db1=> SELECT * FROM emp ORDER BY empno LIMIT 2 OFFSET 1;
 empno | ename | deptno
-------+-------+--------
  7499 | ALLEN |     30
  7521 | WARD  |     30
(2 rows)
```

© LPI-Japan 2013. All rights reserved.

検索結果のソートと行数制限

LIMIT句の注意点

- 記述位置はORDER BY句の後（SELECT文の最後）
- 原則的にORDER BY 句とセットで使用する
  - ORDER BY句を指定しないと並び順が不定のため、行数制限しても意味がない

```
db1=> SELECT * FROM emp ORDER BY empno LIMIT 3;
 empno | ename | deptno
-------+-------+--------
  7369 | SMITH |     20
  7499 | ALLEN |     30
  7521 | WARD  |     30
(3 rows)

db1=> SELECT * FROM emp LIMIT 3;
 empno | ename | deptno
-------+-------+--------
  7566 | JONES |     20
  7499 | ALLEN |     30
  7521 | WARD  |     30
(3 rows)
```

© LPI-Japan 2013. All rights reserved.

重複行の排除 - DISTINCT句

OSS-DB

- SELECT DISTINCT **列名リスト** FROM **テーブル名**
  - DISTINCT**の後の全ての項目が一致するもののみ除去される**

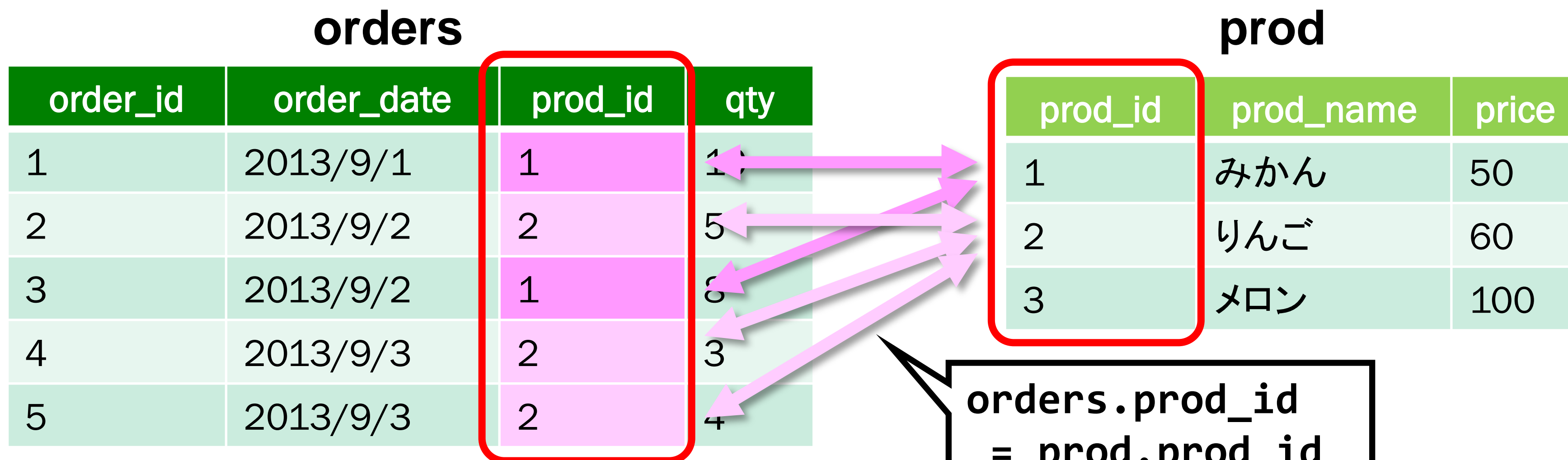
```
db1=> SELECT deptno FROM emp;
 deptno
--------
     20
     30
     30
     20
(4 rows)

db1=> SELECT DISTINCT deptno FROM emp;
 deptno
--------
     20
     30
(2 rows)
```

© LPI-Japan 2013. All rights reserved.

SELECT 文 - 結合(JOIN)

OSS-DB

■ 複数の表のデータを結びつける



order_id	order_date	prod_id	prod_name	qty
1	2013/9/1	1	みかん	10
2	2013/9/2	2	りんご	5
3	2013/9/2	1	みかん	8
4	2013/9/3	2	りんご	3
5	2013/9/3	2	りんご	4

© LPI-Japan 2013. All rights reserved.

結合のSQL構文

OSS-DB

```
=> SELECT ord_id, ord_date, pname FROM orders , prod
->    WHERE orders.prod_id = prod.prod_id;

 ord_id |  ord_date   | pname
--------+-------------+--------
      3 | 2013-09-02 | みかん
      1 | 2013-09-01 | みかん
      5 | 2013-09-03 | りんご
      4 | 2013-09-03 | りんご
      2 | 2013-09-02 | りんご
(5 rows)
```

FROM句にカンマ区切りで表をリスト
WHERE句に結合条件を記載

FROM句にJOIN句の左右に表を記載
ON句に結合条件を記載

```
=> SELECT ord_id, ord_date, pname FROM orders JOIN prod
->    ON orders.prod_id = prod.prod_id;

 ord_id |  ord_date   | pname
--------+-------------+--------
                 :
```

結合のSQL構文


```
=> SELECT ord_id, ord_date, pname FROM orders JOIN prod
->   USING (prod_id);
```

結合に使用する列名が同じ場合は
USING句で表記を簡単にできる

```
=> SELECT ord_id, ord_date, pname FROM orders
->  NATURAL JOIN prod;
```

同じ列名を持つ列を結合条件とする場合は
NATURAL JOIN句で列名の指定を省略できる

© LPI-Japan 2013. All rights reserved.

列名にテーブル名を指定する

```
db1=> ¥d tab1
             Table "public.tab1"
 Column |          Type           | Modifiers
--------+-------------------------+-----------
 id1    | numeric(2,0)            |
 col    | character varying(14) |

db1=> ¥d tab2
             Table "public.tab2"
 Column |          Type           | Modifiers
--------+-------------------------+-----------
 id2    | numeric(2,0)            |
 col    | character varying(14) |

db1=> SELECT id1, col, id2 FROM tab1, tab2
db1->   WHERE tab1.id1 = tab2.id2;
ERROR:  column reference "col" is ambiguous
LINE 1: SELECT id1, col, id2 FROM tab1, tab2
                    ^
```

結合するテーブルに同じ名前の列があると列を区別できない

列名にテーブル名を指定する

