JDBC

How to connect MySQL using JDBC

1. JDK를 인스톨한다. 이미 JDK 가 인스톨되어 있으면 이 단계는 생략: Oracle JDK, IBM JDK
2. Eclipse (Indigo 또는 그 후 버전) 사용.
3. JDK 와 Eclipse 가 동일 비트 레블(bit level)을 사용하여야 함 (JDK 가 32bit(64bit)이면 Eclipse 도 32bit (64bit)를 사용)
4. Eclipse 에서 아래의 샘플 코드를 사용하여 새로운 자바 프로젝트를 만든다.
5. 첨부파일에서 MySQL JDBC Driver(mysql-connector-java-8.0.15.jar)를 적절한 위치에 다운로드한다.
6. Eclipse 에서 Package Explorer 탭 프로젝트 이름에서 우클릭하여 Build Path -> Add External Archives ... 을 클릭하고, 드라이버를 저장한 폴더를 찾아서 mysql-connector-java-8.0.15-bin.jar 를 선택하여 열기를 클릭한다.(New Project할 때 마다 실행한다)
7. 샘플 프로그램을 실행하여 에러 메시지 없이 EMPLOYEE 테이블의 lname 이 출력됨을 확인한다.

Sample Code

```
import java.sql.*;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;

public class DBConnectionTest
{
    public static void main (String args[])
    {
        try
        {
            Class.forName ("com.mysql.cj.jdbc.Driver");
            String host = "ksohndb.c8hr4dzksmx8.us-east-2.rds.amazonaws.com:330
6/";
            String db= "companydb";
            String user = "admin";
            String password = getPassword();
            Connection con = DriverManager.getConnection("jdbc:mysql://" + host
+ db + "?useSSL=false", user, password);
            Statement stmt = con.createStatement();
            ResultSet rset = stmt.executeQuery("select lname from EMPLOYEE");
            while (rset.next())
                 System.out.println(rset.getString(1));

            rset.close();
            stmt.close();
            con.close();
        }
        catch (SQLException ex)
        {
            System.out.println("SQLException" + ex);
        }
        catch (Exception ex)
        {
            System.out.println("Exception:" + ex);
        }
    }
```

java.sql.Connection

- 데이터베이스에 접근하기 위한 객체를 생성하는 인터페이스

메소드 이름	설명
createStatement()	SQL 질의들과 갱신을 수행하기 위해 사용되어지는 Statement 객체를 생성한다.
prepareStatement()	미리 컴파일 된 구문을 포함하고 있는 PreparedStatement 객체를 생성한다.
close()	현재 연결을 즉시 닫음
setAutoCommit()	해당 연결에 대한 커밋 모드를 설정한다. 만일 자동 커밋이 true 이면 모든 구문은 실행이 완료되자 마자 커밋된다.
getAutoCommit()	해당 연결에 대한 자동 커밋 모드를 얻어온다.
commit()	마지막 커밋 이후에 제출된 모든 구문을 커밋한다.
rollback()	마지막 커밋 이후에 제출된 모든 구문의 영향을 무효화한다.

java.sql.Statement

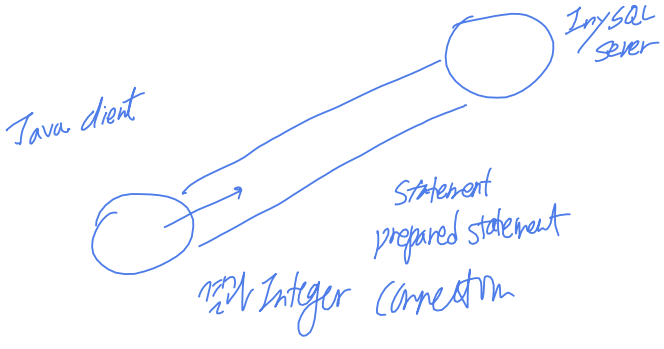
- query문을 실행하고 그것에 대한 결과 값을 가져오기 위해서 사용되는 인터페이스

메소드 이름	설명
execute()	주어진 SQL 문장을 실행시킨다.
executeQuery()	주어진 SQL 문장을 수행하고 질의 결과로 ResultSet을 반환한다.
executeUpdate()	SQL INSERT , UPDATE 또는 DELETE 문장을 실행한다. 영향 받은 레코드들의 수를 반환한다.
close()	연결을 즉시 끊는다.

