LPI-JAPAN
Linux Professional Institute Japan

OSS-DB Exam Silver 技術解説無料セミナー

2011/10/15

特定非営利活動法人エルピーアイジャパン
テクノロジー・マネージャー
松田　神一

Agenda

OSS-DB

- OSS-DB技術者認定試験の概要
- PostgreSQLのインストール
- ポイント解説：運用管理
- ポイント解説：SQL
- OSS-DB Exam Silverの例題

自己紹介

OSS-DB

■松田　神一（まつだ　しんいち）
LPI-JAPAN テクノロジー・マネージャー

■NEC、オラクル、トレンドマイクロなどで約20年間、ソフトウェア開発に従事（専門はアプリケーション開発）
うち10年間はデータベース、およびデータベースアプリケーションの開発（Oracle、C言語、SQL言語）

■2010年7月から現職

今日のゴール

OSS-DB

■受験準備のために何をすべきかの理解
- 実機で試せる環境の準備
- 出題範囲、試験の目的、合格基準

■OSS-DB技術者認定試験についてのポイントの理解
- PostgreSQLの設定、運用管理
- SQLによるデータ操作
- 他のRDBMSとの主な違い

OSS-DB技術者認定試験の概要

■認定の種類
- Silver（ベーシックレベル）
  - OSS-DB Exam Silverに合格すれば認定される
- Gold（アドバンストレベル）
  - OSS-DB Silverの認定を取得し、OSS-DB Exam Goldに合格すれば認定される

■Silver認定の基準
- 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
- 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

OSS-DB Exam Silverの出題範囲

OSS-DB

- 一般知識（20%）
  - OSS-DBの一般的特徴
  - ライセンス
  - コミュニティと情報収集
  - RDBMSに関する一般的知識
- 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
  - トランザクションの概念

 7

出題範囲に関する注意事項

OSS-DB

- 最新の出題範囲は
  http://www.oss-db.jp/outline/examarea.shtml
  で確認できる
- 前提とするRDBMSはPostgreSQL 9.0
- Silverではosに依存する問題は出題しないが、記号や用語がOS によって異なるものについては、Linuxのものを採用している
  - 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
- 出題範囲に関するFAQ
  http://www.oss-db.jp/faq/#n02

 8

傾向と対策

OSS-DB

- Silverの合格基準は、各機能やコマンドについて
  - その目的を正しく理解していること
    - XXXコマンドを使うと何が起きるか
    - YYYをするためにはどのコマンドを使えば良いか
  - 利用法を正しく理解していること
    - コマンドのオプションやパラメータ
    - 設定ファイルの記述方法
- 出題範囲にあるすべての項目について、試験問題が用意されている
- 出題範囲詳細に載っている項目すべてについて、マニュアルなどで調査した上で、実際に試して理解する
  - 実機で試すことは極めて重要

RDBMSの種類による共通点と違い

OSS-DB

- 主な商用RDBMS: Oracle, DB2 (IBM), SQL Server (MS)
- 主なOSS RDBMS: PostgreSQL, MySQL, Firebird
- 共通点
  - RDBMSとしての各種機能
    - データ管理／入出力
    - ユーザ管理
    - アクセス権限管理、セキュリティ
    - バックアップ、リカバリ
  - SQL言語（ANSI/ISOで標準化）
- 違い
  - 各種機能の使い方
    - コマンドとオプション
    - 設定ファイルとパラメータ
  - SQLの方言
  - 独自拡張機能

データベース学習のヒント

OSS-DB

- どの製品にも共通の機能もあれば、同じ機能でも製品によって実行方法の異なるもの、特定の製品にしかない機能もある
- まずはDBの種類による差分はあまり気にせずに、特定のDBについて学習し、マスターする

次のステップは…

- 横展開
  他のDBについて、最初に学習したDBとの差分に注意しながら学習する
- 深掘り
  その製品のエキスパートとなるべく、更に深く学ぶ

OSS-DB

PostgreSQLのインストール

必要な環境

OSS-DB

■インストールに必要な環境
- インターネットにつながっているマシン（Windows/Mac/Linux）
- インストーラの入ったメディアがあれば、オフラインのPCでもインストール可能

■おススメの環境
- ある程度、Linuxの知識がある方にはLinuxを使うことを勧める。
- VMware Playerなどを使えば、Windows PC上に仮想Linux環境を構築し、そこにPostgreSQLをインストールして学習することができる。
- 仮想環境の良い点は、それを破壊しても、簡単に最初からやり直せるところ
- もちろん、WindowsやMacの環境に直接、PostgreSQLをインストールするのでもOK。

■参考書などを読むだけでは、十分な学習をすることはできません。
自分専用の環境を作り、そこでいろいろ試すことで学習してください。

 13

PostgreSQLのインストールと初期設定

OSS-DB

■インストール方法
- ソースコードから自分でビルドしてインストール
- ビルド済みのパッケージをインストール（様々なビルド済みパッケージがある）

■ダウンロードサイト
- http://www.postgresql.org/download/

■インストール後の初期設定
- データベースのスーパーユーザ（postgresユーザ）の作成
- 環境変数（PATH, PGDATAなど）の設定
- データベースの初期化（データベースクラスタの作成）
- データベース（サーバープロセス）の起動
- データベース（サーバープロセス）起動の自動化

■インストール方法によっては、初期設定の一部が自動的に実行される

■インストール方法によって、プログラムがインストールされる場所、データベースファイルが作られる場所が大きく異なるので注意

 14

ワンクリックインストール

OSS-DB

- Windows/Mac/Linuxいずれでも利用可能
  - EnterpriseDB社のサイトから、ビルド済みのパッケージをダウンロードして インストールする
    http://www.enterprisedb.com/products-services-training/pgdownload
  - GUIの管理ツール (pgAdmin III) も同時にインストールされる
  - ApacheやPHPなど、PostgreSQLと一緒に使われるソフトウェアも、同時に インストール可能
  - WindowsではワンクリックインストールのIを推奨
- インストールガイド (英語) は
  http://www.enterprisedb.com/resources-community/pginst-guide
- 多くの項目はデフォルト値のままで良い
  - スーパーユーザ (postgres) のパスワードの設定を求められるので、適切に設定し、それを忘れないようにすること
  - ロケール (Locale) の設定を求められるが、"Default locale"となっているのを"C"に変更することを推奨する
  - インストール終了時にスタックビルダ (Stack Builder) を起動するかどうか尋ねられるが、ここはチェックボックスを外して終了してよい。必要なら後でスタックビルダを起動することができる