```
db1=> SELECT id1, tab1.col, id2 FROM tab1, tab2
db1->   WHERE tab1.id1 = tabl2.id2;
 id1 | col | id2
-----+-----+-----
   1 | A   |   1
   2 | B   |   2
(2 rows)
```

列名の前に
テーブル名を明示する

テーブル名に別名をつけると
テーブル名を別名で参照できるため、表記がシンプルに

```
db1=> SELECT id1, a.col, id2 FROM tab1 a, tab2 b
db1->   WHERE a.id1 = b.id2;
db1=> SELECT id1, a.col, id2 FROM tab1 AS a, tab2 AS b
db1->   WHERE a.id1 = b.id2;
```

© LPI-Japan 2013. All rights reserved.

結合条件に合致しない行の扱い

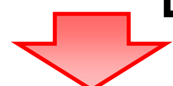
orders

order_id	order_date	prod_id	qty
1	2013/9/1	1	10
2	2013/9/2	2	5
3	2013/9/2	1	8
4	2013/9/3	2	3
5	2013/9/3	2	4

prod

prod_id	prod_name	price
1	みかん	50
2	りんご	60
3	メロン	100

```
orders.prod_id
 = prod.prod_id
```



order_id	order_date	prod_id	prod_name	qty
1	2013/9/1	1	みかん	10
2	2013/9/2	2	りんご	5
3	2013/9/2	1	みかん	8
4	2013/9/3	2	りんご	3
5	2013/9/3	2	りんご	4

→ prod_id=3 (メロン)に対応するデータが表示されない

© LPI-Japan 2013. All rights reserved.

外部結合（OUTER JOIN）

OSS-DB

```
=> SELECT ord_id, ord_date, pname FROM orders JOIN prod
->    ON orders.prod_id = prod.prod_id;
 ord_id |  ord_date   | pname
--------+------------+--------
      3 | 2013-09-02 | みかん
      1 | 2013-09-01 | みかん
      5 | 2013-09-03 | りんご
      4 | 2013-09-03 | りんご
      2 | 2013-09-02 | りんご
(5 rows)
db1=> SELECT ord_id, ord_date, pname
db1->    FROM orders RIGHT OUTER JOIN prod
db1->    ON orders.prod_id = prod.prod_id;
 ord_id |  ord_date   | pname
--------+------------+--------
      1 | 2013-09-01 | みかん
      3 | 2013-09-02 | みかん
      2 | 2013-09-02 | りんご
      4 | 2013-09-03 | りんご
      5 | 2013-09-03 | りんご
        |            | メロン
(6 rows)
```

通常の結合では表示されなかった
prod_id=3（メロン）のデータが
表示された

© LPI-Japan 2013. All rights reserved.

外部結合の種類

- 結合条件に合致しない行を表示するにはOUTER JOIN(外部結合)を使う
  - LEFT OUTER JOIN :左側の表に対応行がなくてもOK
  - RIGHT OUTER JOIN :右側の表に対応行がなくてもOK
  - FULL OUTER JOIN :どちらの表に対応行がなくてもOK
  - 「OUTER」は省略可能
    - テーブルA LEFT JOIN テーブルB ON 結合条件
  - 対応行がなかったテーブル側の値はNULL
- [注意] 外部結合の注意点
  - WHERE句で結合条件を書けない
    - 通常の結合でも外部結合でも、常に JOIN ・・・ ON 結合条件 という構文を用いればいちいち考えなくて済む
    - Oracle Databaseでは独自の「(+)」記法を使用して、外部結合でWHERE句に結合条件を書けるようになっている

© LPI-Japan 2013. All rights reserved.

- SELECT 文で、データを集約(合計、平均、最大、最小などを計算)できる

```
db1=> SELECT * FROM orders;
 ord_id |  ord_date   | prod_id | qty
--------+------------+---------+-----
      1 | 2013-09-01 |       1 |  10
      2 | 2013-09-02 |       2 |   5
      3 | 2013-09-02 |       1 |   8
      4 | 2013-09-03 |       2 |   3
      5 | 2013-09-03 |       2 |   4
(5 rows)

db1=> SELECT max(qty), min(qty) FROM orders;
 max | min
-----+-----
  10 |   3
(1 row)
```

列名ではなく、集約関数を指定

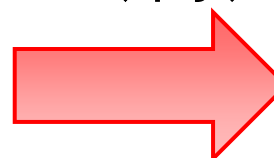
© LPI-Japan 2013. All rights reserved.

集約関数

■ 複数のデータを受け取り、集約した値を返す

```
 ord_id |  ord_date   | prod_id | qty
--------+-------------+---------+-----
      1 | 2013-09-01 |       1 |  10
      2 | 2013-09-02 |       2 |   5
      3 | 2013-09-02 |       1 |   8
      4 | 2013-09-03 |       2 |   3
      5 | 2013-09-03 |       2 |   4
```

sum(qty)



```
 max
-----
  30
```

■ 主な集約関数

集約関数	結果
count()	件数を返す。引数に*を指定した場合、NULLもカウントする
max()	最大値を返す
min()	最小値を返す
sum()	合計値を返す
avg()	平均値を返す

© LPI-Japan 2013. All rights reserved.

- **指定した列の値に応じてグループ分けし、グループ単位で集約する**
- SELECT prod_id, sum(qty) FROM orders GROUP BY prod_id;

```
 ord_id |  ord_date   | prod_id | qty
--------+-------------+---------+-----
      1 | 2013-09-01 |       1 |  10
      2 | 2013-09-02 |       2 |   5
      3 | 2013-09-02 |       1 |   8
      4 | 2013-09-03 |       2 |   3
      5 | 2013-09-03 |       2 |   4
```

GROUP BY prod_id

```
 ord_id |  ord_date   | prod_id | qty
--------+-------------+---------+-----
      1 | 2013-09-01 |       1 |  10
      3 | 2013-09-02 |       1 |   8
```

```
 ord_id |  ord_date   | prod_id | qty
--------+-------------+---------+-----
      2 | 2013-09-02 |       2 |   5
      4 | 2013-09-03 |       2 |   3
      5 | 2013-09-03 |       2 |   4
```

prod_id,
sum(qty)

```
 prod_id | sum
---------+-----
       1 |  18
```

```
 prod_id | sum
---------+-----
       2 |  12
```

© LPI-Japan 2013. All rights reserved.

GROUP BY句とWHERE句の組み合わせ

OSS-DB

■ WHERE句による絞り込みの後、GROUP BYによるグループ化＋集約

