élogos

ACTUALIZACIÓN Y MANIPULACIÓN DE DATOS

élogos

ÍNDICE

ACTUALIZACIÓN Y MANIPULACIÓN DE DATOS

1. Ejecutar sentencias SQL

Sentencias SQL

Para ejecutar las sentencias SQL se va necesitar un objeto que implemente la interfaz **Statement**.

Para obtener un objeto Statement utilizaremos el siguiente método de la interfaz **Connection.**

```
Statement <Connection> create Statement () throws SQLException
```

Una vez que se tenga el Statement se podrá utilizar ejecutando las instrucciones SQL que se quiera. Con Statement se pueden ejecutar dos tipos de sentencias SQL:

1. Sentencias de actualización
2. Sentencias de consultas

Sentencias de actualización

- **Insert Into.** Añade un nuevo registro en la base de datos
- **Update.** Actualiza los valores de un registro existente
- **Delete From.** Elimina un conjunto de registros

Para ejecutar las sentencias anteriores se utiliza el siguiente método:

```
int<Statement>.executeUpdate(String sql)
throws SQLException
```

El método devuelve el número de registros actualizados. Si se produce un fallo se lanza la Excepción.

Devuelve -1 si afecta a toda la tabla.

Devuelve 0 cuando ningún registro se ha visto afectado por la modificación.

También podemos ejecutar una consulta de actualización con el método:

```
Void execute (String sql)
```

Este método no devuelve ningún resultado.

Sentencias de consulta

```
SELECT Nombre, Direccion FROM Clientes
        WHERE (Saldo > 25000)
```

Para ejecutar las sentencias de consulta se tiene el siguiente método:

```
ResultSet <Statement>.excuteQuery (String sql)
        throws SQLException
```

Un ResultSet es el resultado de la consulta compuesto por filas y columnas.

Tras obtener el Resultset habrá que tratar la información contenida en él. Para ello se tienen los siguientes métodos:

```
boolean <ResultSet>.next()
```

Se mueve al siguiente registro del Resultset

Devuelve true cuando encuentra el siguiente registro. Cuando devuelve false se ha llegado al final de la consulta.

```
boolean <ResultSet>.getBoolean(int    columnIndex)
boolean <ResultSet>.getBoolean(String columnName)
int     <ResultSet>.getInt    (int    columnindex)
int     <ResultSet>.getInt    (String columnName)
String  <ResultSet>.getString (int    columnindex)
String  <ResultSet>.getString (String columnName)
```

Ejemplos de inserción de registros

```
import java.sql.*;
public class Insercion {
   //este programa de ejemplo añade un registro nuevo
    //en una hipotética tabla "clientes" de una base de datos
    //de empresa que está en MySQL
   public static void main(String[] args) {
       Connection cn=null;
       try{
          Class.forName ("com.mysql.jdbc.Driver");
           cn=DriverManager.getConnection ("jdbc:mysql://localhost:3306/empresas","root","root");
           Statement st=cn.createStatement ();
           String sql="insert into clientes (usuario,password,email,telefono) values";
           sql+="('manolito','xxxxx','manolito@jjj.com',555555)";
            //ejecución consulta
           st.execute (sql);
           System.out.println ("El cliente ha sido dado de alta");
       }
       catch(ClassNotFoundException ex){
              ex.printStackTrace();
           }
           catch(SQLException ex){
                      ex.printStackTrace();
           }
       finally{
          if(cn!=null){
             try{
                cn.close (); //cierre obligado de la conexión
             }
             catch(SQLException ex){
                      ex.printStackTrace();
             }
          }
       }
    }


}
```

Ejemplo de actualización de registros

```
import java.sql.*;
public class Actualizacion {
    //este programa de ejemplo modifica los datos
    //de un conjunto de registros de una hipotética tabla de productos
    //de una base de datos de empresa que está en MySQL
   public static void main(String[] args) {
        Connection cn=null;
        try{

            Class.forName("com.mysql.jdbc.Driver");

            cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/empresas","root","root");

            Statement st=cn.createStatement();

            String sql="update productos set precio=precio*0.95 where precio>100";
             //ejecución consulta
            st.execute(sql);

            System.out.println("Datos actualizados");
        }
        catch(ClassNotFoundException ex){
                ex.printStackTrace();
             }
             catch(SQLException ex){
                        ex.printStackTrace();
             }
        finally{
          if(cn!=null){
             try{
                cn.close(); //cierre obligado de la conexión
             }
             catch(SQLException ex){
                        ex.printStackTrace();
             }
          }
       }
    }
}
```