ワンクリックインストール後の初期設定

OSS-DB

- postgresユーザは自動的に作成される。
- データベースの初期化、起動はインストール時に実行されるので、インストール後、すぐにデータベースに接続できる。
- データベースの自動起動の設定がされるので、マシンを再起動したときもデータベースが自動的に起動する。
- Windowsでは C:¥Program Files¥PostgreSQL¥9.0 の下にインストールされる。データベースは C:¥Program Files¥PostgreSQL¥9.0¥data の下に作られる。環境変数PATHに C:¥Program Files¥PostgreSQL¥9.0¥bin を追加するか、あるいは C:¥Program Files¥PostgreSQL¥9.0 の下の pg_env.bat を実行する。
- Linuxでは /opt/PostgreSQL/9.0 の下にインストールされる。データベースは /opt/PostgreSQL/9.0/data の下に作られる。環境変数PATHに /opt/PostgreSQL/9.0/bin を追加するか、あるいは /opt/PostgreSQL/9.0 の下の pg_env.sh を読み込む。
  (". pg_env.sh" を実行する)

Linuxへのインストール(OpenSCGのパッケージ)

OSS-DB

- http://www.openscg.org/se/postgresql/packages.jsp
  に、RedHat系、Debian系、それぞれのバイナリパッケージが用意されているので、ダウンロードして、rpmコマンド、dpkgコマンドを使ってインストールすることが可能。
  インストール方法、インストール後のセットアップなどの詳細は、上記のページの"Installing RPM's"および"Installing DEB's"のリンクに記述されている。
- RedHat系は"rpm -ihv filename"を、Debian系は"dpkg -i filename"をrootで実行すると、/opt/postgres/9.0 の下にプログラムがインストールされる。
- # /etc/init.d/postgres-9.0-openscg start
  を実行すると、postgres ユーザの作成、データベースの初期化、自動起動の設定などが行われる。データベースは/opt/postgres/9.0/dataの下に作られる。
- PATHに /opt/postgres/9.0/bin を追加するか、/opt/postgres/9.0の下の pg90-openscg.env を読み込む。
  (". pg90-openscg.env"を実行)

Linux (RedHat系) へのインストール

OSS-DB

- CentOSやFedoraでは、yumコマンドでインストールするのが基本だが、
  # yum install postgresql-server
  とすると、PostgreSQL 8.4 (あるいはもっと古いバージョン) がインストールされるので注意。
- http://yum.postgresql.org/packages.php にPostgreSQLのバージョン、およびLinuxディストリビューションに応じたパッケージのリンクがある。
- 例えば、PostgreSQL 9.0のCentOS 5.x用の32ビット版は
  http://yum.postgresql.org/9.0/redhat/rhel-5-i386/repoview/
- Available Groupsのリンクをクリックし、postgresql90 (クライアント), postgresql90-libs (ライブラリ), postgresql90-server (サーバ) の3つのパッケージをダウンロード
- rpmで、ライブラリ→クライアント→サーバ、の順でインストール
  - # rpm –ivh postgresql90-libs-9.0.5-1PGDG.rhel5.i386.rpm
    # rpm –ivh postgresql90-9.0.5-1PGDG.rhel5.i386.rpm
    # rpm –ivh postgresql90-server-9.0.5-1PGDG.rhel5.i386.rpm

Linux (RedHat系) へのインストール後の初期設定

OSS-DB

- postgres ユーザは自動的に作成される。
- プログラムは /usr/pgsql-9.0 の下にインストールされる。データベースは /var/lib/pgsql/9.0/data の下に作成される。
- 主なコマンドは /usr/bin の下にシンボリックリンクが作られるが、pg_ctl や initdb など一部のコマンドについてはリンクが作成されないので、PATHを設定するか、絶対パスで起動する必要がある。
- インストールしただけでは、データベースの初期化、起動、自動起動の設定などはされない。
  - # service postgresql-9.0 initdb （データベース初期化）
  - # service postgresql-9.0 start （データベース起動）
  - # chkconfig postgresql-9.0 on （データベース自動起動の設定）

RedHat系へのyumによるインストール

OSS-DB

- PostgreSQL 9.0をyumコマンドでインストールする場合について
  http://yum.pgrpms.org/howtoyum.php
  にパッケージとインストールガイド (英語) がある。

- リポジトリをrpmでインストールし、パッケージをyumでインストールする。

- 上記ページの“Please click here and download...”の“here”をクリック。
  http://yum.postgresql.org/repopackages.php
  に表示されているリストから、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合ったリンクをクリック。
  PostgreSQL 9.0をCentOS 5.x (32bit版) にインストールする場合は
  http://yum.postgresql.org/9.0/redhat/rhel-5-i386/pgdg-centos90-9.0-5.noarch.rpm をダウンロード。
  # rpm -ivh pgdg-centos-9.0-5.noarch.rpm
  としてリポジトリをインストールする。

RedHat系へのyumによるインストール

OSS-DB

- http://yum.pgrpms.org/howtoyum.php
  の中ほどにあるImportant noteの指示に従い、/etc/yum.repos.dの下のCentOS-Base.repoを編集する。[base]と[updates]に
  exclude=postgresql*
  を追加する。
- 最後に
  # yum install postgresql90-server
  とすればパッケージがインストールされる。

- yumでインストールした後の状態は、rpmコマンドでライブラリ、クライアント、サーバをインストールした時と同じなので、同様に初期設定を行う

Linux(Ubuntu)へのインストール

OSS-DB

- $ sudo apt-get install postgresql
  とすると、やはりPostgreSQL 8.4がインストールされてしまう。
- PPA (Personal Package Archives) を利用すれば、以下の手順でインストール可能。
  $ sudo add-apt-repository ppa:pitti/postgresql
  $ sudo apt-get update
  $ sudo apt-get install postgresql
- postgres ユーザが自動的に作られる。データベースも作成され、自動起動の設定もされる。
- プログラムは /usr/lib/postgresql/9.0 の下、データベースは /var/lib/postgresql/9.0/main の下に作られる。
- 主なコマンドは /usr/bin の下にシンボリックリンクが作られるが、pg_ctl や initdb など一部のコマンドについてはリンクが作成されないので、PATHを設定するか、絶対パスで起動する必要がある。
- インストール後の環境がちょっと特殊なので、学習環境としては推奨しない。

ソースコードからのインストール

OSS-DB

- Linuxでは、コンパイラなどの開発環境が標準で用意されており（インストールされていなくても簡単にセットアップ可能）、ソースコードから自分でビルドしてインストールするのも難しくない。
- ソースコードはPostgreSQLの公式サイトからダウンロード
  http://www.postgresql.org/ftp/source/
- ビルド、および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。
- 基本的には、
  ```
  $ ./configure
  $ make
  # make install
  ```
  を実行するだけ。
- 多くの環境では configure の実行でいくつかエラーが出るが、これを自力で解決できる人には、ソースからのインストールを勧める。
- 市販書籍では、ソースからビルドを前提に解説された記述が多い

