OSS-DB

OSS-DB Silver
技術解説無料セミナー
2021/04/10 開催

主題	運用管理（出題範囲 52 %）
副題	標準付属ツールの使い方【重要度：5】
主題	開発/SQL（出題範囲 32 %）
副題	SQLコマンド【重要度：13】

本日の講師

SRA OSS, Inc. 日本支社
正野　裕大

LPI-JAPAN

講師紹介

#OSS-DB

- **正野　裕大**
- SRA OSS, Inc. 日本支社所属
- PostgreSQL の技術サポート・コンサルティング
- PostgreSQL トレーニング講師

- **SRA OSS, Inc. 日本支社 (https://www.sraoss.co.jp/)**
- PostgreSQL などの OSS プロダクトの技術サポート・コンサルティング
- PowerGres 製品の製造、販売
- PostgreSQL トレーニング

#OSS-DB

OSS-DBとは

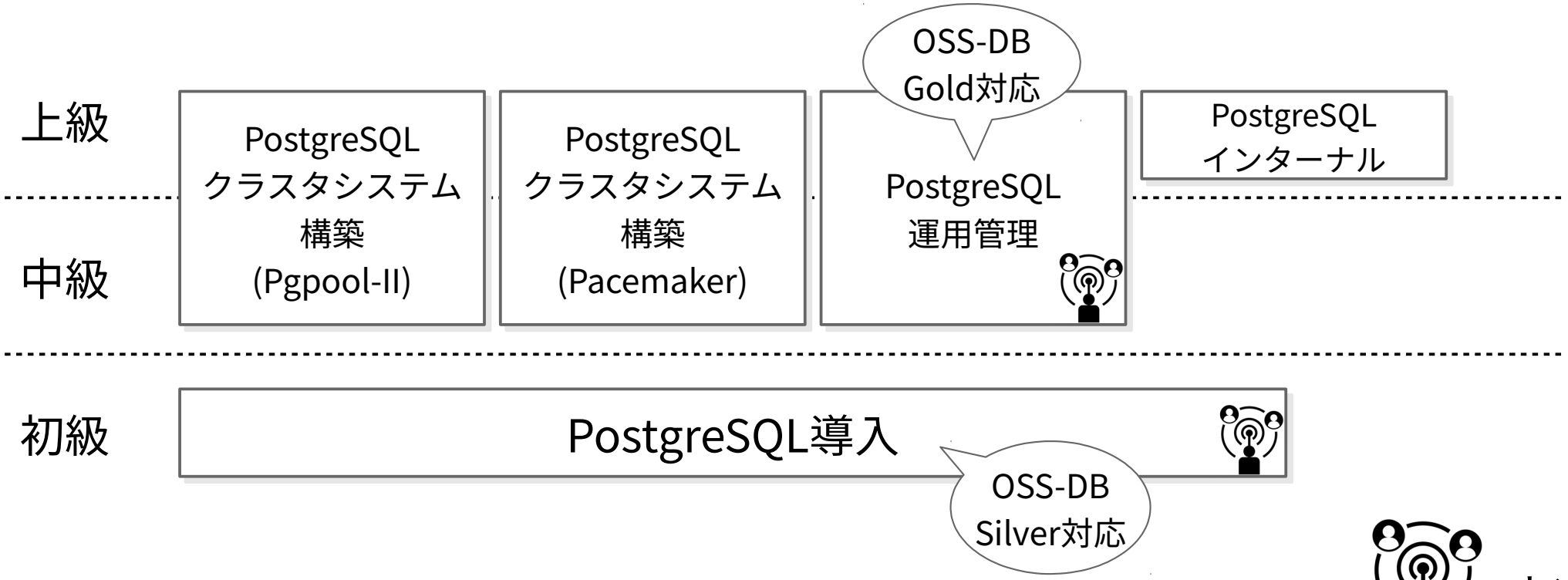
オープンソースのデータベースソフトウェア「PostgreSQL」を扱うことができる技術力の認定です。様々な分野でPostgreSQLの利用拡大が進む中でOSS-DBの認定を持つことは、自分のキャリアのアピールにもつながります。

- ✓OSS-DB Goldは設計やコンサルティングができる技術力の証明
  PostgreSQLについての深い知識を持ち、データベースの設計や開発のほか、パフォーマンスチューニングやトラブルシューティングまで行えることが証明できます
- ✓OSS-DB Silverは導入や運用ができる技術力の証明
  PostgreSQLについての基本的な知識を持ち、データベースの運用管理が行えるエンジニアとしての証明ができます
- ✓対象のバージョンはPostgreSQL 11

