LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Silver 技術解説無料セミナー

2014/2/16

株式会社メトロシステムズ
佐藤　千佳

© LPI-Japan 2014. All rights reserved.

自己紹介

METRO OSS-DB

■氏名
- 佐藤　千佳（さとう　ちか）

■所属
- 株式会社メトロシステムズ（東京・池袋・サンシャイン６０）

■略歴
- ２００６年に株式会社メトロシステムズに入社
  - Oracleを用いた業務システムの保守・管理を担当
- ２００７年からオープンソースデータベースを担当する部署に所属
  - PostgreSQL機能調査、周辺ツールを含めた性能評価を担当
  - PostgreSQLと他DBMSとの機能比較、性能比較評価なども担当

© LPI-Japan 2014. All rights reserved.

本セミナーの流れ

METRO OSS-DB

■OSS-DB試験の紹介
- 試験の概要
- PostgreSQLの特徴

■インストールと基本操作
- PostgreSQLのアーキテクチャ
- インストール
- 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
- SQL文の実行

■運用管理
- データベースとユーザの管理
- 基本的な設定
  - クライアント認証設定
  - パラメータ設定
- バックアップとリカバリ
- 定常的なメンテナンス
- システム情報の取得

Break

■SQLによる開発
- SQL文の基本
- オブジェクトと権限設定
- DMLによるデータ操作
- 関数と演算子
- トランザクション

© LPI-Japan 2014. All rights reserved.

OSS-DB試験の紹介

© LPI-Japan 2014. All rights reserved.

試験の概要

© LPI-Japan 2014. All rights reserved.

OSS-DB試験とは

■オープンソースデータベース（OSS-DB）に関する技術と知識を認定するＩＴ技術者認定です

データベースシステムの設計・導入・運用ができる技術者

大規模データベースシステムの
改善・運用管理・コンサルティングができる技術者

Linux技術者認定制度のLPICと同じく、LPI-Japanが実施

© LPI-Japan 2014. All rights reserved.

オープンソースデータベースとは

METRO OSS-DB

- データベースソフトウェアのうち、ソースコードが公開されているものを「オープンソースデータベース」と呼びます
  - PostgreSQL
  - MySQL、MariaDB
  - Firebird
  - Etc.

OSS-DB試験では
基準のデータベースとして
PostgreSQLを採用

- 商用のデータベースソフトウェアは、ソースコードが公開されていないものがほとんどです
  - Oracle
  - MS SQL Server
  - DB2
  - HiRDB

© LPI-Japan 2014. All rights reserved.

OSS-DB Silver試験の出題範囲

METRO OSS-DB

■一般知識（２０％）

- オープンソースデータベースの一般的特徴
- ライセンス
- コミュニティと情報収集
- リレーショナルデーターベースの一般的知識

■運用管理（５０％）

- インストール方法
- 標準付属ツールの使い方
- 設定ファイル
- バックアップ方法
- 基本的な運用管理作業

■開発/SQL（３０％）

- SQLコマンド
- 組み込み関数
- トランザクション概念

試験時間　：９０分※
問題数　：５０問
合格点　：６４点

※アンケート時間等を含む

© LPI-Japan 2014. All rights reserved.

OSS-DB試験の注意事項

METRO OSS-DB

■最新の試験範囲はWebで確認！

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

■対象のPostgreSQLバージョンは「9.0」

- ２０１４年２月時点の対応バージョンは「9.0.15」

■OSに依存しない内容だが、表記はLinuxベース

- シェルのコマンドプロンプトは「$」
- 「フォルダ」でなく「ディレクトリ」
- ディレクトリ区切り文字は「¥」や「\」でなく「/」

© LPI-Japan 2014. All rights reserved.

PostgreSQLの特徴

© LPI-Japan 2014. All rights reserved.

PostgreSQLの特徴（１）

METRO OSS-DB

■長い歴史を持つオープンソースのデータベースサーバ

- １９８６年〜　Postgresという名称で開発開始
- １９９６年〜　SQLサポートを機に「PostgreSQL」に名称変更
- 現在も活発に開発が続いており、毎年メジャーバージョンアップ

■メジャーバージョンとマイナーバージョンとは

- PostgreSQLのバージョン表記は数字三つの「x.y.z」
  - 現在の最新版は「9.3.2」
- 最初の二つの数字「x.y」部分がメジャーバージョン
  - 大きな機能追加・変更を含むもので、一年に一度程度
  - メジャーバージョンリリース後、５年間はコミュニティが無償サポートを提供
- 最後の「z」部分がマイナーバージョン
  - バグフィックスやセキュリティ対策を含むもので、２〜３ヶ月に一度程度

© LPI-Japan 2014. All rights reserved.

■機能が豊富
- 複雑なクエリも実行できるＳＱＬエンジン
- 様々な言語で開発できるストアドプロシージャ
- BIやDWHで活用できる集約関数やWindow関数
- 組み込みのレプリケーション
- 最新バージョンではNoSQL系の機能も
  - key-valueストアやJSONデータ型のサポート

■性能も継続的に改善
- 同一ハードウェアで商用データベースと比べても遜色ない性能
- 最新バージョンの9.2では64コアまでスケール

© LPI-Japan 2014. All rights reserved.

PostgreSQLの特徴（3）

METRO OSS-DB

■自由度の高いライセンス

- BSDベースの独自ライセンス「PostgreSQLライセンス」を採用
- 商用・非商用を問わず、無償で利用が可能
- 著作権表記とライセンス全文を含めれば、改変および再配布は自由
- GPLなどと異なりソースコードの公開義務はないため、商用製品への組み込みが容易
  - SRA OSS Inc. 「PowerGres Plus」（高信頼データベース）
  - EnterpriseDB「Postgres Plus」（Oracle互換データベース）
  - Heroku 「Heroku Postgres」（SQL Database-as-a-Service）
  - EMC 「Greenplum」（大規模並列分散データベース）
  - Amazon 「Amazon Redshift」（データウェアハウスサービス）

© LPI-Japan 2014. All rights reserved.

■企業ではなくコミュニティが開発

- PostgreSQL Global Development Group（PGDG：PostgreSQLグローバル開発グループ）が開発している
- ソースコードの著作権は、このコミュニティが保持している

■様々なコミュニティ活動

- WEBサイト運営（http://www.postgresql.org）
  - ソースコードやバイナリパッケージ、ドキュメントなどを無償で公開
  - アーカイブには過去バージョンから開発中バージョンまで網羅
  - バグレポートはWebフォームで登録可能
- メーリングリスト運営
  - 一般ユーザ向けから開発者向けから、仕事紹介まで幅広いジャンル
- イベント開催
  - 世界各地でカンファレンスやユーザ会の会合を開催

© LPI-Japan 2014. All rights reserved.

PostgreSQLの特徴（5）

- 日本でのコミュニティ活動が盛ん
  - 日本PostgreSQLユーザ会　（JPUG）が中心となって活動
    - ユーザ会Webサイト（http://www.postgresql.jp）の運営
    - 技術情報ポータルサイト「Let's Postgres」の運営
    - 日本語メーリングリストの運営
    - カンファレンス（年一回）や勉強会（年数回）の開催
    - 日本各地（北海道から沖縄まで）に地方支部がありイベントを開催

© LPI-Japan 2014. All rights reserved.

インストールと基本操作

- PostgreSQLのアーキテクチャ
- インストール
- 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
- SQL文の実行

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

PostgreSQLのアーキテクチャ

© LPI-Japan 2014. All rights reserved.

■インスタンス

- データベースを構成するファイルや共有メモリ、プロセスなどを合わせたもので、PostgreSQLサーバの起動単位
- 一つのサーバ内で複数起動することが可能

© LPI-Japan 2014. All rights reserved.

■データベースクラスタ

- データベースインスタンスに関連するファイルを格納する領域
- データベースインスタンスごとにひとつ
- 慣習的にディレクトリパスを$PGDATAという環境変数に設定する
- ひとつのサーバ内に複数作成することが可能

© LPI-Japan 2014. All rights reserved.

■データベース

- テーブルやインデックス等のオブジェクトの格納先
- ひとつのデータベースクラスタ内に複数作成することが可能
- データベースクラスタを初期化した時点で３つのデータベースが存在

データベース名	接続	用途
template0	不可	初期状態のテンプレート
template1	可	データベース作成時のコピー元
postgres	可	標準ツールのデフォルト接続先で、通常のデータベースと同様に使用可能

© LPI-Japan 2014. All rights reserved.

PostgreSQLのファイル構成

METRO OSS-DB

■データベースクラスタ内の主なファイルとディレクトリ

pgdata/	データベースクラスタのルートディレクトリ
base/	データベースごとのファイル群を格納するディレクトリ
1/	template0データベース用ディレクトリ
11818/	template1データベース用ディレクトリ
11826/	postgresデータベース用ディレクトリ
global/	グローバルカタログのファイルを配置するディレクトリ
pg_log/	ログファイルを保存するディレクトリ
pg_xlog/	WAL(トランザクションログ)を保存するディレクトリ
PG_VERSION	メジャーバージョンが記録されたファイル
postgresql.conf	パラメータ設定ファイル
pg_hba.conf	クライアント認証設定ファイル
postmaster.pid	起動中のサーバのPIDなどを記録したファイル

© LPI-Japan 2014. All rights reserved.

PostgreSQLのプロセス構成

■プロセスの種類

- マスタプロセス（pid=6822）
  - PostgreSQLインスタンスの中心となるプロセス
  - クライアントからの接続ごとにバックエンドプロセスを起動
- バックエンドプロセス（pid=6865）
  - クライアントからのSQLリクエストを処理するプロセス
  - プロセス名の後ろに接続情報や処理内容を表示
- 管理用プロセス群（その他）
  - データベースサーバに必要な役割の一部を担当するプロセス
  - プロセス名の後ろに役割や処理内容を表示

■プロセス名は全て「postgres」

```
$ ps -ef | grep postgres
501  6824  6822   0 11:16AM ??        0:00.12 postgres: writer process
501  6825  6822   0 11:16AM ??        0:00.09 postgres: wal writer process
501  6826  6822   0 11:16AM ??        0:00.05 postgres: autovacuum launcher process
501  6827  6822   0 11:16AM ??        0:00.06 postgres: stats collector process
501  6865  6822   0 11:16AM ??        0:00.01 postgres: postgres postgres [local] SELECT
501  6822     1   0 11:16AM ttys001   0:00.10 /Users/hanada/pgsql-90/bin/postgres
```

© LPI-Japan 2014. All rights reserved.

PostgreSQL独特の用語

METRO OSS-DB

■共有バッファ
- データファイルへの変更を一時的に保持しておくメモリ領域
- データベースへの変更は、必ず共有バッファを介する

■チェックポイント
- 共有バッファからデータファイルに変更点を書き出す処理
- それなりにディスクI/Oが発生するので、頻繁な発生は避けたい

■WAL（トランザクションログ）
- データベースへの変更を共有バッファ操作の前に記録しておくログ
- トランザクションコミット時にファイルに書き出される
- 障害発生時には、コミット済みだが未書き込みの変更内容をここから取得して復旧する

© LPI-Japan 2014. All rights reserved.

PostgreSQL利用までの流れ

METRO OSS-DB

- バイナリファイルのインストール
- データベースクラスタの初期化
- データベースインスタンスの起動
- 接続、利用
- データベースインスタンスの停止

© LPI-Japan 2014. All rights reserved.

インストール

© LPI-Japan 2014. All rights reserved.

インストール方法

METRO OSS-DB

■大きく分けると以下の三種類

- ソースコードからビルドしてインストール
  - 細かいビルドオプションを指定可能
- パッケージ管理システムを使ってインストール
  - 依存関係の自動解決など、管理が容易
- ワンクリックインストーラを使ってインストール
  - インストールが簡単
  - 基本的な設定や追加ツールも同梱

今回はこれを中心に説明

© LPI-Japan 2014. All rights reserved.

OSユーザ作成

METRO OSS-DB

■なぜOSユーザを作成するの？

- PostgreSQLサーバは一般ユーザ（非rootユーザ）でのみ実行可能

■postgresユーザ作成手順

- PostgreSQLサーバを実行するOSユーザは慣習的に「postgres」

```
$ su -
# useradd postgres
# passwd postgres
…
# su - postgres
$
```

これ以降の手順はpostgresユーザで実行

© LPI-Japan 2014. All rights reserved.

ソースコードからインストール

METRO OSS-DB

■ソースコードをダウンロード

- 下記サイトから9.0.xの最新版をダウンロード
- http://www.postgresql.org/ftp/source/

■ビルド＆インストール

- デフォルトインストール先は「/usr/local/pgsql」
- configureスクリプトの「--prefix」オプションでインストール先を変更可能（例：--prefix=$HOME/pgsql）

```
$ tar zxvf postgresql-9.0.13.tar.gz
…
$ cd postgresql-9.0.13
$ ./configure
…
$ make
$ sudo make install # /usr/local/への書き込みはrootで
```

© LPI-Japan 2014. All rights reserved.