ソースコードからインストールした後の初期設定

OSS-DB

- make install は、プログラムを /usr/local/pgsql の下にコピーするだけなので、その後の初期設定をすべて実行する必要がある。
- 初期設定の手順はオンラインマニュアルの17章に解説がある
- postgres ユーザの作成
  ```
  # useradd postgres
  ```
- 環境変数の設定（~postgres/.bash_profile、およびPostgreSQLを利用するユーザの~/.bash_profile に追記）
  ```
  export PATH=$PATH:/usr/local/pgsql/bin
  export PGDATA=/usr/local/pgsql/data
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/pgsql/lib
  export MANPATH=$MANPATH:/usr/local/pgsql/share/man
  ```
- データベース用ディレクトリの作成
  ```
  # mkdir /usr/local/pgsql/data
  # chown postgres /usr/local/pgsql/data
  # chmod 700 /usr/local/pgsql/data
  ```

ソースコードからインストールした後の初期設定

OSS-DB

■データベースの初期化と起動（postgresユーザで実行）
$ initdb -E UTF8 --no-locale
$ pg_ctl start

■自動起動の設定（RedHat系）
contrib/start-scripts/linux を /etc/rc.d/init.d/postgresql-9.0 にコピー
chkconfig --add postgresql-9.0
chkconfig postgresql-9.0 on

■自動起動の設定（Debian系）
contrib/start-scripts/linux を /etc/init.d/postgresql-9.0 にコピー
$ sudo update-rc.d postgresql-9.0 defaults 98 02

インストールに関する注意事項

OSS-DB

■インストール方法によっては、initdb, pg_ctlなど（試験範囲に含まれる）一部のコマンドへのPATHが通っていないので、PATH変数を変更する、あるいは/usr/local/binにリンクを張る、などの必要がある

■PostgreSQLの実行ファイル、ライブラリなどが置かれる場所、データベースファイルが作成される場所がどこか、インストール後に確認しておくこと（インストール方法によって大きく異なるので注意）

■yum, rpm, apt-get, dpkg等、OSやパッケージに依存したインストールコマンドや手順は出題しない

ポイント解説：運用管理

データベース運用管理の目的

- 必要な人に、適切なDBサービスを提供すること（セキュリティ管理）
  - 必要ない人にはサービスを提供しない
  - 不正なアクセスを拒絶する
  - 設定と監視
- サービスレベルの維持
  - 定められた水準のサービスを提供し続けること
    - サービスを提供する時間
    - パフォーマンスの維持
- トラブルシューティング（予防と対処）
  - DBに接続できない
  - DBが遅い
  - DBが起動しない
  - ディスク、ファイル、データの破損
  - バックアップ、リストア、リカバリ

他のRDBMSとの違い

OSS-DB

- 運用管理に必要とされる機能、実現されている機能はほぼ同じだが、使用するコマンド、パラメータ、設定ファイルなどは全く異なる
- それぞれのRDBMSについて基本からマスターする
- 同じ用語を使っていても、その意味がRDBMSの種類によって異なることもあるので注意が必要

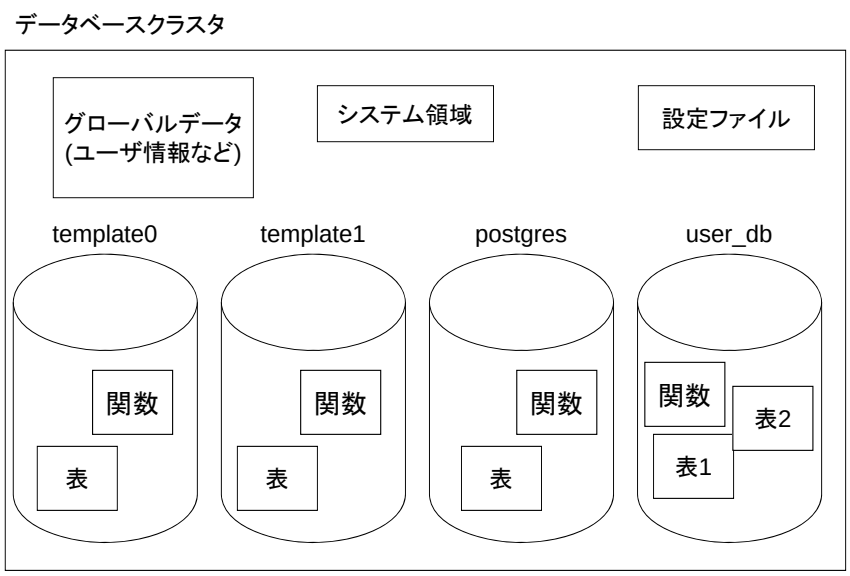
データベース構造

OSS-DB

- データベースインスタンス
  - データベースを構成するプロセス、共有メモリ、ファイルを合わせたものをインスタンスと呼ぶ
  - PostgreSQLのサーバプロセスはマルチプロセス構成で、データアクセス、ログ出力などのために、それぞれ別のプロセスが起動している
  - データベースファイルについては、その置き場所となるディレクトリを指定すると、PostgreSQLサーバがその下にファイルを作成する
- データベースクラスタ
  - 初期化された直後のPostgreSQLのインスタンスには、template0, template1という2つのテンプレートデータベースと、postgresというデータベースが含まれる。これら複数のデータベースの集合体をデータベースクラスタと呼んでいる（PostgreSQL独自の用語）
  - PostgreSQLのサーバプロセスは、1つのデータベースクラスタを管理できる、つまりクラスタ内の複数のデータベースを管理できる

データベースクラスタのイメージ

OSS-DB



データベースの初期化、起動と終了

OSS-DB

- **データベースクラスタの新規作成**
  - initdb コマンド
  - 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
    - -E：デフォルトのエンコーディング（UTF8など）
    - --locale：ロケール（ja_JPなど）
- **データベースの起動**
  - pg_ctl start
  - 主なオプション
    - -D：データベースクラスタのあるディレクトリ
- **データベースの終了**
  - pg_ctl stop
  - 主なオプション
    - -D：データベースクラスタのあるディレクトリ
    - -m：停止モード（smart/fast/immediate）
- **-D オプションを省略すると、環境変数PGDATAが使われる**

設定ファイル (postgresql.conf)

OSS-DB

- DBサーバーのリソースなど、各種パラメータの設定をするファイル
  - '#'で始まる行はコメント
  - "パラメータ名 = 値" という形式でパラメータを設定
  - 主なパラメータと設定の例
    - listen_address = '*' (TCP接続を許可する)
    - log_destination = 'syslog' (サーバーのログをsyslogに出力する)
    - log_line_prefix = '%t %p' (ログ出力時に、時刻とプロセスIDを付加
  - この他、パフォーマンスチューニングなどのための多数のパラメータが設定できるが、OSS-DB Silverの試験で問われるのは、以下の4つ(数字は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)

設定ファイル (pg_hba.conf)