```
db1=> SELECT prod_id, sum(qty) FROM orders
db1->   WHERE ord_id < 5
db1->   GROUP BY prod_id;
 prod_id | sum
---------+-----
       1 |  18
       2 |   8
(2 rows)
```

■ GROUP BYを指定しない実行結果と比較すると理解しやすいか

```
db1=> SELECT prod_id, qty FROM orders
db1->   WHERE ord_id < 5 ORDER BY prod_id ;
 prod_id | qty
---------+-----
       1 |  10
       1 |   8
       2 |   5
       2 |   3
(4 rows)
```

© LPI-Japan 2013. All rights reserved.

- 集約したデータをHAVING句に指定した条件で絞り込む
- WHERE句を用いた絞り込みとの違いに注意
  - HAVING句による絞り込みタイミング　：GROUP BY の後
  - WHERE句による絞り込みタイミング　：GROUP BY の前

```
db1=> SELECT prod_id, sum(qty) FROM orders
db1->    GROUP BY prod_id;
 prod_id | sum
---------+-----
       1 |  18
       2 |  12
(2 rows)

db1=> SELECT prod_id, sum(qty) FROM orders
db1->    GROUP BY prod_id
db1->    HAVING sum(qty) > 15;
 prod_id | sum
---------+-----
       1 |  18
(1 row)
```

© LPI-Japan 2013. All rights reserved.

HAVING句に指定する条件

OSS-DB

```
db1=> SELECT prod_id, sum(qty) FROM orders
db1->   GROUP BY prod_id;
 prod_id | sum
---------+-----
       1 |  18
       2 |  12
(2 rows)

db1=> SELECT prod_id, sum(qty) FROM orders
db1->   GROUP BY prod_id HAVING sum(qty) > 15;
 prod_id | sum
---------+-----
       1 |  18
(1 row)

db1=> SELECT prod_id, sum(qty) FROM orders
db1->   GROUP BY prod_id HAVING prod_id = 2;
 prod_id | sum
---------+-----
       2 |  12
(1 row)
```

HAVING句の条件には集約関数を適用した値か、

グループを一意に識別する値を指定できる

© LPI-Japan 2013. All rights reserved.

集約の留意点

- **処理順序とデータの絞り込み**
  - WHERE, GROUP BY, HAVING**の処理順序**
    - 1. **テーブルの行のうち、**WHERE**句 の条件に合致した行を抽出**
    - 2. GROUP BY**句 の式(列など)に従ってグループ分けして集約**
    - 3. **集約した結果(行)のうち、**HAVING**句 の条件に合致した行を抽出**
  - **絞り込み条件を指定するときの考え方**
    - WHERE ： GROUP BY**による集約処理前に適用したい絞り込み条件**
    - HAVING: **集約処理後でないと適用できない絞り込み条件**

© LPI-Japan 2013. All rights reserved.

集約の留意点

- WHERE句に指定できる条件、HAVINGに指定できる条件
  - エラーとなる SELECT の例:
    - SELECT cid, count(*), avg(score) FROM exam WHERE avg(score) > 75
GROUP BY cid;
    - SELECT cid, count(*), avg(score) FROM exam GROUP BY cid
HAVING grade = 'Pass';
- GROUP BY句に指定していない列はSELECT句に指定できない
  - SELECT emp_id, min(emp_date) FROM employee GROUP BY dept_id;
    - emp_idはグループ基準にない
- NULL値の扱い
  - 集約対象の値がNULL値の行は集約対象に含まれない
    - {1,2,NULL}という集合の平均は「1.5」、個数は「2」

© LPI-Japan 2013. All rights reserved.

副問い合わせ

OSS-DB

- 副問い合わせとは
  - あるSELECT文の中に埋め込んだSELECT文
  - 3つの埋め込みパターンがあるが、WHERE句にSELECT文を埋め込むパターンを抑えておけばSilver対策には十分なはず
- 1. WHERE句にSELECT文を埋め込むパターン
  - 具体的には、条件内の演算子の右辺(または左辺)に副問い合わせを埋め込む
  - IN演算子の右辺に副問い合わせを埋め込んだ例

```
db1=> SELECT pname FROM prod
db1->   WHERE prod_id IN
db1->     (SELECT prod_id FROM orders WHERE qty > 4);
 pname
--------
 みかん
 りんご
(2 rows)
```

(…) が副問い合わせ

© LPI-Japan 2013. All rights reserved.

■ 2. FROM句にSELECT文を埋め込むパターン

```
db1=> SELECT * FROM orders;
 ord_id |  ord_date   | prod_id | qty
--------+------------+---------+-----
      1 | 2013-09-01 |       1 |  10
      2 | 2013-09-02 |       2 |   5
      3 | 2013-09-02 |       1 |   8
      4 | 2013-09-03 |       2 |   3
      5 | 2013-09-03 |       2 |   4
(5 rows)

db1=> SELECT AVG(count_by_date) AS avg_orders
db1->   FROM (SELECT count(*) AS count_by_date
db1(>           FROM orders
db1(>           GROUP BY ord_date
db1(>        ) AS A;
     avg_orders
--------------------
 1.6666666666666667
(1 row)
```

© LPI-Japan 2013. All rights reserved.

■ 3. SELECT句の列リストにSELECT文を埋め込むパターン

```
db1=> SELECT * FROM orders;
 ord_id |  ord_date   | prod_id | qty
--------+------------+---------+-----
      1 | 2013-09-01 |       1 |  10
      2 | 2013-09-02 |       2 |   5
      3 | 2013-09-02 |       1 |   8
      4 | 2013-09-03 |       2 |   3
      5 | 2013-09-03 |       2 |   4
(5 rows)

db1=> SELECT prod_id, count(*), (SELECT count(*) FROM orders) count_all
db1->   FROM orders
db1->   GROUP BY prod_id;
 prod_id | count | count_all
---------+-------+-----------
       1 |     2 |         5
       2 |     3 |         5
(2 rows)
```

© LPI-Japan 2013. All rights reserved.

複数の値を返す副問い合わせ

CO-Solutions OSS-DB

- 演算子によっては、副問い合わせの結果が複数件になるとエラーとなる
  - IN演算子は結果が複数件になってもエラーにならない

