

# 上級プログラミング2(第5回)

---

工学部 情報工学科  
木村昌臣

# 今日のテーマ

---

## □ データベース入門

- データベースシステムの用意の仕方
- データベースを作ってみる
- データベースを使ってみる

## □ データベースプログラミング

- JDBCを使った検索プログラム
  - JDBCを使った更新プログラム
-

# データベース入門

---

※今回の説明は、Windows環境を前提としているため、Linuxなど他のプラットフォーム上で作業を行う場合は、読み替えが必要(基本的にやることは同じ)

# データベース

---

- 大量データを蓄積し、検索を可能にするシステム
  - 様々な構造のシステムが考案され使われてきた
    - 階層型データベース
    - ネットワークデータベース
    - リレーショナルデータベース
    - オブジェクト指向データベース
    - XMLデータベース など
  - データベースのデータを管理するシステムをデータベース管理システム(DBMS)という
-

# リレーショナルデータベース


---

- データは表形式で保持される
    - 表のことをテーブルと呼ぶ
  - 検索言語はSQL(国際標準)
    - DML(データ操作言語)
      - データ検索はSELECT文
      - データ更新はUPDATE文
      - データ挿入はINSERT文
      - データ削除はDELETE文
    - DDL(データ記述言語)
-

# 前提: 下記のサイトからインストーラー を入手、インストール済みとする

□ <http://www.postgresql.org/download/>

[Text Size: Normal / Large](#) | [Donate](#) | [Contact](#) |

 **PostgreSQL**


The world's most advanced open source database.

[Home](#) | [About](#) | [Downloads](#) | [Documentation](#) | [Community](#) | [Developers](#) | [Support](#)

## PostgreSQL 9.1 Beta 2 now available!

Beta release 9.1 Beta 2 is now available for download. We encourage developers and users to test this version. The release includes synchronous replication, writeable common table expressions, per-column collation, serializable snapshot isolation and many other features.

» [Release Announcement](#)  
» [Download](#)



### » LATEST RELEASES

9.0.4 · 2011-04-18 · [Notes](#)  
8.4.8 · 2011-04-18 · [Notes](#)  
8.3.15 · 2011-04-18 · [Notes](#)  
8.2.21 · 2011-04-18 · [Notes](#)

[Download](#) | [RSS](#)  
[Why should I upgrade?](#)

### » SHORTCUTS

» [Security](#)  
» [International Sites](#)  
» [Mailing Lists](#)  
» [pgFoundry](#)  
» [Wiki](#)  
» [Report a Bug](#)  
» [FAQs](#)

### » SUPPORT US

PostgreSQL is free. Please support our work by making a [donation](#).

### » FEATURED USER

Overall, PostgreSQL has been faster than the commercial product from which we converted.

**Kevin Grittner, [Wisconsin Court System](#)**

» [Case Studies](#) | [More Quotes](#) | [Featured Users](#)

# インストール後にすること

---

- データベースクライアントpsql.exeを実行してみる
  - スタートメニューから実行すると管理ユーザーとしてデフォルトデータベースtemplate1に接続される



# データベースを作成する

---

```
template1=# create database test1;  
CREATE DATABASE  
template1=# .
```

**create database データベース名**

というDDL文を実行すると、その名前のデータベースが新規に作成される。さらに接続用ユーザーを作成する。

```
template1=# create user oops password 'pass';  
CREATE USER  
template1=# .
```