OSS-DB

- HBA=Host Based Authentication
- DBへの接続を許可(あるいは拒否)する接続元、データベース、ユーザの組み合わせを設定
  - 先頭行から順に調べて、マッチする組み合わせが見つかったところで終了
  - マッチする組み合わせが見つからなければ、接続拒否
- 記述形式
  - local database名 ユーザ名 認証方法
  - host database名 ユーザ名 接続元IPアドレス 認証方法
- 記述例
  - local all postgres md5 (postgresユーザでの接続はパスワードを要求)
  - local all all ident (OSのユーザ名とDBのユーザ名が一致すれば接続可)
  - host all all 127.0.0.1/32 trust (ローカルホストからは接続可)
  - host db1 all 192.168.0.0/24 reject (192.168.0.1-255からdb1には接続不可)
  - host all all 192.168.0.0/24 trust (192.168.0.1-255から接続可)

設定の確認と変更

OSS-DB

- 実行時パラメータの設定値は、データベースに接続して SHOW コマンドを実行することで確認できる
  - => SHOW log_destination;
  - => SHOW ALL;
- 実行時パラメータの多くは、データベースに接続して SET コマンドを実行することで変更できる。ただし、その変更は現行セッション（あるいはトランザクション）内でのみ有効。
  - => SET client_encoding TO 'UTF8';
- postgresql.conf や pg_hba.conf の設定変更は、ファイルを変更しただけでは有効にならない。多くのパラメータはpostgresユーザで
  $ pg_ctl reload
  を実行することで反映される。一部のパラメータはデータベースの再起動
  $ pg_ctl restart
  をしないと変更が反映されない。
- Linuxの場合、pg_ctl を使う代わりに、root ユーザで
  # service postgresql-9.0 reload あるいは
  # /etc/rc.d/init.d/postgresql-9.0 reload
  としても良い（試験対策としては pg_ctl を覚えること）

psqlツールの利用

OSS-DB

- データベースに接続してSQLを実行するにはpsqlコマンドを使う
  psql [option...] [dbname [username]]
- 主なオプション
  - -d, --dbname : 接続先データベース名
  - -U, --username : 接続時のユーザ名
  - -h, --host : 接続先サーバのホスト名
  - -p, --port : 接続先ホストのポート番号
  - -f, --file : 使用するファイル名（psqlでは入力スクリプト）
    - 以上は他のツールでも共通に使われるオプション
  - -l, --list : 利用可能なデータベースの一覧表示して終了
- '¥'で始まるのはpsqlの独自コマンド（メタコマンド）。
  改行によって終了し、psqlツールによって処理される。
- それ以外のものはSQL文と判断され、データベースのサーバープロセスに送信される。SQL文は";"（セミコロン）で終了する。改行では終了せず、次行以降に継続される（改行はスペースと同じ）。

psqlのメタコマンド

OSS-DB

■主なpsqlのメタコマンド（'=>' はpsqlのプロンプト）
- => ¥d （テーブル一覧の表示）
- => ¥d 表名 （指定した表の列名、データ型の表示）
- => ¥du（ユーザー一覧の表示）
- => ¥set（内部変数の表示・設定）
- => ¥c db名（他のデータベースに接続）
- => ¥? （psql で使える各種コマンドに関するヘルプの表示）
- => ¥h （SQL に関するヘルプの表示）
  - => ¥h SELECT （SELECTの使い方に関するヘルプの表示）
- => ¥! OSコマンド （OSコマンドの実行）
  - => ¥! ls （カレントディレクトリのファイル一覧の表示）
- => ¥q （終了）

システム情報などの取得

OSS-DB

■システム情報取得関数
- マニュアルの9.23節のセッション情報関数を確認
- 例えば、SELECT version ()；とすると、接続先PostgreSQLサーバのバージョンが表示される

■情報スキーマ
- ANSI標準準拠
- マニュアルの34章を簡単に確認
- すべてのデータベースにinformation_schemaというスキーマが存在する
- データベース内のオブジェクトに関する情報を参照できるビュー群
  - SELECT * FROM information_schema.tables;

■システムカタログ
- PostgreSQL独自
- マニュアルの45章、特に45.1節と45.46節について簡単に確認
  - SELECT * FROM pg_tables;

ユーザ管理

OSS-DB

■一般ユーザと管理者ユーザ（スーパーユーザ）

- OSに一般ユーザと管理者ユーザがあるのと同じように、データベースにも一般ユーザと管理者ユーザがある。
- 一般ユーザには限られた権限しかないが、管理者ユーザにはすべての権限がある。
- OSの管理者ユーザと、データベースの管理者ユーザは異なる。例えば、root で pg_ctl コマンドを実行することはできない。

■権限とは？

- 多くの種類の権限があるが、例えば
  - 新規にテーブルを作成する権限、あるいは削除する権限
  - テーブルからデータを検索（SELECT）する権限
  - テーブルのデータを更新（UPDATE）する権限
- デフォルトでは、テーブルの所有者（作成者）だけが、そのテーブルに対するSELECT/UPDATEなどの権限を持つ（管理者ユーザは別）。つまり、権限を与えられなければ、他人のDBやテーブルを参照/更新できない。

ユーザ作成と削除

OSS-DB

■ユーザ作成

- postgres ユーザで createuser コマンドを使う。
  - $ createuser [option] [username]
- オプションで指定しなかった場合、以下を対話的に入力する。
  - 新規ユーザ名
  - 新規ユーザを管理者ユーザとするかどうか
  - 新規ユーザにデータベース作成の権限を与えるかどうか
  - 新規ユーザにユーザ作成の権限を与えるかどうか
- あるいは、CREATEROLE権限のあるユーザでpsqlを使って接続し、CREATE USER文を使う。
  - =# CREATE USER name [option];
- createuserコマンドよりも細かい設定がオプションで指定できるが、対話的な指定はできない。

■ユーザ削除

- dropuserコマンド、またはDROP USER文を使う

権限管理

OSS-DB

■データベース権限の管理
- CREATEDB, CREATEROLEなどデータベースシステムに関する権限は、ユーザ作成時に付与するか、あるいはALTER USER文で付与・剥奪する
  - =# ALTER USER username CREATEDB NOCREATEROLE;

■オブジェクト権限の管理
- テーブルなどのオブジェクトに対する権限の付与・剥奪には、GRANT文とREVOKE文を使う。
- 個々のユーザに対して、GRANT/REVOKEすることもできるが、ユーザ名としてpublicを指定すれば、全ユーザに対するGRANT/REVOKEも可能。
  - => GRANT SELECT ON table1 TO public;
  - => GRANT SELECT, UPDATE ON table2 TO user3;
  - => REVOKE DELETE ON table4 FROM public;

データベースの作成・削除

OSS-DB

■データベースクラスタ内に新規にデータベースを作成するには、createdbコマンドを使う、あるいはデータベースに接続して、CREATE DATABASE文を使う
- $ createdb [option...] dbname [comment]
- => CREATE DATABASE dbname [option];
- いずれの場合もCREATEDB権限が必要

