Práctica 2: Utilización de JDBC para conectar una base de datos

Fecha: 26 julio 2017 Fecha de entrega 28 de julio 2017 (por cada día de retraso tendrá un valor de menos 2 puntos)

Valor 54 puntos (debes de documentar cada uno de los apartados e integrar los conceptos necesarios para el desarrollo de la practica)

**OBJETIVOS**

Los objetivos de la práctica son los siguientes:

* Integrar sentencias JDBC en programas Java (8 puntos)
* Aprender cómo conectarse a la base de datos usando el driver JDBC de Oracle (8 puntos)
* Realizar consultas SQL utilizando JDBC (8 puntos)
* Utilizar ResultSet para obtener los resultados (10 puntos)
* Saber cómo obtener cualquier tipo de resultado sin tener que conocer previamente el tipo de dato con ResultSetMetaData (10 puntos)
* Aprender a crear sentencias preparadas (10 puntos)

JDBC es un api que posibilita el acceso a datos almacenados en las tablas de una base de datos. Su principal ventaja es que permite acceder a los diferentes sistemas de bases de datos existentes (Oracle, Sybase, IBM DB2, ...) con la misma interfaz. Utilizando esta API podemos realizar todas las tareas sobre la base de datos para las que nuestro usuario tenga permiso (creación y borrado de tablas, inserción de datos, ...)

Puedes descargar la librería de JDBC que proporciona Oracle ojdbc14\_g.jar (Nota: puede variar la versión depende de la versión del gestor) desde la propia página de Oracle. Esta librería incluye el driver de Oracle thin. Es un driver de tipo 4 que se comunica directamente con el servidor de bases de datos utilizando el protocolo nativo de Oracle llamado Net 8 y que funciona sobre TCP.

Para el desarrollo de la práctica te será de utilidad consultar el API de java

**CONECTAR A LA BASE DE DATOS**

En JDBC existen dos formas de conectarnos a la base de datos:

* Utilizando la clase DriverManager
* Usando un objeto DataSource para realizar la conexión

**Conexiones con DriverManager**

Con DriverManager podemos crear una conexión a la base de datos utilizando un driver que debemos cargar previamente. Un driver es una clase que implementa una interfaz para acceder a la base de datos. Para crear una conexión tenemos que seguir los siguientes pasos:

// Especifica cómo nos conectaremos a la base de datos

String dbURL = "jdbc:oracle:thin:BASEDATOS@NOMBRE\_DE\_MAQUINA:PUERTO:CADENA\_DE\_CONEXION"

// Carga del driver que usaremos (en este caso, Oracle thin está en esta clase)

Class.forName("oracle.jdbc.driver.OracleDriver");

// Obtención de una conexión

Connection conn = DriverManager.getConnection

(URL, "USUARIO", "PASSWORD");

**Conexiones con DataSource**

Utilizar conexiones con DataSource en lugar de con DriverManager es preferible ya que nos proporciona algunas características adicionales más avanzadas, como la creación de pools de conexiones o transacciones distribuidas con poca modificación del código. Para crear una conexión con DataSource:

// Creación de un objeto DataSource

oracle.jdbc.pool.OracleDataSource ds = new oracle.jdbc.pool.OracleDataSource();

// Es la misma URL que creamos en DriverManager

ds.setURL(URL);

// Obtención de una conexión

Connection conn = ds.getConnection(USUARIO, PASSWORD);

**Ejercicios**

Crea una clase que al ejecutar cree estos dos tipos de conexiones e imprima un mensaje de texto indicando si ha podido conectar o no a la base de datos

**EJECUTAR SENTENCIAS SQL**

En JDBC, las sentencias SQL se ejecutan usando un objeto de la clase Statement asociado a una conexión. Este objeto podemos crearlo a partir de una conexión y luego reutilizarlo para ejecutar todas las sentencias SQL que queramos.

Según el tipo de operación que realicemos, podemos ejecutar dos tipos de sentencias SQL:

Consulta - executeQuery() Es una operación SELECT que devuelve el resultado de una consulta encapsulado en un objeto de tipo ResultSet

Actualización - executeUpdate() Es una operación INSERT, UPDATE o DELETE que realiza una modificación sobre la base de datos; su resultado es un valor entero indicando el número de tuplas afectadas por esta operación.

