東京ITスクール
TOKYO IT SCHOOL

JDBC による DB プログラミング

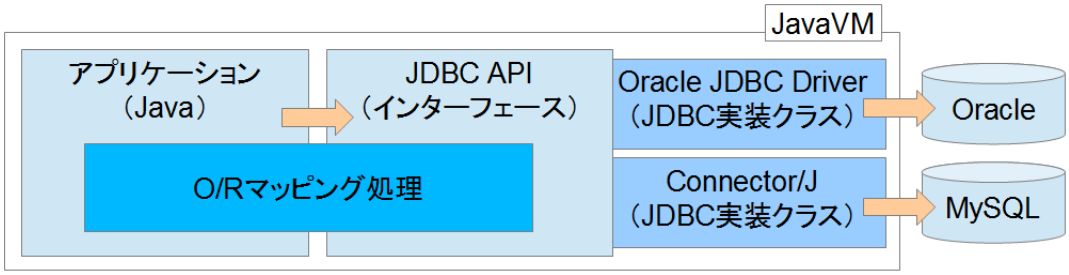
目次

1. JDBC とは

1　JDBC と ORM

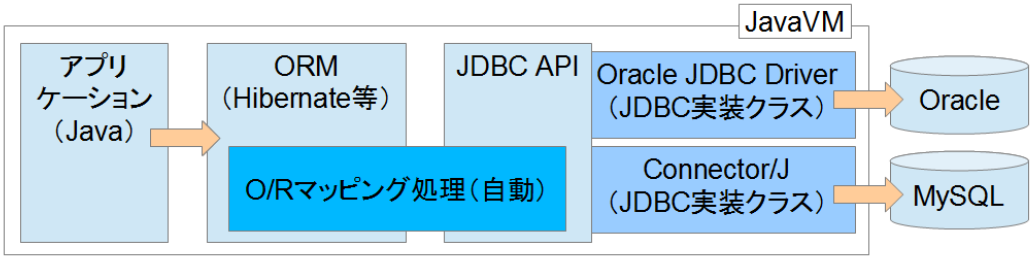
Java で DB を扱う際には、JDBC や ORM を用います。

JBDC（Java DataBase Connectivity）とは、Java から DB にアクセスするための標準 API です。JDBC 自体はインターフェース（仕様）で、使用する際には DB 固有の JDBC 実装ライブラリ（JDBC ドライバ）が別途必要になります。従って、基本的には DB 間の違いは JDBC が吸収してくれる仕様となっています。以下に、JDBC の概要を示します。



ORM（Object-Relational Mapping）とは、DB のレコードと Java オブジェクトを自動的に紐づける O/R マッピング処理や DB への接続/切断、トランザクション処理など、DB 関連の処理を自動化してくれるフレームワークです。

代表的な ORM として、Hibernate、iBatis、S2JDBC などがあります。ORM によっては、SQL 文を書かずに DB アクセスが出来るものもあります。以下に、ORM の概要を示します。

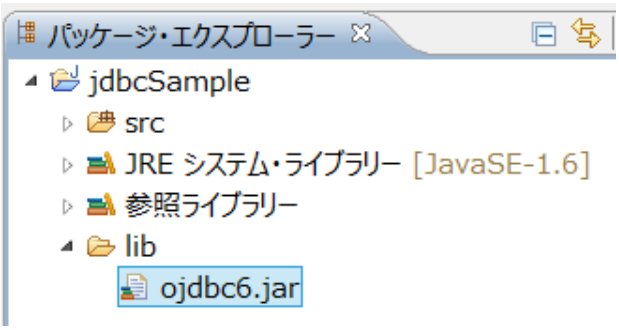


2. DB への接続/切断プログラムの作成

1　JDBC ドライバの準備

JDBC による DB アプリケーションを作成する際は、接続する DB 用の JDBC ドライバを入手し、クラスパスを通す必要があります。
また、Oracle Database 11g Release 2 用の JDBC ドライバは以下の Web サイトからダウンロードページ出来ます。

http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html



2　DB 接続の手順

それでは、早速 JDBC を利用して DB へ接続してみましょう。DB への接続は、以下の手順で行います。

1．JDBC ドライバを lib フォルダーに格納し、ビルド・パスを通す
2．JDBC ドライバクラスを JVM に登録する
3．接続先 URL、ユーザー、パスワードを指定し、DB 接続する