■新規に作成されるデータベースは、(オプションで指定しなければ) テンプレートデータベースtemplate1のコピーとなる
- 複数のデータベースで共通に利用したいオブジェクトや関数定義などは、事前にtemplate1に作成しておく

■データベースを削除するには、dropdbコマンド、またはDROP DATABASE文を使う
- 元に戻せないので要注意
- データベースの所有者、または管理者ユーザだけが実行できる

データベースのバックアップ

OSS-DB

- 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重要。
- データベースのファイルは常に更新され続けている。メモリ上のデータ（キャッシュ）とディスク上のデータファイルの内容が一致するとは限らない、つまり、OSコマンドを使ってファイルをコピーしてもバックアップにはならない。
  - データベースのバックアップには特殊な方法が必要。
- データベースがクラッシュしたとき、一週間前のバックアップからデータベースが復元（リストア）できても、ありがたくないかもしれない。
  - クラッシュ直前の状態にデータを復旧（リカバリ）するためのバックアップ手段がある。
- バックアップの方法とリストア・リカバリの方法をセットで覚えること
  - バックアップを作っても、いざというときに使えなければ役に立たない

バックアップの手段

OSS-DB

- 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
- 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、データベー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）
  - 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
- 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

pg_dumpによるバックアップとリストア

OSS-DB

■データベースを停止せずに、データベース単位のバックアップを取得
- $ pg_dump [options] –f dumpfilename dbname　あるいは
- $ pg_dump [options] dbname > dumpfilename
- -F オプションで、出力形式を指定できる。p (plain) はテキスト形式 (デフォルト) 、c (custom) はカスタム (バイナリ) 形式、t (tar) はTAR形式
- データベースクラスタ内のすべてのデータベースのバックアップを取得するには、pg_dumpall コマンドを使う。(出力形式はテキストのみ)

■テキスト形式のバックアップは psql コマンドで、バイナリ形式のバックアップは pg_restore コマンドでリストアする。
- $ psql –f dumpfilename dbname　あるいは
- $ psql dbname < dumpfilename
- $ pg_restore –d dbname dumpfilename

■pg_dumpが作成するテキスト形式のバックアップはSQLのスクリプト (CREATE TABLE, INSERTなど) となっており、エディタで修正可能

コールドバックアップ

OSS-DB

■ディレクトリコピーによるバックアップ
- データベースを停止すれば、物理的なデータファイルをディレクトリごとコピーすることでバックアップを作成できる。(コールドバックアップ)
- コピーの方法は自由に選んで良い。(cp, tar, cpio, zip…)
  - $ cp –r data backupdir
  - $ tar czf backup.tgz data
- 簡単で確実な方法だが、頻繁には実行できない

■バックアップを、同じ構成の別のマシンにコピーして動かすこともできる
- バックアップ作成と逆のことをすればリストアできる
  - $ cp –r backupdir data
  - $ tar xzf backup.tgz

■コールドバックアップに対し、データベースの稼働中に取得するバックアップをホットバックアップと呼ぶ

ポイント・イン・タイム・リカバリ (PITR)

OSS-DB

■PITR (Point In Time Recovery)
- 障害の直前の状態までデータを復旧 (リカバリ) できる。
- 間違ってデータを削除した場合でも、任意の時点まで戻すことができる。

■PITRの仕組み
- WAL (Write Ahead Logging) により、データファイルへの書き込み前に、変更操作についてログ出力される。(トランザクションログ)
- 最後のバックアップ (ベースバックアップ) に対して、障害発生直前までのWALを適用することで、データを復旧できる。

■PITRによるベースバックアップの取得手順
- スーパーユーザで接続し、バックアップ開始をサーバに通知
  - =# SELECT pg_start_backup ('label') ;
- tar, cpioなどのOSコマンドでバックアップを取得 (サーバーは止めない)
- 再度、スーパーユーザで接続し、バックアップ終了をサーバに通知
  - =# SELECT pg_stop_backup ('label') ;

CSVファイルの入出力

OSS-DB

■psql の ¥copy メタコマンドを使うと、データベースのテーブルと、OSファイルシステム上のファイル (CSVなど) の間で入出力ができる。

■基本的な使い方
- => ¥copy table_name to file_name [options]
- => ¥copy table_name from file_name [options]
- デフォルトではタブ区切りのテキストファイルを入出力、オプションに"csv"と指定すれば、カンマ区切りのCSVファイルになる。

■SQLのCOPY文 (PostgreSQLの独自拡張機能) もあるが、¥copy との使い方の違いに注意。
- =# COPY table_name TO 'file_name' [options] ;
- =# COPY table_name FROM 'file_name' [options] ;
- ¥copyはクライアント上のファイル、COPYはサーバ上のファイルの入出力。
- COPYによるファイル入出力は、データベース管理者ユーザのみ実行できる。

VACUUM

OSS-DB

- PostgreSQLのデータファイルは追記型の構造。データが更新されると、旧データには削除マークが付けられ、新データはファイルの末尾に追加される。削除マークの付いた領域は、そのままでは再利用されない
- データの更新が繰り返されると、ファイルサイズが増大し、ディスク容量不足やパフォーマンス問題を引き起こす。
- VACUUMは削除マークがついたデータ領域を回収し、再利用可能にする
- コマンドラインから vacuumdb コマンド、あるいはデータベースに接続してVACUUM文を実行する。
- VACUUM, vacuumdbの主なオプション
  - ANALYZE, -z, --analyze：統計情報の取得も同時に実施
  - FULL, -f, --full：データを移動し、ファイルサイズを小さくする
  - VERBOSE, -v, --verbose：処理内容の詳細を画面に出力する
  - -a, --all：クラスタ内の全データベースに対してVACUUMを実施

自動バキューム (autovacuum)

OSS-DB

- VACUUMを自動的に実行する機能
- デフォルトの設定では、自動的に実行されるようになっており、これが推奨の設定でもある
- VACUUMとANALYZEが自動的に実行される
- データの変更量が設定値を超えると実行される

- PostgreSQLの古いバージョンでは、手動で、あるいはcronで定期的にVACUUM を実行する必要があった
- autovacuumにより、管理者がVACUUMを意識する必要性が低くなっているが、機能については理解しておくこと

ポイント解説：SQL

他のRDBMSとの違い

- SQLはANSIで標準化されており、RDBMSの種類による違いは小さい
- SQL文（DML/DDL/DCL）については差分が小さいが、関数（特に文字列関数や時間関数）はRDBMSの種類による違いが大きい
- 標準準拠の程度はRDBMSの種類によるが、PostgreSQLは準拠度が比較的高い
- PostgreSQLのマニュアルでは、各所にその機能がANSI標準なのか、PostgreSQLの独自拡張なのかの別が記述されている
- OracleなどANSI標準の策定前から存在していたRDBMSには、標準にない仕様が数多く残っているが、現在のバージョンでは標準の仕様の多くが取り入れられている