Actualización y manipulación de datos

Ejemplo de eliminación de registros

```
import java.sql.*;
public class Eliminacion {
    //este programa de ejemplo elimina los datos
    //de un conjunto de registros de una hipotética tabla de productos
    //de una base de datos de empresa que está en MySQL
   public static void main(String[] args) {
        Connection cn=null;
        try{

            Class.forName("com.mysql.jdbc.Driver");

            cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/empresas","root","root");

            Statement st=cn.createStatement();

            String sql="delete from productos where codigo=24765";
             //ejecución consulta
            st.execute(sql);

            System.out.println("Producto eliminado");
        }
        catch(ClassNotFoundException ex){
               ex.printStackTrace();
             }
             catch(SQLException ex){
                        ex.printStackTrace();
             }
        finally{
          if(cn!=null){
             try{
                cn.close(); //cierre obligado de la conexión
             }
             catch(SQLException ex){
                        ex.printStackTrace();
             }
          }
       }
    }
}
```

Ejemplo de recuperación de registros

```
import java.sql.*;
public class Recuperacion {
    //este programa muestra el nombre de los productos
    //de un conjunto de registros de una hipotética tabla de productos,
    //localizada en una base de datos de empresa que está en MySQL
   public static void main(String[] args) {
        Connection cn=null;
        try{

            Class.forName("com.mysql.jdbc.Driver");

            cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/empresas","root","root");

            Statement st=cn.createStatement();

            String sql="Select nombre From productos";
             //ejecución consulta
            ResultSet rs=st.executeQuery(sql);
            while(rs.next()){
                System.out.println("Nombre del producto: "+rs.getString("nombre"));
            }
        }
        catch(ClassNotFoundException ex){
                ex.printStackTrace();
             }
             catch(SQLException ex){
                        ex.printStackTrace();
             }
        finally{
           if(cn!=null){
              try{
                 cn.close(); //cierre obligado de la conexión
              }
              catch(SQLException ex){
                        ex.printStackTrace();
              }
           }
        }
    }
}
```

Metadatos

Estos métodos sirven para recuperar la información contenida dentro de las columnas, bien por el índice (columnIndex) o bien por el nombre de la columna (columnName). El índice de la primera columna tiene el valor 1.

También se pueden recoger los Metadatos del Resultset. Los Metadatos es la información de la estructura del sistema de almacenamiento. Se tienen los siguientes métodos:

- **int<Resultset>. findColumn(String columnName)**
- **ResultSetMetaData<ResultSet>.getMetaData()**

ResultSetMetaData tiene los siguientes métodos:

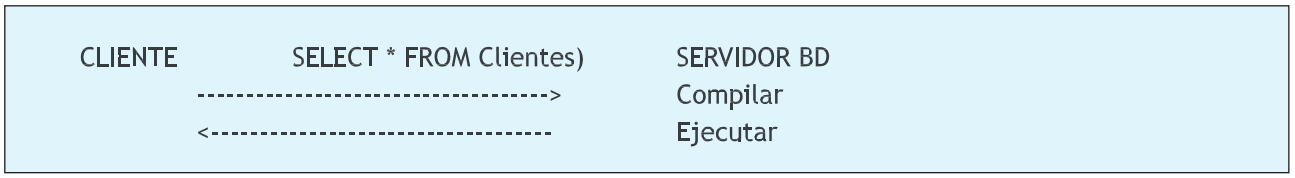
- **int<ResultSetMetaData>.getColumnCount()**
- **String <ResultSetMetaData>.getColumnName(int col)**
- **int<ResultSetMetaData>.getColumnType(int col)**
- **boolean<Statement>execute(String sql) throwsSQLException**

