データベースS

第5回　PostgreSQLによるデータベース実践演習

システム創成情報工学科　尾下 真樹

2018年度 Q2

今回の内容

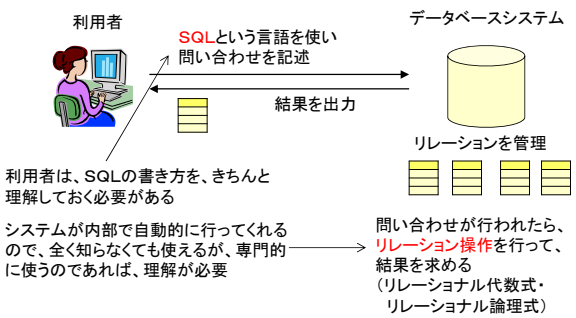
- 前回の復習
- SQLの利用形態
- 問い合わせ以外のSQL
- PostgreSQL演習
  - 演習環境
  - PostgreSQLの概要
  - PostgreSQLの基本的な使い方
  - 演習課題

前回の復習

SQL

- リレーショナルデータベース言語SQL
  - SQL（エスキューエル）
  - データベースの操作、特に問い合わせを行うための言語
  - 使いやすい
  - リレーショナル代数式、論理式などよりも記述が簡単・高機能
    - 代数演算はリレーショナルモデルの操作を規定するもの（利用者が直接使用することはあまりない）
    - SQLはデータベースのインターフェース（具体的な操作ではなく、操作の目的を記述する。）

SQLとリレーション操作の関係



リレーショナル代数 と SQL

- リレーショナル代数式
  - どのような処理でデータを出力するか（How）
- SQL
  - どのような条件のデータを出力したいか（What）
- SQLはリレーショナル完備
  - リレーショナル代数式で記述できる問い合わせは全て記述できる
  - SQLでしか記述できないような問い合わせもある

SQLによる問い合わせの記述

- SQLの基本的な書き方
- 条件（WHERE）の書き方
- 出力（SELECT）の書き方
- 順序付け（ORDER BY）
- グループ表（GROUP BY）
- 結合（JOIN）
- 集合演算
- 副問い合わせ（入れ子型質問）

SQLの記述方法

```
SELECT   表.属性（値式）, …
FROM     表, …
WHERE    条件式  AND ...
```

– SELECT 節
  - 問い合わせの結果として取り出す属性（値式）を指定

– FROM 節
  - どの表（テーブル）から検索するかを指定

– WHERE 節
  - 検索の条件を指定

– ORDER BY節、GROUP BY節、HAVING節（後述）

集約関数

- 検索結果の表に対して集計演算を行い、表の全データを出力する代わりに、集約演算の結果を1行だけ出力する
  – COUNT（行数）、SUM（合計値）, AVG（平均点）、MAX（最大値）、MIN（最小値）
  – 出力されるテーブルは1行だけになることに注意

Q. 科目番号 001 の平均点、最小点、最高点

```
SELECT   AVG（成績）, MIN（成績）, MAX（成績）
FROM     履修
WHERE    科目番号 = '001'
```

グループ表（GROUP BY）

- GROUP BY
  – 指定した属性によりデータをグループ化
    - 属性値が等しいデータ同士をグループにまとめる
  – 集約関数と組み合わせて用いる（重要）
    - 集約関数は、全データではなく、各グループごとの全データに適用される（グループ数分のデータを出力）

Q. 全科目の科目番号と平均点の一覧を出力

```
SELECT   科目番号, AVG（成績）
FROM     履修
GROUP BY 科目番号
```

グループ表の例

- グループ表の例

集約演算は、各グループごとに適用されることに注意

科目番号	学生番号	成績
001	001001	65
001	001002	75
001	001004	70
002	001002	80
002	001003	60
002	001004	90
002	001005	70
003	001004	70
…	…	…

各グループをひとつのデータ（行）として出力

科目番号	成績
001	70
002	75
…	…

グループ表＋グループ選択（HAVING）

- GROUP BY + HAVING
  – HAVING により出力するグループの条件を指定

Q. 履修者が30名以上の科目の
科目番号、履修者数、平均点の一覧

```
SELECT   科目番号, COUNT（＊）, AVG（成績）
FROM     履修
GROUP BY 科目番号
HAVING   COUNT（＊）>= 30
```