SELECT文

OSS-DB

■複雑な使い方があるが、RDBMS依存（方言）の部分は少ないので、基本をしっかり理解する
- SELECT cols FROM tables WHERE cond;
- ORDER BY, DISTINCT, GROUP BY, HAVING
- 副問い合わせ、EXISTS, IN

■準備1：（実行例は次のページ）
- CREATE TABLE sales（id INTEGER, person VARCHAR(10), amount INTEGER);
- INSERT INTO sales VALUES（1, 'aaa', 5000),（2, 'bbb', 3000)…;

id	person	amount
1	aaa	5000
2	bbb	3000
3	ccc	12000
4	ddd	4000
5	aaa	6000
6	bbb	5000

SELECT文（GROUP BYとHAVING）

OSS-DB

■WHEREとHAVINGの使い方の違い、GROUP BYとの関係について、正しく理解する
- WHEREの条件に合致した行の抽出→GROUP BYに従って集約→HAVINGの条件に合致した集約行の抽出、の順で処理される
- HAVING句には集約後でなければ判定できない条件を記述、集約前に判定できる記述はWHERE句に記述する
- 誤った記述例
  - SELECT person, sum(amount) FROM sales WHERE sum(amount) > 10000 GROUP BY person;
- 正しい使い方の例
  - SELECT person, sum(amount) FROM sales GROUP BY person HAVING sum(amount) > 10000;
- 動作はするが、正しくない使い方の例
  - SELECT person, sum(amount) FROM sales GROUP BY person HAVING person = 'aaa' OR person = 'bbb';
- 正しく書き直すと…
  - SELECT person, sum(amount) FROM sales WHERE person = 'aaa' OR person = 'bbb' GROUP BY person;

SELECT文（行数の制限）

OSS-DB

■SELECTする行数を制限するため、LIMIT/OFFSETが利用できる
（PostgreSQL/MySQLなど一部のRDBMSでのみ利用可能）
- SELECT * FROM table1 LIMIT 10 OFFSET 20;
  - 20行をスキップして、21行目から10行を表示
- LIMIT/OFFSETはORDER BYと組み合わせて利用可能
  - OracleのROWNUMは擬似列なので、ORDER BYと組み合わせられない

■準備2：（実行例は次のページ）
- CREATE TABLE t1 (id INTEGER, val VARCHAR(10));
- CREATE TABLE t2 (id INTEGER, val VARCHAR(10));
- INSERT INTO t1 VALUES (1, 'aaa'), (2, 'bbb');
- INSERT INTO t2 VALUES (1, 'xxx'), (3, 'yyy');

id	val
1	aaa
2	bbb

id	val
1	xxx
3	yyy

SELECT文（表の結合）

OSS-DB

■外部結合
- (+)や * を使うのは、OracleとSQL Serverの独自の外部結合
  - SELECT col... FROM table1 t1, table2 t2 WHERE t1.id = t2.id(+); (Oracle)
  - SELECT col... FROM table1 t1, table2 t2 WHERE t1.id* = t2.id; (MS-SQL)
- ANSI標準ではJOIN句を使って表を結合する
  - SELECT col... FROM table1 t1
    LEFT JOIN table2 t2 ON t2.id2 = t1.id1...
- JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, CROSS JOINの違いを理解する

■実験
- SELECT * FROM t1
  (INNER/LEFT/RIGHT/FULL) JOIN t2 ON t2.id = t1.id;
- SELECT * FROM t1 CROSS JOIN t2;

INSERT文

OSS-DB

- INSERT … SELECT によるものと、INSERT … VALUES によるものについて、理解する
  - INSERT INTO table1 (col…)
    SELECT col… FROM table2…;
  - INSERT INTO table1 (col…) VALUES (data…);
- VALUES句を使うときでも、複数行をまとめてINSERTできる
  - INSERT INTO table1 (col1, col2) VALUES
    (1, 'aaa'), (2, 'bbb'), (3, 'ccc');
  - 例えば、MySQLでは同じことができるが、Oracleではできないので注意

UPDATE文

OSS-DB

- 標準的な使い方を理解する
  - UPDATE table1 SET col1 = 'xxx', col2 = 'yyy' WHERE…;
- PostgreSQLは追記型構造なので、更新された行は、次のSELECTでは最終行に表示されることが多い
- 他のテーブルを参照したUPDATEについては、RDBMSの種類によって制限や独自の仕様があるので注意
  - UPDATE table1 t1 SET a =
    (SELECT b FROM table2 t2 WHERE t2.id = t1.id); (Oracle)
  - UPDATE table1 t1 SET a = t2.b
    FROM table2 t2 WHERE t1.id = t2.id; (PostgreSQL)
  - UPDATE table1 t2, table2 t2
    SET t1.a = t2.b
    WHERE t1.id = t2.id; (MySQL)

DELETE文

OSS-DB

- 標準的な使い方を理解する
  - DELETE FROM table1 WHERE...;
- 他のテーブルを参照する場合、USING句を使うと簡単に記述できる上、パフォーマンス上も有利なことが多い（ただしPostgreSQL独自の拡張）
  - DELETE FROM table1 t1 USING table2 t2
    WHERE t1.id = t2.id
- 同じことをANSI標準のSQLで記述すると
  - DELETE FROM table1
    WHERE id IN (SELECT id FROM table2);
  - DELETE FROM table1 t1
    WHERE EXISTS
    (SELECT * FROM table2 t2 WHERE t2.id = t1.id);

トランザクション

OSS-DB

- PostgreSQLでは、BEGINまたはSTART TRANSACTION文でトランザクションが開始され、COMMITまたはROLLBACK文で終了する
- SAVEPOINT, ROLLBACK TO savepointなどの基本を理解する
- トランザクションの外部で実行されるSQL文（INSERT/UPDATE/DELETE）は自動的にCOMMITされる（Oracleに慣れた人は要注意）
- PostgreSQLではCREATE TABLE, DROP TABLEなどのDDLもトランザクションの一部になるので、DDLによる自動COMMITは発生せず、ROLLBACKすればDROP TABLEされたテーブルも元に戻る
  - Oracleなどでは、DDLを実行すると、トランザクションが自動的にCOMMITされる

トランザクション

OSS-DB

■PostgreSQLでは、トランザクションの途中でエラーが発生すると、以後のSQLはすべてエラーとなり、ROLLBACKするしかなくなるので注意が必要

- SQLの文法エラー、DBの制約違反(一意性、外部参照など)によるエラー、いずれの場合もROLLBACKが必要
- この状態でCOMMITを発行すると、ROLLBACKが実行される
- 回避策は、エラーになる可能性のあるSQLを実行する前にSAVEPOINTを実行し、エラーが発生したらそのSAVEPOINTまでROLLBACKすること
- Oracleなどでは、エラーが発生しても、処理の継続が可能