- **ResultSet<Statement>.getResultSet()**
- **int<Statement>.getUpdateCount()**

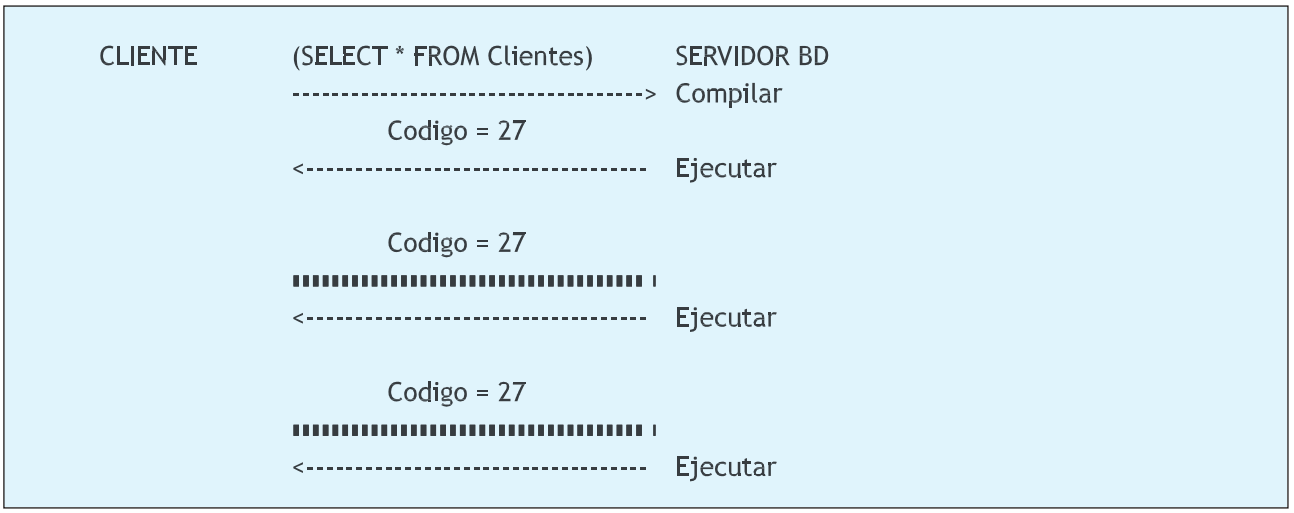
2. Sentencias SQL Precompiladas(Prepared Statement)

Sentencias SQL Precompiladas

Las sentencias SQL Precompiladas (Prepared Statement) son:



En caso de ejecutar varias veces una misma sentencia SQL si el SERVIDOR las tiene que compilar baja el rendimiento en la aplicación. Para que esto no ocurra se utilizan las sentencias precompiladas de SQL.



Parámetros de las sentencias SQL precompiladas

En vez de mandar la sentencia de nuevo se enviará sólo el código. Éste es devuelto por la base de datos. Todas las sentencias pueden ser sentencias precompiladas y además se podrán parametrizar:

```
INSERT INTO CLIENTES  VALUE (?,?,?,?)
```

De esta forma se podrá repetir la instrucción INSERT las veces necesarias. La primera vez enviando la propia sentencia y las restantes veces se enviará el código devuelto por la base de datos.

Las sentencias precompiladas se representan mediante el interface PreparedStament el cual deriva de Statement. Para las sentencias precompiladas se le dirá a la conexión que se quiere una PreparedStatement.

Ejemplo

```
import java.sql.*;
public class EjemploPrepared {
   //realiza la inserción de un grupo de objetos Cliente
   public void insercionClientes(ArrayList<Cliente>() clientes){
      Connection cn=null;
      try{
         Class.forName("com.mysql.jdbc.Driver");

         cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/empresas","root","root");
         String sql="insert into clientes (usuario,password,email,telefono) values(?,?,?,?)";

         PreparedStatement st=cn.prepareStatement(null);
```

```
            //recorre cada uno de los clientes y realizar la inserción de
            //los datos
            for(Cliente c:clientes){
               st.setString(1, c.getUsuario());
               st.setString(2, c.getPassword());
               st.setString(3, c.getEmail());
               st.setInt(4, c.getTelefono());
               st.execute();
            }

         }
         catch(ClassNotFoundException ex){
               ex.printStackTrace();
          }
          catch(SQLException ex){
                     ex.printStackTrace();
          }
         finally{
           if(cn!=null){
              try{
                 cn.close(); //cierre obligado de la conexión
              }
              catch(SQLException ex){
                     ex.printStackTrace();
              }
           }
        }

    }
}
```