プロモーション 1/2

#OSS-DB

■**SRA OSS, Inc. 日本支社は**
PostgreSQLトレーニングコースを各種用意しています



オンライン対応

■OSS-DB Exam認定校第1号ならではの充実した内容
■受講者のレベルに合わせたコースでOSS-DB試験にも対応した内容

プロモーション 2/2

#OSS-DB

直近のトレーニング開催スケジュール

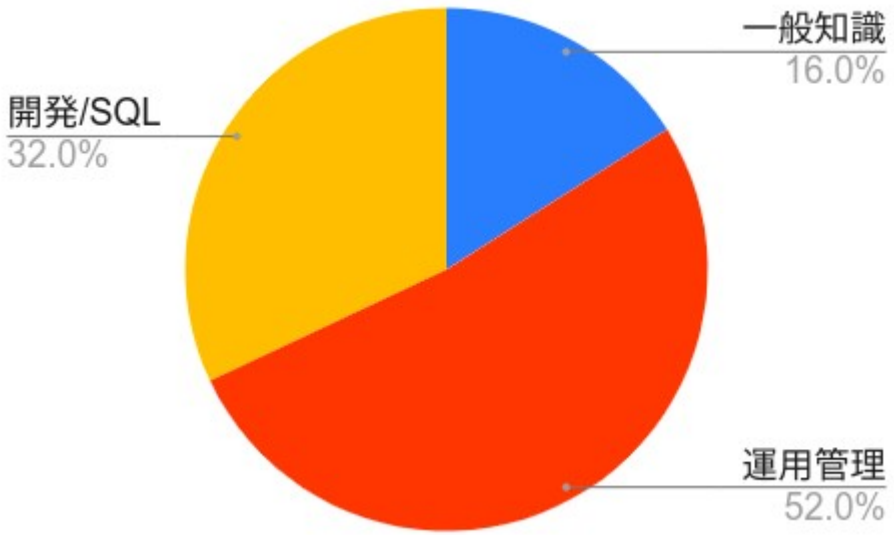
2021 年 4 月 21 日 ～ 22 日 締切：2021 年 4 月 5 日	PostgreSQL運用管理トレーニング	オンライン
2021 年 5 月 10 日 ～ 11 日 締切：2021 年 4 月 16 日	PostgreSQL導入トレーニング	東京
2021 年 5 月 12 日 ～ 13 日 締切：2021 年 4 月 20 日	PostgreSQL運用管理トレーニング	東京
2021 年 5 月 14 日 締切：2021 年 4 月 22 日	Pacemaker/DRBDによるPostgreSQLクラスタ構築トレーニング	東京
2021 年 5 月 17 日 ～ 18 日 締切：2021 年 4 月 23 日	PostgreSQL導入トレーニング	オンライン
2021 年 5 月 19 日 ～ 20 日 締切：2021 年 4 月 27 日	PostgreSQL運用管理トレーニング	オンライン
2021 年 5 月 26 日 締切：2021 年 5 月 10 日	PostgreSQLインターナル講座	オンライン

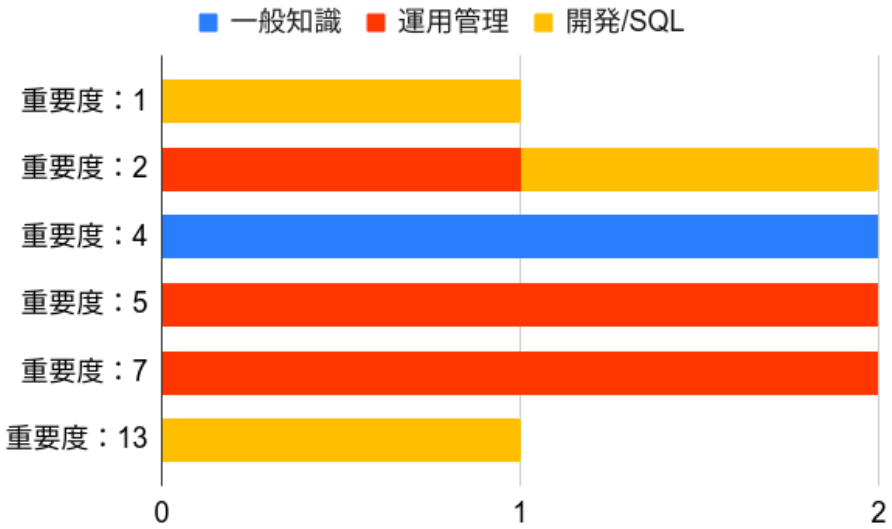
詳しくは https://www.sraoss.co.jp/prod_serv/training/pgsql/