インストール後の設定

METRO OSS-DB

■環境変数設定

- PATHに「インストール先/bin」を追加
- LD_LIBRARY_PATHに「インストール先/lib」を追加
- 一般的には、postgresユーザのログインスクリプトに設定
  - .bash_profileや.profile

© LPI-Japan 2014. All rights reserved.

yumインストール（Redhat系Linux）

■rpmでリポジトリ設定をインストール

- http://yum.postgresql.org/repopackages.php
- 上記ページの「PostgreSQL 9.0」からOS別RPMをダウンロード
- ダウンロードしたRPMをrpmコマンドでインストール

■yumでPostgreSQL自体をインストール

- yum install postgresql90-server
  - 自動的にpostgresql90とpostgresql90-libsもインストール
- /usr/pgsql-9.0に入り、/etc/alternatives経由で/usr/binにシンボリックリンクができる

■ソースコードインストールとの違い

- パッケージインストール中にpostgresユーザを自動的に作成
- initdbやpg_ctlがない（service postgresql-9.0を使う）
  - service postgresql-9.0 initdb
  - service postgresql-9.0 start

© LPI-Japan 2014. All rights reserved.

ワンクリックインストーラ

METRO OSS-DB

■OS別インストーラをダウンロード
- EnterpriseDB社のサイトで配布
- http://www.enterprisedb.com/products-services-training/pgdownload

■インストールガイドは英語のみ
- http://www.enterprisedb.com/resources-community/pginst-guide

■初期設定からの変更点
- スーパーユーザ（管理者ユーザ）のパスワードを忘れないように！
- ロケール（Locale）は「C」に設定
  - デフォルトのままでは日本語ソートが正しく動きません
- コマンドインストールパスをPATH環境変数に追加
  - Windows　:C:\Program Files\PostgreSQL\9.0\bin
  - Unix系　:/opt/PostgreSQL/9.0/bin

■自動起動設定
- インストール時にデータベース初期化・起動・自動起動設定が完了

© LPI-Japan 2014. All rights reserved.

OSS-DB試験対策

METRO OSS-DB

■どのOSがいいの？

- 可能であれば、CentOSなどのRedhat系Linuxがお勧め
- VMwareなどの仮想環境でもＯＫ

■どのインストール方法がいいの？

- とりあえず使ってみるなら、パッケージorワンクリックインストーラ
- 試験対策にはソースインストール
- Ubuntu等のパッケージでは、起動・停止などのコマンドが異なる
- Mac OS XではPostgres.appという選択肢も
- Windowsの場合はワンクリックインストーラを推奨

■どのバージョンがいいの？

- 試験の基準は9.0
- SQLや基本的なコマンドは9.1以降でもほとんど変わりなし

© LPI-Japan 2014. All rights reserved.

データベースクラスタの初期化

© LPI-Japan 2014. All rights reserved.

データベースクラスタの初期化

METRO OSS-DB

■initdbコマンドでデータベースクラスタを初期化

- オプション
  - -D データベースクラスタのパス
  - --locale ロケール（日本語環境では「C」推奨）
  - -E エンコーディング（日本語はUTF-8かEUC-JP）
  - -U スーパーユーザ名

```
$ initdb -D /home/postgres/pgdata --locale=C -E UTF-8 -U
postgres
…
Success. You can now start the database server using:

    postgres -D /home/postgres/pgdata
or
    pg_ctl -D /home/postgres/pgdata -l logfile start

$
```

- pg_ctl initdbコマンドでも可能
  - pg_ctl initdb -D データベースクラスタ -o "initbdオプション"

© LPI-Japan 2014. All rights reserved.

インスタンスの起動と停止

© LPI-Japan 2014. All rights reserved.

■pg_ctl startコマンドでインスタンスを起動

```
$ pg_ctl start -D /home/postgres/pgdata -w
waiting for server to start....LOG:  database
system was shut down at 2013-04-08 21:10:13 JST
LOG:  database system is ready to accept
connections
LOG:  autovacuum launcher started
 done
server started
$ pg_ctl status -D
/home/postgres/pgdata
pg_ctl: server is running (PID: 6822)
/usr/local/pgsql/bin/postgres
$
```

インスタンスが正常に稼働中

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

■psqlコマンドで接続してSQLを実行

```
$ psql
psql (9.0.13)
Type "help" for help.

postgres=# SELECT current_user;
 current_user
--------------
 postgres
(1 row)

postgres=# \q
$
```

psqlコマンドはSQLコマンドインタプリタ

データベースから切断するには「\q」

© LPI-Japan 2014. All rights reserved.

■pg_ctl stopコマンドでインスタンスを停止

```
$ pg_ctl stop -D /home/postgres/pgdata
waiting for server to shut down....LOG:  received smart
shutdown request
LOG:  autovacuum launcher shutting down
LOG:  shutting down
LOG:  database system is shut down
 done
server stopped
$ pg_ctl status -D
/home/postgres/pgdata
pg_ctl: no server running
$
```

インスタンスは停止済み

■終了モードを-mオプションで指定

- -m smart ：クライアントが全て切断するのを待って終了
- -m fast ：全接続を強制切断して終了
- -m immediate ：即時停止、次回起動時に復旧処理が必要

© LPI-Japan 2014. All rights reserved.

■PGDATA環境変数

- pg_ctlやinitdbは、$PGDATA環境変数が設定されていればそこからデータベースクラスタパスを取得する
- 一般的には、postgresユーザの環境変数としてログインスクリプトに記述しておくことが多い
  - PGDATA=/home/postgres/pgdata
  - export PGDATA

© LPI-Japan 2014. All rights reserved.

SQL文の実行

© LPI-Japan 2014. All rights reserved.

■psqlコマンドとは

- PostgreSQL標準の対話的ターミナル（コマンドラインツール）
- SQL文を実行して結果を取得したり、メタコマンド（後述）でデータベースオブジェクトの情報を取得したりできる

■コマンド書式

- psql ［接続オプション］［オプション］ ［データベース名 ［ユーザ名］］

オプション	意味・指定する値
-l|--list	データベースを一覧表示する（引数なし）
-c|--command	指定したコマンドを実行しpsqlを終了する
-f|--file	指定したファイルの内容をソースとして使用し処理終了後にpsqlを終了する

© LPI-Japan 2014. All rights reserved.

■接続オプション

- データベース接続情報を指定するオプションは全コマンド共通
- オプションを省略した項目は、対応する環境変数から値を取得する
- 環境変数が設定されていない場合は、デフォルト値を使用する

オプション	指定する値	環境変数	デフォルト値
-h|--hostname	サーバのホスト名またはIPアドレス	PGHOST	UNIXドメインソケットで接続
-p|--port	サーバのポート番号	PGPORT	5432
-U|--username	データベースユーザ名	PGUSER	OSユーザ名
-d|--dbname	データベース名	PGDATABASE	接続に使うデータベースユーザ名

データベースに接続する標準ツールにも共通

© LPI-Japan 2014. All rights reserved.

■プロンプトに対してSQL文を入力して改行すると実行される

```
$ psql
psql (9.0.13)
Type "help" for help.

postgres=# SELECT now(),
postgres-# current_user;
              now              | current_user
-------------------------------+--------------
 2013-04-09 23:28:21.326966+09 | postgres
(1 row)

postgres=# \q
$
```

SQL文は途中で改行可能
※継続行では「=」が「-」に

SQL文はセミコロンで終了

切断するときは「\q」または「\quit」

© LPI-Japan 2014. All rights reserved.

psqlのメタコマンド

■メタコマンドとは

- バックスラッシュ「\」で始まるpsql独自のコマンド
- SQLと異なり、改行で終わりを判断（途中で改行できない）

■機能

- データベースに存在するオブジェクトの情報を表示
- SQL実行結果の表示形式や出力先を変更
- データベースとクライアント側のファイルの間でデータをやりとり
- 接続先データベースやユーザを変更
- ラージオブジェクトの操作
- Etc.

■ヘルプ表示

- psqlのプロンプトで「\?」を実行するとメタコマンドのヘルプが表示される

© LPI-Japan 2014. All rights reserved.

オブジェクト情報の表示

METRO OSS-DB

■現在のユーザで参照できるオブジェクトの情報を表示

- コマンド単独で実行すると一覧を表示
- コマンドの後ろにパターンを指定すると、名前が一致するオブジェクトの詳細を表示

コマンド	対象
\l	データベース
\d	テーブル･ビュー･シーケンス
\du	ユーザー
\dn	スキーマ
\dt	テーブル
\dv	ビュー
\ds	シーケンス
\di	インデックス
\dS	システムカタログ
\df	関数

© LPI-Japan 2014. All rights reserved.

その他のメタコマンド

■オブジェクト情報表示以外の主なコマンド

コマンド	効果
\x [onloff]	拡張テーブル形式を切り替える （on/off省略時はトグル動作）
\i ファイル名	ファイルの内容を実行
\o ファイル名	実行結果をファイルに出力
\timing [onloff]	クエリ実行時間表示のon/off （on/of省略時はトグル動作）
\! [シェルコマンド]	シェルコマンドを実行
\h [SQL]	SQL文のヘルプを表示

© LPI-Japan 2014. All rights reserved.

メタコマンドの実行例（１）

METRO OSS-DB

■リレーション一覧（\d）

```
postgres=# \d
                List of relations
 Schema |        Name        |   Type    |  Owner
--------+--------------------+-----------+----------
 public | pgbench_accounts | table     | postgres
 public | pgbench_branches | table     | postgres
 public | pgbench_history  | table     | postgres
 public | pgbench_tellers  | table     | postgres
 public | seq_accounts     | sequence | postgres
 public | v_accounts       | view      | postgres
(6 rows)
postgres=#
```

© LPI-Japan 2014. All rights reserved.

メタコマンドの実行例（2）

METRO OSS-DB

■リレーション詳細（\d パターン）

```
postgres=# \d pgbench_branches
   Table "public.pgbench_branches"
  Column   |      Type       | Modifiers
-----------+-----------------+-----------
 bid       | integer         | not null
 bbalance  | integer         |
 filler    | character(88)   |
Indexes:
    "pgbench_branches_pkey" PRIMARY KEY, btree
(bid)

postgres=#
```

© LPI-Japan 2014. All rights reserved.

メタコマンドの実行例（3）

METRO OSS-DB

■拡張テーブル形式（\x）

```
postgres=# \x
Expanded display is on.
postgres=# SELECT * FROM pgbench_tellers;
-[ RECORD 1 ]-----
tid      | 8
bid      | 1
tbalance | 6885
filler   |
-[ RECORD 2 ]-----
tid      | 6
bid      | 1
tbalance | -36903
filler   |
...
postgres=#
```

各列が縦に展開されて表示される

「(n rows)」が表示されない

© LPI-Japan 2014. All rights reserved.

運用管理

- データベースとユーザの管理
- 基本的な設定
- バックアップとリカバリ
- 定常的なメンテナンス
- システム情報の取得

© LPI-Japan 2014. All rights reserved.

データベースとユーザの管理

© LPI-Japan 2014. All rights reserved.

データベースの作成

- ひとつのインスタンスで複数のデータベースを管理可能
  - 参照できるのは、接続しているデータベース内のオブジェクトのみ
  - データベースをまたがった検索などはできない
- システムごとにデータベース分割
  - プログラムバグなどで不用意にデータを参照・変更することを防止
  - データベースごとに接続可能なユーザやクライアントを設定可能
  - ログでの問題きり分けが容易になる
- createdbコマンドで作成

```
$ createdb jinji
$ psql jinji
jinji=#
```

プロンプトには接続中の
データベース名が表示される

© LPI-Japan 2014. All rights reserved.

■テンプレートデータベースとは

- 新しいデータベースを作るときに、ひな形となるデータベース
  - createdbコマンドの「-T」オプションで指定可能
  - デフォルトでは、template1が使われる
  - ユーザが作成したデータベースも指定可能
- ロケールやエンコーディングをテンプレートと変えたい場合はtemplate0をテンプレートに指定する必要がある
  - createdb -E エンコーディング -T template0

© LPI-Japan 2014. All rights reserved.

データベースの削除

METRO OSS-DB

- dropdbコマンドで削除
  - dropdb [-i] データベース名
- 注意点
  - ユーザが接続しているデータベースは削除できない
  - データベースの削除は不可逆なので事前確認をしっかり！
  - -iオプションをつけると、確認メッセージが出る
    - alias dropdb='dropdb -i'
- dropdbコマンドの実行例

```
$ dropdb -i jinji
Database "jinji" will be permanently removed.
Are you sure? (y/n) y
$ psql jinji
psql: FATAL:  database "jinji" does not exist
$
```

削除したので接続できない

© LPI-Japan 2014. All rights reserved.