```
db1=> SELECT pname FROM prod
db1->   WHERE prod_id IN
db1->     (SELECT prod_id FROM orders WHERE qty > 4);
 pname
--------
 みかん
 りんご
(2 rows)

db1=> SELECT pname FROM prod
db1->   WHERE prod_id =
db1->     (SELECT prod_id FROM orders WHERE qty > 4);
ERROR:  more than one row returned by a subquery used as an expression
db1=> SELECT pname FROM prod
db1->   WHERE prod_id =
db1->     (SELECT prod_id FROM orders WHERE qty > 9);
 pname
--------
 みかん
(1 row)
```

© LPI-Japan 2013. All rights reserved.

複数の値を返す副問い合わせと演算子

- 副問い合わせが複数の値を返す場合に使える演算子
  - 式 IN (副問い合わせ)
    - 式と副問い合わせ結果のいずれかが一致する場合trueとなる
  - 式 演算子 ANY (副問い合わせ)
    - 式と副問い合わせ結果のいずれかとの演算結果がtrueの場合trueとなる
  - EXISTS (副問い合わせ)
    - 副問い合わせが1件以上結果を返した場合trueとなる

© LPI-Japan 2013. All rights reserved.

SELECT文の整理

- SELECT文の形式
  - SELECT 式[, 式...]
    FROM テーブル[JOIN テーブル ON 結合条件...]
    WHERE 絞り込み条件[AND 絞り込み条件...]
    GROUP BY 集約列[, 集約列...]
    HAVING 集約後絞り込み条件[AND 集約後絞り込み条件...]
    ORDER BY ソート基準
    OFFSET スキップ件数
    LIMIT 抽出件数;
- 処理順に注意
  - FROM句の結合
    → WHERE句の絞り込み
    → GROUP BY句の集約 →HAVING句の絞り込み
    → ORDER BY句のソート →OFFSET句のスキップとLIMIT句の件数制限

© LPI-Japan 2013. All rights reserved.

INSERT, UPDATE, DELETE

- INSERT文
  - データを挿入（追加）する
- UPDATE文
  - データを更新（修正）する
- DELETE文
  - データを削除する

© LPI-Japan 2013. All rights reserved.

データの挿入 - INSERT

CO-Solutions OSS-DB

- **データを挿入するには**INSERT**文を使う**
- INSERT**文の構文**
  - INSERT INTO **テーブル** [(**列名リスト**)] VALUES (**値リスト**);
- INSER**文の動作**
  - **指定した値を持つ行を挿入**
  - **列名リストに指定しなかった列には**NULL**値が入る**
  - **列名リストを省略した場合は、全列分の値を値リストに記述する**
- INSERT INTO **テーブル** [(**列名リスト**)] SELECT**文**; **という構文もある**
  - SELECT**結果をそのまま挿入(複数行一括挿入も可能)**
  - **列名リストと**SELECT**結果は同じ項目数でないとエラー**

© LPI-Japan 2013. All rights reserved.

INSERTの実行例


```
db1=> -- 列名を指定したINSERT文の実行例
db1=> INSERT INTO prod1 (prod_id, pname, price)
db1->    VALUES(1, 'トマト', 98);
INSERT 0 1
db1=> -- 列名指定を省略したINSERT文の実行例
db1=> INSERT INTO prod1 VALUES(2, 'にんじん', 40);
INSERT 0 1
db1=> -- 列値の指定を省略したINSERT文の実行例
db1=> -- → 省略した列の値はNULLとなる
db1=> INSERT INTO prod1 VALUES(3, 'だいこん');
INSERT 0 1
db1=> SELECT * FROM prod1;
 prod_id |  pname    | price
---------+----------+-------
       1 | トマト    |    98
       2 | にんじん  |    40
       3 | だいこん  |
(3 rows)
```

© LPI-Japan 2013. All rights reserved.

データの更新 – UPDATE文

CO-Solutions OSS-DB

- データを更新するにはUPDATE文を使う
- UPDATE文の構文
  - UPDATE テーブル SET 列 = 値 [,列 = 値...] WHERE 条件;
  - UPDATE テーブル (列リスト) = (値リスト) WHERE 条件;
    - 列リストと値リストはカンマ区切り
- UPDATE文の動作
  - 条件を満たす行を指定した値で更新する
    - 条件の書き方はSELECT文と同じ
  - 値にはその行の列の値を使用できる
    - SET count = count + 1
- [注意]
  - WHERE 句を省略すると、すべての行が更新される
  - psqlではオートコミットが有効であるため、UPDATE実行でデータは更新＋コミットされ、取り消しできない

© LPI-Japan 2013. All rights reserved.

UPDATE文の実行例


```
db1=> SELECT * FROM prod1 ORDER BY prod_id;
 prod_id |   pname    | price
---------+------------+-------
       1 | トマト     |    98
       2 | にんじん |    40
       3 | だいこん |
(3 rows)

db1=> UPDATE prod1 SET pname = 'すいか' WHERE prod_id = 1;
UPDATE 1
db1=> UPDATE prod1 SET price = price + 10 WHERE prod_id = 2;
UPDATE 1
db1=> SELECT * FROM prod1 ORDER BY prod_id;
 prod_id |   pname    | price
---------+------------+-------
       1 | すいか     |    98
       2 | にんじん |    50
       3 | だいこん |
(3 rows)
```

© LPI-Japan 2013. All rights reserved.

データの削除 - DELETE文

- データを削除するには DELETE 文を使う
- DELETE文の構文
  - DELETE FROM テーブル名 WHERE 条件;
- [注意]
  - WHERE 句を省略すると、すべての行が削除される
    - (参考)すべての行を削除するには、DELETE よりも TRUNCATE が高速 TRUNCATE [TABLE] テーブル名;
  - psqlではオートコミットが有効であるため、DELETE実行でデータは削除＋コミットされ、取り消しできない

© LPI-Japan 2013. All rights reserved.