今回のテーマとゴール

#OSS-DB

- 運用管理（出題範囲52%）の「標準付属ツールの使い方【重要度: 5】」
- 開発/SQL（出題範囲32%）の「SQL コマンド【重要度: 13】」





- 初めてデータベースを学ぶ人がPostgreSQLやSQLの学習の最初の一歩を踏み出せる

環境情報

仮想マシンossdbを用意

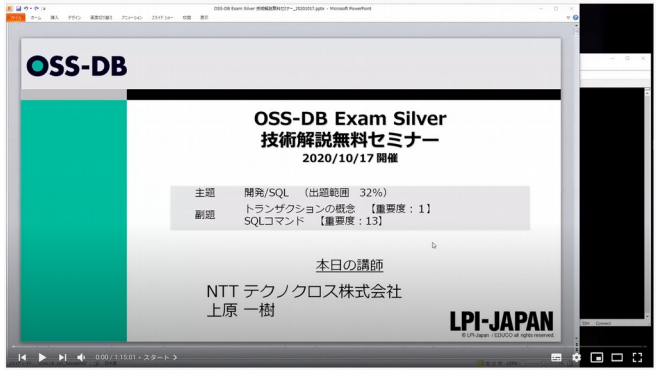
#OSS-DB

- CentOS 8.3

```
[postgres@ossdb ~]$ cat /etc/redhat-release
CentOS Linux release 8.3.2011
```

PostgreSQL

- 両マシンとも開発コミュニティ公式のパッケージからPostgreSQL 11をインストールして環境変数(*)を調整済み
- 具体的なインストール方法は LPI-Japan の Youtube チャンネルを参照



- https://www.youtube.com/watch?v=tnXTRG__lXg&t=701s
- [2020/10/17]OSS-DB Silver技術解説セミナー「運用管理、開発／SQL」11:41 ～ 環境の構築（PostgreSQL のインストールなど事前準備）

(*) 環境変数: コマンドやアプリケーションの実行時に参照される変数のこと。

標準付属ツール > initdbでデータベースクラスタの初期化

initdbの実行

#OSS-DB

- データベースクラスタ(*)を初期化するPostgreSQLコマンド

```
[postgres@ossdb ~]$ initdb --encoding=UTF8 --no-locale
```

- --encoding=ENCODING
  データベースのデフォルト文字エンコーディングを指定

- --no-locale
  ロケールを使用しない（慣習的にローケールを使用しないことが推奨されている）

- データベースクラスタパスの指定方法
  $PGDATAで指定するか--pgdata=《データベースクラスタパス》で指定

(*) データベースクラスタ: ストレージに記録されるPostgreSQLのデーター式を格納するディレクトリ。

標準付属ツール > pg_ctlでデータベースクラスタの起動

#OSS-DB

起動 (start)

```
[postgres@ossdb ~]$ pg_ctl start
```

- 操作対象となるデータベースクラスタをセットで指定する
- 指定方法は$PGDATAか--pgdata=DATADIR

停止 (stop)

```
[postgres@ossdb ~]$ pg_ctl stop
```

- --modeで停止モードを指定

fast（デフォルト）	既存の接続を切断してから停止
smart	新規の接続を受け付けず、既存の接続の終了を待ってから停止
immediate	クラッシュさせて強制終了

その他

- restart: 再起動
- reload: 設定ファイルの読み込み
- staus: 起動状態の確認

標準付属ツール > 接続パラメータについて

#OSS-DB

- **PostgreSQLコマンドの分類**
  - サーバアプリケーション（initdb, pg_ctl など）
    同じマシン内に配置されているデータベースクラスタを対象にコマンド実行
  - クライアントアプリケーション（createuser, dropuser, createdb, dropdb, psql など）
    起動しているPostgreSQLにアクセスしてコマンド実行
- **クライアントアプリケーションが起動しているPostgreSQLにアクセスするために必要な情報 = 接続パラメータ**
  - どこのマシンの
  - どのポートで起動しているPostgreSQLの
  - どのデータベースユーザを使って
  - どのデータベースに
    アクセスするのか