El siguiente código realiza una consulta sobre la tabla Empleado que creamos en la práctica anterior y muestra su contenido por pantalla:

import java.sql.\*;

public class Consulta {

protected String consulta;

protected String[] atributos;

protected Connection con;

protected ResultSet resultado;

public Consulta() {

// Aqui debemos obtener la conexión a la base de datos

...

}

public void consulta(){

try{

Statement sentencia= con.createStatement();

resultado= sentencia.executeQuery(consulta);

System.out.println("¡Consulta hecha!");

}

catch (SQLException sqle) {

System.err.println(sqle);

}

} //consulta

public void imprimeResultado(){

//Recorramos el ResultSet e imprimamos el resultado

...

}

ResultSet getResultSet() {

return resultado;

}//getResultSet()

public static void main(String [] args) {

String consulta="SELECT \* FROM EMPLEADO";

String[] atributos= {"NOMBRE", "INC", "APELLIDO", "NSS", "FECHA\_NCTO", "DIRECCION", "SALARIO", "NSS\_SUPERV", "ND"};

Consulta miConsulta= new Consulta (consulta, atributos);

System.out.println("Resultado de la consulta \"" + consulta + "\":");

miConsulta.consulta();

miConsulta.imprimeResultado();

}//main()

}//class Consulta

Básicamente sigue los siguientes pasos:

Crea una sentencia de la clase Statement a partir de un objeto de tipo Connection.

Ejecuta la sentencia y obtiene un objeto de la clase ResultSet

Imprime el resultado de la ejecución de la sentencia a partir del ResultSet

Ejercicios

Completa el código anterior para que escriba en pantalla el contenido de la tabla Empleado. Para obtener el contenido de cada una de las columnas de una tupla, utiliza los métodos getXXX() de ResultSet donde XXX representa el tipo de dato de la columna (consultar este listado de conversiones). Asegurate también de que para crear la conexión a la base de datos utilizas un objeto de la clase DataSource.

Si todo ha ido bien, las ejecuciones nos darán los mismos datos que si hubieramos hecho la consulta desde SQLPLUS:

*¡Conexión correcta!*

*¡Consulta hecha!*

Resultado de la consulta "SELECT \* FROM EMPLEADO":

Fila 1: John B Smith 123456789 1965-01-09 00:00:00.0 731 Fondren, Houston, TX H 30000 333445555 5

Fila 2: Franklin T Wong 333445555 1955-12-08 00:00:00.0 638 Voss, Houston, TX H 40000 888665555 5

Fila 3: Alicia J Zelaya 999887777 1968-07-19 00:00:00.0 3321 Castle, Spring, TX M 25000 987654321 4

Fila 4: Jennifer S Wallace 987654321 2041-06-20 00:00:00.0 291 Berry, Bellaire, TX M 43000 888665555 4

Fila 5: Ramesh K Narayan 666884444 1972-09-15 00:00:00.0 975 Fire Oak, Humble, TX H 38000 333445555 5

Fila 6: Joyce A English 453453453 1972-07-31 00:00:00.0 5631 Rice, Houston, TX M 25000 333445555 5

Fila 7: Ahmad V Jabbar 987987987 1969-03-29 00:00:00.0 980 Dallas, Houston, TX H 25000 987654321 4

Fila 8: James E Borg 888665555 2037-11-10 00:00:00.0 450 Stone, Houston, TX M 55000 null 1

**ACCESO A METADATOS**

Como has podido comprobar, el acceso a la información de un ResultSet requiere conocer previamente el tipo de datos de la información que estás accediendo. Existe una forma de conocer su tipo en tiempo de ejecución accediendo a los metadatos de un ResultSet. Toda esta información la podemos obtener a partir del objeto ResultSetMetadata obtenido por el método getMetaData().

**Algunos métodos interesantes sobre esta clase son:**

* getColumnType() - Devuelve el tipo de dato de la columna especificada como un valor entero. Los tipos de datos existentes pueden consultarse en la clase java.sql.Types
* getColumnName() - Devuelve la cadena de texto que representa el nombre de la columna
* getColumnCount() - Devuelve el número de columnas existentes

Se recomienda echar un vistazo a los métodos de esta clase ResultSetMetadata para poder ver otra información interesante que se puede extraer de la misma.