データベースユーザの概要

- データベースに接続するには、データベースユーザが必要
  - データベースクラスタ初期化時にスーパーユーザが作成済み
    - 今回の例では「postgres」
- データベースユーザごとにアクセス権や接続認証を設定可能
  - セキュリティレベルや用途別にユーザを分けることを推奨
  - 普段からスーパーユーザで作業していると、操作ミスでデータベースを破壊してしまうことも！
- データベースユーザはデータベース間で共通
  - ユーザ情報はデータベース間共通のグローバルカタログで管理
    - pg_authidシステムカタログに管理情報を保持
      - 一般ユーザは参照不可
    - pg_roleビューやpg_userビューは人間が読みやすい形式

© LPI-Japan 2014. All rights reserved.

データベースユーザの作成・削除

METRO OSS-DB

■createuserコマンドで対話的に作成

```
$ createuser sato
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n
$ psql -U sato
postgres=>
```

■dropuserコマンドで削除

- dropuser [-i|--interactive] ユーザ名
- 注意点
  - オブジェクトを所有しているユーザは削除できないので、事前にオブジェクトを削除するか所有権を他ユーザに移す
  - -iオプションをつけないと確認なしで即削除

© LPI-Japan 2014. All rights reserved.

基本的な設定

～クライアント認証設定～

© LPI-Japan 2014. All rights reserved.

クライアント認証設定

METRO OSS-DB

■クライアント認証設定とは

- クライアントの特性に応じて認証方式を切り替える設定

■設定ファイル

- $PGDATA/pg_hba.conf
- HBA＝Host Based Authentication

■利用可能な特性

- 接続方式（UNIXドメイン接続 or TCP/IP接続）
- 接続先データベース
- 接続ユーザ
- 接続元IPアドレス（TCP/IP接続のみ）

■認証方式

- 常に許可
- パスワード認証（平文）
- パスワード認証（MD5ハッシュ）
- 常に拒否

© LPI-Japan 2014. All rights reserved.

クライアント認証の設定例

■クライアント認証ポリシー

postgresデータベースへのリモート接続は常に拒否

ローカル接続はどちらのデータベースでも常に許可

LAN外からのリモート接続は常に拒否

jinjiデータベースへのLANからのアクセスはMD5認証

WAN

postgres

jinji

10.0.0.0/8

192.168.1.0/24

© LPI-Japan 2014. All rights reserved.

pg_hba.confでの設定

■書式

- クライアントからの接続の認証ルールを一行に一つ記述
- 「#」から行末まではコメント

■記述項目（カッコ内はpg_hba.confファイル中の見出し）

- 接続方式（TYPE）
  - local（UNIXドメイン接続）またはhost（TCP/IP接続）
- 接続先データベース名（DATABASE）
- 接続ユーザ名（USER）
- 接続元IPアドレス（CIDR-ADDRESS）
  - CIDRアドレスまたはアドレス+ネットマスクで範囲指定
  - TYPE=localの行では記述しない
- 認証方法（METHOD）
  - trust　：常に許可
  - password　：平文パスワード認証
  - md5　：MD5ハッシュパスワード認証
  - reject　：常に拒否

© LPI-Japan 2014. All rights reserved.

pg_hba.confの内容

■例

```
# TYPE   DATABASE        USER            CIDR-ADDRESS            METHOD

# ローカル接続は全て許可
local    all             all                                     trust

# 同一サーバ上からの接続も全て許可
host     all             all             127.0.0.1/32            trust
host     all             all             ::1/128                 trust

# postgresデータベースへのリモートアクセスは全て拒否
host     postgres        all             0.0.0.0/0               reject

# LAN内からの接続はMD5パスワード認証
host     jinji           all             192.168.1.0/24          md5
host     jinji           all             10.0.0.0 255.0.0.0      md5
```

ローカル接続の行では
IPアドレスは不要

IPv6も
サポート

アドレス+ネットマスク指定も可能

上から順に照合し一致するエントリが
見つかったらその行の認証方式を適用

一致するエントリがなければ接続拒否

© LPI-Japan 2014. All rights reserved.

クライアント認証設定の変更

- 設定変更手順
  - pg_hba.confを書き換える
  - pg_ctl reloadを実行、またはマスタプロセスにSIGHUPを送信
    - パラメータ設定（後述）も併せてリロードされるので注意！
- 注意点
  - 既に接続が確立しているセッションには効果なし
  - 接続済みのセッションを強制的に切断したい場合は、psコマンドなどでPIDを調べ、スーパーユーザでpg_terminate_backend(pid)関数を実行

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

基本的な設定

〜パラメータ設定〜

© LPI-Japan 2014. All rights reserved.

パラメータ設定ファイル

METRO OSS-DB

■パラメータ設定ファイル
- $PGDATA/postgresql.conf

■書式
- パラメータ = 設定値
- 文字列の設定値はシングルクォートで囲む
- パラメータのデータ型に応じて設定値に単位をつける
  - サイズ：kB、MB、GB（kのみ小文字)
  - 時間：ms（ミリ秒）、s（秒）、min（分）、h（時間）、d（日）
- 「#」から行末まではコメント

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

■例

```
#------------------------------------------------------------------------------
# RESOURCE USAGE (except WAL)
#------------------------------------------------------------------------------

# - Memory -

shared_buffers = 32MB                                   # min 128kB
                                                        # (change requires restart)
#temp_buffers = 8MB                                     # min 800kB
#max_prepared_transactions = 0                          # zero disables the feature
                                                        # (change requires restart)
# Note:  Increasing max_prepared_transactions costs ~600 bytes of shared memory
# per transaction slot, plus lock space (see max_locks_per_transaction).
# It is not advisable to set max_prepared_transactions nonzero unless you
# actively intend to use prepared transactions.
#work_mem = 1MB                                         # min 64kB
#maintenance_work_mem = 16MB                            # min 1MB
#max_stack_depth = 2MB                                  # min 100kB
```

パラメータグループごとにまとめられている

デフォルト値がコメントアウトで記載されている

有効範囲や要再起動などの説明

© LPI-Japan 2014. All rights reserved.

設定変更方法（１）

METRO OSS-DB

■SET文実行

- セッション中でSET文を実行する
- 一部のパラメータはスーパーユーザのみ変更可能
- 一部のパラメータのみ変更可能
- 影響範囲は実行したセッションのみ

■SET分の構文

- SET パラメータ =|TO 値|'値'|DEFAULT;
- 値が数値や論理値、列挙値でない場合はシングルクォートで囲む
- 値の代わりに「DEFAULT」を指定するとデフォルト値に戻る

```
postgres=# SET work_mem TO '100MB';
SET
postgres=#
```

© LPI-Japan 2014. All rights reserved.

設定変更方法（2）

METRO OSS-DB

■設定リロード

- 設定ファイルを書き換えてリロードを指示する
  - `pg_ctl reload`またはマスタプロセスにSIGHUP送信
- 共有メモリやネットワーク関連など一部のパラメータは変更できない
- 影響範囲は既存セッション以外
  - SETで変更可能なパラメータは起動済みバックエンドに反映されない
  - クライアント認証設定も併せてリロードされる

■インスタンス再起動

- 設定ファイルを書き換えた後、`pg_ctl restart`コマンドでインスタンスを起動する
- あらゆるパラメータを変更できる

© LPI-Japan 2014. All rights reserved.

設定確認方法

METRO OSS-DB

■SHOW文実行

- セッション中でSHOW文を実行すると現在の設定値を参照できる

```
postgres=# SHOW work_mem;
 work_mem
----------
 10MB
(1 row)
```

- SHOW ALL;を実行すると、全てのパラメータの設定とそれらの説明が表示される

■pg_settingsビュー参照

- パラメータの設定値だけでなく、データ型や変更に関する制限、設定経緯（設定ファイル・環境変数・SET文など）も分かる

© LPI-Japan 2014. All rights reserved.

■listen_addresses

- TCP/IP接続を待ち受けるIPアドレスを指定する
- 複数指定する場合は、カンマで区切る
- '*'を指定すると全てのIPアドレスで待ち受ける
- 空（''）に設定してもUNIXドメイン接続は常に待ち受ける

データ型	文字列
デフォルト値	'localhost'
変更可能タイミング	インスタンス起動時

■port

- 接続を待ち受けるポート番号を指定する

データ型	数値
デフォルト値	5432（IANAにPostgreSQL用として登録済み）
変更可能タイミング	インスタンス起動時

© LPI-Japan 2014. All rights reserved.

■max_connections

- 同時最大接続数を指定する
- この設定を増やすと共有メモリ消費量が増える

データ型	数値
デフォルト値	100
変更可能タイミング	インスタンス起動時

© LPI-Japan 2014. All rights reserved.

エラー報告とログ取得（1）

METRO OSS-DB

log_destination

- サーバ側ログの出力先を指定する
  - stderr：テキスト形式で標準エラーに出力
  - csvlog：CSV形式で標準エラーに出力（要logging_collector=on）
  - syslog：テキスト形式でsyslogに出力
  - eventlog：Windowsのイベントログに出力（Windows環境のみ）
    - 文字化けするという情報あり
- 複数指定する場合はカンマで区切る

データ型	文字列
デフォルト値	'stderr'
変更可能タイミング	リロード時

© LPI-Japan 2014. All rights reserved.

エラー報告とログ取得（2）

METRO OSS-DB

■logging_collector

- サーバプロセスの標準エラー出力をファイルにリダイレクトするかを指定する
- log_destination=csvlogを使うときは「on」にする必要がある
- 外部ツールでログローテーションする場合は「off」にする

データ型	論理値
デフォルト値	off
変更可能タイミング	インスタンス起動時

この設定を「on」にすると、「logger process」という役割のpostgresプロセスが起動する

© LPI-Japan 2014. All rights reserved.

log_line_prefix

- ログ行の先頭に付加する文字列パターンを指定する
  - %t：タイムスタンプ（年月日時分秒までの精度）
  - %u：ユーザ名
  - %d：データベース名
  - %p：プロセスID
  - %c：セッションID
  - %x：トランザクションID（トランザクション外の場合は「0」）
  - %%：「%」そのもの

データ型	文字列
デフォルト値	''（何も付加しない）
変更可能タイミング	リロード時

ログ本文とくっついてしまうので、設定値の最後に空白を入れましょう

© LPI-Japan 2014. All rights reserved.

エラー報告とログ取得（4）

METRO OSS-DB

■log_rotation_age

- ログファイルの最大寿命（ファイル切り替え間隔）を指定する
- 「0」を指定すると切り替えなし
- logging_collector=onのときのみ有効

データ型	時間（単位を省略すると分単位）
デフォルト値	1d（1日）
変更可能タイミング	リロード時

■log_rotation_size

- ログファイルの最大サイズを指定する
- 「0」を指定すると切り替えなし

データ型	サイズ（単位を省略するとkB単位）
デフォルト値	10MB
変更可能タイミング	リロード時

© LPI-Japan 2014. All rights reserved.

■お勧め設定（ファイル出力）

```
# サーバログをファイルに出力
log_destination = 'stderr'
logging_collector = on

log_directory = 'pg_log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'

# 日付が変わるか256MBになったらログファイルを切り替える
# サイズは運用しながら調整
log_rotation_age = 1d
log_rotation_size = 256MB

log_line_prefix = '%t %d %u %p %x '  #最後の空白を忘れずに
```

© LPI-Japan 2014. All rights reserved.

リソース制限（１）

METRO OSS-DB

■shared_buffers

- 共有バッファとして使用するメモリサイズを指定する
- 他のサービスがなければ物理メモリサイズの２５％程度
  - OSのディスクキャッシュも期待できるので、あまり大きくしない
- 大きい値にする場合、OSのカーネルパラメータ変更が必要
  - SHMMAX、SHMALLなど

データ型	サイズ
デフォルト値	32MB(環境により異なる)
変更可能タイミング	インスタンス起動時

© LPI-Japan 2014. All rights reserved.

リソース制限（2）

■work_mem

- バックエンドがクエリ処理中に使用するヒープメモリ量を指定する
  - ソートやハッシュ結合などで使われるメモリ
- インスタンス全体の制限でなく、個々の処理での上限なので注意！
  - 大きくしすぎるとOut-of-Memoryが発生する
- postgresql.confの設定は小さめを推奨
  - バッチ処理などのセッションで個別にSET文で増やす

データ型	サイズ
デフォルト値	10MB
変更可能タイミング	いつでも

© LPI-Japan 2014. All rights reserved.