標準付属ツール > 接続パラメータについて

接続パラメータの指定方法

#OSS-DB

- 環境変数
- PostgreSQLコマンドのオプション

用途	環境変数	オプション	省略時の振る舞い
接続ホスト （どのマシンに）	$PGHOST	--host	Unix ドメインソケットが使われる
接続ポート （どのポート番号で）	$PGPORT	--port	暗黙的に5432が使われる
接続データベースユーザ （どのデータベースユーザで）	$PGUSER	--username	暗黙的にPostgreSQLコマンド実行時のLinuxのOSユーザ名が使われる
接続データベース （どのデータベースで）	$PGDATABASE	-dbname	

標準付属ツール > createuserでデータベースユーザの作成

#OSS-DB

データベースユーザ

- initdb直後はpostgresユーザしか存在しない
- initdbを実行したLinux OS ユーザ名が暗黙的に管理者権限を持ったデータベースユーザになる

データベースユーザ「joe」の作成

- --interactiveで対話的に作成可能

```
[postgres@ossdb ~]$ createuser --interactive joe
Shall the new role be a superuser? (y/n) n                    # 管理者権限を付与？
Shall the new role be allowed to create databases? (y/n) y    # データベース作成権限を付与？
Shall the new role be allowed to create more new roles? (y/n) n # データベースユーザ作成権限を付与？
```

データベースユーザの削除

```
[postgres@ossdb ~]$ dropuser 削除したいデータベースユーザ名
```

標準付属ツール > creatdbでデータベースの作成

#OSS-DB

データベース

- PostgreSQLのデータを格納する領域
- initdbを実行直後はメンテナンス用データベースしか存在しない
  template0, template1, postgres
- アプリのデータを格納するためのデータベースは別途作成すること

データベース「demo」の作成

```
[postgres@ossdb ~]$ createdb --username joe demo
```

- データベース作成時に使用したデータベースユーザがそのデータベースの所有権を持つ

データベースの削除

```
[postgres@ossdb ~]$ dropdb 削除したいデータベース名
```

- 削除できるのは管理者権限か削除対象のデータベースの所有者権限を持ったデータベースユーザだけ

標準付属ツール > psqlでPostgreSQLにアクセスしてSQL実行

#OSS-DB

psqlコマンド

- PostgreSQLにアクセスしてSQLを実行できる PostgreSQL コマンド

```
[postgres@ossdb ~]$ psql --username joe --dbname demo
demo=> SELECT * FROM -- 以降、行末までコメント扱い
demo-> pg_user;      -- SQL 文の終わりは改行ではないので複数行に渡って入力可能
```

プロンプト

demo=>	接続データベース名
demo=> demo->	SQL文の入力状態（通常は = で、複数行入力中は -）
demo=# demo=>	管理者ユーザか一般ユーザ（# が管理者で、> が一般ユーザ）

SQLの実行方法

- ; が文の終わり
- -- でコメント
- SQL構文中の空白、タブ、改行は1つの空白として扱う
- SQL構文は大文字、小文字は区別されない
- Ctrl + CでSQL入力をキャンセル
- \qかCtrl + dでpsql接続を終了

SQLコマンド > SQLとは

#OSS-DB

SQL

- リレーショナルデータベースを操作するプログラミング言語
- ISO標準規格
  SQL92、SQL:1999、SQL:2003、SQL:2008、SQL:2011、SQL:2016
- 多くのデータベースソフトウェアで共通規格として取り入れられている

SQLの分類

- DDL（Data Definition Language; データ定義言語）
  データを入れるテーブル作成に関する SQL 文
- DML（Data Manipulateion Language; データ操作言語）
  テーブルの中に入ったデータを操作する SQL 文
- DCL（Data Control Language; データ制御言語
  上記以外の SQL 文）

基本的なSQL

目的	SQL	例
データの生成 (Create)	INSERT 文	INSERT INTO products VALUES (1, '帽子');
データの読み取り (Read)	SELECT 文	SELECT name FROM products;
データの更新 (Update)	UPDATE 文	UPDATE products SET name = '帽子';
データの削除 (Delete)	DELETE文	DELETE FROM products