GROUP BY での処理適用順序

① FROM ……（入力とするテーブル）
② WHERE ……（出力するデータを選択）
③ GROUP BY ……（出力されたデータをグループ化）
④ HAVING ……（出力するグループを選択）
⑤ SELECT ……（出力する属性）

- FROM 節のテーブルの各データの組み合わせから、WHERE 節で書かれている条件を満たす組を選択
- 選択されたデータに対して、GROUP BY 節に書かれている属性の値が同じもの同士でグループ化
- 各グループごとに、HAVING 節に書かれている条件を満たすかどうか判定し、条件を満たすもののみを選択
- 選択されたデータ or グループの属性のうち、SELECT 節に書かれているものを出力

GROUP BY 使用時の注意

- GROUP BY 使用時の注意

```
SELECT    科目番号, COUNT(*), AVG(成績)
FROM      履修
GROUP BY  科目番号
HAVING    COUNT(*) >= 30
```

- GROUP BY 節があるときは、各グループが出力の単位となるので、SELECT 節や HAVING 節にはグループで共通な属性（GROPU BY に使った属性）や集約関数しか書けない（重要!）

WHEREとHAVINGの使い分け

- WHERE
  - データ（インスタンス）に関する条件
    - グループにまとめる前に評価される
    - データに関する条件なので、集約関数は使えない
- HAVING
  - グループに関する条件
    - グループにまとめた後で評価される
    - グループに関する条件なので、集約関数しか使えない

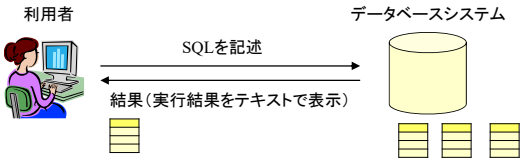
SQLの利用形態

SQLの利用形態

- SQLの利用形態
  - 直接起動（direct invocation）
    - 利用者がSQLを直接入力する
    - 結果は表として表示される
  - 埋込みSQL（embed SQL）
    - プログラミング言語の中に固定のSQLを記述しておき、プログラム実行時に呼び出す
    - 現在は、次の動的SQLの方が一般的に使われる
  - 動的SQL（dynamic SQL）
    - 埋込みSQLの問い合わせ文を、プログラムの実行時に動的に生成し、呼び出す

SQLの直接起動

- 利用者がSQLを直接入力する
- 結果は表として表示される
- （今回の演習で体験する利用方法）

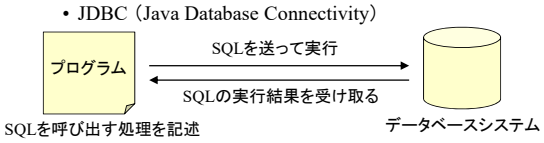


埋め込みSQL

- 親言語の一部にSQLを埋め込むことで、プログラムの途中でSQLを実行
  - SQLの結果をプログラムで扱える
  - SQLの変数と親言語の変数がどのように対応するかを定義
  - 問い合わせの結果が複数の行になる場合は、各行ごとに結果の取得を繰り返す
- コンパイル時に、データベースシステムと情報をやり取りするような処理が自動的に追加される

動的SQL

- プログラムから動的にSQLを実行して、結果を受け取り処理することができる
  - データベースとのやり取りにはライブラリを使用
  - さまざまなリレーショナルデータベースに共通的にアクセスできるようなAPI(ライブラリ)もある
    - ODBC（Open Database Connectivity）
    - JDBC（Java Database Connectivity）



SQLを呼び出す処理を記述

SQLの利用

- 実際の利用方法は、演習の講義で説明
  - 直接起動 は、今回の演習で体験
  - 動的SQL は、今後の演習で扱う
- 埋め込みSQL は、最近はあまり使われない
  - 動的SQL の方が、使い勝手が良いため

問い合わせ以外のSQLの記述

問い合わせ以外のSQL

- リレーション(表)の生成
- データ(行)の挿入
- データ(行)の削除
- データ(行)の更新

テーブルの生成

- CREATE TABLE 文を使用

```
CREATE TABLE 表名
（属性名　型　[属性に関する制約(省略可)],
 属性名　型　[属性に関する制約(省略可)],
 ・・・・・・
 表全体や複数の属性に関する制約(省略可),
 ・・・・・・
）
```