また、DB への接続をした場合、必ず行う必要があるのが以下の 2 つの処理です。

1．例外処理（SQLException）
2．DB との接続を切断する

3　JDBC ドライバを lib フォルダーに格納し、ビルド・パスを通す

Java から DB に接続する際に用いる JDBC ドライバ「ojdbc6.jar」ファイルは、プロジェクト直下の lib フォルダーに格納します。その後、ビルド・パス上に jar ファイルを追加しましょう。

1．プロジェクト直下に lib フォルダーを作成
[プロジェクト上で右クリック] > [新規] > [フォルダー] > [フォルダー名に lib と入力]
2．lib フォルダーに ojdbc6.jar ファイルを格納
3．ビルド・パスを通す
[プロジェクト上で右クリック] > [プロパティ] > [Java のビルド・パス] > [ライブラリー] > [Jar 追加] > [lib フォルダーに格納した ojdbc6.jar ファイルを選択]

4　JDBC ドライバクラスを JVM に登録する

JDBC ドライバクラスを JVM に登録する際は、Class#forName()メソッドを用います。このメソッドは、引数で指定した文字列のクラスを JVM に登録するための static メソッドです。ここで、引数に指定するドライバクラス名は、JDBC ドライバごとに異なるものになります。Oracle の ojdbc6 のドライバクラス名は「oracle.jdbc.driver.OracleDriver」です。

JVM への登録は、一度実行すれば基本的に JVM が終了するまでの間、再登録の必要はありません。

5　DB へ接続する

DB へ接続する際は、DriverManager#getConnection()メソッドを用います。第 1 引数に接続先情報を示した URL（ホスト名、ポート番号、データベース名）、第 2 引数に DB に接続するためのユーザー名、第 3 引数にパスワードを指定します。また、このメソッドを実行すると戻り値として Connection オブジェクトが返ってきます。

DB 操作をする際は、ここで取得した Connection オブジェクトを使用していくことになります。

6　例外処理（SQLException）

JDBC では、多くのメソッドがエラー発生時に SQLException をスローします。この例外クラスからは、getErrorCode()メソッドで DB のエラーコード、getSQLState()メソッドで SQL ステートコードが取得出来ます。

7　接続を切断する

DB 接続の切断は、Connection#close()によって行います。接続の終了処理は必ず行なう必要があるため、finally ブロックの中で close()メソッドを実行します。

切断処理をしていない場合、システムが停止するなど、致命的なトラブルに繋がる場合がありますので、十分気を付けてください。

8　接続サンプルを確認する

それでは、DB 接続のサンプルを確認しましょう。以下のサンプルは、DB 接続を行う DBManager クラスと実行クラスの Sample01 クラスで構成されています。

jp.co.sss.sample.jdbc.chapter01/DBManager.java

```
package jp.co.sss.sample.jdbc.chapter01;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBManager {


    /**
     * DBと接続する
     *
     * @return DBコネクション
     * @throws ClassNotFoundException
     * @throws SQLException
     */
    public Connection getConn() throws ClassNotFoundException, SQLException {
        // JDBCドライバクラスをJVMに登録
        Class.forName("oracle.jdbc.driver.OracleDriver");
        // DBへ接続
        Connection conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:XE",
                "jdbcuser", "systemsss");
        System.out.println("DBに接続しました");
        return conn;
    }

    /**
     * DB接続を切断する
     *
     * @param conn
     *              DBコネクション
     */
    public void close(Connection conn) {
```

```
        try {
            // 切断処理
            if (conn != null) {
                conn.close();
                System.out.println("切断しました");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

jp.co.sss.sample.jdbc.chapter01/Sample01.java