Ejercicios

Añade una cabecera a los datos presentados en el programa anterior, de forma que aparezca para cada columna, su nombre; debería tener una salida similar a la siguiente:

Nº Fila NOMBRE INC APELLIDO NSS FECHA\_NCTO DIRECCION SEXO SALARIO NSS\_SUPERV ND

Fila 1: John B Smith 123456789 1965-01-09 00:00:00.0 731 Fondren, Houston, TX H 30000 333445555 5

Fila 2: Franklin T Wong 333445555 1955-12-08 00:00:00.0 638 Voss, Houston, TX H 40000 888665555 5

...

Añade el método imprimeResultSetDesordenado que sea capaz de imprimir todos los datos de una tupla independientemente del orden en que aparezcan y del nombre que tenga su columna.

Usa el método anterior para presentar en pantalla la información devuelta por esta consulta (De cada proyecto ubicado en 'Stafford', haga una lista con el número de proyecto, el número de departamento controlador y el apellido, dirección y fecha de nacimiento del jefe de departamento):

SELECT NUMEROP, NUMD, APELLIDO, DIRECCION, FECHA\_NCTO

FROM PROYECTO, DEPARTAMENTO, EMPLEADO

WHERE LOCALIZACIONP='Stafford' AND NUMD=NUMEROD AND NSS\_JEFE=NSS

La consulta debería devolver:

*¡Conexión correcta!*

*¡Consulta hecha!*

Resultado de la consulta "SELECT NUMEROP, NUMD, APELLIDO, DIRECCION, FECHA\_NCTO FROM PROYECTO, DEPARTAMENTO, EMPLEADO WHERE LOCALIZACIONP='Stafford' AND NUMD=NUMEROD AND NSS\_JEFE=NSS":

Nº Fila NUMEROP NUMD APELLIDO DIRECCION FECHA\_NCTO

Fila 1: 10 4 Wallace 291 Berry, Bellaire, TX 2041-06-20 00:00:00.0

Fila 2: 30 4 Wallace 291 Berry, Bellaire, TX 2041-06-20 00:00:00.0

**SENTENCIAS PREPARADAS**

Muchas veces tenemos que ejecutar varias veces una misma sentencia sobre una tabla (como puede ser el caso de insertar varias tuplas similares). Para estos casos, existe la posibilidad de crear sentencias preparadas; estas son sentencias optimizadas para ser ejecutadas muchas veces. Previamente a su ejecución, su código es optimizado para tardar menos al ejecutar, pero por contra este proceso también consume un tiempo que puede penalizar la ejecución si solo lo utilizamos para ejecutar pocas sentencias. Una sentencia preparada (PreparedStatement) se crea de la misma forma que un Statement convencional:

PreparedStatement updateSales = con.prepareStatement("UPDATE COFFEES SET SALES = ? WHERE COF\_NAME = ?");

En el ejemplo anterior se crea una sentencia de actualización de una supuesta tabla llamada COFFEES. Como se puede ver, hay dos valores no definidos. Una sentencia preparada permite obtener parámetros externos a la sentencia, y aqui radica la potencia de este tipo de sentencias. Para dar valor a estos parámetros, podemos utilizar los métodos setXXX(pos, value) donde XXX representa el tipo de dato utilizado, pos es el número de parámetro y value el valor dado a este parámetro. Por ejemplo, para ejecutar la sentencia "UPDATE COFFEES SET SALES=75 WHERE COF\_NAME='Colombian'":

PreparedStatement updateSales = con.prepareStatement("UPDATE COFFEES SET SALES = ? WHERE COF\_NAME = ?");

updateSales.setInt(1,75);

updateSales.setString(2,"Colombian");

updateSales.executeUpdate();

Ejercicios adicionales

Modifica la clase anterior para que utilice una sentencia preparada para realizar la consulta. Presenta en pantalla la consulta para las localizaciones 'Stafford' y 'Houston' con la misma sentencia preparada.

Utilizando sentencias preparadas, crea un método creaEmpleados() que, pasándole un valor entero, inserte dicho número de tuplas de contenido aleatorio en la tabla Empleado. El contenido de los campos pueden ser cadenas de texto aleatorias.