■checkpoint_segments

- チェックポイント間で出力できる最大WALセグメント数を指定する
- この設定が小さいと頻繁にディスク書き出しが発生し性能が低下する
- 最初は「10」程度に設定し、様子を見て調整する
- 大きくしすぎると、$PGDATA/pg_xlogが肥大化するので注意

データ型	数値
デフォルト値	3
変更可能タイミング	リロード

■wal_buffers

- WAL出力用バッファのサイズを指定する
- １トランザクションが生成するWALのサイズより大きくする

データ型	サイズ
デフォルト値	64kB
変更可能タイミング	インスタンス再起動

© LPI-Japan 2014. All rights reserved.

バックアップとリカバリ

© LPI-Japan 2014. All rights reserved.

データベースのバックアップ

METRO OSS-DB

■バックアップの重要性

- データベースで管理している情報は重要なもの
- ストレージをはじめとするハードウェアはいつか必ず故障するので、バックアップをとって備えることが肝要

■きちんとしたストレージを使っていれば大丈夫？

- 大丈夫ではありません！！！
- RAIDで冗長構成にしていても、オペレーションミスやプログラムバグによるデータ破損には対応できません

■どのようにバックアップをとったらよい？

- 管理するデータの重要度や復旧に許される時間を考慮して、適切なバックアップ計画を立てることが必要
- 計画を立てたら、必ずリハーサルして手順や所要時間を明確に！

© LPI-Japan 2014. All rights reserved.

バックアップ手法（1）

METRO OSS-DB

- pg_dump/pg_dumpall
  - インスタンス稼働状態で論理バックアップを取得
  - データはバックアップ開始時点のもので、一貫性あり
    - バックアップした時点にのみリカバリ可能
  - pg_dumpはデータベース単位
  - pg_dumpallはユーザ情報を含むデータベースクラスタ全体
  - 設定ファイルは別途バックアップしておく必要がある
  - バックアップ取得元とリカバリ先でメジャーバージョンが異なっていてもリカバリ可能

© LPI-Japan 2014. All rights reserved.

バックアップ手法（2）

METRO OSS-DB

■コールドバックアップ
- インスタンス停止状態でデータベースクラスタ全体をコピー
  - rsyncやtarを使うのが一般的
  - ストレージのスナップショット機能も利用可能
- バックアップした時点にのみリカバリ可能
- 設定ファイルなどもまとめてバックアップされるので手順がシンプル
- リカバリ時間は他の手法と比べて短い
- バックアップを類似構成の別マシンにコピーすることも可能
  - ただし、メジャーバージョンが一致していること

© LPI-Japan 2014. All rights reserved.

バックアップ手法（3）

■PITR（Point In Time Recovery）

- 運用中に出たWALをアーカイブしておき、定期的にインスタンス稼働中のデータベースクラスタ全体をコピー（ベースバックアップ）
- ベースバックアップ取得前後にインスタンスにバックアップ開始・終了を通知する必要がある
- ベースバックアップ取得から障害発生までの間の任意の時点の状態にリカバリできる
  - オペレーションミスなどへの対応に便利
- 手順が複雑だが最も自由度が高い

© LPI-Japan 2014. All rights reserved.

使い方

- pg_dump [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|--filename=出力ファイル名（省略すると標準出力にダンプ）
- -F|--format＝出力フォーマット
  - p/plain ：テキスト形式（デフォルト）
  - c/custom ：バイナリ形式・自動的に圧縮
  - t/tar ：tar形式

リカバリ方法

- 事前にデータベースクラスタを作成してインスタンスを起動
- リカバリ先データベースも事前に作成（別データベースでも可）
- テキスト形式はSQL文でダンプされるのでpsqlコマンドでリカバリ
  - psql -f ダンプファイル名 データベース名
- それ以外の形式はpg_restoreコマンドでリカバリ
  - pg_restore -d データベース名 ダンプファイル名

© LPI-Japan 2014. All rights reserved.

■使い方
- pg_dumpall [接続オプション] [オプション]

■オプション
- -f|--filename=出力ファイル名（省略すると標準出力にダンプ）
- -g|--globals-only：グローバルオブジェクトのみバックアップ
  - 一部のデータベースのみバックアップしたい場合に使用

■リカバリ方法
- 事前にデータベースクラスタを作成しておく
- テキスト形式なのでpsqlコマンドでリカバリ
  - psql -f ダンプファイル名 postgres

© LPI-Japan 2014. All rights reserved.

コールドバックアップ

■バックアップ方法

- 事前にデータベースインスタンスが停止する
  - 稼働中にバックアップすると一貫性がなくリカバリできない
- データベースクラスタ全体をコピーする方法は自由
  - cp -rp $PGDATA $BACKUP_DIR/
  - tar zcf $PGDATA $BACKUP_DIR/pgdata.tar.gz
  - ストレージのスナップショット機能でボリューム全体をコピー

■リカバリ方法

- 古いデータベースクラスタを退避または削除
- 元のデータベースクラスタの位置にバックアップを展開
  - 展開方法はバックアップ方法に応じて
- データベースインスタンスを起動

© LPI-Japan 2014. All rights reserved.

■事前設定

- 普段からWALを安全な場所にアーカイブしておく
  - wal_level=archive
  - archive_mode=on
  - archive_command='cp -i %p /path/to/archive/%f'

■バックアップ方法

- ベースバックアップ取得（関数実行はスーパーユーザで）
  - SELECT pg_start_backup('バックアップラベル');
  - データベースクラスタ全体をコピー
    - コピー開始後のファイル変更をエラーとしないように注意
  - SELECT pg_stop_backup();
- ベースバックアップが取れたらそれ以前のアーカイブWALは破棄可能

PostgreSQL 9.1以降ではpg_basebackupコマンドで ベースバックアップが簡単に取得可能

© LPI-Japan 2014. All rights reserved.

PITR（2）

METRO OSS-DB

■リカバリ方法

- インスタンスが稼働している場合は停止
- $PGDATA/pg_xlogの中身を安全な場所に退避（※）
  - 障害直前に変更内容はここにしかないので忘れずに！
- データベースクラスタを退避または削除
- リカバリしたい時点の直前のベースバックアップを元の位置に復元
- $PGDATA/pg_xlogの中を全て削除し、そこに（※）をコピー
- recovery.confを作成し$PGDATAに配置
  - restore_command = 'cp /path/to/archive/%f %p'
- pg_hba.confを編集して一般ユーザの接続を拒否
- インスタンスを起動すると自動的にリカバリが始まる
- リカバリが完了するとrecovery.confがrecovery.doneにリネームされる
- データベースの内容を確認
- pg_hba.confを元の設定に戻して設定をリロード

© LPI-Japan 2014. All rights reserved.

定常的なメンテナンス

© LPI-Japan 2014. All rights reserved.

統計情報の取得

METRO OSS-DB

■統計情報とは

- PostgreSQLのプランナがクエリのコスト計算に使用する列単位やテーブル単位のデータ特性
- コストはデータの内容から推測するため、適切なプランを選択するには現在の状態を反映した統計情報が必要

■統計情報の取得方法

- SQL文
  - ANALYZE [テーブル名]:
- OSコマンド
  - vacuumdb -Z|--analyze-only [接続オプション]
    [--t|--table='テーブル名'] [データベース名]
- 両方とも、テーブル名省略時はデータベース全体の統計情報を取得

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

■不要領域とは

- PostgreSQLは追記型アーキテクチャを採用しており、更新処理によってデータファイル内に更新前データが蓄積されていく
  - 複数のユーザからの同時処理を実現するためにMVCC（MultiVersion Concurrency Control：多版型同時実行制御）を採用している
  - この方式では、更新や削除が実行されると対象データに更新済みマークをつけ、更新後データは別の場所に保存する
  - この仕組みによって、更新したトランザクション以外が更新前のデータをロック競合なしで参照できる
- このまま運用を進めていくと、更新前のデータが残りデータファイルのサイズが実データ量よりも大きくなっていく
  - ファイルサイズが大きくなると、ストレージ容量の不足やパフォーマンス悪化といった問題が発生する
- そこで、「VACUUM」処理を実行して不要領域を回収してデータファイル肥大化を防ぐ必要がある

© LPI-Japan 2014. All rights reserved.

VACUUMの動作イメージ

■データファイルの内部構造イメージ

チーム	得点	削除
東京	0	
千葉	0	
埼玉	0	

UPDATE

チーム	得点	削除
東京	0	X
千葉	0	
埼玉	0	
東京	100	

UPDATE

チーム	得点	削除
東京	0	X
千葉	0	X
埼玉	0	
東京	100	
千葉	40	

VACUUM

チーム	得点	削除
埼玉	0	
東京	100	
千葉	40	

UPDATE

チーム	得点	削除
埼玉	80	
埼玉	0	X
東京	100	
千葉	40	

INSERT

チーム	得点	削除
埼玉	80	
神奈川	90	
埼玉	0	X
東京	100	
千葉	40	

© LPI-Japan 2014. All rights reserved.

VACUUMの方法（1）

METRO OSS-DB

■SQL文

- 書式
  - VACUUM [オプション] [テーブル名];
- オプション
  - FULL：不要領域回収後、データファイルを切り詰める

■OSコマンド

- 書式
  - 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、データファイルを切り詰める
  - -z|--analyze　：ANALYZEを併せて実行する

© LPI-Japan 2014. All rights reserved.

VACUUMの方法（2）

■自動VACUUM

- 8.1以降ではインスタンス内にVACUUM実行専用のプロセスがあり、一定以上の割合で更新されたテーブルを自動的にVACUUMする
  - ANALYZEも同時に実行される
- 自動VACUUMは8.3以降はデフォルトでonになっているため、更新量があまり多くないシステムでは手動VACUUMは不要の場合もある
- 自動VACUUMは、通常のクエリにあまり影響を与えないように、ある程度処理したらしばらくスリープする、というサイクルで実行される

© LPI-Japan 2014. All rights reserved.

SQLによる開発

- SQL文の基本
- オブジェクトと権限
- DMLによるデータ操作
- 関数と演算子
- トランザクション

© LPI-Japan 2014. All rights reserved.

SQL文の基本

© LPI-Japan 2014. All rights reserved.

■SQLとは

- 関係データベースでデータの定義や操作をするための言語

■SQLの規格

- ANSI、ISOなどで標準化されている
- 基本部分はデータベース製品に依存しないが、一部独自拡張がある
  - PostgreSQLはドキュメントに標準・独自拡張を明記
- 制定された年号で規格を区別（SQL92、SQL99、SQL:2008など）

■SQLの種類

- DDL:Data Definition Language（データ定義言語）
  - テーブルやインデックスの作成・変更・削除など
- DML:Data Manipulation Language（データ操作言語）
  - データの追加・更新・削除・検索など
- DCL:Data Control Language（データ制御言語）
  - データのアクセス権設定・トランザクション制御など

© LPI-Japan 2014. All rights reserved.

SQLの特徴

METRO OSS-DB

■大文字・小文字区別なし
- キーワード（予約語）や識別子（テーブル名など）は大文字と小文字を区別しない
  - PostgreSQLでは内部的に小文字に変換
- 大文字・小文字の区別したい場合は、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

■文字列リテラル
- 文字列リテラルはシングルクォートで囲む
  - 文字列中のシングルクォートは二重にしてエスケープ
  - 'abc''123'は「abc'123」として解釈される

■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 2014. All rights reserved.

NULL値

METRO OSS-DB

■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では空文字列＝NULL
- 一致、大小比較、文字列連結などの演算の結果はNULLとなる
- NULLであるか？の判断は専用の演算子を用いる
  - 式 IS NULL　：式がNULLであれば真
  - 式 IS NOT NULL　：式がNULLでなければ真

■psqlでの区別

- \pset null '(NULL)'とすると「(NULL)」と表示される

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

オブジェクトと権限

© LPI-Japan 2014. All rights reserved.

■テーブルとは

- データを格納する容器
- 複数の列からなり、各列に独立した値を格納
- 複数の行を保持できる

■テーブルの例

- employeeテーブル

emp_id	emp_name	dept_id	emp_date
1	佐藤	1	1971-01-23
2	鈴木	3	1983-10-12
3	渡辺	4	1957-07-01
4	佐藤	4	2000-01-01
5	山本	3	1973-10-03

© LPI-Japan 2014. All rights reserved.

■テーブルの作成

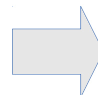
- CREATE TABLE文で作成

■CREATE TABLE文の基本形

- CREATE TABLE テーブル名 (
  列名 データ型[, 列名 データ型...]
  );

■CREATE TABLE文の実行例

```
jinji=>CREATE TABLE dept (
jinji->    dept_id   int,
jinji->    dept_name text,
jinji->    place     text
jinji->);
CREATE TABLE
jinji=> \d
        List of relations
 Schema | Name | Type  | Owner
--------+------+-------+-------
 public | dept | table | jinji
(1 row)
```



```
jinji=> \d dept
       Table "public.dept"
  Column   |  Type   | Modifiers
-----------+---------+-----------
 dept_id   | integer |
 dept_name | text    |
 place     | text    |

jinji=>
```

© LPI-Japan 2014. All rights reserved.