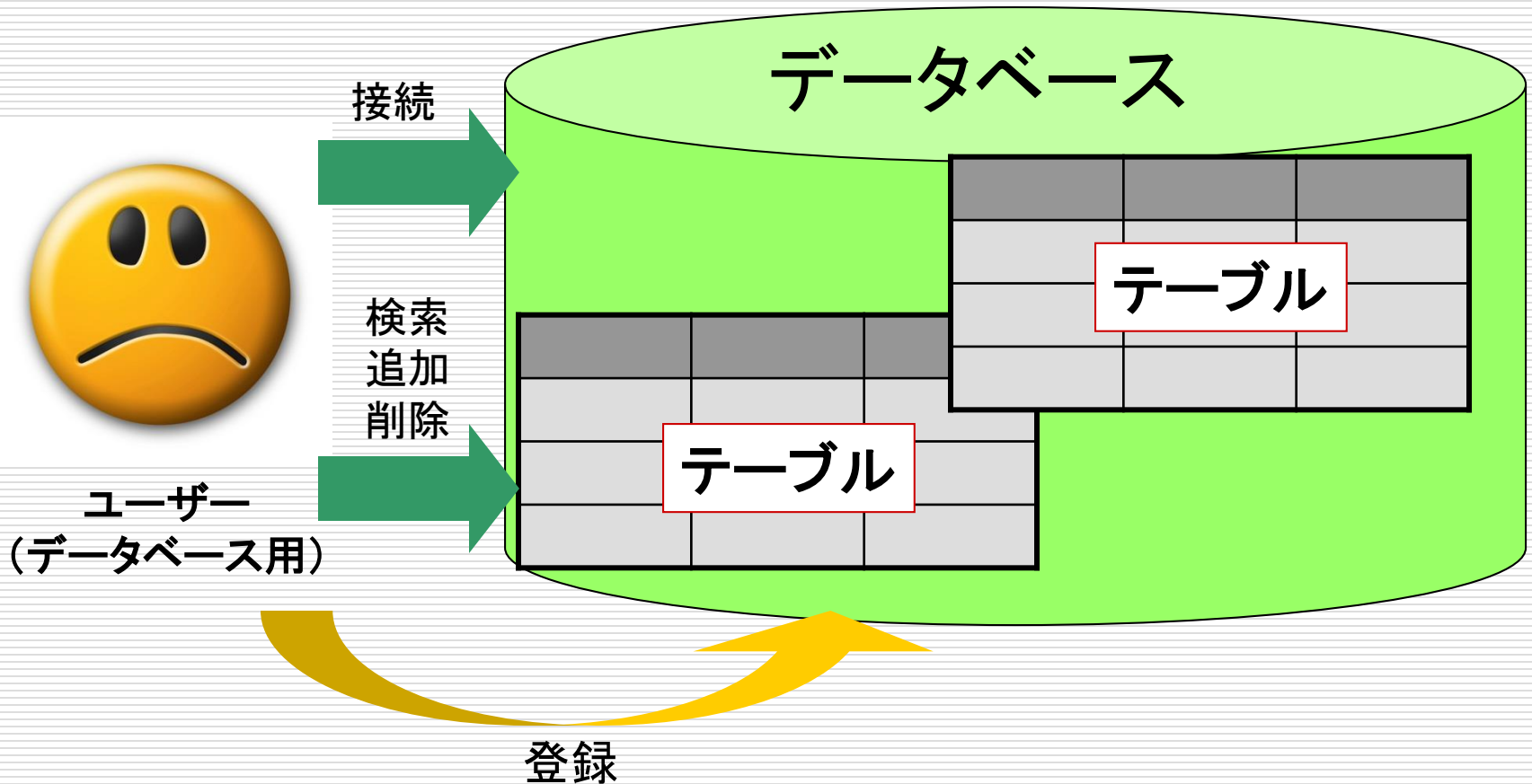
**create user ユーザー名 password 'パスワード'**

---



# データベースとテーブル・ユーザの関係

---



# テーブルを作る

```
test1=> create table test_table  
test1-> ( ID VARCHAR(5),  
test1(>  NAME VARCHAR(50),  
test1(>  AGE  INTEGER,  
test1(>  ADDRESS VARCHAR(50)  
test1(> );  
CREATE TABLE  
test1=>
```

データ型の例:

可変文字列    VARCHAR(バイト数)

整数値        INTEGER

実数値        REAL

正確な数値    NUMBER(有効桁数, 小数点下桁数)

**create table** テーブル名  
( 列名 データ型,  
  列名 データ型,  
   ....  
  列名 データ型)

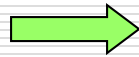
# データの検索 (SELECT文)