- いわゆるCRUD操作

SQLコマンド > テーブル作成

CREATE TABLE文

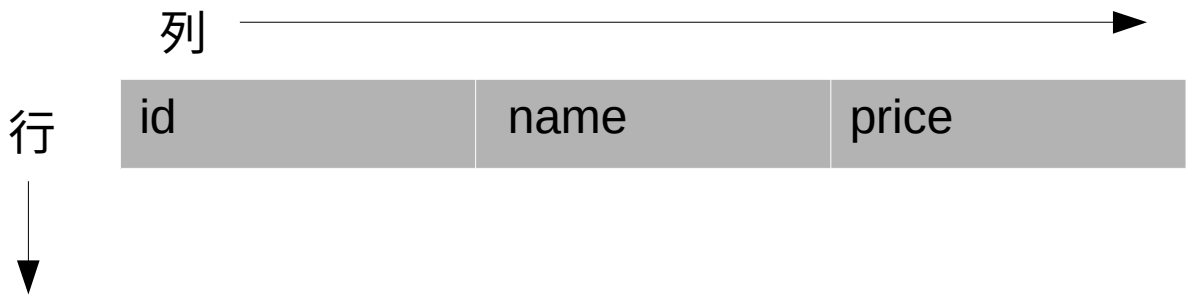
#OSS-DB

```
CREATE TABLE テーブル名（列名1 データ型1[, 列名2 データ型2, ...]);
```

- テーブル名、列名について
  - 大文字、小文字が区別されない（小文字推奨）
  - PostgreSQLの予約語は使えない
  - マルチバイト文字は使えるが非推奨

fruitsテーブルを作成する

```
demo=> CERATE TABLE fruits(id int, name text, price int);
```



- fruitsテーブル
- id列、name列、price列で構成
- まだ行データは存在しない

SQLコマンド > データ登録

INSERT文

#OSS-DB

```
INSERT INTO テーブル名（列名1[, 列名2, ...]）VAULES（データ1[, データ2, ...]);
```

- 指定した列名の順序に対応した順序でデータを指定する
- 文字データはシングルクォートでくくる
- 文字データはマルチバイト文字も格納可能

fruitsテーブルにデータを3行登録する

```
demo=> INSERT INTO fruits(id, name, price) VALUES (1, 'ringo', 100);
demo=> INSERT INTO fruits(name, id, price) VALUES ('banana', 2, 150); -- 列の指定順序は任意
demo=> INSERT INTO fruits VALUES (3, 'mikan', 200); -- テーブル定義と同じ順序でデータを登録するなら列名は省略可能
```

id	name	text
1	ringo	100
2	banana	150
3	mikan	200

SQLコマンド > データ読み取り

SELECT文

#OSS-DB

```
SELECT 列名1, 列名2, ... FROM テーブル名 [オプション];
```

- 列名に * を指定するとテーブルの全列を表示
- 「*列名* AS *別名*」で列名の表示を変更できる

fruitsテーブルの全データ行を読み取る

```
demo=> SELECT * FROM fruits; -- 全列取得
demo=> SELECT id, name FROM fruits; -- 任意の列だけ取得
demo=> SELECT id, name AS 品名, price AS 価格 FROM fruits; -- AS を使って列名を変更
```

SQLコマンド > データ読み取り > データの絞り込み

WHERE句で行データを絞り込める

#OSS-DB

```
SELECT 列名1[, 列名2, ...] FROM テーブル名 WHERE 検索条件;
```

- 「*検索条件*」にtrue / falseを返す条件式を書ける
- 条件式がtrueになる行だけが返る

fruitsテーブルのデータ行を絞り込む

```
demo=> SELECT * FROM fruits WHERE price > 120;
```

検索条件でよく使われる演算子 / キーワード	例
=	WHERE price = 150
!=, <>	WHERE price <> 150
<, >, <=, >=	WHERE price >= 100
AND, OR	WHERE price > 100 AND price < 200

SQLコマンド > データ読み取り > 演算子を使う

列指定箇所や検索条件には演算子が使える

#OSS-DB

+, -, *, /	加減乗算
%	剰余
^	べき上
&, |, #	ビットごとの AND、OR、XOR

<<, >>	算術左、右シフト
<, <=, =, >, >=, !=, <>	大小比較、等号、不等号
||	文字列結合
~	正規表現マッチ

fruitsテーブルの価格を1割増しにする

```
demo=> SELECT id, name, (price * 1.10) AS new_price FROM fruits;
```

fruitsテーブルの価格に単位をつける

```
demo=> SELECT id, name, price || '円' AS new_price FROM fruits;
```