■例

```
CREATE TABLE table1 (id INTEGER UNIQUE, val VARCHAR(10));
BEGIN;
INSERT INTO table1 VALUES (1, 'aaa'), (2, 'bbb');
SAVEPOINT sp1;
INSERT INTO table1 VALUES (2, 'ccc'); ←エラー!!
ROLLBACK TO sp1;
COMMIT;
```


主なデータ型

OSS-DB

■数値型

- SMALLINT(2バイト)、INTEGER(4バイト)、BIGINT(8バイト)
- NUMERIC(最大1000桁)、DECIMAL(NUMERICと同じ)
- REAL(4バイト)、DOUBLE PRECISION(8バイト)
- SERIAL(自動増分4バイト)、BIGSERIAL(自動増分8バイト)

■文字列型

- CHARACTER VARYING(可変長、最大4096文字)、VARCHAR(CHARACTER VARYINGと同じ)
- CHARACTER(固定長)、CHAR(CHARACTERと同じ)
- TEXT(可変長、無制限)

■日付型

- DATE(日付のみ)
- TIME(時刻のみ)
- TIMESTAMP(日付+時刻)

データ型（他のRDBMSとの比較）

OSS-DB

- 共通のものが多いが、微妙に仕様が異なることがある
- 多くのRDBMSでほぼ同じように使えるもの
  - INTEGER, NUMERIC
  - CHAR, VARCHAR
  - DATE, TIMESTAMP
- PostgreSQL独自のデータ型
  - SERIAL：自動的にシーケンスが作成され、列を連番にできる
  - BOOLEAN：論理値型
    - TRUE/'t'/'true'/'y'/'yes'/'on'/'1'
    - FALSE/'f'/'false'/'n'/'no'/'off'/'0'
    - 大文字・小文字は区別しない、TRUE/FALSEはキーワード、他は文字列
- Oracleのデータ型との比較
  - NUMBER, BINARY_FLOAT, BINARY_DOUBLE
  - VARCHAR2, NCHAR, NVARCHAR2, CLOB
  - DATE

文字列リテラル

OSS-DB

- 文字列リテラル
  - SQLの文字列リテラルはシングルクォートで囲まれ、大文字と小文字は区別される
    - 'STRINGstring'
    - 文字列の外側のSQL文では大文字と小文字は区別されない
    - MySQLのように、文字列の大文字と小文字を区別しないRDBMSもある
    - ダブルクォートで囲った文字列をリテラルとして使えるRDBMSもあるが、一般には列別名などシングルクォートとは異なる特定の用途でしか使えない
      - SELECT col1 "col #1" FROM table1 WHERE…;
  - 文字列中にシングルクォートを入れるにはシングルクォートを2つ並べる
    - 'I can''t do it.'
  - tag を使って文字列リテラルを記述することも可能（PostgreSQL独自）
    - xyzI can't do it.xyz : 'I can''t do it.'と同じ
    - tagはなくても良く、$$I can't do it.$$ という記述でもOK
    - Oracleでは、Q'XstringX'（Xは任意の文字、Qは小文字でも可）という記述がある 例えば、q'xI can't do it.x'

シーケンス

OSS-DB

- CREATE SEQUENCE文で明示的に作成することができる他、SERIAL型（4バイト）またはBIGSERIAL型（8バイト）の列を作ることで自動的に作成される
  - CREATE SEQUENCE seq_name [options];
  - デフォルトでは8バイト
- シーケンス名と同じ名前のテーブルが自動的に作成される
  - SELECT * FROM seq_name;
- シーケンスの現在値はcurrval()，次の値はnextval()関数で取得。
  - SELECT currval('seq_name');
  - SELECT nextval('seq_name');
- 現在値の変更はsetval()関数を使う
  - SELECT setval('seq_name', 100);
- SERIAL/BIGSERIAL型の列については、INSERT時に列を指定しない、あるいは列の値としてDEFAULTを指定すると、シーケンスの次の値が使われる

集約関数、算術演算子、算術関数

OSS-DB

- 集約関数
  - count, sum, avg, max, min
  - NULL値の扱いに注意
    - count(*)はすべての列がNULLであっても1件のデータとしてカウントする
    - count(col)は、colの値がNULLのものを除いたデータ数を返す
    - avg(col)はNULLを除いたデータの平均値を返す
- 算術演算子、算術関数
  - ＋、－、＊、／ の算術演算子は標準通り
  - 剰余計算にMOD関数の他、％ 演算子が使える(Oracle, DB2などはMODのみ)
  - 乱数発生にRANDOM関数が用意されており、0と1の間の小数値を返す(PostgreSQL独自)

文字列演算子

OSS-DB

■文字列演算子
- LIKEで、_% を使ったマッチングは非常に重要
  - SELECT * FROM table1 WHERE col1 LIKE 'a_c%';
- 文字列結合で 'aaa'||NULL はNULLになる
  - Oracleでは'aaa'になるので注意
  - || はANSI標準の文字列結合演算子だが、利用できないRDBMSや + を文字列結合に使うRDBMSもあるので注意
  - concat関数で文字列結合するRDBMSもあるが、PostgreSQLにはconcat関数はない

■正規表現
- ～ 演算子で、指定の正規表現を含む文字列とマッチさせられる
  - SELECT * FROM table1 WHERE col1 ~ '^[a-c]';
- SIMILAR TOはLIKEとほぼ同じ使い方だが、正規表現の一部をサポートする
  - SELECT * FROM table1 WHERE col1 SIMILAR TO '[a-c]%';
- 正規表現は多くのRDBMSが何らかの方法でサポートしているが、実装方法はRDBMSの種類によって大きく異なる

文字列関数

OSS-DB

■文字列関数
- RDBMSの種類によって実装されている関数に違いがある
- 文字列の変換：UPPER, LOWER
- 文字列の置換：REPLACE, TRANSLATE
- 文字の削除：TRIM, RTRIM, LTRIM
- 文字列の長さ：LENGTH, CHAR(ACTER)_LENGTH, OCTET_LENGTH
- 部分文字列：SUBSTRING, POSITION
- ASCII変換：ASCII, CHR

■現在では、どのRDBMSでもマルチバイト文字は当然のようにサポートされており、(CHARACTER_)LENGTH関数はバイト数ではなく文字数を返す。バイト数を調べたいときはOCTET_LENGTH関数を使う(OracleではLENGTHB)。

変換関数

OSS-DB

■変換関数
- TO_CHAR, TO_NUMER, TO_DATEなどは、OracleでもPostgreSQLでも使えるが、他のRDBMSには使えないものが多い
- DECODE, NVLはOracle独自 (PostgreSQL／MySQLではDECODEは復号化)
- TO_xxx → CAST (ANSI標準)
  - SELECT cast ('2011-10-01' AS DATE) + 10;
- PostgreSQL独自の型変換方式として :: 演算子を使う方法がある
  - SELECT '2011-10-01'::DATE + 10;
- DECODE → CASE/WHEN/THEN/ELSE/END
  - SELECT CASE col1 WHEN val1 THEN 'xxx' WHEN val2 THEN 'yyy' ELSE 'zzz' END FROM table1;
- NVL → COALESCE
  - SELECT coalesce (val1, val2...)
    → val1, val2...のうち、最初のNULLでないものが返る