```
package jp.co.sss.sample.jdbc.chapter01;

import java.sql.Connection;
import java.sql.SQLException;

public class Sample01 {

    public static void main(String[] args) {
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        Connection conn = null;
        try {
            // 接続する
            conn = manager.getConn();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // 切断処理
            manager.close(conn);
        }
    }
}
```

3. Statement による検索

1 Statement クラス

JDBC を用いて DB に対して、SQL 文を発行して検索を行う方法はいくつかあります。その最も単純な方法が Statement クラスによる検索です。Statement クラスのオブジェクトは Connection クラスの createStatement()メソッドで生成します。また、検索用の SQL 文の発行は、Statement クラスの executeQuery()メソッドで行います。このメソッドは引数に発行したい SQL 文を文字列として指定します。

executeQuery()メソッドを実行すると検索結果は ResultSet クラスのオブジェクトとして戻ってきます。

実装例

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM employee");
```

2 ResultSet クラス

ResultSet クラスは表形式のデータとなっており、カーソルを利用しています。カーソルとは、現在、処理しているレコードを指します。

カーソルの移動は、ResultSet クラスの next()メソッドで行います。このメソッドを実行すると、次のレコードへカーソルを移動させることが出来ます。また、このメソッドは、boolean 型の戻り値を返します。次のレコードが存在する場合は true を、存在しない場合は false を返します。

カーソル位置のレコードの各列の値を取得するには、ResultSet クラスの getString()メソッドを用います。このメソッドは、引数に渡した列名に対応した値を取得することが出来ます。

ResultSet クラスを利用する上で注意しなければならないのが、カーソルの初期位置は 1 レコード目では無く、0 レコード目になことです。従って、レコードを読み込む際は、最低 1 回は next()メソッドを実行する必要があります。

実装例

```
while (rs.next()) {
    System.out.println(rs.getString("emp_id"));
    System.out.println(rs.getString("emp_name"));
}
```

3　検索機能を実装

それでは、Statement クラスによる検索機能を確認しましょう。以下のサンプルは、DBManager クラスと検索を実施する Sample02 クラスで成り立っています。また、今回使用するテーブル情報も同時に確認しましょう。

作成するユーザー：jdbcuser
パスワード：systemsss
データベース名：xe（Oracle Database 11g Express Edition のデフォルト）

employee テーブル

No	論理名称	物理名称	データ型	桁数	PK	FK	制約	備考
1	社員 ID	emp_id	数値	NUMBER	◎			
2	パスワード	emp_pass	文字列	VARCHAR2(10)			not null	
3	社員名	emp_name	文字列	VARCHAR2(20)			not null	
4	性別	gender	数値	NUMBER(1)			not null	"1"：男 "2"：女
5	住所	address	文字列	VARCHAR(30)				
6	誕生日	birthday	日付	DATE				
7	部署 ID	post_id	数値	NUMBER(2)		◎	not null	

post テーブル

No	論理名称	物理名称	データ型	桁数	PK	FK	制約	備考
1	役職 ID	post_id	数値型	NUMBER	◎			
2	役職名	post_name	文字型	VARCHAR2(20)			not null	"1"：社長 "2"：部長 "3"：課長 "4"：一般

SQL 文

```
CREATE USER jdbcuser IDENTIFIED BY systemsss;
GRANT ALL PRIVILEGES TO jdbcuser;

CONNECT jdbcuser/systemsss

CREATE TABLE post (
  post_id NUMBER PRIMARY KEY,
  post_name VARCHAR2(20) NOT NULL
);

CREATE TABLE employee (
  emp_id NUMBER PRIMARY KEY,
  emp_pass VARCHAR2(10) NOT NULL,
  emp_name VARCHAR2(20) NOT NULL,
  gender NUMBER(1) NOT NULL,
  address VARCHAR(30),
  birthday DATE,
  post_id NUMBER(2) NOT NULL,
  FOREIGN KEY ( post_id ) REFERENCES post ( post_id )
);

INSERT INTO post values (1,'社長');
INSERT INTO post values (2,'部長');
INSERT INTO post values (3,'課長');
INSERT INTO post values (4,'一般');

INSERT INTO employee values (1,'1', 'system shared', 1, '千葉県', '2009/09/09', 1);
INSERT INTO employee values (2,'1', 'systemsss', 1, '松戸市', '2009/09/10', 2);
INSERT INTO employee values (3,'1', 'administrator', 2, '東京都', '2009/09/10', 3);
INSERT INTO employee values (4,'1', 'edudbuser', 2, '埼玉県', '2009/09/14', 4);

COMMIT;

SELECT * FROM post;
SELECT * FROM employee;
```