---

テーブルT\_SHOHINから  
商品コードが23333である商品の単価を取り出す

T\_SHOHIN

商品コード	商品名	単価
10001	携帯電話	20000
10002	パソコン	200000
23333	プリンタ	35000
43990	ハードディスク	10000



```
SELECT 単価  
FROM T_SHOHIN  
WHERE 商品コード='23333'
```

# データの挿入(INSERT文)

---

新製品 MP3プレーヤー(単価25000円)を  
商品コード10003として登録したい

T\_SHOHIN

商品コード	商品名	単価
10001	携帯電話	20000
10002	パソコン	200000
23333	プリンタ	35000
43990	ハードディスク	10000

```
INSERT INTO T_SHOHIN VALUES  
( '10003', 'MP3プレーヤー', 25000)
```

---

# データの変更(UPDATE文)

---

商品コード'10001'の商品の単価を15000円に修正したい

T\_SHOHIN

商品コード	商品名	単価
10001	携帯電話	20000
10002	パソコン	200000
23333	プリンタ	35000
43990	ハードディスク	10000

UPDATE T\_SHOHIN

SET 単価=15000

WHERE 商品コード='10001'

同じレコードの二つ以上のフィールドを変更したい場合は  
SET 列名1='XXXX', 列名2='YYYY' のように列記する。

---

# データの削除(DELETE文)

---

商品コード'10001'の商品をマスタから削除したい

T\_SHOHIN

商品コード	商品名	単価
10001	携帯電話	20000
10002	パソコン	200000
23333	プリンタ	35000
43990	ハードディスク	10000

```
DELETE FROM T_SHOHIN  
WHERE 商品コード='10001'
```