SQLコマンド > データ更新

UPDATE文

#OSS-DB

```
UPDATE テーブル名 SET 列名1 = データ1[, 列名2 = データ2 , ...] WHERE 検索条件;
```

- WHERE句で更新対象行を絞り込む（そうしないと全行が更新される）
- UPDATEする検索条件を事前にSELECTして確認するのも手

fruitsテーブルのリンゴを10円引きにする

```
demo=> UPDATE fruits SET price = price -10 WHERE name = 'ringo';
demo=> SELECT * FROM fruits;
```

id	name	text
1	ringo	100
2	banana	150
3	mikan	200

UPDATE

id	name	text
2	banana	150
3	mikan	200
1	ringo	90

SQLコマンド > データ削除

DELETE文

#OSS-DB

```
DELETE FROM テーブル名 WHERE 検索条件;
```

- WHERE句で削除対象行を絞り込む（そうしないと全行が削除される）

fruitsテーブルのリンゴの情報を削除する

```
demo=> DELTE FROM fruits WHERE name = 'ringo';
demo=> SELECT * FROM fruits;
```

id	name	text
2	banana	150
3	mikan	200
1	ringo	90

DELETE

id	name	text
2	banana	150
3	mikan	200

高速なデータの全削除（WHERE句の無いDELETE文よりも高速）

```
TRUNCATE TABLE テーブル名;
```

SQLコマンド > データ型 > 数値データ型

#OSS-DB

データ型名	別名	バイト数	値の範囲
SMALLINT	INT2	2	-32768 ～ 32767
INTEGER	INT, INT4	4	-2147483648 ～ 2147483647
BIGINT	INT8	8	-9223372036854775808 ～ 9223372036854775807
REAL	FLOAT4	4	6桁少数精度
DOUBLE PRECISION	FLOAT8	8	15桁少数精度
NUMERIC(*精度*, *位取り*)		可変長	小数点より上は131072桁まで、小数点より下は16383桁まで
SERIAL	SERIAL4	4	1 ～ 2147483647 重複の無い連番を生成する
BIGSERIAL	SERIAL8	8	1 ～ 9223372036854775807 重複の無い連番を生成する

SQLコマンド > データ型 > 数値データ型 > 連番の割り当て

#OSS-DB

シリアル型

- 整数の連番を自動で割り振れる
- SERIAL　　INT型の範囲
- BIGSERIAL　　BIGINT型の範囲

テーブル作成時にシリアル型を設定

```
demo=> CREATE TABLE customers (id serial, name text); -- int 型の代わりに serial 型
demo=> INSERT INTO customers (name) VALUES ('user1'); -- id 列を省略してデータ登録
demo=> INSERT INTO customers (name) VALUES ('user2');
demo=> SELECT * FROM customers;
```

- テーブル定義時にシリアル型を指定すると内部にシーケンス（連番ジェネレータ）が生成される
- INSERT時に自動で連番が割り振られる
- 「重複しない整数値」がINSERTされることが保証される
  → 行の一意な識別子に使える

SQLコマンド > データ型 > 文字データ型

#OSS-DB

データ型名	別名	説明
CHARACTER(n)	CHAR(n)	固定長文字列 nを超えたらエラー、満たない場合はスペース詰め
CHARACTER VARYING(N)	VARCHAR(n)	制限付き可変長文字列 nを超えたらエラー
TEXT		可変長文字列 文字制限無し

- nには文字数を指定（バイト数ではない）
- PostgreSQLの場合は3つの型に性能差はほとんど無し

LIKE / NOT LIKE 述語

#OSS-DB

```
文字列 LIKE パターン  -- 大文字小文字を区別する
文字列 ILIKE パターン -- 大文字小文字を区別しない
```

パターン内で使える特殊文字

```
_  -- 任意の1文字
%  -- 0以上の任意の文字列
```

- 文字列としての「_」や「%」をマッチさせる場合は「\」でエスケープする

```
demo=> SELECT 'abc' LIKE 'abc'; -- true
demo=> SELECT 'abc' LIKE 'a%';  -- true（前方一致検索）
demo=> SELECT 'abc' LIKE '%c';  -- true（後方一致検索）
demo=> SELECT 'abc' LIKE '_b_'; -- true
demo=> SELECT 'abc' LIKE 'c';   -- false
```

上記のSQL実行例の通り、SELECT文はテーブルを参照するだけでなく、何らかの演算結果を返すことも可能です。

SQLコマンド > 日付・時間データ型

#OSS-DB

データ型名	説明	範囲
TIMESTAMP [(p)] [WITHOUT TIME ZONE]	年月日時分秒	BC4713 ~ AD5874897
TIMESTAMP [(p)] WITH TIME ZONE	タイムゾーン付き年月日時分秒	BC4713 ~ AD5874897
DATE	日付	BC4713 ~ AD5874897
TIME [(p)] [WITHOUT TIME ZONE]	時刻	00:00:00 ~ 24:00:00
TIME [(p)] WITH TIME ZONE	タイムゾーン付き時刻	00:00:00+1459 ~ 24:00:00-1459
INTERVAL [(p)]	日付時刻の差	-178000000年 ~ 178000000年