データ型（1）

METRO OSS-DB

■数値データ型

- 整数や実数を保存するデータ型

データ型	サイズ(byte)	説明
smallint	2	16bit符号付き整数
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bigint

■論理値データ型

- 論理値を保持するデータ型

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

© LPI-Japan 2014. All rights reserved.

データ型（2）

METRO OSS-DB

■文字列データ型

- 文字列を保存するデータ型（長さ制限はバイト数ではなく文字数）

データ型	説明
text	文字列
character varying(n)/ varchar(n)	文字数制限付きの文字列
character(n)/char(n)	文字数制限付きの固定長文字列 （末尾空白詰め）
char	1文字
name	63バイトまでのオブジェクト名称 （システム内部でのみ使用）

■通貨データ型

- 金額を保持するデータ型

データ型（別名）	サイズ（byte）	説明
money	8	金額（精度はlc_monetaryパラメータで決定）

© LPI-Japan 2014. All rights reserved.

データ型（3）

METRO OSS-DB

■日付・時刻データ型

- 日付や時刻を保存するデータ型

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

■OIDデータ型

- OID（オブジェクトID）を保持するデータ型

データ型	サイズ(byte)	説明
oid	4	データベースオブジェクト(テーブルや行など)を識別するID

© LPI-Japan 2014. All rights reserved.

■制約とは

- テーブル内のデータの妥当性を保証する仕組み
- テーブル作成時に指定する他、作成後にALTER TABLE文で追加・削除することも可能
  - 既存のデータが制約に合致しない場合は挿入・更新時にエラーとなる

■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
- ドメイン制約

© LPI-Japan 2014. All rights reserved.

NOT NULL制約

METRO OSS-DB

■NOT NULL制約とは

- 列の値としてNULL値を許容しないという制約
- 常に有効な値を持つ列に定義する

■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 2014. All rights reserved.

検査（チェック）制約

■検査（チェック）制約とは

- 基準に合わ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 CHECK(チェック基準), ...);
  - CREATE TABLE テーブル名 (列定義...,
    CONSTRAINT 制約名 CHECK(チェック基準));
- 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(チェック基準);

■チェック式の書き方

- チェック式は、SQLの条件式（論理値を返す式）で記述する
  - name = upper(name) ：小文字を含まないこと

© LPI-Japan 2014. All rights reserved.

一意（ユニーク）制約

METRO OSS-DB

■一意（ユニーク）制約とは

- ある列の組み合わせがテーブル内で重複する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 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...]);

■注意点

- NULL値同士は一致しないので、(1, NULL)と(1, NULL)のような組み合わせは許可

© LPI-Japan 2014. All rights reserved.

主キー（プライマリキー）制約

METRO OSS-DB

■主キー（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
- 一意制約と異なり定義列のNULL値を許容しないので、テーブル内のデータを一意に特定可能

■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...]);

■注意点

- 暗黙でインデックスを作成する
- テーブルに一つだけ定義可能

© LPI-Japan 2014. All rights reserved.

外部キー（参照整合性）制約

METRO OSS-DB

■外部キー（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...]);

■注意点

- 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 2014. All rights reserved.

ドメイン制約

■ドメインとは

- 制約条件を付加した独自データ型
- CREATE DOMAIN文で作成し、テーブル定義でデータ型として使用
- 複数のテーブルに共通する制約を一元管理できる

■ドメインの作成

- CREATE DOMAIN ドメイン名 AS
  データ型 [NOT NULL | NULL] [CHECK(チェック基準)];
- チェック基準では、列値を「VALUE」というキーワードで参照可能

■ドメインの例

```
jinji=> CREATE DOMAIN dept_code_type AS char(4)
jinji->     NOT NULL CHECK(VALUE = upper(VALUE));
CREATE DOMAIN
jinji=> CREATE TABLE dept (dept_code dept_code_type, dept_name
text...);
CREATE TABLE
jinji=> INSERT INTO dept VALUES('a002', '人事部');
ERROR:  value for domain dept_code_type violates check
constraint "dept_code_type_check"
```

© LPI-Japan 2014. All rights reserved.

制約の削除

OSS-DB

- 制約の削除はALTER TABLE文で
  - ALTER TABLE DROP CONSTRAINT 制約名;

© LPI-Japan 2014. All rights reserved.

デフォルト値

METRO OSS-DB

■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 2014. All rights reserved.

テーブル定義変更

テーブル名と列名の変更

- テーブル名と列名はALTER TABLE RENAME文で変更
- 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

- データ型はALTER TABLE文で変更
- 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

- 列の追加・削除はALTER TABLE文
- 新しい列は最後に追加される
- 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
- ALTER TABLE テーブル名 DROP COLUMN 列名;

© LPI-Japan 2014. All rights reserved.

テーブルの削除

METRO OSS-DB

テーブルの削除

- DROP TABLE文で削除
- 格納されたデータもまとめて削除

DROP TABLE文の基本形

- DROP TABLE テーブル名;

DROP TABLE文の実行例

```
jinji=> \d
        List of relations
 Schema | Name | Type  | Owner
--------+------+-------+-------
 public | dept | table | jinji
(1 row)

jinji=> DROP TABLE dept;
DROP TABLE
jinji=> \d
No relations found.
jinji=>
```

© LPI-Japan 2014. All rights reserved.

ビュー

METRO OSS-DB

■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
- CREATE VIEW文で作成する
- ビューは読み取り専用で、書き込みはできない

■定義方法

- CREATE VIEW ビュー名 AS 問い合わせ;

■削除方法

- DROP VIEW ビュー名;

© LPI-Japan 2014. All rights reserved.

スキーマ

METRO OSS-DB

■スキーマとは

- データベースオブジェクトが所属する名前空間
  - 同じ名前のオブジェクトでも、スキーマが異なれば共存が可能
- 「スキーマ名.オブジェクト名」が完全修飾名
- ひとつのデータベースに複数作成することが可能（ネストは不可）
- 初期状態では「public」というスキーマがある

■スキーマの作成

- CREATE SCHEMA文で作成
  - CREATE SCHEMA スキーマ名;

■オブジェクトの探索

- search_pathパラメータの設定に沿ってオブジェクトを探索
  - デフォルトは'"$user",public'（"$user"は接続ユーザ名）
  - 環境変数のPATHの考え方とほぼ同じ
- オブジェクト作成時、スキーマを指定しないとsearch_path内の最初のスキーマに作成される

© LPI-Japan 2014. All rights reserved.

シーケンス

METRO OSS-DB

■シーケンスとは

- 数列生成器
- 高速だが、トランザクション（後述）をロールバックしても一度使用した値は戻らないので、完全な連番の実現には使えない
- CREATE SEQUENCE文で作成
- serial型、bigserial型はシーケンスを使って実現

■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

- 現在値の取得（進まない）　：currval('シーケンス名');
- 次の値の取得（進む）　：nextval('シーケンス名');
- 次の値の設定　：setval('シーケンス名', 次の値);

© LPI-Japan 2014. All rights reserved.

オブジェクトに対する権限（１）

METRO OSS-DB

■オブジェクト権限とは

- あるユーザがあるオブジェクトに対して可能な操作のこと
  - jinjiユーザはemployeeテーブルのデータを変更してもよいがdeptテーブルは参照のみで変更はできない、など
- スーパーユーザはあらゆるオブジェクト権限チェックをバイパスする
  - 安易にスーパーユーザ権限を付与しない！
- 権限の付与はGRANT文を、剥奪はREVOKE文を使用

■権限の付与と剥奪

- GRANT 権限 ON オブジェクト名 TO ユーザ名
  [WITH GRANT OPTION];
  - 「WITH GRANT OPTION」をつけると、そのユーザは他者に権限を付与できる
- REVOKE 権限 ON オブジェクト名 FROM ユーザ名;

© LPI-Japan 2014. All rights reserved.

オブジェクトに対する権限（2）

METRO OSS-DB

■テーブル・ビューの主な権限

- SELECT/INSERT/UPDATE/DELETE/TRUNCATE
  - SQL文を実行可能
  - SELECT/INSERT/UPDATEは列単位で指定可能
- REFERENCE
  - 外部キーで参照可能（参照元テーブルにもこの権限が必要）
  - 列単位で指定可能
- TRIGGER
  - トリガーを作成可能
- ALL
  - 全ての権限を一括で付与/剥奪

© LPI-Japan 2014. All rights reserved.

オブジェクトに対する権限（3）

■データベースの主な権限

- CONNECT
  - データベースに接続が可能
- CREATE
  - データベース内にスキーマを作成可能

■スキーマの主な権限

- USAGE
  - スキーマ内のオブジェクトへのアクセスが可能
- CREATE
  - スキーマ内にオブジェクトを作成可能

■シーケンスの主な権限

- USAGE
  - currval()およびnextval()の呼び出しが可能
- UPDATE
  - nextval()およびsetval()の呼び出しが可能

© LPI-Japan 2014. All rights reserved.

オブジェクトに対する権限（4）

METRO OSS-DB

■デフォルト権限

- 作成直後のオブジェクトについて、所有者（一般に作成ユーザ）は全権限を持つが、他のユーザは一切の権限を持たない
- ALTER DEFAULT PRIVILEGES文であるユーザがこれから作成するオブジェクトのデフォルト権限を変更可能
  - 既存のオブジェクトの権限は変更されないので注意

■変更方法

- ALTER DEFAULT PRIVILEGES FOR USER 作成ユーザ名
      GRANT 権限 ON オブジェクト種別 TO 付与ユーザ名;
- ALTER DEFAULT PRIVILEGES FOR USER 作成ユーザ名
      REVOKE 権限 ON オブジェクト種別 FROM 剥奪ユーザ名;
- オブジェクト種別
  - TABLES/SEQUENCES/FUNCTIONS

© LPI-Japan 2014. All rights reserved.

権限付与対象ユーザ

METRO OSS-DB

■権限管理の対象ユーザ

- GRANT文の権限付与対象ユーザに「public」を指定すると、全てのユーザ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Aから剥奪、という手順をとってもユーザAはpublicの権限を経由してアクセス可能なままになる

■ロール権限

- GRANT文で権限をロールに付与し、そのロールをユーザに付与できる
  - PostgreSQLではユーザとロールは同一のもので区別されない
- GRANT ロール名 TO ユーザ名 [WITH GRANT OPTION];
- このとき、ロールの持つLOGIN権限やSUPERUSER権限も同時に付与されるので注意

© LPI-Japan 2014. All rights reserved.

権限の確認方法

METRO OSS-DB

- psqlのメタコマンド
  - \dp
    - テーブル、ビュー、シーケンスの一覧に権限を表示
  - \du+
    - ユーザ一覧に権限を表示
  - \dn+
    - スキーマ一覧に権限を表示
  - \ddp
    - デフォルト権限の一覧を表示

© LPI-Japan 2014. All rights reserved.

情報スキーマとシステムカタログ

METRO OSS-DB

■情報スキーマとは

- データベースオブジェクトに関する情報を保持するスキーマ
- SQL標準で規定されており、データベース製品に依存しない

■システムカタログとは

- PostgreSQL独自の内部テーブルで、データベースクラスタ全体の管理に必要な情報を保持している

© LPI-Japan 2014. All rights reserved.

DML文によるデータ操作

© LPI-Japan 2014. All rights reserved.

DMLの種類

■DMLはデータを操作する言語

- 検索
  - 蓄積されたデータを特定の形で取得する
- 挿入
  - 新しいデータを追加する
- 更新
  - 特定のデータの内容を変更する
- 削除
  - 特定のデータを削除する

© LPI-Japan 2014. All rights reserved.

サンプルデータ

METRO OSS-DB

■employeeテーブル

emp_id	emp_name	dept_id	emp_date
1	佐藤	1	1971-01-01
2	鈴木	3	1983-10-01
3	渡辺	4	1971-07-01
4	佐藤	4	2000-01-01
5	山本	3	1973-10-03

■deptテーブル

dept_id	dept_name	place
1	経理部	新宿
2	総務部	新宿
3	営業部	品川
4	開発部	池袋

© LPI-Japan 2014. All rights reserved.

データの検索（1）

METRO OSS-DB

■検索するにはSELECT文
- 最も単純な形は「SELECT * FROM テーブル」
- 「SELECT *」は「全列」という意味
  - 列の並びはテーブル定義に従う

■実行例

```
jinji=> SELECT * FROM employee;
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      1 | 佐藤     |       1 | 1971-01-01
      2 | 鈴木     |       3 | 1983-10-01
      3 | 渡辺     |       4 | 1971-07-01
      4 | 佐藤     |       4 | 2000-01-01
      5 | 山本     |       3 | 1973-10-03
(5 rows)
```

© LPI-Japan 2014. All rights reserved.