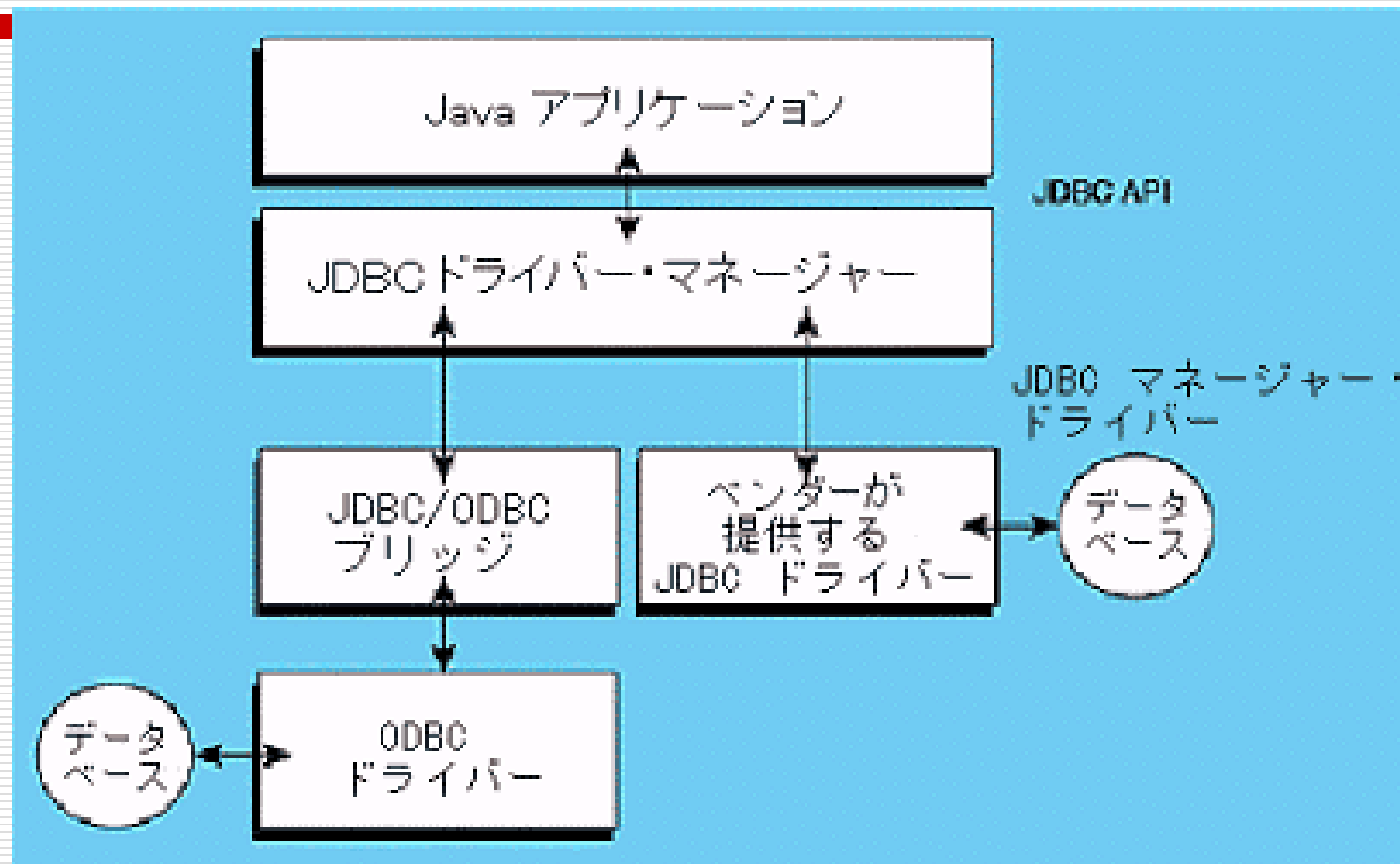
---

# JDBC

---

- Java Database Connectivity
  - Javaプログラムからリレーショナルデータベースに対し検索・更新するためのインターフェイス(API)
    - SQLをリレーショナルデータベースへ渡し、結果を取得するためのインターフェイス
  - 標準化されているが、実際に接続するためには、個々のデータベースに対応したJDBCドライバ(クラス)を利用する必要あり
-

