Diseño y Desarrollo de Sistemas de Información

Sentencias de consulta y modificación. Transacciones

Sentencias de consulta

- Clase **Statement**. Representación de un objeto "sentencia" y de su entorno de ejecución (sobre el objeto de conexión activo)

- El método **executeQuery()** sobre un objeto *Statement* devuelve un objeto *ResultSet* (contiene el conjunto de filas obtenidas del resultado de una consulta)

  - Método **next()**. Avanza el cursor fila a fila. Devuelve *true* si se encuentra en una fila concreta y *false* si alcanza una posición posterior a la última fila

  - Cuando el *ResultSet* se encuentra en una fila concreta se pueden usar los métodos de acceso a las columnas:

    tipo getTipo(nombre o número de columna), donde Tipo indica el tipo de datos de la columna que queremos recuperar

Algunos get*Tipo*()

Tipo Standard SQL	Método getTipo
CHAR	getString
VARCHAR	getString
SMALLINT	getShort
INTEGER	getInt
FLOAT	getFloat/getDouble
DOUBLE	getDouble
DECIMAL	getDecimal
DATE	getDate
MONEY	getDouble
TIME	getTime

- Esta lista no incluye todas las posibilidades, tan solo las más básicas
- Muchos get*Tipo*() se pueden utilizar para actuar sobre diferentes tipos SQL, haciendo que Java aplique una transformación

getString se puede aplicar para recuperar cualquier tipo SQL

Sentencias de consulta

```
System.out.println("Los datos de la tabla son:");
try  {
     Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery("select * from p");
     while (rs.next()) {
        int v1 = rs.getInt(1);
        int v2 = rs.getInt(2);
        System.out.println(v1 + "   " + v2);
     }
     stmt.close();
}
catch(SQLException e) {
     System.out.println("Error al leer datos de la tabla");
     System.out.println(e.getMessage());
     System.out.println(e.getSQLState());
     System.out.println(e.getErrorCode();
}
```

Importante: no olvidar liberar los recursos con ***close()***

Sentencias de modificación

- Operaciones: INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE, etc.
- Método ***executeUpdate()***. Devuelve el número de filas que han sido modificadas por la ejecución de la sentencia

Sentencias de modificación

```
System.out.println("Realizando operaciones...");
System.out.println("Insertando datos");
try {
     Statement stmt = conn.createStatement();
     stmt.executeUpdate("insert into p values (40,40)");
     stmt.executeUpdate("insert into p values (50,50)");
     stmt.close();
}
catch(SQLException e) {
     System.out.println("Error al insertar datos en la tabla");
     System.out.println(e.getMessage());
     System.out.println(e.getSQLState());
     System.out.println(e.getErrorCode());
}
```

Todas las sentencias SQL se pasan como parámetros de tipo "cadena de caracteres"
Debemos tener cuidado a la hora de crear estas sentencias
No olvidar nunca que SQL utiliza ‘ y no “ para los tipos char y varchar

Sentencias preparadas y parametrizadas

- Lo más habitual es construir consultas genéricas a las que, posteriormente, se le asignan los valores de los parámetros
- Para ello, se utiliza la clase **PreparedStatement**, que es una extensión de la clase Statement
- Además, es una de las formas que tiene Java de evitar los "SQL injection"

```
PreparedStatement ps = null;
ps = conn.prepareStatement("UPDATE COFFEES SET SALES = ? WHERE COF_NAME = ?");

ps.setInt(1,75);
ps.setString(2,"Colombian");
ps.executeUpdate();
```

Transacciones

- Antes de seguir, debes repasar el concepto de **transacción** y su utilidad
- Para trabajar con transacciones desde Java, tenemos los siguientes métodos de la clase *Connection*:

  - **setAutoCommit(*true*)**. Cada sentencia es tratada como una transacción. Es el valor por defecto
  - **setAutoCommit(*false*)**. Podemos especificar transacciones de más de una sentencia
  - **rollback()**. La BD vuelve al estado que tenía después del último *commit*
  - **commit()**. Los cambios realizados por la transacción se vuelven efectivos

Transacciones - Ejemplo

```
try  {
  conn.setAutoCommit(false);
  Statement stmt = conn.createStatement();
  stmt.execupeUpdate("insert into p values (80,80)");
  stmt.execupeUpdate("insert into p values (90,90)");
  System.out.println("Confirma la inserción (S/N):");
  BufferedReader new BufferedReader(new InputStreamReader(System.in));
  String s = null;
  try {
     s = in.readLine();
  }
   catch (IOException exc) {
     System.out.println(exc);
  }
  if (s.startsWith("S"))
     conn.commit();
  else
     conn.rollback();
  stmt.close();
  conn.setAutoCommit(true);
}
catch (SQLException e) {
  ...
}
```

Transacciones

- Puede usarse un bloque **finally** para asegurarnos de que se ejecuta cierto código incluso tras ocurrir una excepción:

```
try {
    conn.setAutoCommit(false);
    // Sentencias de la transacción
    conn.commit();
}
catch (SQLException e) {
    try {
          conn.rollback();
    }
    catch (SQLException ex) {
          System.out.println("Fallo en el roolback: " + e.getMessage());
    }
}
finally {
    try {
          conn.setAutoCommit(true);
    }
    catch (SQLException e) {
          System.out.println(e.getMessage());
    }
}
```