■特定の列だけを取得（射影：Projection）

- SELECT句に取得したい列だけを指定
- 「列名 AS 別名 」とすると結果の列名を変更可能
- 順番の入れ替えも可能
- 計算結果など列値以外も取得可能

■実行例

```
jinji=> SELECT current_date - emp_date + 1 as dulation, emp_name
jinji-> FROM employee;
 dulation | emp_name
----------+----------
    15443 | 佐藤
    10787 | 鈴木
    15443 | 渡辺
     4851 | 佐藤
    14437 | 山本
(5 rows)
```

在籍日数：入社日から今日までの日数

© LPI-Japan 2014. All rights reserved.

データの検索（3）

METRO OSS-DB

■特定の行を取得（選択：Selection）

- WHERE句に指定した条件が真になる行だけが抽出される
  - ➔ SELECT * FROM dept WHERE place = '新宿';
- 上記例では「新宿勤務の部署」の全列を取得

■実行例

```
jinji=> SELECT * FROM dept WHERE place = '新宿'
 dept_id | dept_name | place
---------+-----------+--------
       1 | 経理部     | 新宿
       2 | 総務部     | 新宿
(2 rows)
```

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

■組み合わせた例

```
jinji=> SELECT * FROM dept;
 dept_id | dept_name | place
---------+-----------+--------
       1 | 経理部    | 新宿
       2 | 総務部    | 新宿
       3 | 営業部    | 品川
       4 | 開発部    | 池袋
(4 rows)

jinji=> SELECT dept_name FROM dept WHERE place = '新宿';
 dept_name
-----------
 経理部
 総務部
(2 rows)

jinji=>
```

© LPI-Japan 2014. All rights reserved.

比較演算子

■検索条件は「論理値を返す式」

- ANDやORで条件を組み合わせて複雑な条件を指定することも可能
  - ANDでつないだ条件は、全てが真の場合に真
  - ORでつないだ条件は、いずれかが真の場合に真

■比較演算子

演算子	真となる条件
式 = 式	両辺が一致した場合
式 != 式、式 <> 式	両辺が一致しない場合
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 最小 AND 最大	式の値が「最小」以上「最大」以下の場合 「式 >= 最小 AND 式 <= 最大」と同義
式 IN (値, 値, 値...)	式が値リストのいずれかと一致した場合

© LPI-Japan 2014. All rights reserved.

■LIKE演算子

- 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真（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真（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 2014. All rights reserved.

■SIMILAR TO演算子

- 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真
- 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 2014. All rights reserved.

■POSIX正規表現

- 文字列 ~ パターン
  - パターンが文字列の一部にマッチしたら真
  - 大文字小文字区別あり
- 文字列 ~* パターン
  - パターンが文字列の一部にマッチしたら真
  - 大文字小文字区別なし
- 文字列 !~ パターン
  - パターンが文字列の一部にマッチしなかったら真
  - 大文字小文字区別あり
- 文字列 !~* パターン
  - パターンが文字列の一部にマッチしなかったら真
  - 大文字小文字区別なし

© LPI-Japan 2014. All rights reserved.

重複行の排除

■結果をユニークにしたい
- 例：オフィス一覧が欲しい→dept.placeにはどれだけ種類がある？
- SELECT句でDISTINCTを指定
- 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

■実行例

```
jinji=> SELECT DISTINCT place FROM dept;
 place
-------
 品川
 池袋
 新宿
(3 rows)
```

© LPI-Japan 2014. All rights reserved.

データの並び順（１）

■SELECT結果の並び順は不定

- 一見「データを挿入した順」に見える場合もあるが、保証なし
- 結果の並び順はORDER BY句で制御

■ORDER BYの書き方

- ORDER BY ソート基準
- ソート基準は列名や計算結果、列インデックス（１始まり）
- カンマ区切りで複数指定可能
  - ORDER BY 年齢 ：年齢でソート
  - ORDER BY 氏名, 年齢 ：氏名順、同一氏名の場合は年齢順
  - ORDER BY 1, 2 ：一列目順、一列目が同じ値ならば二列目順
- 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
- NULL値の扱いも指定可能
  - ORDER BY 列名 NULLS FIRST ：NULL値＜非NULL
  - ORDER BY 列名 NULLS LAST ：NULL値＞非NULL（デフォルト）

© LPI-Japan 2014. All rights reserved.

データの並び順（2）

■実行例

```
jinji=> SELECT emp_id, emp_name, emp_date FROM employee
jinji=> ORDER BY emp_name, emp_date DESC;
 emp_id | emp_name |  emp_date
--------+----------+------------
      4 | 佐藤     | 2000-01-01
      1 | 佐藤     | 1971-01-01
      5 | 山本     | 1973-10-03
      3 | 渡辺     | 1971-01-01
      2 | 鈴木     | 1983-10-01
(5 rows)
```

同じ「佐藤」は入社が遅い順

基本的にはemp_nameの昇順

■注意点

- ORDER BY句の中では列の別名は使えない
- 列値以外の基準でソートする場合は式等を改めて書く必要がある

© LPI-Japan 2014. All rights reserved.

検索結果の件数・範囲指定

METRO OSS-DB

■結果の一部範囲だけが欲しい（一覧のページングなど）
- OFFSET nで先頭n件をスキップ
- LIMIT nで取得件数を最大n件に制限
- 記述位置はORDER BY句の後（SELECT文の最後）
- 抽出結果を一定にするためには、ORDER BY句と併用すること！
- LIMIT句はPostgreSQLの独自拡張

■実行例

```
jinji=> SELECT * FROM employee ORDER BY emp_id OFFSET 1 LIMIT 2;
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      2 | 鈴木     |       3 | 1983-10-01
      3 | 渡辺     |       4 | 1971-07-01
(2 rows)
```

emp_id=1の行をスキップ

結果は２件だけ取得

■SQL標準の記法
- OFFSET count FETCH FIRST|NEXT ROW count ONLY;

どちらでもOK

© LPI-Japan 2014. All rights reserved.

テーブル同士の関連付け（１）

■employeeテーブル

emp_id	emp_name	dept_id	emp_date
1	佐藤	1	1971-01-01
2	鈴木	3	1983-10-01
3	渡辺	4	1971-07-01
4	佐藤	4	2000-01-01
5	山本	3	1973-10-03

■deptテーブル

dept_id	dept_name	place
1	経理部	新宿
2	総務部	新宿
3	営業部	品川
4	開発部	池袋

© LPI-Japan 2014. All rights reserved.

テーブル同士の関連付け（2）

METRO OSS-DB

■結合（Join）

- 部署名入りの従業員一覧が欲しい
  - dept_idでemployeeテーブルとdeptテーブルをひもづける
- 欲しい結果

emp_name	dept_name
佐藤	経理部
鈴木	営業部
渡辺	開発部
佐藤	開発部
山本	営業部

© LPI-Japan 2014. All rights reserved.

■WHERE句で結合

カンマ区切りで
テーブル名を列挙

```
• SELECT emp_name, dept_name FROM dept, employee
    WHERE dept.dept_id = employee.dept_id;
```

テーブル名をつけて
列名の曖昧さを回避

dept_idが一致する
行同士をひもづける

■JOIN構文で結合

テーブル名を「JOIN」でつなぐ

```
• SELECT emp_name, dept_name
    FROM dept d JOIN employee e
    ON d.dept_id = e.dept_id;
```

テーブルに別名をつけると
記述がシンプルに

ON」に続けてひもづける条件を記述

© LPI-Japan 2014. All rights reserved.

データの集計（1）

■データの集計とは

- 複数のデータから一つの結果値を求めること
  - 例：最大値、平均値、合計値、個数、Etc.
- テーブルの行をグループに分け、グループ別に集計することも可能

■データを集計するには

- SELECT 文で集約関数とGROUP BY句

```
jinji=> SELECT min(emp_date) FROM employee;
    min
------------
 1971-01-01
(1 row)

jinji=> SELECT dept_id, min(emp_date) FROM employee GROUP BY dept_id;
 dept_id |    min
---------+------------
       4 | 1971-01-01
       1 | 1971-01-01
       3 | 1973-10-03
(3 rows)
```

min()は最小値＝最も古い入社日

部署ごとの最古参

© LPI-Japan 2014. All rights reserved.

■JOINの書き方

- テーブル名１ JOIN テーブル名２
    ON テーブル名1.列名 = テーブル名２.列名

■結合キーが同じ列名の場合はUSINGで簡略化

- SELECT emp_name, dept_name FROM dept d
    JOIN employee e USING (dept_id);

■同じ列名が結合キーだけの場合はNATURAL JOINでさらに簡略化

- SELECT emp_name, dept_name FROM dept d
    NATURAL JOIN employee e;
- 「name」などの列が両方にあることがあるので注意！

© LPI-Japan 2014. All rights reserved.

テーブル同士の関連付け（5）

METRO OSS-DB

```
jinji=> SELECT * FROM employee;
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      1 | 佐藤     |       1 | 1971-01-01
      2 | 鈴木     |       3 | 1983-10-01
      3 | 渡辺     |       4 | 1971-01-01
      4 | 佐藤     |       4 | 2000-01-01
      5 | 山本     |       3 | 1973-10-03
(5 rows)
```

```
jinji=> SELECT * FROM dept;
 dept_id | dept_name | place
---------+-----------+-------
       1 | 経理部    | 新宿
       2 | 総務部    | 新宿
       3 | 営業部    | 品川
       4 | 開発部    | 池袋
(4 rows)
```

所属のいない「総務部」は結果に出てこない

```
jinji=> SELECT emp_name, dept_name FROM dept d
jinji-> JOIN employee e ON d.dept_id = e.dept_id;
 emp_name | dept_name
----------+-----------
 佐藤     | 経理部
 鈴木     | 営業部
 渡辺     | 開発部
 佐藤     | 開発部
 山本     | 営業部
(5 rows)
```

© LPI-Japan 2014. All rights reserved.

テーブル同士の関連付け（6）

METRO OSS-DB

■外部結合（Outer Join）
- 所属社員の分かる部署一覧が欲しい
  - dept_idでemployeeテーブルとdeptテーブルをひもづける
- 欲しい結果

dept_id	emp_name
経理部	佐藤
総務部	
営業部	鈴木
営業部	山本
開発部	渡辺
開発部	佐藤

所属社員がいなくても
結果に出てほしい

© LPI-Japan 2014. All rights reserved.

テーブル同士の関連付け（7）

■JOIN構文

- SELECT dept_name emp_name FROM dept d

  LEFT OUTER JOIN employee e

  ON d.dept_id = e.dept_id;

© LPI-Japan 2014. All rights reserved.

```
jinji=> SELECT * FROM employee;
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      1 | 佐藤     |       1 | 1971-01-01
      2 | 鈴木     |       3 | 1983-10-01
      3 | 渡辺     |       4 | 1971-01-01
      4 | 佐藤     |       4 | 2000-01-01
      5 | 山本     |       3 | 1973-10-03
(5 rows)
```

```
jinji=> SELECT * FROM dept;
 dept_id | dept_name | place
---------+-----------+-------
       1 | 経理部    | 新宿
       2 | 総務部    | 新宿
       3 | 営業部    | 品川
       4 | 開発部    | 池袋
(4 rows)
```

所属のいない「総務部」も結果に出てくる！

```
jinji=> SELECT emp_name, dept_name FROM dept d
jinji-> LEFT OUTER JOIN employee e ON d.dept_id = e.dept_id;
 emp_name | dept_name
----------+-----------
 佐藤     | 経理部
          | 総務部
 鈴木     | 営業部
 山本     | 営業部
 渡辺     | 開発部
 佐藤     | 開発部
(6 rows)
```

© LPI-Japan 2014. All rights reserved.

OSS-DB

- OUTER JOIN（外部結合）とは
  - どちらかのテーブルに対応する行がない場合にも結果に残る
    - 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
  - 対応行がなかった側の結果値はNULL
- 構文の制約
  - 外部結合はWHERE句での結合では記述できない
    - 「(+)」記法は一部製品の拡張構文
  - 基本的にはJOIN構文で慣れておけば間違いない
- 通常の結合は「内部結合（INNER JOIN）」
  - テーブルA INNER JOIN テーブルB ON 結合条件

© LPI-Japan 2014. All rights reserved.