- 表や複数の属性に関する制約は末尾に記述
- 一つの属性に関する制約は属性名・型の直後 or 末尾に記述（どちらに記述しても可）

指定可能な型(1)

分類	型	意味
数値型	int2	整数（2バイト、-32378～+32767）
	int / int4	整数（4バイト、±約21億）
	int8	整数（8バイト、±約18桁）
	real / float4	浮動小数点表現による実数（4バイト）
	float8	浮動小数点表現による実数（8バイト）
文字列型	text	可変長文字列（長さ制限なし）（PostgreSQL独自）
	varchar(n)	可変長文字列（最大のn文字）
	char(n)	固定長文字列（常にn文字、自動的に空白が追加される）

指定可能な型(2)

分類	型	意味
日付時刻型	date	日付
	time	時間
	timestamp	日付と時間
	interval	日付時間の間隔
他	boolean	真偽値（TRUE, FALSE, NULL のどれかをとる）

※ これら以外の型については、参考書等を参照

指定可能な制約

- NOT NULL
  - 属性が空値(NULL)とらない(必ず値を持つ)
- UNIQUE
  - 複数のインスタンスが同じ属性値をとらない（＝候補キー属性）
- PRIMARY KEY
  - 主キー
- FOREIGN KEY ＋ REFERENCES
  - 外部キー(参照整合性制約)(詳細は後で説明)

テーブルの生成の例

- 科目(科目番号, 科目名, 単位数)の生成例

```
CREATE TABLE 科目
（科目番号　CHAR(3) NOT NULL,
 科目名　　HCHAR(12) NOT NULL,
 単位数　　INTEGER,
 PRIMARY KEY (科目番号),
 CHECK (単位数 BETWEEN 1 AND 12) )
```

  - CHAR･･･文字列、HCHAR･･･漢字文字列、INTEGER･･･整数、(??)･･･文字列のサイズ
  - NOT NULL(空値を許さないように制約)
  - PRIMARY KEY(主キー), CHECK(制約条件)

データの挿入

- 挿入
  - INSERT節, INTO節, VALUES節 を使用
- 挿入例
  - 履修(科目番号, 学生番号, 成績)
  - 学生番号 001001 の学生が、科目番号 005 の科目を履修した(成績は未定)

```
INSERT
INTO　　履修
VALUES　('005', '001001', NULL )
```

データの削除

- 削除
  - DELETE文を使用(FROM節, WHERE節も使用)
- 削除例
  - 履修(科目番号, 学生番号, 成績)
  - 学生番号 001001 の学生が、科目番号 005 の科目の履修を取り消した

```
DELETE
FROM　　履修
WHERE　科目番号= '005' AND 学生番号= '001001'
```

データの更新

- 更新
  - UPDATE節, SET節を使用（WHERE節も使用）
- 更新例
  - 履修（科目番号, 学生番号, 成績）
  - 学生番号 001001 の学生の、科目番号 005 の科目の成績が 80点になった

```
UPDATE   履修
SET      成績 = 80
WHERE    科目番号= '005' AND 学生番号= '001001'
```

PostgreSQL演習

演習の流れ

- 演習のやり方を講義で説明
- 講義外の空き時間に、各自、CL（学科端末室）で演習を行う
- 各回の演習が終わったら、課題を提出
- 最終レポート課題（後日の講義で説明）

演習の参考書

- 「PHP5 徹底攻略」
  堀田 倫英、桑村 潤 著
  ソフトバンクパブリッシング（3,800円）
  - PHP（後日説明）+ PostgreSQL についての詳しい参考書

- 「PostgreSQLによるLinuxデータベース構築」
  廉升烈 著、翔泳社 出版（2,200円）
  - 演習では基本的に最低限の資料は用意するので無理に買わなくても良い

演習資料

- 演習資料（データベース演習資料 第1回）
  - この資料に従って、今回の演習を進める
    - 末尾のトラブル対応方法の説明等もきちんと読むこと
  - 次回以降の演習でも、同様の資料を使用