jp.co.sss.sample.jdbc.chapter02/Sample02.java

```
package jp.co.sss.sample.jdbc.chapter02;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample02 {

    public static void main(String[] args) throws SQLException {
        Connection conn = null;
        Statement st = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            st = conn.createStatement();
            String sql = "SELECT * FROM employee";
            rs = st.executeQuery(sql);
            System.out.println(
                "emp_id¥temp_pass¥temp_name¥tgender¥taddress¥tbirthday");
            while (rs.next()) {
                System.out.print(rs.getString("emp_id") + "¥t");
                System.out.print(rs.getString("emp_pass") + "¥t");
                System.out.print(rs.getString("emp_name") + "¥t");
                System.out.print(rs.getString("gender") + "¥t");
                System.out.print(rs.getString("address") + "¥t");
                System.out.println(rs.getString("birthday"));
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
```

```
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // Statementをクローズ
            if (st != null) {
                st.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```

4. PreparedStatement による検索

1 PreparedStatement クラス

JDBC を利用した最も一般的な SQL 文の発行方法が PreparedStatement クラスによる実装となります。PreparedStatement とは、SQL 文の一部を変数として定義した、SQL 文のひな型のようなものです。

PreparedStatement クラスのオブジェクトは Connection クラスの prepareStatement ()メソッドで生成します。また、検索用の SQL 文の発行は、PreparedStatement クラスの executeQuery()メソッドで行います。このメソッドは引数に発行したい SQL 文を文字列として指定します。

executeQuery()メソッドを実行すると検索結果は ResultSet クラスのオブジェクトとして戻ってきます。

また、PreparedStatement クラスによる SQL 文の発行の際は、プレースホルダを用いて、SQL 構文の動的に変化する箇所を「?」で記述することが出来ます。プレースホルダに値をバインドする際は、PreparedStatement クラスの setString()メソッド等を利用します。第 1 引数には位置、第 2 引数には値を指定します。

実装例

```
String sql = "SELECT * FROM employee WHERE emp_id = ?";
PrepareStatement ps = conn.prepareStatement(sql);
ps.setString(1,  "0001");
rs = ps. executeQuery();
```

2 検索機能を実装

それでは、PreparedStatement クラスによる検索機能を確認しましょう。以下のサンプルは、DBManager クラスと検索を実施する Sample03 クラスで成り立っています。

jp.co.sss.sample.jdbc.chapter03/Sample03.java

```
package jp.co.sss.sample.jdbc.chapter03;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample03 {

    public static void main(String[] args) throws SQLException, IOException {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            // SQL構文を構築
            String sql = "SELECT * FROM employee WHERE emp_name LIKE ?";
            ps = conn.prepareStatement(sql);
            // 検索値の入力
            System.out.println("社員名を入力してください。");
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    System.in));
            // 入力値をバインド
            String empName = br.readLine();
            ps.setString(1, "%" + empName + "%");
            rs = ps.executeQuery();
            System.out.println(
                "emp_id¥temp_pass¥temp_name¥tgender¥taddress¥tbirthday");
            while (rs.next()) {
```

```
                System.out.print(rs.getString("emp_id") + "¥t");
                System.out.print(rs.getString("emp_pass") + "¥t");
                System.out.print(rs.getString("emp_name") + "¥t");
                System.out.print(rs.getString("gender") + "¥t");
                System.out.print(rs.getString("address") + "¥t");
                System.out.println(rs.getString("birthday"));
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // PreparedStatementをクローズ
            if (ps != null) {
                ps.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```

5. SQL インジェクション

1　SQL インジェクションとは

DB プログラミングでは、SQL 文のコーディング方法によっては SQL インジェクション脆弱性と呼ばれるセキュリティホールを発生させてしまう危険があります。

SQL インジェクション（SQL Injection）とは、Web アプリケーションが利用している DB に対し、想定していない SQL コマンドを不正に実行する攻撃手法です。