SQLコマンド > 日付・時間データ型

日付・時間データの入力方法

#OSS-DB

- 日付・時刻を表現した文字列をキャスト（型変換）する

もとのデータ型 :: 変換後のデータ型

- 様々な様式の文字列をPostgreSQLは日付・時刻として解釈してくれる

```
demo=> SELECT '2021/04/10'::date;           -- 「年/月/日」という文字列を date 型にキャスト
demo=> SELECT 'April 10, 2021'::timestamp; -- 「アメリカ式の日付」の文字列を timestamp 型にキャスト
demo=> SELECT '15:00'::time;                -- 「時:分」という文字列を time 型にキャスト
```

現在のトランザクション(*)開始時刻を返す関数

```
demo=> SELECT CURRENT_DATE;       -- 日付を返す
demo=> SELECT CURRENT_TIME;       -- 時刻を返す
demo=> SELECT CURRENT_TIMESTAMP; -- 日付と時刻を返す
```

(*) トランザクションの解説は省略します。詳細は LPI-Japan の Youtube チャンネルを参照してください。

https://www.youtube.com/watch?v=tnXTRG__lXg&t=2150s

SQLコマンド > 日付・時間データ型

INTERVAL型

- 「時間間隔」を扱うデータ型

日付・時間データ型同士の引き算

```
demo=> SELECT '2021-04-10'::date - '1999-12-31'::date;
 ?column?
-----------
 7771 days
(1 row)
```

時間間隔を足す

- 「時間間隔」表す単位を含んだ文字列はINTERVA型にキャストできる

```
demo=> SELECT '2021-04-10'::date + '100 day'::interval;
      ?column?
---------------------
 2021-07-19 00:00:00
(1 row)
```

interval の単位：

year　month　week　day
hour　minute　second

SQLコマンド > その他のデータ型

#OSS-DB

配列型

- 配列型にした列は多次元配列を格納できる

```
demo=> CREATE TABLE t1 (col text[]);  -- 1次元のtext型配列を定義
demo=> INSERT INTO t1 VALUES ('{Red, Apple, Tomato}'); -- 配列は { } で表現、要素はカンマ区切り
demo=> SELECT col[2], col[3] FROM t1;  -- 配列へのアクセスは [ ] を使う、添字は 1 スタート
  col  ¦  col
-------+--------
 Apple ¦ Tomato
(1 row)
```

BOOLEAN型

- TRUE（真）・FALSE（偽）・NULL（値が存在しないことを示す）を格納
- 真偽の出力結果は「t」か「f」

```
demo=> SELECT true;
 bool
------
 t
(1 row)
```

```
demo=> SELECT false;
 bool
------
 f
(1 row)
```

SQLコマンド > 色々なSELECT文

#OSS-DB

SELECT文にはWHERE句以外のオプションがある

- ORDER BY　　SELECT結果のソート
- LIMIT, OFFSET　　SELECT結果の範囲指定
- DISTINCT　　SELECT結果から重複行を取り除く
- GROUP BY, HAVING　　SELECT結果の集計

下準備

- カルフォルニア州の天気情報を格納するテーブルを作成

```
demo=> CREATE TABLE weather (
  city varchar(80),
  temp_lo int, -- 最低気温
  temp_hi int, -- 最高気温
  prcp real,   -- 降水量
  date date
);
demo=> INSERT INTO weather VALUES
  ('San Francisco', 46, 50, 0.25, '1994-11-27'),
  ('San Francisco', 43, 57, 0.0, '1994-11-29'),
  ('Hayward', 37, 54, NULL, '1994-11-29');
```

```
     city      | temp_lo | temp_hi |  prcp  |    date
---------------+---------+---------+--------+------------
 San Francisco |      46 |      50 |   0.25 | 1994-11-27
 San Francisco |      43 |      57 |      0 | 1994-11-29
 Hayward       |      37 |      54 |        | 1994-11-29
(3 rows)
```

#OSS-DB

ORDER BY句

- SELECT 文の結果を並び替える（デフォルトは昇順）

```
SELECT 列指定 FROM テーブル名 ORDER BY 列名 [, 列名, ...];
```