java.sql.PreparedStatement

- 동일한 query문이 여러 번 반복적으로 수행될 때 주로 사용되는 인터페이스

메소드 이름	설명
setXXX(int n, XXX x)	XXX 는 int, double, String, Date 같은 자료형을 의미하며, x를 n번째 전달인자 값으로 설정한다.
clearParameters()	준비된 구문에서 현재의 모든 전달인자를 해제한다.
executeQuery()	준비된 SQL 질의를 실행하고 ResultSet 객체를 반환한다.
executeUpdate()	PreparedStatement 객체에 의해서 표현되는 준비된 SQL INSERT, UPDATE 또는 DELETE구문을 실행한다.



java.sql.ResultSet

- Statement 인터페이스의 executeQuery() 메서드가 반환하는 결과로 얻은 데이터를 저장

메소드 이름	설명
next()	불러온 레코드에서 다음 값의 유무를 반환한다.
getString()	받아온 레코드 컬럼의 데이터를 String형으로 반환한다.
getBoolean()	받아온 레코드 컬럼의 데이터를 boolean형으로 반환한다.
getInt()	받아온 레코드 컬럼의 데이터를 int형으로 반환한다.
getFloat()	받아온 레코드 컬럼의 데이터를 float형으로 반환한다.

Closing DB Connection

Java는 메모리 관리를 따로 하지 않아도 자체적으로 garbage collection 기능이 있어 사용되지 않는 객체는 자동적으로 처리된다. 그러나 자원을 효율적으로 관리라기 위해서는 사용하지 않은 자원은 close하여야 한다. JDBC API는 이를 위해 close() 메소드를 제공하는데, close() 메소드를 사용해야 할 클래스는 Connection, Statement, PreparedStatement, ResultSet 등이다. 따라서, 데이터베이스에 접속하여 SQL 구문을 모 두 수행했다면 다음처럼 close() 메소드를 사용하여 객체를 모두 소멸 시키도록 한다.

- rs.close(); // 결과값 객체 소멸
- stmt.close(); // SQL문 객체 소멸
- conn.close(); // 연결 객체 소멸

Connection은 상당한 Overhead를 가져온다. 따라서 최적화된 상태 를 유지하기 위해서는 반드시 Connection을 닫아 주어야 한다.

Example 1 : Print the last names and salary of an employee given SSN using Statement

```java
import java.sql.*;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import java.io.*;

public class JDBCExample1 {
    public static void main (String args [])
    throws SQLException, IOException {
        try
        {
            // Connect to the database
            Class.forName ("com.mysql.cj.jdbc.Driver");
            String host = "ksohndb.c8hr4dzksmx8.us-east-2.rds.amazonaws.com:330
6/";
            String db= "companydb";
            String user = "admin";
            String password = getPassword();
            Connection con = DriverManager.getConnection("jdbc:mysql://" + host
+ db + "?useSSL=false", user, password);
            // Perform query using Statement
            // by providing SSN at run time
            String ssn = readEntry("Enter a Social Security Number: ");
            Statement stmt = con.createStatement();
            ResultSet rset = stmt.executeQuery("select LNAME,SALARY from EMPLOYE
E where SSN = " + ssn);

            // Process the ResultSet
            if (rset.next ()) {
                String lname = rset.getString(1);
                double salary = rset.getDouble(2);
                System.out.println(lname + "'s salary is $" + salary);
            }
            else {
                System.out.println("No Employees whose ssn is " + ssn);
            }

            // Close objects
            rset.close();
            stmt.close();
            con.close();
        }
        catch (SQLException ex)
        {
            System.out.println("SQLException" + ex);
        }
        catch (Exception ex)
        {
            System.out.println("Exception:" + ex);
        }
    }
```

```
    private static String getPassword() {
        final String password, message = "Enter password";
        if(System.console() == null)
        {
              final JPasswordField pf = new JPasswordField();
              password = JOptionPane.showConfirmDialog(null, pf, message,
                      JOptionPane.OK_CANCEL_OPTION,
                      JOptionPane.QUESTION_MESSAGE ) == JOptionPane.OK_OPTION ?
                              new String(pf.getPassword()) : "";
        }
        else
            password = new String(System.console().readPassword("%s> ", message
));

        return password;
    }

    // ReadEntry function -- to read input string
    private static String readEntry(String prompt) {
        try {
            StringBuffer buffer = new StringBuffer();
            System.out.print(prompt);
            System.out.flush();
            int c = System.in.read();
            while (c != '\n' && c != -1) {
            buffer.append((char)c);
            c = System.in.read();
            }
            return buffer.toString().trim();
        } catch (IOException e) {
            return "";
        }
    }
}
```