2　SQL インジェクションを体験してみる

例えば以下のような SQL 文を見てください。Statement クラスを利用した一般的な検索のように見えます。

```
Statement st = conn.createStatement();
String sql = "SELECT * FROM employee WHERE emp_id = '";
String empId   = br.readLine();
sql = sql + empId   + "'";
rs = st.executeQuery(sql);
```

しかしユーザーが、「' OR 1 = 1 -- 」を入力すると、「SELECT * FROM employee WHERE emp_id = '' OR 1 = 1 -- '」という SQL 文が最終的に出来上がります。

この SQL 文では、「1 = 1」は必ず成り立ってしまうため、正しい emp_id の値を知らなくても情報を抜き出すことが出来てしまいます。

以下のサンプルで動作を確認してみましょう。

jp.co.sss.sample.jdbc.chapter04/Sample0401.java

```
package jp.co.sss.sample.jdbc.chapter04;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample0401 {

    public static void main(String[] args) throws SQLException, IOException {
        Connection conn = null;
        Statement st = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            st = conn.createStatement();
            String sql = "SELECT * FROM employee WHERE emp_id = '";
            // 検索値の入力
            System.out.println("社員IDを入力してください。");
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    System.in));
            // 検索の実行
            String empId  = br.readLine();
            sql = sql + empId  + "'";
            rs = st.executeQuery(sql);
            System.out.println(
                "emp_id¥temp_pass¥temp_name¥tgender¥taddress¥tbirthday");
            while (rs.next()) {
                System.out.print(rs.getString("emp_id") + "¥t");
```

```
                System.out.print(rs.getString("emp_pass") + "¥t");
                System.out.print(rs.getString("emp_name") + "¥t");
                System.out.print(rs.getString("gender") + "¥t");
                System.out.print(rs.getString("address") + "¥t");
                System.out.println(rs.getString("birthday"));
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // Statementをクローズ
            if (st != null) {
                st.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```

3　SQL インジェクション対策

SQLインジェクションの対策として最も代表的なものがPreparedStatementクラスの利用です。PreparedStatement を利用すると、ユーザーからの入力部分はバインド変数となるため、SQL 文の構文を確定させることが出来ます。

ユーザーが入力した文字は全て単純な文字列としてバインド変数を置き換えるため、どのような文字列を指定しても構文の形を崩すことは出来ません。

以下のサンプルで動作を確認してみましょう。

jp.co.sss.sample.jdbc.chapter04/Sample0402.java

```
package jp.co.sss.sample.jdbc.chapter04;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample0402 {

    public static void main(String[] args) throws SQLException, IOException {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            // SQL構文を構築
            String sql = "SELECT * FROM employee WHERE emp_id = ?";
            ps = conn.prepareStatement(sql);
            // 検索値の入力
            System.out.println("社員IDを入力してください。");
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    System.in));
            // 入力値をバインド
            String empId = br.readLine();
            ps.setString(1, empId);
            rs = ps.executeQuery();
            System.out.println(
                "emp_id¥temp_pass¥temp_name¥tgender¥taddress¥tbirthday");
            while (rs.next()) {
```

```
                System.out.print(rs.getString("emp_id") + "¥t");
                System.out.print(rs.getString("emp_pass") + "¥t");
                System.out.print(rs.getString("emp_name") + "¥t");
                System.out.print(rs.getString("gender") + "¥t");
                System.out.print(rs.getString("address") + "¥t");
                System.out.println(rs.getString("birthday"));
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // PreparedStatementをクローズ
            if (ps != null) {
                ps.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```

6. 登録、更新、削除

1　登録、更新、削除方法

Statement クラスに利用して登録、更新、削除を行う際は、executeUpdate()メソッドを使用します。引数には検索の時と同様に発行したい SQL 文を渡します。戻り値は処理件数となります。

PreparedStatement クラスを利用して登録、更新、削除を行う際も、Statement クラス同様に executeUpdate()メソッドを使用します。戻り値も Statement クラス同様に処理件数となります。また、PreparedStatement クラスを用いた場合はプレースホルダも使用することが出来ます。