```
db1=> SELECT * FROM prod2;
 prod_id | pname  | price
---------+--------+-------
       1 | みかん |    50
       2 | りんご |    50
       3 | メロン |   100
(3 rows)

db1=> DELETE FROM prod2 WHERE price > 70;
DELETE 1
db1=> SELECT * FROM prod2;
 prod_id | pname  | price
---------+--------+-------
       1 | みかん |    50
       2 | りんご |    50
(2 rows)
```

© LPI-Japan 2013. All rights reserved.

制約

制約とは

- テーブルに格納されたデータを制限する方法
- データが制約に合致しない場合、挿入・更新時にエラーとなる
- 不適切なデータが格納されることを抑制できる
  - ただし、制約だけでデータの整合性を担保するのは難しいため、アプリケーション側でもデータチェックの仕組みを用意するのが一般的なやり方

制約の種類

- NOT NULL（非NULL）制約
- 検査（チェック）制約
- 主キー（プライマリーキー）制約
- 一意（ユニーク）制約
- 外部キー（参照整合性）制約

© LPI-Japan 2013. All rights reserved.

NOT NULL制約

- NOT NULL制約とは
  - 列にNULL値を格納することを許さない制約
  - 列に必ず何らかの値が格納されることを保証する
- 定義方法
  - テーブル作成時
    - CREATE TABLE **テーブル名** (**列名 データ型** NOT NULL, ...);
  - 既存列をNULL値不可に変更
    - ALTER TABLE **テーブル名** ALTER COLUMN **列名** SET NOT NULL;
  - 既存列をNULL値可に変更
    - ALTER TABLE **テーブル名** ALTER COLUMN **列名** SET NULL;

© LPI-Japan 2013. All rights reserved.

検査制約(チェック制約)

- **検査制約(チェック制約)とは**
  - **条件を満たさないデータを格納することを許さない制約**
- **定義方法**
  - **テーブル作成時**
    - CREATE TABLE **テーブル名** (**列名 データ型** CHECK(**検査制約式**), ...);
または
CREATE TABLE **テーブル名** (**列定義**..., CONSTRAINT **制約名** CHECK(**検査制約式**));
  - **既存テーブルに制約を追加**
    - ALTER TABLE **テーブル名** ADD CONSTRAINT **制約名** CHECK(**検査制約**);
- **検査制約式の書き方**
  - SQL**の条件式(論理値を返す式)で記述する**
  - **例**) price > 0 ： price**列は必ず正の値**

© LPI-Japan 2013. All rights reserved.

一意制約(ユニーク制約)

- 一意制約(ユニーク制約)とは
  - ある列の値(または複数の列における値の組)が、テーブル内で一意であることを保証する制約
  - 列の値(または複数の列における値の組)が、テーブル内で重複する場合はデータの格納を拒否する
- 定義方法
  - テーブル作成時
    - CREATE TABLE **テーブル名** (**列名 データ型** UNIQUE, ...);
    - CREATE TABLE **テーブル名** (**列定義**..., CONSTRAINT **制約名** UNIQUE(**列名**[, **列名**...]));
  - 既存テーブルに制約を追加
    - ALTER TABLE **テーブル名** ADD CONSTRAINT **制約名** UNIQUE(**列名**[, **列名**...]);
- [注意]
  - 制約を付与した列に対して、自動的にインデックスが作成される
  - (1, NULL)と(1, NULL)のような組み合わせは重複していないとみなす
    - NULL値は比較できない不明な値とみなされるため

© LPI-Japan 2013. All rights reserved.

主キー制約(プライマリキー制約)

- **主キー制約(プライマリキー制約)とは**
  - **テーブル内のデータを一意に識別できるようにする制約**
  - **一意制約**＋NOT NULL**制約**
- **定義方法**
  - **テーブル作成時**
    - CREATE TABLE **テーブル名** (**列名 データ型** PRIMARY KEY, ...);
    - CREATE TABLE **テーブル名** (**列定義**..., CONSTRAINT **制約名** PRIMARY KEY(**列名**[, **列名**...]));
  - **既存テーブルに制約を追加**
    - ALTER TABLE **テーブル名** ADD CONSTRAINT **制約名** PRIMARY KEY(**列名**[, **列名**...]);
- **[注意]**
  - **制約を付与した列に対して、自動的にインデックスが作成される**
  - **テーブルに一つだけ定義可能**

© LPI-Japan 2013. All rights reserved.

外部キー制約(参照整合性制約)

- 外部キー制約(参照整合性制約)とは
  - 参照先テーブルにないデータを拒否する制約
- 定義方法
  - テーブル作成時
    - CREATE TABLE **テーブル名** (**列名 データ型** REFERENCES **参照先テーブル名** (**列名**), ...);
    - CREATE TABLE **テーブル名** (**列定義**..., CONSTRAINT **制約名** FOREIGN KEY (**列名**[, **列名**...])REFERENCES **参照先テーブル名** (**列名**[, **列名**...]));
  - 既存テーブルに制約を追加
    - ALTER TABLE **テーブル名** ADD CONSTRAINT **制約名** PRIMARY KEY(**列名**[, **列名**...])REFERENCES **参照先テーブル名** (**列名**[, **列名**...]);
- [注意]
  - 外部キーがあるとデータの挿入や削除の順序に考慮が必要

© LPI-Japan 2013. All rights reserved.

制約の削除と制約名

- 制約の削除方法
  - ALTER TABLE DROP CONSTRAINT **制約名**;
- 制約名の確認
  - ¥d **テーブル名 などで制約名を確認できる**
  - **制約作成時に制約名の指定を省略した場合、自動的に名前が付けられる**

```
db1=> ¥d products
     Table "public.products"
   Column    |  Type    | Modifiers
------------+---------+-----------
 product_no | integer |
 name       | text    |
 price      | numeric |
Check constraints:
    "products_price_check" CHECK (price > 0::numeric)