Example 2 : Print the last names and salary of an employee given SSN using PreparedStatement

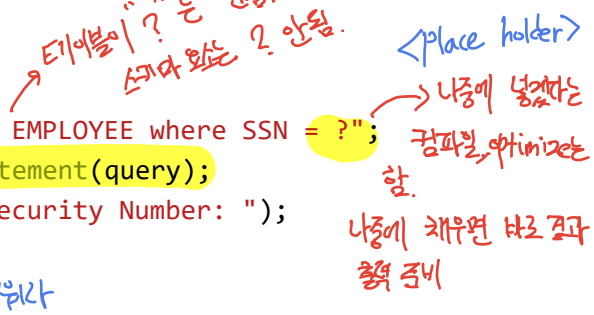
```java
import java.sql.*;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import java.io.*;

public class JDBCExample2 {
    public static void main (String args [])
    throws SQLException, IOException {
        try
        {
            // Connect to the database
            Class.forName ("com.mysql.cj.jdbc.Driver");
            String host = "ksohndb.c8hr4dzksmx8.us-east-2.rds.amazonaws.com:330
6/";
            String db= "companydb";
            String user = "admin";
            String password = getPassword();
            Connection con = DriverManager.getConnection("jdbc:mysql://" + host
+ db + "?useSSL=false", user, password);

            // Perform query using PreparedStatement
            // by providing SSN at run time
            String query = "select LNAME,SALARY from EMPLOYEE where SSN = ?";
            PreparedStatement pstmt = con.prepareStatement(query);
            String ssn = readEntry("Enter a Social Security Number: ");
            pstmt.clearParameters();
            pstmt.setString(1,ssn);
            ResultSet rset = pstmt.executeQuery();

            // Process the ResultSet
            if (rset.next ()) {
                String lname = rset.getString(1);
                double salary = rset.getDouble(2);
                System.out.println(lname + "'s salary is $" + salary);
            }
            else {
                System.out.println("No Employees whose ssn is " + ssn);
            }

            // Close objects
            rset.close();
            pstmt.close();
            con.close();
        }
        catch (SQLException ex)
        {
            System.out.println("SQLException" + ex);
        }
        catch (Exception ex)
        {
```



```
            System.out.println("Exception:" + ex);
        }
    }

...
}
```

Example 3 : Print the last names and salary of an employee given DNUMBER using PreparedStatement

- 하나의 SQL 문이 동일 Session에서 DB Server에서 반복적으로 수행될 때는 PreparedStatement문이 훨씬 더 효율적이다. Parsing, Compiling과 Optimization Overhead를 피할 수 있다.
- Oracle Library Cache는 Statement를 사용해도 효과적으로 실행되게 해 주는 기술이다. 자주 사용되는 SQL문에 대해 Oracle DBMS가 알아서 실행계획을 캐쉬한다.
- 입력값을 검증함으로서 SQL Injection을 피할 수 있다.

데이터베이스 해킹 사용자로부터 받는 값 (스캐너X)

```
import java.sql.*;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import java.io.*;

public class JDBCExample3 {
    public static void main (String args [])
    throws SQLException, IOException {
        try
        {
            // Connect to the database
            Class.forName ("com.mysql.cj.jdbc.Driver");
            String host = "ksohndb.c8hr4dzksmx8.us-east-2.rds.amazonaws.com:330
6/";
            String db= "companydb";
            String user = "admin";
            String password = getPassword();
            Connection con = DriverManager.getConnection("jdbc:mysql://" + host
+ db + "?useSSL=false", user, password);

            String query = "select LNAME,SALARY from EMPLOYEE where DNO = ?";
            PreparedStatement pstmt = con.prepareStatement (query);
            int dno = Integer.parseInt(readEntry("Enter a DEPARTMENT Number: "
));
            pstmt.clearParameters();
            pstmt.setInt(1, dno);
            ResultSet rset = pstmt.executeQuery();
            //Process ResultSet
            while (rset.next()) {
                String lname = rset.getString(1);
                double salary = rset.getDouble(2);
                System.out.println(lname + "'s salary is $" + salary);
            }

            // Close objects
            rset.close();
            pstmt.close();
            con.close();
        }
        catch (SQLException ex)
        {
            System.out.println("SQLException" + ex);
        }
        catch (Exception ex)
        {
            System.out.println("Exception:" + ex);
        }
    }
```

```
...
}


Enter a DEPARTMENT Number: 5
Smith's salary is $30000.0
Wong's salary is $40000.0
English's salary is $25000.0
Narayan's salary is $38000.0
```