CLの利用方法

- Computing Laboratory (CL) 研究棟 6F
  - 学科の端末室
    - Windows 8.1 端末
    - 情報科学センターの端末とは環境が異なる
  - 端末数 33台
    - 全員同時には使えないので注意
  - アカウントは情報科学センタと共通
  - 端末の利用について問題等が発生したら、学科の技術職員室（研究棟E526）に相談に行くこと
    - 教員のところに来ても、対応できない

演習環境の説明

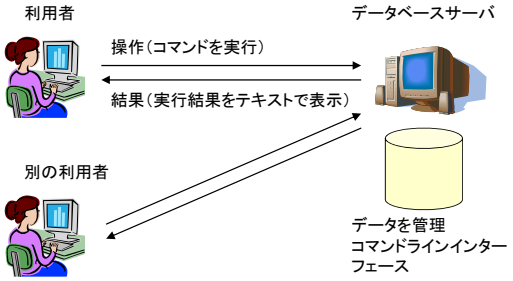
- PostgreSQL（ぽすとぐれす、ぽすとぐれすきゅーえる）
  - フリーのリレーショナルデータベースシステム
  - 広く使われている
  - それほどパフォーマンスが要求されない用途であれば、十分実用になる
  - Unix、Windows、Mac等さまざまな環境で利用可能
    - 希望者は、自宅のパソコンに PostgreSQL をインストールして演習やレポート課題を行っても構わない
    - ただし提出レポートはCL環境で動作する必要がある

PostgreSQLの実行環境

- クライアント・サーバ環境
  - PostgreSQLサーバ
    - 全員のデータベースを管理
    - 今回の演習では、popuradb.ces.kyutech.ac.jp という学科のサーバを使用
  - クライアント
    - データベースにコマンドやSQLなどを送り、結果を受け取る
    - 今回の演習では、CL端末室のPC（pcXX.ces.kyutech.ac.jp）

クライアント・サーバ環境

- サーバがクライアントにサービスを提供する



psql

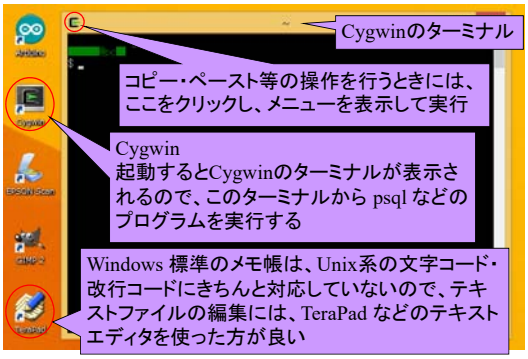
- psql
  - クライアントで使用するフロントエンド・プログラム
  - 対話的にコマンドやSQLを実行できる
  - キャラクタ（文字）ベースのプログラム
    - グラフィカルユーザインターフェース（GUI）はなく、文字で情報の表示・入力が行われる

Cygwin

- CL端末では、Cygwin環境で psql を使うように設定されている
- Cygwin（シグウィン）
  - Unix と同様の環境を Windows で実現するためのソフトウェア
  - 最新の Windows 版の PostgreSQL は、Cygwin を使わなくとも利用できるが、現在の CL端末には、Cygwin を使うものが設定されている

CL端末のソフトウェア

CL端末のソフトウェア



PosgreSQLの使い方

- データベースの作成
- psqlの起動
- テーブルの作成
- データの挿入
- SQLによる問い合わせ
- データの更新と削除
- 複数のテーブルと外部参照整合性制約

データベース作成、psqlの起動

- createdb コマンド（プログラム）
  - 最初にデータベースを作成する必要がある
  - データベース名（本演習ではユーザ名と同じにする）、PostgreSQLサーバ名、文字コードを指定
  - 最初に一度だけ行う
  - 詳しい使用方法は、資料を参照
- psql コマンド（プログラム）
  - 同じくデータベース名とサーバ名を指定して起動
  - コマンド・SQLを使ってデータベースを操作