db1=> ALTER TABLE products DROP CONSTRAINT products_price_check;
ALTER TABLE
```

© LPI-Japan 2013. All rights reserved.

デフォルト値

- **デフォルト値とは**
  - **データ挿入時に値が省略された列に使用する値**
- **定義方法**
  - **テーブル作成時**
    - CREATE TABLE **テーブル名** (**列名 データ型** DEFAULT **デフォルト値**, ...);
  - **既存列にデフォルト値を設定**
    - ALTER TABLE **テーブル名** ALTER COLUMN **列名** SET DEFAULT **デフォルト値**;
  - **既存列のデフォルト値を削除**
    - ALTER TABLE **テーブル名** ALTER COLUMN **列名** DROP DEFAULT;

© LPI-Japan 2013. All rights reserved.

テーブル定義の変更

- テーブル名と列名の変更
  - ALTER TABLE **テーブル名** RENAME TO **新しい名前**;
  - ALTER TABLE **テーブル名** RENAME COLUMN **列名** TO **新しい名前**;
- データ型の変更
  - ALTER TABLE **テーブル名** ALTER COLUMN **列名** SET DATA TYPE **新しいデータ型** [USING **式**];
  - USING**句は新しいデータ型にキャストできない場合に必要**
- 列の追加・削除
  - ALTER TABLE **テーブル名** ADD COLUMN **列名 データ型 制約**;
    - **列定義部分は**CREATE TABLE**文と同じ**
    - **新しい列は最後に追加される**
  - ALTER TABLE **テーブル名** DROP COLUMN **列名**;

© LPI-Japan 2013. All rights reserved.

テーブルの削除

- DROP TABLE テーブル名;
- 格納されたデータもまとめて削除する

© LPI-Japan 2013. All rights reserved.

他のRDBMSとの差異 - テーブルのDDL

- CREATE TABLE
  - 文法上の差異はほとんどない
  - ただし、データ型の差異があるため、既存のCREATE TABLE文を機械的に流用できない
- ALTER TABLE
  - 文法上の細かい差異が多い
  - 差異が発生する主なポイント
    - ADD/DROP/ALTER/RENAME の後に COLUMN と書くかどうか
    - ADD などで指定する列定義を括弧で囲うかどうか
    - 列属性の変更は ALTER か MODIFY か

© LPI-Japan 2013. All rights reserved.

インデックス（索引）

CO-Solutions OSS-DB

- インデックスとは
  - テーブル内のデータへ効率的にアクセスするためのオブジェクト
  - WHERE条件に指定される列に対してインデックスを作成することで、一般にSQLの処理効率を高めることができる
  - 様々な種類のインデックスが存在するが、Bツリーインデックスが最もよく使用される
- Bツリーインデックスの作成方法
  - CREATE INDEX インデックス名 ON テーブル名 (列名);
  - 複数列に対してインデックスを作成することも可能
    CREATE INDEX インデックス名 ON テーブル名 (列名, 列名, ...);

© LPI-Japan 2013. All rights reserved.

ビュー

- ビューとは
  - SELECT文に名前をつけて、クエリ内で単なるテーブルのように使えるようにしたもの
    - アプリケーション側で複雑なクエリを記述してなくても済む
    - ビューの実装(具体的な検索方法)を隠蔽できる
  - ビューは読み取り専用で、書き込みはできない
- ビューの定義方法
  - CREATE VIEW ビュー名 AS SELECT ... ;
- ビューの削除方法
  - DROP VIEW ビュー名;

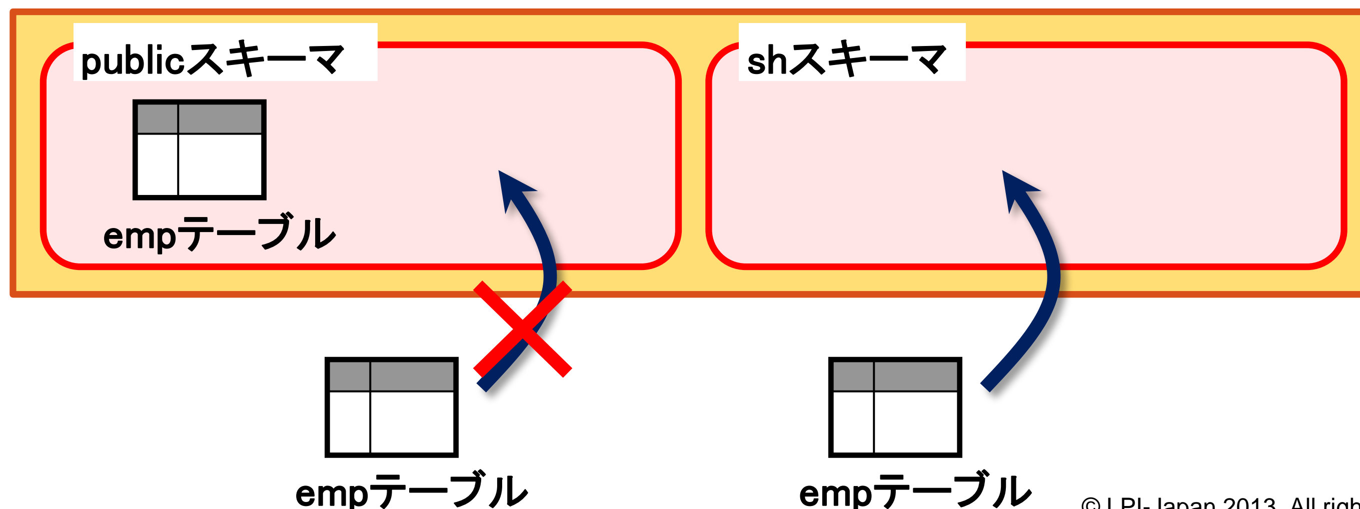
© LPI-Japan 2013. All rights reserved.

スキーマと名前空間

OSS-DB

- 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
    - 「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
  - オブジェクト名の名前空間として機能する
    - 別のスキーマであれば、同じ名前のオブジェクトを作成可能
- 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
  - ひとつのデータベースに複数のスキーマを作成できる

データベース



© LPI-Japan 2013. All rights reserved.

スキーマ

- スキーマの作成
  - CREATE SCHEMA スキーマ名;
- スキーマの削除
  - DROP SCHEMA スキーマ名;
  - "public"スキーマを削除することもできる

- オブジェクトの検索ルールとスキーマ
  - search_pathパラメータの設定に沿ってスキーマを探索
  - デフォルトは'"$user",public'("$user"は接続ユーザー名)
    - まず、接続ユーザー名と同名のスキーマ内のオブジェクトを検索
    - 次にpublicスキーマ内のオブジェクトを検索
- 新規作成オブジェクトの格納先スキーマ
  - スキーマを指定しないとsearch_path内の最初のスキーマに格納される
  - スキーマを指定すると、指定したスキーマに格納される

© LPI-Japan 2013. All rights reserved.

シーケンス

OSS-DB

■ シーケンスとは

- ■ 連番を作成するオブジェクト
- ■ ただし、トランザクションをロールバックした場合など、番号に「飛び」が発生する可能性がある
  - ■ 完全な連番を生成することよりも、処理の高速性を優先している

■ シーケンスの定義方法

- ■ CREATE SEQUENCE **シーケンス名**[INCREMENT [BY] **増分**] [START [WITH] **初期値**];
- ■ **初期値のデフォルトは**1、**増分のデフォルトは**1

■ シーケンスの操作関数

- ■ **連番を取得する** ： nextval('**シーケンス名**');
- ■ **取得した連番を参照する** ： currval('**シーケンス名**');
- ■ **連番をリセットする** ： setval('**シーケンス名**', **次の値**);

© LPI-Japan 2013. All rights reserved.

シーケンスの作成と使用