# JDBCを使ったJAVAアプリケーションとリレーショナルデータベースの通信方法



出展:<http://www-06.ibm.com/jp/software/data/developer/library/techdoc/jdbc.html>

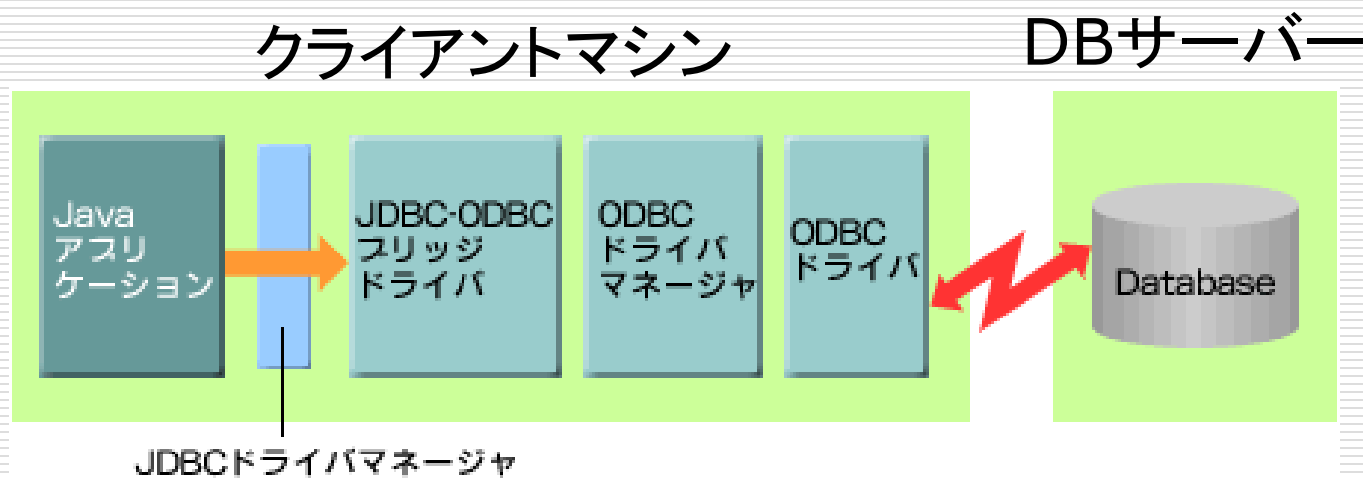


# JDBCの種類

---

## □ Type1 (JDBC-ODBCブリッジ)

- Microsoft社のデータベース接続API経由で接続
- ODBCドライバが導入されていることが前提

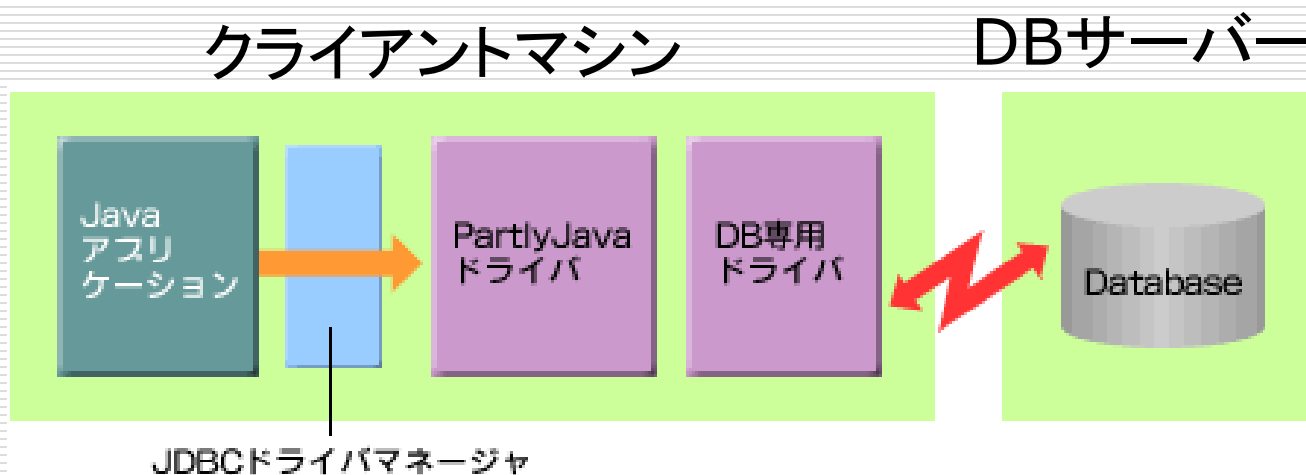


# JDBCの種類

---

## □ Type2 (ネイティブブリッジドライバ)

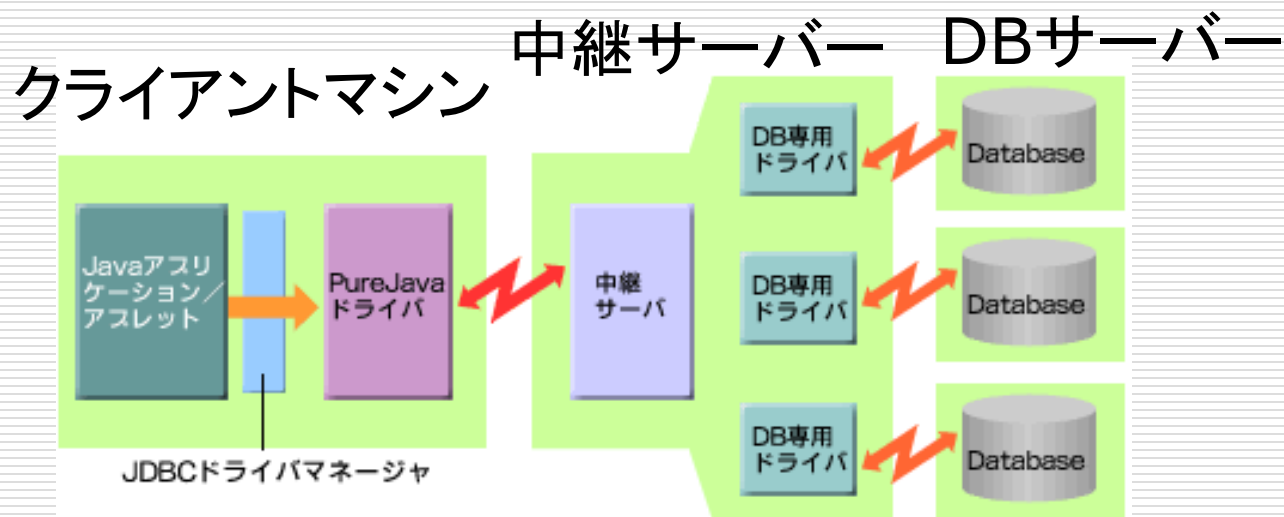
- データベース固有の接続方式経由で接続
- データベース専用クライアントソフト(ドライバ)を利用するため、その導入が前提