- 降順にする場合はDESCを指定

```
SELECT 列指定 FROM テーブル名 ORDER BY 列名 [, 列名, ...] DESC;
```

都市名でソート

```
demo=> SELECT * FROM weather ORDER BY city;
```

ORDER BYを指定しないとデータの並びは不定

- SELECT文を実行して得られたデータの並び順が、次回も同じになる保証は無い
- データの並び順を保証したければ、必ず ORDER BY を使う必要がある

LIMIT, OFFSET句

#OSS-DB

- SELECT文の結果を範囲指定する

```
SELECT 列指定 FROM テーブル名 LIMIT 件数 OFFSET 開始行;
```

- SELECT結果の「OFFSET行」から「LIMIT」件だけ表示する
- 「OFFSET 0」で一行目から

先頭から2行だけ取得

```
demo=> SELECT * FROM weather ORDER BY city LIMIT 2 OFFSET 0;
```

ORDER BY句と併用しなければならない

- SELECT文を実行して得られたデータの並び順が、次回も同じになる保証は無い
- ORDER BYで順序を決めない限り、行のどこが返ってくるか予想がつかないため

#OSS-DB

DISTINCT句

- SELECT文の結果から重複した行を取り除く

```
SELECT DISTINCT 列名1[, 列名2, ...] FROM テーブル名 ...;
```

```
demo=> SELECT DISTINCT city FROM weather;
```

- SELECT文の結果の全列を使うのでこれは意味が無い

```
demo=> SELECT DISTINCT city, date FROM weather;
```

DISTINCT ON句

- ONで指定した列のみを使って重複を取り除く
- 重複削除条件と表示させたい列が異なる場合に使う
- ORDER BY句と併用しないと削除される行が不定になるので注意

```
SELECT DISTINCT ON (列名1[, 列名2, ...]) 列指定 FROM テーブル名 ...;
```


SQLコマンド > 色々なSELECT文 > SELECT結果の集計

#OSS-DB

集約関数

- 関数に渡した列の値を使って1行の結果を返す
  - max() 最大値
  - min() 最小値
  - avg() 平均値
  - sum() 合計
  - count() 行数

```
demo=> SELECT max(temp_hi) FROM weather; -- 最高気温を求める
```

GROUP BY句

- GROUP BYで指定した列の値毎に集約関数を実行する

```
demo=> SELECT city, max(temp_hi) FROM weather GROUP BY city; -- 都市毎の最高気温を求める
```

GROUP BY句 + HAVING句

- GROUP BYしたSELECT結果行を絞り込む

```
demo=> SELECT city, max(temp_hi) FROM weather
         GROUP BY city HAVING max(temp_hi) > 55; -- 上の結果からさらに気温が 55 度より高い行を求める
```

SQLコマンド > インデックス

#OSS-DB

インデックス

- テーブルの行データを素早く見つけるための仕組み
- テーブルの検索や並び替えが早くなる

作成

```
CREATE INDEX [インデックス名] ON テーブル名 (列名[1, 列名2, ...]);
```

削除

```
DROP INDEX インデックス名;
```

使いどころ(*)

- WHERE条件で使う列
- ORDER BYで使う列

(*) 無駄にインデックスを作成すると、検索性能の向上に効果がないだけでなく、UPDATE時にインデックスの更新がオーバーヘッドとなり、性能が低下してしまいます。

SQLコマンド > インデックス

インデックスの作成前後のSELECT性能を比較

#OSS-DB

```
demo=> CREATE TABLE index_test (num int); -- インデックステスト用テーブル
demo=> -- generate_series 関数を使って 1 から 10 万まで値を登録する
demo=> INSERT INTO index_test VALUES (generate_series(1, 100000));
demo=> \timing -- SQL の実行時間が出力されるようになる（psql のメタコマンドと呼ばれる機能）
demo=> SELECT * FROM index_test WHERE num = 50000; -- インデックスを使わない SELECT
  num
-------
 50000
(1 row)

Time: 34.597 ms
demo=> CREATE INDEX ON index_test (num); -- インデックス作成
demo=> SELECT * FROM index_test WHERE num = 50000; -- インデックスを効かせてもう一度 SELECT
  num
-------
 50000
(1 row)

Time: 1.622 ms
```

まとめ

- **PostgreSQL学習の最初の一歩を踏み出せるように以下を解説**
  - PostgreSQLの起動・停止
  - データベースクラスタの初期化
  - PostgreSQLクライアントアプリケーションの基本的な使い方
  - SQLの基本的な使い方

#OSS-DB

- **このスライドの内容はすべて試験に出るか、試験に問われる前提知識です**

- **このスライドの内容を起点にして他領域の学習へと進んでみてください**