データベース作成、psqlの起動

```
username@pcXX ~
# createdb dbname -h popuradb.ces.kyutech.ac.jp -U username -E UTF8
CREATE DATABASE
# psql dbname -h popuradb.ces.kyutech.ac.jp -U username
Welcome to psql 7.3.2, the PostgreSQL interactive terminal.
Type:  ¥copyright for distribution terms
       ¥h for help with SQL commands
       ¥? for help on internal slash commands
       ¥g or terminate with semicolon to execute query
       ¥q to quit
dbname=#
```

username には、PostgreSQLユーザ名を指定

dbname には、データベースの名前を指定

ユーザ名・データベース名

- PostgreSQL ユーザ名（*username*）
  - 本授業の履修登録者は、各自のアカウント名（九工大ID）と同じものを PostgreSQLユーザ名として登録済み
- データベース名（*dbname*）
  - 上の PostgreSQLユーザ名と同じものを、データベース名とする
    - 通常はデータベース名は作成者が自由に決められるが、他の人と名前が重複すると困るので、本授業の演習では、ユーザ名＝データベース名とする

psqlの操作

- psqlの内部コマンド
  - ¥ で始まるコマンド（環境によっては ¥ は ＼（バックスラッシュ）として表示される）
  - メタ情報（テーブル一覧）の表示や、ファイルの読み込みなど
- SQL
  - テーブルの作成、データの挿入、問い合わせ
  - SQLなどのコマンドの最後には ; を入力
    - ; を入力するまで、複数行に渡って入力しても良い

psqlの内部コマンド

- コマンドの例

コマンド	機能
¥q	psqlを終了。
¥l	データベース一覧の表示。
¥d	テーブル一覧の表示。
¥i *filename*	オプション*filename*で指定したファイルを読み込み、そのコマンドを実行する。
¥copy	ファイルとテーブルの間でデータを読み書きする。詳しい使い方は後述。
¥?	psqlで使用可能なコマンドの一覧を表示。

PosgreSQLの使い方

- データベースの作成
- psqlの起動
- テーブルの作成
- データの挿入
- SQLによる問い合わせ
- データの更新と削除
- 複数のテーブルと外部参照整合性制約

テーブルの作成の例

- 以下のテーブル（リレーション）を作成してみる

従業員

従業員番号	部門番号	氏名	年齢
0001	01	織田 信長	48
….	..	……	..

4桁の数字　2桁の数字　最大12文字　整数

  - テーブル名や属性名にはアルファベットを使うのが無難（日本語を使うと問題が出ることがある）

employee

id	dept_no	name	age

テーブルの作成

- テーブルの作成（CREATE TABLE 文）
  - employee (id, dept_no, name, age)

```
create table employee(
  id        varchar(4)    not null unique,
  dept_no   varchar(2),
  name      varchar(12)   not null,
  age       int2,
  primary key( id )
);
```

- 入力方法（どちらの方法で行っても構わない）
  1. psqlのコマンドラインから直接入力
  2. ファイルに記述して、¥i コマンドで読み込み実行

テーブルの作成（方法1）

- create table を実行
  - psqlのコマンドラインから直接入力
  - 前スライドの例のように途中に改行を入れてもOK（psql は ; （セミコロン）までを一つのコマンドとして解釈する）

```
dbname=# create table employee ( id varchar(4) not null unique, dept_no varchar(2), name varchar(12) not null,age int2, primary key( id ) ) ;
```

テーブルの作成（方法2）

- テキストファイルから実行することもできる
  - あらかじめ、テキストファイルに処理の内容を記述した上で、ファイルを呼び出して実行

employee_table.txt

```
create table employee(
      id          varchar(4)      not null unique,
      dept_no     varchar(2),
      name        varchar(12)     not null,
      age         int2,
      primary     key( id )
);
```

```
dbname=# ¥i employee_table.txt
```

PosgreSQLの使い方

- データベースの作成
- psqlの起動
- テーブルの作成
- データの挿入
- SQLによる問い合わせ
- データの更新と削除
- 複数のテーブルと外部参照整合性制約

データの挿入（方法1）

- INSERT文を使用

```
insert into employee( id, dept_no, name, age )
                    values( '0001', '007', 'taro', 20 );
```

- 入力方法（どの方法で行っても構わない）
  1. psqlのコマンドラインから INSERT文を直接入力
     - 一度にひとつのデータしか挿入できない
  2. ファイルに INSERT文を記述して、¥i コマンドで読み込み実行（テーブルの作成と同じ方法）
  3. ファイルにデータを記述して、¥COPYコマンドで読み込み設定