# JDBCの種類

## □ Type3 (ネットプロトコルドライバ)

- 中継サーバー経由でデータベースサーバーに接続
- JDBCドライバはJavaのみで作られているため、クライアントのプラットフォームに依存しない

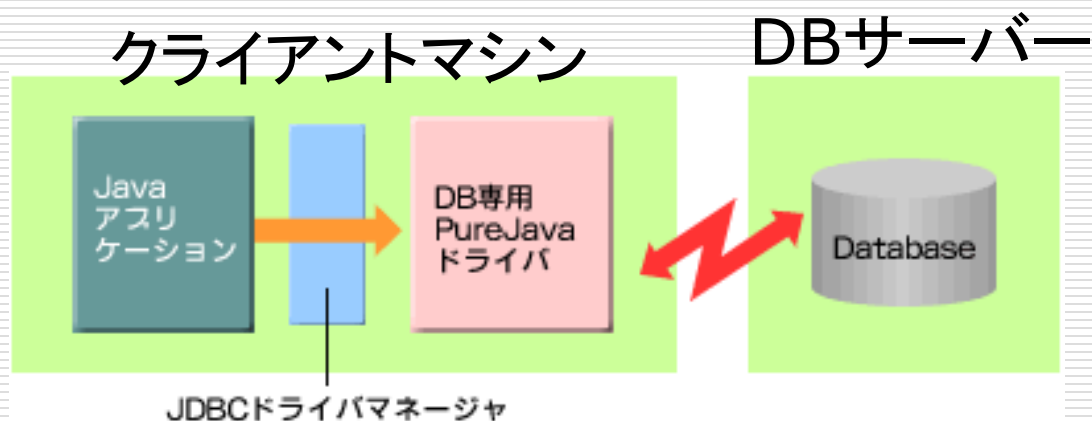


# JDBCの種類

---

## □ Type4

- データベース製品に対し、専用のJDBCが直接接続
- JDBCドライバはJavaのみで作られているため、クライアントのプラットフォームに依存しない



# JDBCを使ったプログラムを動かすために必要な設定

---

- JDBCドライバが含まれているディレクトリもしくはJARファイルがある場所(絶対パス)を環境変数CLASSPATHに追加
  - データベース検索をするJavaプログラムのコンパイルや実行前に次のコマンドを実行※
    - SET CLASSPATH=C:¥temp¥postgresql-9.4.1208.jre6.jar;%CLASSPATH%
  - Windowsの「マイコンピュータ」の「システムのプロパティ」の「詳細設定」で設定することも可