```
db1=> CREATE SEQUENCE seq1 CYCLE;
CREATE SEQUENCE
db1=> ¥d seq1
            Sequence "public.seq1"
    Column     |  Type   |        Value
---------------+---------+---------------------
 sequence_name | name    | seq1
 last_value    | bigint  | 1
 start_value   | bigint  | 1
 increment_by  | bigint  | 1
 max_value     | bigint  | 9223372036854775807
 min_value     | bigint  | 1
 cache_value   | bigint  | 1
 log_cnt       | bigint  | 0
 is_cycled     | boolean | t
 is_called     | boolean | f

db1=> INSERT INTO tab0 values(nextval('seq1'), 'AAA');
INSERT 0 1
db1=> SELECT * FROM tab0;
 id | col1
----+------
  1 | AAA
(1 row)
```

© LPI-Japan 2013. All rights reserved.

シーケンス操作関数

OSS-DB

```
db1=> ¥connect db1 user1
You are now connected to database "db1".
db1=> SELECT currval('seq1');
ERROR:  currval of sequence "seq1" is not yet defined in this session
db1=> SELECT nextval('seq1');
 nextval
---------
       2
(1 row)

db1=> SELECT currval('seq1');
 currval
---------
       2
(1 row)

db1=>
```

現行セッションでnextval()を実行する前にcurvar()を実行するとエラーになる（現行セッションでシーケンスの値が取得されていないため）

nextval()を実行した後であれば、curvar()で値を取得できる

© LPI-Japan 2013. All rights reserved.

関数(ファンクション)

- 関数(ファンクション)とは
  - プログラミング言語の関数と類似した概念
  - 関数名(と引数)を指定して、関数に対応する処理を実行できる
- 関数の実行方法
  - SELECT 関数名(引数);

```
db1=> SELECT lower('ABC');
 lower
--------
 abc
(1 row)
```

- [注意] 関数の実行方法
  - Oracle Databaseでは EXECUTE 関数名; で関数を実行する
  - PostgreSQLではEXECUTEはプリペアド文の実行に使用する
    - プリペアド文：名前付きのバインド変数化SQLのようなもの
      単一セッションで類似したSQLを繰り返し実行する際にパフォーマンス向上を実現できる

© LPI-Japan 2013. All rights reserved.

組み込み関数

- 組み込み関数とユーザー定義関数
  - 組み込み関数
    - PostgreSQLにデフォルトで定義済みの関数
    - 上記のlower()は引数に指定された文字列を小文字に変換する処理を行う組み込み関数
  - ユーザー定義関数
    - CREATE FUNCTION文を用いて、ユーザーが独自に定義した関数
- 組み込み関数の分類
  - 集約関数 （SELECT文の箇所ですでに説明済み）
  - 算術関数
  - 文字列関数
  - 時間関数
  - 主にSilver試験対策では上記関数を押さえておけばOK
  - その他の関数については http://www.postgresql.jp/document/9.0/html/functions.html を参照のこと

© LPI-Japan 2013. All rights reserved.

組み込み関数 – 算術関数

関数名	説明	使用例	結果
abs(x)	xの絶対値	abs(-17.4)	17.4
div(y, x)	y/xの整数商	div(9, 4)	2
mod(y, x)	y/xの剰余	mod(9, 4)	1
sqrt(x)	xの平方根	sqrt(2.0)	1.4142135623731
power(x, y)	xのy乗	power(2,3)	8
ceil(x)	xより小さくない最少の整数	ceil(10.5) ceil(-42.8)	11 -42
floor(x)	xより大きくない最少の整数	floor(10.5) floor(-42.8)	10 -43

- **参考）**
  http://www.postgresql.jp/document/9.0/html/functions-math.html#FUNCTIONS-MATH-FUNC-TABLE

© LPI-Japan 2013. All rights reserved.

組み込み関数 – 文字列関数

CO-Solutions OSS-DB

関数名	説明	使用例	結果
length(s) char_length(s)	文字列の文字数	length('XYZ')	3
bit_length(s)	文字列のビット数	bit_length('XYZ')	3
substring(s FROM n)	部分文字列を取り出す	substring('XYZ' FROM 2)	'YZ'
upper(s)	文字列を大文字に変換	upper('xyz')	'XYZ'
lower(s)	文字列を小文字に変換	lower('XYZ')	'xyz'
replace(s, t, u)	文字列s内の文字列tを文字列uに置換	replace('abcdecd', 'cd', 'XX')	'abXXeXX'
trim([leading | trailing | both] c FROM s)	文字列sから文字列の先頭または末尾またはその両方の文字cを削除	trim(both 'x' FROM 'xTomxx')	'Tom'

- **参考）**
  http://www.postgresql.jp/document/9.0/html/functions-string.html

© LPI-Japan 2013. All rights reserved.

組み込み関数 – 時間関数

関数名	説明
now()	現在の日付と時刻を返す
current_timestamp	現在の日付と時刻を返す
current_time	現在の時刻を返す
current_date	現在の日付を返す
age(timestamp, timestamp)	時刻の差分を返す
to_char(timestamp, text)	時刻を文字列に変換する
extract(フィールド FROM timestamp)	時刻から指定されたフィールドを取り出す

- **参考）**
  http://www.postgresql.jp/document/9.0/html/functions-datetime.html

© LPI-Japan 2013. All rights reserved.

to_char () の使用例

```
db1=> SELECT to_char(TIMESTAMP '2013-01-02 23:10:11', 'Dy. DD-MON-YY');
    to_char
----------------
 Wed. 02-JAN-13
(1 row)

db1=> SELECT to_char(TIMESTAMP '2013-01-02 23:10:11', 'HH12:MI:SS');
 to_char
----------
 11:10:11
(1 row)
```

■ to_char()の詳細な定義または使用方法については以下を参照のこと
http://www.postgresql.jp/document/9.0/html/functions-formatting.html#FUNCTIONS-FORMATTING

© LPI-Japan 2013. All rights reserved.

extract () の使用例