データの挿入（方法3）

- ¥COPYコマンドによる方法
  - あらかじめテキストファイルにデータを記述し、¥COPY コマンドで一度にテーブルに読み込み

employee_data.txt

```
0001,01,織田 信長,48
0002,02,豊臣 秀吉,45
0003,03,徳川 家康,39
0004,01,柴田 勝家,60
0005,02,伊達 政宗,15
0006,03,上杉 景勝,26
0007,01,島津 家久,35
```

```
dbname =# ¥COPY employee FROM 'employee_data.txt'
                USING DELIMITERS ','
```

テーブルの削除

- drop table コマンドで、テーブルを削除可能
  - 演習中に、間違えてテーブルのデータがおかしくなったりしたときには、一度テーブルを削除して作り直すと良い

```
dbname =# drop table employee;
```

PosgreSQLの使い方

- データベースの作成
- psqlの起動
- テーブルの作成
- データの挿入
- SQLによる問い合わせ
- データの更新と削除
- 複数のテーブルと外部参照整合性制約

問い合わせ

- SQLによる問い合わせ
  - SQLを使って、さまざまな問い合わせ（検索）を実行することができる
- 同じくSQLを使用して、データの削除（DELETE）や変更（UPDATE）もできる
  - 詳しくは、資料を参照

SQLの記述方法（復習）

```
SELECT   表.属性（値式）, …
FROM     表, …
WHERE    条件式  AND ...
```

- SELECT 節
  - 問い合わせの結果として取り出す属性（値式）を指定
- FROM 節
  - どの表（テーブル）から検索するかを指定
- WHERE 節
  - 検索の条件を指定
- ORDER BY節、GROUP BY節、HAVING節（後述）

問い合わせの例

- 全従業員の全情報の一覧

select に＊を指定すると全属性を出力
where を省略すると全てのデータを出力

```
select ＊ from employee;
```

- 20歳以下の従業員の氏名の一覧

```
select name from employee where age < 21;
```

- 部門番号01の従業員の人数

select に count（＊）を指定するとデータの数を出力（集約関数）

```
select count（＊）from employee where dept_no = '01';
```

PosgreSQLの使い方

- データベースの作成
- psqlの起動
- テーブルの作成
- データの挿入
- SQLによる問い合わせ
- データの更新と削除
- 複数のテーブルと外部参照整合性制約

参照整合性制約の例（復習）

従業員

従業員番号	部門番号	氏名	年齢
0001	01	織田 信長	48
0002	02	豊臣 秀吉	45
0003	03	徳川 家康	39
0004	01	柴田 勝家	60

主キー　外部キー

部門

部門番号	部門名
01	開発
02	営業
03	総務

主キー

この場合、従業員の部門番号は、必ず部門の部門番号（部門の主キー）に存在する必要がある
→ 参照整合性制約

複数のテーブルの追加

- 部門のテーブル・データも追加してみる

department.txt

```
create table department(
        dept_no varchar(2) not null unique,
        name varchar(12) not null,
        primary key( dept_no )
);
insert into department values( '01', '開発' );
insert into department values( '02', '営業' );
insert into department values( '03', '総務' );
```

```
dbname=# ¥i department.txt
```

外部参照整合性制約の設定

- 既存の従業員テーブルに制約を追加
  - alter table テーブル名 add constraint 制約名 ・・・
    - 制約名は適当（後から制約を削除するときに使用）
  - 外部参照整合性制約
    - foreign key（外部キー属性）
      references 参照先テーブル（参照先属性）
    - 制約により、不整合なデータの挿入を防げる
  - テーブル作成時に最初から指定することも可能

```
dbname =# alter table employee add constraint
employee_dept_key foreign key (dept_no) references
department (dept_no);
```

外部参照整合性制約の確認

- 外部参照整合性制約を満たさないデータを挿入できるかどうか、確認してみる

```
insert into employee values( '0020', '04', 'Jack', 20 );
```