時間関数、期間リテラル

OSS-DB

■時間関数
- RDBMSの種類によって実装されている関数に大きな違いがある
- 現在日時の取得:CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP
  - これらは関数名の後に括弧を付けずに使うことに注意
- 日時から要素の取得:EXTRACT, TO_CHAR

■期間リテラル
- 記述方法はRDBMSの種類によって大きく異なる
- INTERVAL '10' YEAR (Oracle)
- 10 YEARS (DB2)
- INTERVAL '10 YEAR' (PostgreSQL)
- INTERVAL 10 YEAR (MySQL)

- 例えば、1ヶ月後の日付をPostgreSQLで表示するには
  - SELECT current_date + INTERVAL '1 MONTH'; あるいは
  - SELECT current_date + '1 MONTH'::INTERVAL

関数定義

OSS-DB

- PL/pgSQLという、OracleのPL/SQLに似た言語でストアドプログラムを作成できる
  - CREATE FUNCTION test(INTEGER)
    RETURNS INTEGER AS $$
    DECLARE
      di ALIAS FOR $1;
      d INTEGER;
    BEGIN
      d := di * 2;
      RETURN d;
    END;
    $$ LANGUAGE 'plpgsql'
- 事前に、createlang plpgsql を実行して、手続き言語の使用についてDBに登録しておく必要があるが、通常は登録済み(createlang -l で確認)
- FUNCTIONはあるがPROCEDUREはない。ただし、値を返さないVOID型のFUNCTIONを作ることはできる
- PL/pgSQLでなく、SQLで関数を定義することもできる

トリガーとルール

OSS-DB

- トリガー
  - テーブルの更新(INSERT/UPDATE/DELETE)が実行される直前、あるいは直後に呼び出される手続き
  - SQL文の実行前あるいは後に1度だけ呼び出す、あるいは更新される各行について、更新の前あるいは後に呼び出す、いずれも設定可能
  - PostgreSQLでは、PL/pgSQLなどによるFUNCTIONを事前に作成しておき、CREATE TRIGGER文でそれを割り当てる
- ルール
  - ビュー(VIEW)の更新を実現するためのPostgreSQL独自の方式
  - ビューに対するINSERT/UPDATE/DELETEが可能かどうかは、RDBMSの種類およびビューの定義の両方に依存
  - PostgreSQLでは、CREATE RULE文でルールを定義すれば、ビューの更新ができる(ルールが定義されていないビューは更新できない)
    - CREATE RULE view1_ins AS ON INSERT view1
      DO INSTEAD INSERT INTO table1 VALUES ...;

スキーマ

OSS-DB

- スキーマの実装はRDBMSの種類によって異なる
  - Oracleでは、ほぼユーザと同義
  - PostgreSQLでは単なる名前空間（ユーザ名との関連はある）
  - MySQLにはスキーマがない
- PostgreSQLでのスキーマの利用
  - CREATE SCHEMAで作成、DROP SCHEMAで削除
  - ALTER SCHEMAで名前や所有者を変更できる
  - public というスキーマがある
- スキーマ検索パス
  - => SHOW search_path; で確認できる
  - デフォルトでは、"$user", public となっている
  - ユーザ foo が SELECT * FROM table1; を実行
    - foo.table1 → public.table1 の順で検索して SELECT、どちらもなければエラー
  - ユーザ foo が CREATE TABLE table2 ...; を実行
    - スキーマ foo が存在すれば foo.table2 を作成、なければ public.table2 を作成

OSS-DB

例題解説

例題解説1

OSS-DB

■一般知識 – ライセンス

PostgreSQLの利用条件、ライセンスについて、
正しいものを2つ選びなさい。

A. 研究目的、商用を問わず、無料で利用できる。
B. ソースコードを改変したものを配布する場合には、
変更部分についてソースコードを公開する必要がある。
C. ソースコードを改変したものを配布する場合には、
無保証であることをドキュメントなどに明記する必要がある。
D. 致命的な障害については、開発者は修正の義務を負う。
E. 日本では、日本PostgreSQLユーザ会がサポートの義務を負う。

例題解説2

OSS-DB

■運用管理 – 標準付属ツールの使い方

以下の記述から、誤っているものを2つ選びなさい。

A. createdbコマンドでデータベースを作成するにはCREATEDB権限が
必要である
B. dropdbコマンドでデータベースを削除するにはCREATEDB権限が
必要である
C. dropdbコマンドでデータベースを削除する前に、そのデータベース内の
テーブルなどすべてのオブジェクトを削除しておく必要がある
D. dropuserコマンドでユーザを削除するには、CREATEROLE権限が必要である
E. dropuserコマンドでユーザを削除する前に、そのユーザが所有するすべての
テーブルを削除しておく必要がある

例題解説3

OSS-DB

■運用管理 – 基本的な運用管理作業

VACUUM は PostgreSQL の運用管理でどのような役割を持っているか。
誤っているものを2つ選びなさい。

A. 不正なIPアドレスからのデータベースアクセスがないか監視する
B. データベースファイルの巨大化を防ぐ
C. データベースのパフォーマンスの悪化を防ぐ
D. 最適な検索を実施するための統計情報を取得する
E. 長期間、利用されていないデータをアーカイブする

例題解説4

OSS-DB

■開発 – トランザクション

以下SQL文を順次実行した。実行後のテーブル t1 の行数は何行か。

```
CREATE TABLE t1 (id INTEGER, val VARCHAR(10));
BEGIN;
INSERT INTO t1 VALUES (1, 'aaa'), (2, 'aaa');
SAVEPOINT sp1;
DELETE FROM t1 WHERE id = 1;
SAVEPOINT sp2;
INSERT INTO t1 VALUES (3, 'ccc');
ROLLBACK to sp1;
INSERT INTO t1 VALUES (4, 'ddd'), (5, 'eee');
COMMIT;
```

参考資料

- OSS教科書OSS-DB Silver
  - 認定教材
- オープンソースデータベース標準教科書
  - 初心者向けにSQLの初歩からWebアプリケーション開発まで
- PostgreSQL徹底入門
  - PostgreSQL 9.0対応
  - 9.0.1のインストーラ、ソースコード
- SQLポケットリファレンス
  - 他のDBやANSI標準との比較
- 日本PostgreSQLユーザ会
  http://www.postgresql.jp/
- Let's Postgres
  http://lets.postgresql.jp/
- オンラインマニュアル
  http://www.postgresql.jp/document/9.0/html/

OSS-DB Silver/Gold

ご清聴ありがとうございました。

■お問い合わせ■
LPI-Japan
テクノロジー・マネージャー
松田　神一
matsuda@lpi.or.jp