PreparedStatement クラスを用いた実装例

```
String sql = "INSERT INTO customer VALUES (?, ?, ?, ?)";
ps = conn.prepareStatement(sql);
ps.setString(1, "AAA");
ps.setString(2, "太郎");
ps.setString(3, "東京都");
ps.setString(4, 32);
int cnt = ps.executeUpdate();
```

2　登録、更新、削除機能を実装

それでは、PreparedStatement クラスによる登録、更新、削除機能を確認しましょう。以下のサンプルは、DBManager クラスと登録を実施する Sample0501 クラス、更新を実施する Sample0502 クラス、削除を実施する Sample0503 クラスで成り立っています。

jp.co.sss.sample.jdbc.chapter05/Sample0501.java

```
package jp.co.sss.sample.jdbc.chapter05;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample0501 {

    public static void main(String[] args) throws SQLException, IOException {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            String sql = "INSERT INTO post VALUES (?, ?)";
            ps = conn.prepareStatement(sql);
            System.out.println("役職の新規登録を行います。");
            BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));
            System.out.println("役職IDを入力してください。");
            String postId = br.readLine();
            System.out.println("役職名を入力してください。");
            String postName = br.readLine();
            ps.setString(1, postId);
            ps.setString(2, postName);
            int cnt = ps.executeUpdate();
            System.out.println(cnt + "件のデータを登録しました。");
            ps = conn.prepareStatement("SELECT * FROM post");
```

```
            rs = ps.executeQuery();
            System.out.println(
                "post_id¥tpost_name");
            while (rs.next()) {
                System.out.print(rs.getString("post_id") + "¥t");
                System.out.println(rs.getString("post_name") + "¥t");
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // PreparedStatementをクローズ
            if (ps != null) {
                ps.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```

jp.co.sss.sample.jdbc.chapter05/Sample0502.java

```
package jp.co.sss.sample.jdbc.chapter05;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample0502 {

    public static void main(String[] args) throws SQLException, IOException {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            String sql = "UPDATE post SET post_name = ? WHERE post_id = ?";
            ps = conn.prepareStatement(sql);
            System.out.println("役職の更新を行います。");
            BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));
            System.out.println("役職IDを入力してください。");
            String postId = br.readLine();
            System.out.println("役職名を入力してください。");
            String postName = br.readLine();
            ps.setString(2, postId);
            ps.setString(1, postName);
            int cnt = ps.executeUpdate();
            System.out.println(cnt + "件のデータを更新しました。");
            ps = conn.prepareStatement("SELECT * FROM post");
```

```
            rs = ps.executeQuery();
            System.out.println(
                "post_id¥tpost_name");
            while (rs.next()) {
                System.out.print(rs.getString("post_id") + "¥t");
                System.out.println(rs.getString("post_name") + "¥t");
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // PreparedStatementをクローズ
            if (ps != null) {
                ps.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```