外部参照整合性制約の設定

- テーブルを作成する時点で参照整合性制約を設定することもできる
  - ただし、参照先のテーブルは作成済みである必要がある
- 参照整合性制約を含むテーブル作成の例
  - deparement テーブルは作成済みとする

```
dbname=# create table employee ( id varchar(4) not null unique, dept_no varchar(2), name varchar(12) not null,age int2, primary key( id ), foreign key (dept_no) references department (dept_no) );
```

文字コードに関する注意（1）

- 日本語の文字コードは、Shift-JIS、EUC、UTF8（ユニコード）など複数ある
  - Windows 環境では Shift-JIS、UNIX環境ではEUCが使われていた
    - 最近は、どちらの環境もユニコードに対応しつつある
- 環境によって文字コードが異なるため、CL以外の環境で演習を行ったり、レポートを書いたりしたい場合は、注意をする必要がある
  - CL以外の環境（自宅PCなど）は、講義ではサポートしないので、やりたい人は自力でやること

文字コードに関する注意（2）

- 本演習で使用する CL端末、PostgreSQLの文字コードは UTF8（ユニコード）
- データベースのテーブルに日本語を格納する場合は、UTF8 で記述する必要がある
  - テキストファイルにデータを記述し、データベースに読み込ませる場合は、テキストファイルの文字コードを UTF8 で作成する
  - Windowsでテキストファイルを作成すると、文字コードは Shift-JIS になっていることが多いため、あらかじめ変換をする必要がある
    - 文字コードの変換方法は演習資料の末尾を参照

演習課題

演習

- 資料の説明に従って、データベースを作成し、用意されているデータを追加
- 従業員のデータを追加
- 用意されているSQLを実行して、結果を確認
- Moodleに置いているレポート用のファイルに結果を貼り付けて、Moodleから提出
- 提出締め切り 7/2（月）18:00（厳守）

演習1. データベースの作成

- 資料の説明に従って、データベースを作成し、用意されているデータを追加

従業員

従業員番号	部門番号	氏名	年齢
0001	01	織田 信長	48
0002	02	豊臣 秀吉	45
0003	03	徳川 家康	39
0004	01	柴田 勝家	60
0005	01	伊達 政宗	15
0006	02	上杉 景勝	26
0007	03	島津 家久	35

部門

部門番号	部門名
01	開発
02	営業
03	総務

※ テーブル名や属性名はアルファベットにする

演習2. データの追加

- 従業員のデータを、最低 5つ以上追加する
  - データの属性値は、適当に決める
  - データの追加は、どの方法で行っても構わない

従業員

従業員番号	部門番号	氏名	年齢
….	..	…..	..
0008	01	山田 一郎	20

データを追加

演習3. 問い合わせの実行

- 全従業員の全情報の一覧

```
select * from employee;
```

- 30歳以下の従業員の氏名の一覧

```
select name from employee where age < 31;
```

- 部門番号01の従業員の人数

```
select count(*) from employee where dept_no = '01';
```

- 実行結果をファイルにコピーする
  資料の 付録1.ファイルへの出力方法 の説明を参照

演習4.外部参照整合性制約の確認

- 外部参照整合性制約を満たさないデータを挿入できるかどうか、確認してみる

```
insert into employee values( '0020', '04', 'Jack', 20 );
```

- 同じく、実行結果をファイルにコピーする

演習の注意

- きちんと自分で演習を行うこと
  - 必ず、自分のアカウント名で作成したデータベースを使って演習を行うこと。
  - 従業員のデータは、必ず自分で作成したデータを追加すること。
    他の人からもらったデータ(他の人と全く同じデータ)を使っていた場合は、0点とする。
- 次回以降の演習を行うためにも、今回の演習で、psql の使い方に慣れておくことが重要

演習補助資料

- CL端末以外の環境で演習を行いたい場合は、Moodleで公開している演習補助資料を参照すること
  - 情報科学センタの端末からの演習
  - マルチメディア教室の端末からの演習

まとめ

- 前回の復習
- SQLの利用形態
- 問い合わせ以外のSQL
- PostgreSQL演習
  - 演習環境
  - PostgreSQLの概要
  - PostgreSQLの基本的な使い方
  - 演習課題

次回予告

- SQLによる問い合わせの記述方法(2)
- PostgreSQLを使ったSQL演習