Connection의 트랜잭션 관련 메소드

메소드 이름	설명
commit()/rollback()	commit()과 rollback()메소드는 Connection객체의 메소드이다. 하나의 Connection객체로부터 생성된 여러개의 Statement객체가 개별적으로 갱신문을 수행하더라도 Connection객체의 commit()이 호출되기 전에는 DB에 반영되지 않는다. rollback()은 Connection상에서 이루어진 Transaction들을 마지막 commit이나 rollback을 수행한 상태의 DB로 되돌린다. 여러 갱신문이 실행되더라도 rollback을 실행하면 다 무시된다. (단 setAutoCommit(false)인 경우에한정된다.)
setAutoCommit(boolean autoCommit)	기본적으로 Connection객체는 autoCommit Flag가 true로 지정되어 있다. 즉 매번 갱신문이 수행될 때마다 자동으로 commit()메소드를 수행한다. 이와 같이 Auto Commit을 true로 두고 Transaction을 처리한다면 어떤 한 Table에 Data를 성공적으로 INSERT한 다음, 연관된 다른 Table에 Update 작업을 수행하던 중 Error가 발생한다면 Data의 Integrity가 깨어질 수 있다. 그러므로 Transaction을 관련 있는 작업단위로 처리하고자 한다면 AutoCommit을 false로 하고 User가 판단하는 시점에 Transaction을 종료하도록 COMMIT이나 ROLLBACK을 하도록 한다.
setTrasactionIsolation	트랜잭션 고립(Isolation)레벨을 정한다

Example 4: Transaction Program 1

```java
import java.sql.*;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import java.io.*;

public class JDBCExample4 {
    public static void main (String args [])
    throws SQLException, IOException {
        try
        {
            // Connect to the database
            Class.forName ("com.mysql.cj.jdbc.Driver");
            String host = "ksohndb.c8hr4dzksmx8.us-east-2.rds.amazonaws.com:330
6/";
            String db= "companydb";
            String user = "admin";
            String password = getPassword();
            Connection con = DriverManager.getConnection("jdbc:mysql://" + host
+ db + "?useSSL=false", user, password);

            Statement stmt = con.createStatement();
            stmt.executeUpdate("drop table if exists ttt");
            stmt.executeUpdate("create table ttt(id int primary key) engine=Inno
DB");
            boolean success = false;
            con.setAutoCommit(false);
            // when transaction is executed normally, set success to true
            // if success is true, commit; otherwise rollback
            try {
                // transaction starts here
                stmt.executeUpdate("insert into ttt values(1)");
                stmt.executeUpdate("insert into ttt values(2)");
                stmt.executeUpdate("insert into ttt values(1)");
                stmt.executeUpdate("insert into ttt values(3)");
                success = true;
            } catch (Exception e) {
                System.out.println("Exception occurred. Transaction will be roll
backed");
                System.out.println("SQL State: " + ((SQLException)e).getSQLState
());
                System.out.println("SQL Error Message: " + e.getMessage());
            } finally {
                try {
                    if (success) {
                        con.commit();
                    } else {
                        con.rollback();
                    }
                } catch (SQLException sqle) {
                    sqle.printStackTrace();
```

테이블 생성 DDL

DML → Insert, update, delete 등에서 생길경우 exception 핸들링

exception이 발생하든 finally block으로 옴

commit할지 rollback할지

success가 false일경우 try에서 error가 발생함. 이경우 finally block에서 rollback 함.

네트워크 connection이 끊길경우 timeout되면 모두 rollback함.

```
                }
            }

            // Close objects
            con.close();
        }
        catch (SQLException ex)
        {
            System.out.println("SQLException" + ex);
        }
        catch (Exception ex)
        {
            System.out.println("Exception:" + ex);
        }
    }

...
}
```