データの集約（2）

METRO OSS-DB

■集約結果から絞り込むには

- GROUP BY句と一緒にHAVING句を使用
- HAVING句には集約後の絞り込み条件を指定
  - WHERE句は集約前の絞り込み条件

■使用例

```
jinji=> SELECT dept_id, min(emp_date) FROM employee GROUP BY
dept_id
jinji-> HAVING min(emp_date) > '1972-01-01';
 dept_id |    min
---------+------------
       3 | 1973-10-03
(1 row)
```

集計値を使って絞り込み

© LPI-Japan 2014. All rights reserved.

データの集約（3）

■集約の注意点

- 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」
- 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

■集約関数の例

- count(式|*)
  - グループ毎の件数を返す
  - 「*」を指定した場合はNULL値の除外が発生しないので単純に件数を返す
- max(式)/min(式)
  - グループ毎の最大値/最小値を返す
- sum(式)/avg(式)
  - グループ毎の合計値/平均値を返す

© LPI-Japan 2014. All rights reserved.

■副問い合わせとは

- 条件値の取得や擬似的なテーブルとして使われる問い合わせ
- SELECT句、FROM句、WHERE句などで使用できる

■SELECT句の副問い合わせ

- 検索値を求める問い合わせ
- 複数件を返す問い合わせはエラーになる

```
jinji=> SELECT emp_name,
jinji-> (SELECT dept_name FROM dept d WHERE d.dept_id =
e.dept_id)
jinji-> AS dept_name FROM employee e;
 emp_name | dept_name
----------+-----------
 佐藤     | 経理部
 鈴木     | 営業部
 渡辺     | 開発部
 佐藤     | 開発部
 山本     | 営業部
(5 rows)
```

主問い合わせのdept_idをキーにdeptを検索する副問い合わせ

副問い合わせ列の別名

副問い合わせの説明用です
通常は結合を使いましょう

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

■FROM句の副問い合わせ

- 問い合わせ結果を擬似的な表として扱い、別の問い合わせに使う
- 副問い合わせに表別名を付けないとエラーになる

```
jinji=> SELECT * FROM (SELECT emp_id, emp_name, emp_date FROM
employee
jinji-> ORDER BY emp_date LIMIT 3) AS oldest3 ORDER BY emp_name;
 emp_id | emp_name |  emp_date
--------+----------+------------
      1 | 佐藤     | 1971-01-01
      5 | 山本     | 1973-10-03
      3 | 渡辺     | 1971-01-01
(3 rows)
```

© LPI-Japan 2014. All rights reserved.

副問い合わせ（3）

■WHERE句の副問い合わせ

- 問い合わせ結果を検索条件に使う
- 演算の種類によって、複数件を返す問い合わせはエラーになる

```
jinji=> SELECT * FROM employee WHERE dept_id =
jinji-> (SELECT max(dept_id) FROM dept);
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      3 | 渡辺     |       4 | 1971-01-01
      4 | 佐藤     |       4 | 2000-01-01
(2 rows)
```

最も大きいdept_idを得る
副問い合わせを条件値として使用

最大のdept_idの部署に
所属する社員の一覧

© LPI-Japan 2014. All rights reserved.

副問い合わせ（4）

METRO OSS-DB

■複数件を返す副問い合わせ

- 演算の種類によって、複数件を返す問い合わせはエラーになる

```
jinji=> SELECT * FROM employee WHERE dept_id IN
jinji-> (SELECT dept_id FROM employee WHERE emp_date < '1972-01-01');
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      1 | 佐藤     |       1 | 1971-01-01
      3 | 渡辺     |       4 | 1971-01-01
      4 | 佐藤     |       4 | 2000-01-01
(3 rows)
```

1972年より前に入社した人がいる部署を得る副問い合わせを条件値として使用

■複数件を返す副問い合わせと使える演算子

- 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真となる
- 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真の場合真となる
- 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真となる

© LPI-Japan 2014. 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 2014. All rights reserved.

データの挿入（1）

METRO OSS-DB

■データを挿入するにはINSERT文

- INSERT INTO テーブル [(列名リスト)] VALUES (値リスト);
  - 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;
  - 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 2014. All rights reserved.

データの挿入（2）

OSS-DB

■実行例（VALUES指定）

```
jinji=> INSERT INTO dept (dept_id, dept_name) VALUES (5, '人事部');
INSERT 0 1
jinji=> SELECT * FROM dept;
 dept_id | dept_name | place
---------+-----------+--------
       1 | 経理部      | 新宿
       2 | 総務部      | 新宿
       3 | 営業部      | 品川
       4 | 開発部      | 池袋
       5 | 人事部      |
(5 rows)
```

1行追加された

省略した列はNULL値

ただし、place列にデフォルト値があれば、それが使われる

© LPI-Japan 2014. All rights reserved.

■実行例（SELECT文指定）

```
jinji=> INSERT INTO dept
jinji-> SELECT dept_id + 100, dept_name || '（追加）', '不明' FROM dept;
INSERT 0 5
jinji=> SELECT * FROM dept;
 dept_id |    dept_name     | place
---------+------------------+--------
       1 | 経理部           | 新宿
       2 | 総務部           | 新宿
       3 | 営業部           | 品川
       4 | 開発部           | 池袋
       5 | 人事部           |
     101 | 経理部（追加）   | 不明
     102 | 総務部（追加）   | 不明
     103 | 営業部（追加）   | 不明
     104 | 開発部（追加）   | 不明
     105 | 人事部（追加）   | 不明
(10 rows)
```

５行追加された

元の値＋１００で挿入

別テーブルからデータを移す場合や、テストデータを倍々に増やす場合などに便利

© LPI-Japan 2014. All rights reserved.

データの更新（1）

■データを更新するにはUPDATE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
  - 条件に一致した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
  - 値にSELECT結果を使うことも可能
    - SET count = (SELECT count(*) FROM foo...)

© LPI-Japan 2014. All rights reserved.

データの更新（2）

METRO OSS-DB

■実行例（emp_id=4の佐藤さんが高橋姓になり人事部に異動）

```
jinji=> UPDATE employee SET (emp_name, dept_id) = ('高橋', 2)
jinji-> WHERE emp_id = 4;
UPDATE 1
jinji=> SELECT * FROM employee ORDER BY emp_id;
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      1 | 佐藤     |       1 | 1971-01-01
      2 | 鈴木     |       3 | 1983-10-01
      3 | 渡辺     |       4 | 1971-01-01
      4 | 高橋     |       2 | 2000-01-01
      5 | 山本     |       3 | 1973-10-03
(5 rows)
```

１行更新された

© LPI-Japan 2014. All rights reserved.

データの削除（1）

METRO OSS-DB

■データを削除するにはDELETE文

- DELETE FROM テーブル WHERE 条件;
- 条件に一致する行を全て削除する
- 条件の書き方はSELECT文と同じ

■実行例（営業部の社員を削除）

```
jinji=> DELETE FROM employee WHERE dept_id = 3;
DELETE 2
jinji=> SELECT * FROM employee ORDER BY emp_id;
 emp_id | emp_name | dept_id |  emp_date
--------+----------+---------+------------
      1 | 佐藤     |       1 | 1971-01-01
      3 | 渡辺     |       4 | 1971-01-01
      4 | 高橋     |       2 | 2000-01-01
(3 rows)
```

２行削除された

© LPI-Japan 2014. All rights reserved.

関数と演算子

© LPI-Japan 2014. All rights reserved.

PostgreSQLにおける関数

METRO OSS-DB

- 関数＝SQL文で実行できる定義済み処理
  - 文字列操作や算術演算から、システム管理のためのものまで
  - インストール時点で使用できる組み込み関数以外に、ユーザが定義することも可能
  - 関数の実装には、様々な言語が利用可能
    - SQL、PL/pgSQL、C言語はインストール時点で利用可能
    - PL/Perl、PL/Pythonなどは追加機能で利用可能（パッケージインストールだと最初から利用可能）
- 演算子＝データ同士の演算処理
  - 算術演算や文字列演算、比較演算など
  - 組み込みの演算子以外に独自に定義することも可能

© LPI-Japan 2014. All rights reserved.

算術関数と演算子

METRO OSS-DB

■数値データを処理する演算子

- 和、差、積、商はそれぞれ「+」、「-」、「*」、「/」
- 剰余は「%」、累乗は「^」、階乗は「!」
- 平方根は「|/」、立方根は「||/」

■ビット演算（内部データ型のみ）

- AND、OR、XOR、NOTはそれぞれ「&」、「|」、「#」、「~」
- ビットシフトは左シフトが「<<」、右シフトが「>>」

■算術関数の例

- abs(値)　：絶対値
- ceil(値)　：引数より小さい最小の整数
- floor(値)　：引数より大きくない最大の整数
- round/trunc(値, 桁)　：精度指定の四捨五入/切り捨て
- random()　：0.0<=x<1.0の乱数値（double型）
- sin/cos/tan(値)　：正弦/余弦/正接

© LPI-Japan 2014. All rights reserved.

文字列関数と演算子（1）

- ■文字列を処理する関数や演算子
  - 文字列を他の形に変換
  - 非文字列を文字列に変換
  - 一部の文字列関数には、引数区切りにカンマでなくキーワードを使うバージョンがある（substring()やtrim()系など）
- ■文字列演算子の例
  - 文字列 || 文字列
    - ➔ 二つの文字列を連結する

© LPI-Japan 2014. All rights reserved.

■文字列関数や演算子の例

- lower(文字列)/upper(文字列)
  - 文字列を小文字/大文字に変換する
- substring(文字列, 開始位置[, 文字数])
  - 指定位置から指定文字数（デフォルトは残り全部）分の部分文字列を返す
  - 開始位置は文字数ベースで先頭文字が「1」
- char_length(文字列)/octet_length(文字列)
  - 文字列の文字数/バイト数を返す
  - length()はchar_length()と同じ機能
- trim/ltrim/rtrim(元文字列[, 除去する文字列])
  - 文字列の両側/左側/右側の指定文字列（デフォルトは空白）を除去する
- lpad/rpad(元文字列, 文字列長[, 追加文字列])
  - 指定文字数になるまで追加文字列を左/右に繰り返し追加する

© LPI-Japan 2014. All rights reserved.

日付・時刻関数と演算子（1）

METRO OSS-DB

- 日付や時刻を処理する関数や演算子
  - 日付や時刻の演算
  - 日付や時刻と文字列の変換
- 日付・時刻演算の例
  - 日付/時刻　+/−　時間間隔　=　日付/時刻
    - 日付/時刻を指定した時間間隔分だけ進める/戻す
  - インターバル　*　数値
    - インターバルを指定した数値倍する

© LPI-Japan 2014. All rights reserved.

日付・時刻関数と演算子（2）

METRO OSS-DB

■日付・時刻関数の例

- to_date/to_timestamp(文字列, 書式)
  - 文字列を指定書式に従って日付/タイムスタンプに変換する
- to_char(日付・時刻, 書式)
  - 日付・時刻を指定書式の文字列に変換する
- age(タイムスタンプ１, タイムスタンプ２)
  - タイムスタンプ１からタイムスタンプ２を引いた時間間隔を返す
- current_date/current_time/current_timestamp
  - 現在の日付/時刻/タイムスタンプを返す
  - カッコ「()」がつかない点に注意
- extract(フィールド指定 FROM 日付・時刻/時間間隔)
  - 日付・時刻/時間間隔から指定したフィールド部分を取得する
  - フィールド指定はsecond/month/isodow/timezone/centuryなど
  - フィールド指定は文字列ではないのでクォートは不要
- date_part(フィールド指定文字列, 日付・時刻/時間間隔)
  - extract()と同様だが、フィールド指定が文字列型

© LPI-Japan 2014. All rights reserved.

データ型変換関数（1）

METRO OSS-DB

■異なるデータ型に値を変換する関数
- 日付やタイムスタンプと文字列の変換
- 文字列から数値への変換

■日付・時刻型との変換
- to_date/to_timestamp(文字列, 書式)
  - 文字列を指定書式に従って日付/時刻データ型に変換する
- to_char(日付/時刻/タイムスタンプ/時間間隔, 書式)
  - 日付・時刻データ型の値を指定書式の文字列に変換する

■数値型との変換
- to_number(文字列, 書式)
  - 文字列を指定書式に従って数値に変換する

© LPI-Japan 2014. All rights reserved.

■関数はユーザも作成可能

- 関数の作成はCREATE FUNCTION文で
- 関数の定義には、様々な言語が利用可能
  - SQL　：単純な処理ならばSQL文で記述
  - PL/pgSQL　：OracleのPL/SQLに似た手続き言語
  - C言語　：C言語で書いた関数をSQLから呼び出し可能
  - これら以外にも、PerlやTcl、Pythonなどが利用可能だが、利用の前にcreatelangコマンドで手続き言語をデータベースに登録する必要がある

■関数の作成例

```
jinji=> CREATE FUNCTION fullname(first_name text, last_name text)
jinji-> RETURNS text AS $$
jinji$> SELECT first || ' ' || last; $$ LANGUAGE sql;
CREATE FUNCITON
jinji=> SELECT fullname('Taro', 'Yamada');
  fullname
-------------
 Taro Yamada
(1 row)
```

© LPI-Japan 2014. All rights reserved.