jp.co.sss.sample.jdbc.chapter05/Sample0503.java

```
package jp.co.sss.sample.jdbc.chapter05;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample0503 {

    public static void main(String[] args) throws SQLException, IOException {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            String sql = "DELETE FROM post WHERE post_id = ?";
            ps = conn.prepareStatement(sql);
            System.out.println("役職の削除を行います。");
            BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));
            System.out.println("役職IDを入力してください。");
            String postId = br.readLine();
            ps.setString(1, postId);
            int cnt = ps.executeUpdate();
            System.out.println(cnt + "件のデータを削除しました。");
            ps = conn.prepareStatement("SELECT * FROM post");
            rs = ps.executeQuery();
            System.out.println(
                "post_id¥tpost_name");
```

```
            while (rs.next()) {
                System.out.print(rs.getString("post_id") + "¥t");
                System.out.println(rs.getString("post_name") + "¥t");
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("Oracleエラーコード:" + e.getErrorCode());
            System.err.println("SQLStateコード:" + e.getSQLState());
            System.err.println("エラーメッセージ:" + e.getMessage());
            e.printStackTrace();
        } finally {
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // PreparedStatementをクローズ
            if (ps != null) {
                ps.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```

7. トランザクション管理

1　トランザクションとは

　トランザクションとは、一連のデータ処理を 1 つの単位として管理することを指します。簡単に言うと、複数のテーブルに処理をする際に、その一連の流れの中で何かエラーが起きたら、全ての処理を実行前に戻すといったことです。
　この章では、JDBC を用いたトランザクション処理の方法を学んでいきます。

2　ACID 特性

　ACID 特性とは、トランザクション処理に求められる特性です。トランザクション処理を実装する際は、この ACID 特性を意識する必要があります。
　ACID 特性には以下の 4 つがあります。

原子性(Atomicity)
　それ以上は切り離すことが出来ない作業単位であることを保証します。トランザクションは、処理が完全に完了するか、あるいは一切実行されないかのいずれかである必要があります。

一貫性(Consistency)
　トランザクション完了時に、データの整合性が維持されていることを保証します。

独立性(Isolation)
　同時に実行されている複数のトランザクションは、互いに影響を及ぼしてはなりません。他のトランザクションから完全に隔離されている必要があります。

耐久性(Durability)
　システム障害の有無に関わらず、一度コミットされたトランザクションは保存されなければいけません。

3　JDBC を利用したトランザクション

JDBC は自動コミットモードにてトランザクションを管理しています。これは、boolean 型の値となっていて、true が設定されている場合は、処理毎にデータをコミットします。

また、この自動コミットモードのデフォルト値は true のため、トランザクションを利用する際は、false を指定する必要があります。

自動コミットモードは、Connection クラスの setAutoCommit()メソッドで制御します。

トランザクション開始の実装例

```
conn. setAutoCommit(false);
```

トランザクションを開始した場合は、明示的にコミットをする必要があります。コミット処理は、Connection クラスの commit()メソッドで行います。

コミットの実装例

```
conn. commit();
```

トランザクション中に何かエラーが起きた際は、データを処理前に戻す必要があります。この処理をロールバックといいます。ロールバックは、Connection クラスの rollback ()メソッドで行います。

ロールバックの実装例

```
conn. rollback ();
```

4　トランザクション処理を実装

それでは、トランザクション処理を実装したサンプルを確認しましょう。以下のサンプルは、DBManager クラスと post テーブルに 2 件続けて登録を実施する Sample07 クラスで構成されています。

jp.co.sss.sample.jdbc.chapter06/Sample06.java

```
package jp.co.sss.sample.jdbc.chapter06;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import jp.co.sss.sample.jdbc.chapter01.DBManager;

public class Sample06 {

    public static void main(String[] args) throws SQLException, IOException {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        // DBManagerのインスタンスを生成
        DBManager manager = new DBManager();
        try {
            // 接続する
            conn = manager.getConn();
            // トランザクション開始
            conn.setAutoCommit(false);
            String sql = "INSERT INTO post VALUES (?, ?)";
            for (int i = 0; i < 2; i++) {
                ps = conn.prepareStatement(sql);
                System.out.println("役職の新規登録を行います。");
                BufferedReader br = new BufferedReader(new InputStreamReader(
                        System.in));
                System.out.println("役職IDを入力してください。");
                String postId = br.readLine();
                System.out.println("役職名を入力してください。");
                String postName = br.readLine();
                ps.setString(1, postId);
                ps.setString(2, postName);
```

```
                int cnt = ps.executeUpdate();
                System.out.println(cnt + "件のデータを登録しました。");
            }
            // コミット
            conn.commit();
        } catch (Exception e) {
            // ロールバック
            conn.rollback();
            System.out.println("エラーが発生したため、データをロールバックしました。");
        } finally {
            // postテーブルの内容を表示
            ps = conn.prepareStatement("SELECT * FROM post");
            rs = ps.executeQuery();
            System.out.println("post_id¥tpost_name");
            while (rs.next()) {
                System.out.print(rs.getString("post_id") + "¥t");
                System.out.println(rs.getString("post_name") + "¥t");
            }
            // ResultSetをクローズ
            if (rs != null) {
                rs.close();
            }
            // PreparedStatementをクローズ
            if (ps != null) {
                ps.close();
            }
            // 切断する
            manager.close(conn);
        }
    }
}
```