3. Uso de transacciones

La transacción y sus propiedades

Una transacción es un conjunto de operaciones que se quiere enviar a la base de datos de forma que se ejecute toda la transacción o que no se ejecute nada.
Una transacción se puede definir como un cambio de estado de la base de datos.
Este cambio de estado no se puede realizar ya que no existe ninguna instrucción de SQL para realizarla sino que hay que usar varias instrucciones.

Las transacciones tienen tres propiedades:

- Atomicidad
- Serialización

- Persistencia

El auto commit mode

En Java, cuando se realiza la conexión con la base de datos, el sistema se encuentra en auto commit mode, esto quiere decir que cada sentencia que se envíe se considera una transacción.

Para realizar varias sentencias en SQL como una sola transacción habrá que desactivar el auto commit mode. Esto se realiza con el siguiente método:

```
void <Connection>.setAutoCommit (boolean autoCommit)
```

Cuando se quieran ejecutar todas las sentencias SQL lanzadas contra la base de datos habrá que ejecutar el siguiente método:

```
void <Connection>.commit() throws SQLException
```

Si no se confirma la transacción se deshace la transacción.
Si se quiere deshacer la transacción se tiene el siguiente método.

```
void <Connection>.rollback() throws SQLException
```

Las excepciones que lanza rollback no tienen que ver con las transacciones que se han realizado. La excepción nos informan sobre otro tipo de problemas que se puedan dar durante la operación de rollback (caída de la red, etc).
Cuando se abre una transacción se bloquean registros de la base de datos, esto hay que tenerlo en cuenta. Siempre es aconsejable que el ordenador realice todas las operaciones de la transacción.

setAutoCommit

El auto commit mode no se activa tras realizar el método commit sino que habrá que activarlo mediante el método setAutoCommit.

Ejemplo de programa con transacciones:

```
conexion.setAutoCommit (false);
try {
        Statement stmt = conexion.createStatement();
        stmt.executeUpdate ("INSERT INTO Clientes VALUES
                (1,'Luis',"Principe de Vergara 44',4000000)");
        stmt.executeUpdate ("INSERT INTO Clientes VALUES
                (1,'Jose',"O'Donell 10',7000000)");
        conexion.commit();
}

catch (Exception ex) {
        System.out.println("Transacción deshecha "+ex);
}

conexion.setAutoCommit (true);
```

Nivel de aislamiento

Las bases de datos tienen lo que se llaman niveles de aislamiento (Isolation Levels). Los niveles de aislamiento informan de cuánto tiempo está una transacción aislada del resto de transacciones.

Para trabajar con el nivel de aislamiento se tienen los siguientes métodos:

```
int <Connection>.getTransactionIsolation()
                Devuelve el código de nivel de aislamiento
void <Connection>.setTransactionIsolation(int level)
        throws SQLException
                Fija el nivel de aislamiento.
```

Procedimientos almacenados (Stored Procedures)

Los procedimientos almacenados son un conjunto de instrucciones de base de datos que se van a ejecutar

en la base de datos del servidor. Con esto se gana CPU en el cliente pero se pierde CPU en el servidor.

- Una ventaja importante es que al ejecutarse todo en el servidor no hay que realizar transporte de información por lo tanto se acelera el proceso ya que sólo se realiza una petición por el cliente y el resultado producido por el servidor.
- Los procedimientos almacenados son funciones de java que van a tener parámetros de entrada (IN) y parámetros de salida (OUT).
- También existen parámetros INOUT, donde se les pasa un parámetro de entrada y en el mismo parámetro se produce la salida.

La única diferencia que existe con las funciones es que los procedimientos almacenados pueden tener varios parámetros de salida.