```
db1=> SELECT extract(year FROM TIMESTAMP '2013-01-02 10:11:12');
 date_part
-----------
      2013
(1 row)

db1=> SELECT extract(month FROM TIMESTAMP '2013-01-02 10:11:12');
 date_part
-----------
         1
(1 row)

db1=> SELECT extract(day FROM TIMESTAMP '2013-01-02 10:11:12');
 date_part
-----------
         2
(1 row)
```

- extract()の詳細な定義または使用方法については以下を参照のこと http://www.postgresql.jp/document/9.0/html/functions-datetime.html#FUNCTIONS-DATETIME-EXTRACT

© LPI-Japan 2013. All rights reserved.

- 関数(ファンクション)とは
  - ユーザーが独自に関数を定義してデータベースに格納できる
  - 関数名(と引数)を指定して、関数に対応する処理を実行できる
  - 関数を記述する言語を、いくつかの言語(PL/pgSQL、SQLなど)から選ぶことができる
- PL/pgSQLを用いてユーザー定義関数を作成し、実行した例

```
db1=> CREATE OR REPLACE FUNCTION increment(i integer) RETURNS integer AS $$
db1$>         BEGIN
db1$>                 RETURN i + 1;
db1$>         END;
db1$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
db1=> SELECT increment(4);
 increment
-----------
         5
(1 row)
```

© LPI-Japan 2013. All rights reserved.

トリガ

OSS-DB

トリガとは

- あるテーブルに対してデータ更新処理が実行されたときに、自動的にあるユーザー定義関数を実行する機能

トリガの作成とパラメータ

- CREATE TRIGGER文でトリガを作成する
- トリガのパラメータとして以下を指定する
  - トリガの起動単位：行単位（1行に1回）または 文単位（1つのSQL文に1回）
  - トリガを起動するイベント： INSERT、UPDATE、DELETE、TRUNCATE
  - トリガが起動されるタイミング：BEFORE、AFTER、INSTEAD OF
  - 起動されるファンクション

トリガの作成例

- accountsテーブルの行が更新される直前に、行単位でcheck_account_update関数を実行するトリガ

```
CREATE TRIGGER check_update
    BEFORE UPDATE ON accounts
    FOR EACH ROW
    EXECUTE PROCEDURE check_account_update();
```

© LPI-Japan 2013. All rights reserved.

ルール

■ ルールとは

- ■ オブジェクトに対してSQLが発行されたときに、その処理を置き換えたり、別の処理を追加する仕組み

■ ルールの作成

- ■ CREATE RULE文でルールを作成する

■ ルールの作成例

- ■ ビューへのINSERTを可能にするルール

```
db1=> CREATE VIEW view0 AS SELECT id, col1 FROM tab0 WHERE id < 3;
db1=> CREATE RULE view0_ins AS ON INSERT TO view0
db1->     DO INSTEAD
db1->     INSERT INTO tab0 VALUES (NEW.id, NEW.col1 );
db1=> INSERT INTO view0 VALUES(2, 'ZZZ');
db1=> SELECT * FROM view0;
 id | col1
----+------
  2 | ZZZ
(1 row)
```

© LPI-Japan 2013. All rights reserved.

アクセス権限

- 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）

© LPI-Japan 2013. All rights reserved.

アクセス権限の確認方法

- ¥dp メタコマンド
  - ¥zメタコマンドは¥dp メタコマンド の別名
- 他のオブジェクト
  - ¥dn+　　：スキーマ一覧にアクセス権限を表示
  - ¥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 ¥ddp メタコマンドで確認できる

© LPI-Japan 2013. All rights reserved.

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）

© LPI-Japan 2013. All rights reserved.

アクセス権限の種類

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

参考）書籍「PostgreSQL全機能バイブル」初版 第1刷 P142 を元に一部修正ののち作成

© LPI-Japan 2013. All rights reserved.

アクセス権限の種類

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 PRIVILEDGES	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

参考) 書籍「PostgreSQL全機能バイブル」初版 第1刷 P142 を元に一部修正ののち作成

© LPI-Japan 2013. All rights reserved.

デフォルトのアクセス権限

CO-Solutions OSS-DB

- 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;

© LPI-Japan 2013. All rights reserved.

通常ユーザー以外へのアクセス権限付与

- 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

© LPI-Japan 2013. All rights reserved.

他のRDBMSとの差異 － 権限

CO-Solutions OSS-DB

- 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

© LPI-Japan 2013. All rights reserved.

トランザクション

CO-Solutions OSS-DB

- 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**

© LPI-Japan 2013. All rights reserved.

トランザクションのACID特性

CO-Solutions OSS-DB

- 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**

© LPI-Japan 2013. All rights reserved.

トランザクションの制御

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

© LPI-Japan 2013. All rights reserved.

トランザクションの実行例

```
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

© LPI-Japan 2013. All rights reserved.

```
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

© LPI-Japan 2013. All rights reserved.

SAVEPOINT

- 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
---
 1
 2
 3
(3 rows)

db1=> ROLLBACK TO A;
ROLLBACK
db1=> SELECT * FROM tab0;
 n
---
 1
(1 row)
```

セーブポイントAの
状態に戻る

© LPI-Japan 2013. All rights reserved.

トランザクション分離レベル（隔離レベル）

CO-Solutions OSS-DB

- 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 COMMITED, READ UNCOMMITED
- 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

© LPI-Japan 2013. All rights reserved.

トランザクション分離レベルとphenomena

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 COMMITED	発生しない	発生する	発生する	○
READ UNCOMMITED	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 UNCOMMITEDを指定してもREAD COMMITEDとして動作する

© LPI-Japan 2013. All rights reserved.

トランザクション分離レベルの指定方法

- 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 COMMITED

© LPI-Japan 2013. All rights reserved.

注意点：psqlはオートコミット

- 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)

db1=> ROLLBACK;
NOTICE:  there is no transaction in progress
ROLLBACK
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）

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

© LPI-Japan 2013. All rights reserved.

注意点：SQLエラーとトランザクション

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

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

ロールバックするしかない

© LPI-Japan 2013. All rights reserved.

注意点：DDLとトランザクション

- 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

© LPI-Japan 2013. All rights reserved.

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等である**
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: a, c

© LPI-Japan 2013. All rights reserved.

例題解説7

■ **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：a, c

© LPI-Japan 2013. All rights reserved.

最後に

CO-Solutions OSS-DB

- 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。

© LPI-Japan 2013. All rights reserved.

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

© LPI-Japan 2013. All rights reserved.