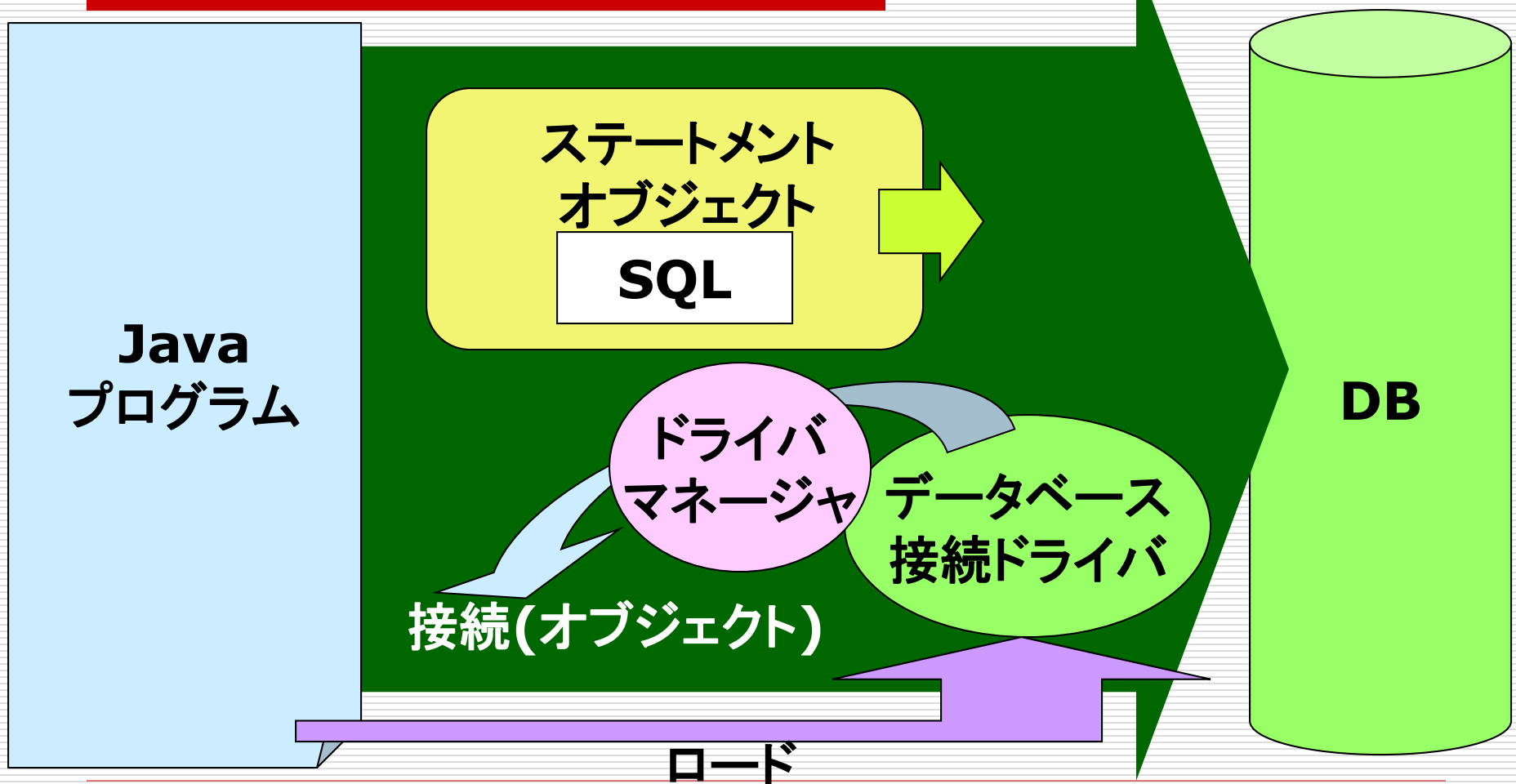
---

※この例では、ドライバが含まれるJARファイルをC:¥tempディレクトリにコピーしている

# データベースプログラミング

---

# データベース利用プログラムの構成



# データベース利用プログラムの構成

---

□ データベース接続ドライバのロード

□ 接続(コネクション)の確立

■ URLやユーザー・パスワードの設定

(例) jdbc:postgresql://localhost:5432/test1

ユーザー oops パスワード pass



**DB名**

□ ステートメントオブジェクトの作成

■ SQLの送信

■ 検索の場合は結果セットの取得も

---



# JDBCを使った検索プログラム (1/3)

---

```
import java.sql.*;

public class JDBC_PSQL {

    public static void main(String[] args) {

        try {
            String url="jdbc:postgresql://localhost:5432/test1"
            Class.forName("org.postgresql.Driver");
            Connection con =
                DriverManager.getConnection(url,"oops","pass");
            Statement stmt = con.createStatement();
```

---

続く

# JDBCを使った検索プログラム(2/3)

---

```
String sql = "SELECT * FROM TEST_TABLE";
```

```
ResultSet rs = stmt.executeQuery(sql);
```

```
while(rs.next()){  
    String sName = rs.getString("NAME");  
    int age = rs.getInt("AGE");  
    String sAddr = rs.getString("ADDRESS");  
    System.out.println(sName+" "+age+" "+sAddr);  
}
```