トリガー（1）

METRO OSS-DB

■トリガーとは

- テーブルへの操作を契機にカスタム処理を実行するしくみ
- 事前にカスタム処理を関数として作成してから、CREATE TRIGGER文でテーブルにトリガーを作成する

■トリガータイミング

- どのタイミングでトリガー関数を実行するかを指定
  - INSERT/UPDATE/DELETE/TRUNCATEとBEFORE/AFTERの組み合わせ
  - SQL種別は組み合わせ可能（例：INSERT OR UPDATE）

■トリガー単位

- トリガーを
  - FOR EACH ROW　　：各行に対して一回ずつトリガー関数を実行
  - FOR EACH STATEMENT　：元SQLに対して一回だけトリガー関数を実行

■トリガー条件

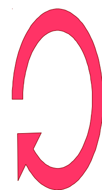
- トリガー関数を実行する条件を指定
- 行単位トリガーでは条件式で行データの変更前後の値を参照可能

© LPI-Japan 2014. All rights reserved.

トリガー（２）

■トリガーと元SQLの関係

- BEFORE STATEMENTトリガー



  - BEFORE ROWトリガー
  - 元SQLの行処理（INSERT、UPDATE、DELETE）
  - AFTER ROWトリガー
- AFTER STATEMENTトリガー

© LPI-Japan 2014. All rights reserved.

■トリガーの作成例

```
jinji=> CREATE FUNCTION log_dept_change() RETURNS trigger AS $$
jinji$> BEGIN
jinji$>     IF TG_OP = "INSERT" THEN
jinji$>         INSERT INTO log_dept VALUES(user, TG_OP, NULL, NEW);
jinji$>     ELSIF TG_OP = "UPDATE" THEN
jinji$>         INSERT INTO log_dept VALUES(user, TG_OP, OLD, NEW);
jinji$>     ELSIF TG_OP = "DELETE" THEN
jinji$>         INSERT INTO log_dept VALUES(user, TG_OP, OLD, NULL);
jinji$>     END IF;
jinji$>     RETURN NULL;
jinji$> END;
jinji$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
jinji=> CREATE TRIGGER dept_audit AFTER INSERT OR UPDATE OR DELETE ON
dept
jinji-> FOR EACH ROW EXECUTE PROCEDURE log_dept_change();
CREATE TRIGGER
```

戻り値はtriggerという疑似型

TG_OPでトリガー契機の処理が分かる

OLD/NEWは変更前後の行データ
OLD.列名で変更前の列値が取得可能

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

■トリガーの実行例

deptを更新

```
jinji=> UPDATE dept SET place = '四谷' WHERE dept_id = 1;
UPDATE 0 1
jinji=> SELECT * FROM dept;
 dept_id |    dept_name     | place
---------+------------------+---------
       1 | 経理部           | 四谷
       2 | 総務部           | 新宿
       3 | 営業部           | 品川
       4 | 開発部           | 池袋
(4 rows)

jinji=> SELECT * FROM log_dept ;
  by   | operation |    old_value     |    new_value
-------+-----------+------------------+-----------------
 jinji | UPDATE    | (1,経理部,新宿) | (1,経理部,四谷)
(1 row)

jinji=> select * From log_dept ;
```

© LPI-Japan 2014. All rights reserved.

ルール

■ルールとは

- 問い合わせの内容を書き換えるルール
- CREATE RULE文で作成
- PostgreSQLではビューを実現するために使っている
  - ビューに対するSELECTクエリをビュー定義のSELECTクエリに置換
- パーティションテーブルやビューへの直感的な更新を可能に
- 強力だが、廃止も検討されている機能なので濫用は禁物

■ルールの属性

- イベント（SELECT/INSERT/UPDATE/DELETE）
- 対象リレーション（テーブル、ビューなど）
- ルール適用条件
- 元処理の扱い（ALSO/INSTEAD）
  - 元の処理も実行するか、ルール処理のみを実行するか
- 処理内容（NOTHING/SELECT/INSERT/UPDATE/DELET/NOTIFY）
  - 何もしないNOTHINGか、SQL文を組み合わせて実行

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

トランザクション

© LPI-Japan 2014. All rights reserved.

■トランザクションとは

- 複数のSQL文の結果をまとめて確定または破棄するしくみ
- 例：銀行口座の振替処理
  - トランザクション開始
  - 振替元口座の残高を−１００００円
  - 振替先口座の残高を＋１００００円
  - トランザクション終了（確定）

どちらかのみだと
矛盾が出てしまう！

■ACID特性

- データベースのトランザクションの特性

特性	意味
Atomicity（原子性）	完全に実行されるか、まったく実行されないのどちらかである
Consistency（整合性）	トランザクションの開始前と終了後はデータベースの内容に整合性がある
Isolation（分離性）	他のトランザクションによる影響を受けない
Durability（永続性）	完了したトランザクションによる変更は確実に記録される

© LPI-Japan 2014. All rights reserved.

トランザクションの制御（1）

METRO OSS-DB

■トランザクションの開始

- BEGIN文またはSTART TRANSACTION文で開始する

■トランザクションの確定

- COMMIT文でそれまでの変更を確定する
- いったん確定した変更は取り消せない

■トランザクションの破棄

- ABORT文またはROLLBACK文でそれまでの変更を破棄する
- データベースの内容はトランザクション開始時点の状態に戻る

■トランザクションを開始しなかったら？

- psqlではSQL文毎にトランザクションを開始・確定（自動コミット）
  - ➔ 重要なデータベースでは明示的なBEGINを忘れずに！
- アプリケーションでは、各APIの仕様による

© LPI-Japan 2014. All rights reserved.

METRO OSS-DB

■実行例

```
jinji=> BEGIN;
BEGIN
jinji=> DELETE FROM employee;
DELETE 5
jinji=> SELECT count(*) FROM employee;
 count
-------
     0
(1 row)

jinji=> ROLLBACK;
ROLLBACK
jinji=> SELECT count(*) FROM employee;
 count
-------
     5
(1 row)
```

全件削除された

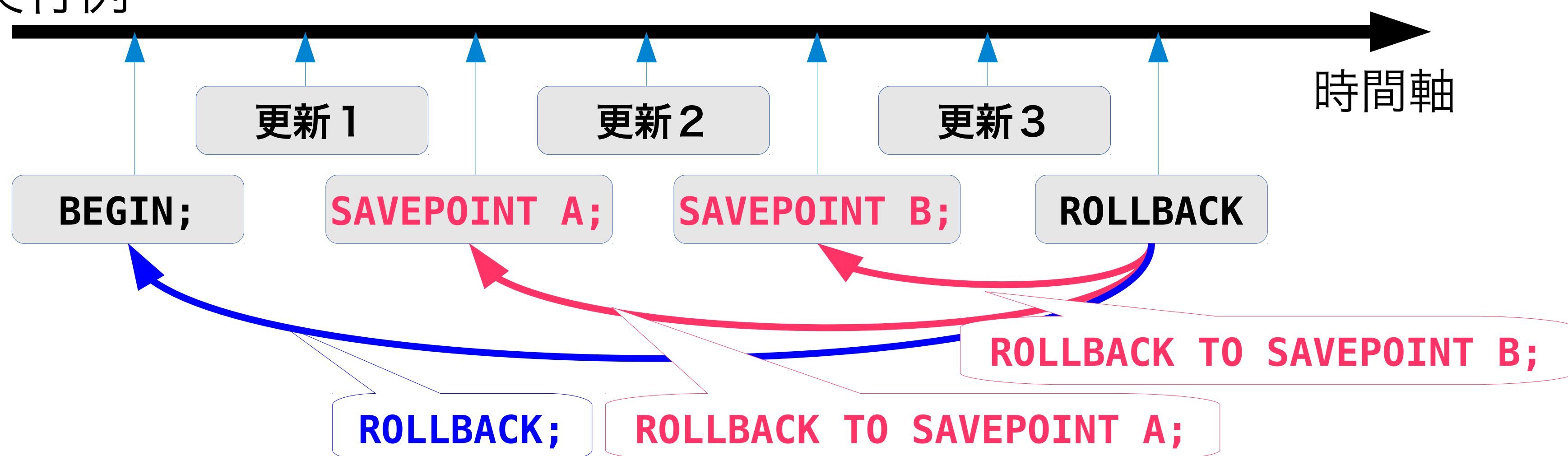
５件に戻っている

© LPI-Japan 2014. All rights reserved.

トランザクションの途中で部分的にロールバックしたい

- SAVEPOINT文でセーブポイントを設置
  - SAVEPOINT セーブポイント名;
- セーブポイント指定ロールバックで、それ以降の変更のみを破棄
  - ROLLBACK TO SAVEPOINT セーブポイント名;
- 不要なセーブポイントはRELEASE SAVEPOINT文で破棄
  - RELEASE SAVEPOINT セーブポイント名;
- コミット範囲はBEGIN以降の更新全部（部分的コミットはできない）

実行例



© LPI-Japan 2014. All rights reserved.

SET文の結果への影響

METRO OSS-DB

- トランザクション内でのSET文
  - そのトランザクションがロールバックされると、SET文で変更されたパラメータのみトランザクション開始時点の値に戻る
  - そのトランザクションがコミットされると、SET文で変更した値がそのまま残る
- トランザクション外のSET文
  - SET文が完了した時点でトランザクションがコミットされるので、変更した値がそのまま残る

© LPI-Japan 2014. All rights reserved.

■トランザクション分離レベル

- 同時に複数のトランザクションが実行されている場合、相互に影響を与える場合がある
- 他のトランザクションの影響によって受ける影響の度合いを「トランザクション分離レベル」と呼ぶ

■他のトランザクションによる影響

- 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（Fazzy 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 2014. All rights reserved.

他のトランザクションの影響（2）

■トランザクション分離レベル

特性	ダーティーリード	ファジーリード	ファントムリード	PostgreSQLでの指定時の動作
SERIALIZABLE	安全	安全	安全	SERIALIZABLE
REPEATABLE READ	安全	安全	あり	SERIALIZABLE
READ COMMITTED	安全	あり	あり	READ COMMITTED
READ UNCOMMITTD	あり	あり	あり	READ COMMITTED

■トランザクション分離レベルの設定

- セッションのデフォルト分離レベル設定
  - SET default_transaction_isolation TO|= '分離レベル';
- トランザクション開始時に分離レベルを明示
  - BEGIN|START TRANSACTION ISOLATION LEVEL 分離レベル;
- トランザクション開始直後に分離レベルを明示
  - SET TRANSACTION ISOLATION LEVEL 分離レベル;

© LPI-Japan 2014. All rights reserved.

PostgreSQLでの注意点

METRO OSS-DB

- psqlはオートコミット
  - psqlではSQL文毎にトランザクションを開始・確定（自動コミット）
    - 重要なデータベースでは明示的なBEGINを忘れずに！
  - アプリケーションでは、各APIの仕様による
- SQLでエラーでトランザクション全体がアボート
  - それまでの全ての変更はロールバックされる
  - 明示的にABORT（ROLLBACK）するまで、全てのSQLがエラーとなる
  - SQLの文法エラーやテーブル名の間違いなどでも同様
  - エラー後に処理継続できる他のRDBMS（Oracleなど）から移行した場合は要注意！
- 一部DDLがトランザクション内で実行可能
  - PostgreSQLでは、CREATE TABLEやDROP TABLEなどのDDLもトランザクション内で実行可能
    - ロールバックするとDDL実行前の状態に戻る
    - TRUNATEもロールバック可能
    - ロールバックできないDDL（DROP DATABASEなど）もあるので注意

© LPI-Japan 2014. All rights reserved.

参考文献

■Webサイト

- PostgreSQL 9.0.4文書（日本語版）
  - http://www.postgresql.jp/document/9.0/html/index.html
- Let's Postgres
  - http://lets.postgresql.jp

■書籍

- OSS教科書 OSS-DB Silver
  - 翔泳社刊 ISBN: 978-4798124421
- 徹底攻略 OSS-DB Silver問題集
  - インプレスジャパン刊 ISBN: 978-4844331933
- 独習ＳＱＬ 第２版
  - 翔泳社刊 ISBN: 978-4798117645

© LPI-Japan 2014. All rights reserved.

ご清聴ありがとうございました。

■お問い合わせ■
株式会社メトロシステムズ
佐藤　千佳
Mail: satock@metrosystems.co.jp
Twitter: @METRO_Seminar

© LPI-Japan 2014. All rights reserved.