---

# JDBCを使った検索プログラム(3/3)

---

```
stmt.close();  
con.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}  
}
```

---

# 結果セット(ResultSet)オブジェクト

---

- 検索結果が格納されるオブジェクト
    - 実際は、検索結果の取得行を指し示す役割を果たすもの(カーソル)であり、検索結果の実体が全て結果セットオブジェクトに格納されるわけではない
  - Statementオブジェクトのメソッド  
executeQuery()の戻り値として取得
    - `ResultSet rs = stmt.executeQuery(SQL文);`
  - nextメソッドで次の行へ
    - 次の行があればtrueを返し、最終行など次の行がなければfalseを返す
-

## テーブルの定義(再掲)

```
create table test_table (  
    ID      VARCHAR(5),  
    NAME    VARCHAR(50),  
    AGE     INTEGER,  
    ADDRESS VARCHAR(50),  
)
```

データ型の例:

可変文字列    VARCHAR(バイト数)

整数値        INTEGER

実数値        REAL

正確な数値    NUMBER(有効桁数, 小数点下桁数)

# 結果セットのメソッド(データ取得用)

---

- get型名(列名)もしくはget型名(列番号)の形をしている
- 例)NAME列(文字列)を取得する際、NAME列が左から1番目にあれば、次の二つは同じ
  - getString("NAME")
  - getString(1)    一列目は1とカウントする点に注意
- 例)AGE列(整数)の場合
  - getInt("AGE")
  - getInt(2)

# JDBCを使った更新プログラム(1/3)

---

```
import java.sql.*;
```

```
public class JDBC_UPDATE {
```

```
    public static void main(String[] args) {
```

```
        try {
```

```
            String url="jdbc:postgresql://localhost:5432/test1"
```

```
            Class.forName("org.postgresql.Driver");
```

```
            Connection con =
```

```
                DriverManager.getConnection(url,"oops","pass");
```

```
            Statement stmt = con.createStatement();
```

---

続く

## JDBCを使った更新プログラム(2/3)

---

```
String sql= "INSERT INTO TEST_TABLE "  
            sql+= " VALUES "  
            sql+= " ('A5','静御前',21,'京都府京都市 ')"
```

```
stmt.executeUpdate(sql);
```

- 更新時にはステートメントオブジェクトの  
executeUpdateメソッドを使う



# JDBCを使った更新プログラム(3/3)

---

```
stmt.close();  
con.close();
```

```
} catch (Exception e) {  
    e.printStackTrace();  
}
```

```
}
```

```
}
```

---

# PreparedStatement

---

- 同じSQL文を何度も利用する場合は、あらかじめSQL文だけを先にコンパイルしておき、それを使いまわす仕組みを使ったほうがお得
    - PreparedStatementクラスを使うと対象とするSQLを一度だけコンパイルし、使いまわせる
    - Statementを用いる方法では、SQLを実行するごとにコンパイルされてしまう
  - SQL文で、あとで埋めたい部分(値などのパラメータ)は、「?」にしておく
    - SQLインジェクション対策として有効
-

# PreparedStatementを使った更新プログラム

---

```
PreparedStatement pstmt;  
String sql = "INSERT INTO TEST_TABLE VALUES  
            (?, ?, ?, ?)";  
pstmt = con.prepareStatement(sql);  
  
String[] ID = {"A6", "A7", "A8"};  
String[] name  
    = {"山田俊雄", "並木幸治", "高橋美紀"};  
int[] age = {62, 75, 61};  
String[] address  
    = {"神奈川県横須賀市", "高知県高知市", "埼玉県越谷市"};
```

# PreparedStatementを使った更新プログラム

---

```
for(int i = 0; i < 3; i++) {  
    pstmt.setString(1, ID[i]);  
    pstmt.setString(2, name[i]);  
    pstmt.setInt(3, age[i]);  
    pstmt.setString(4, address[i]);  
  
    pstmt.executeUpdate();  
}
```

prestmt.setString(1, ID[i]);  
⇒ 一番目の?マークにID[i]の値を置く