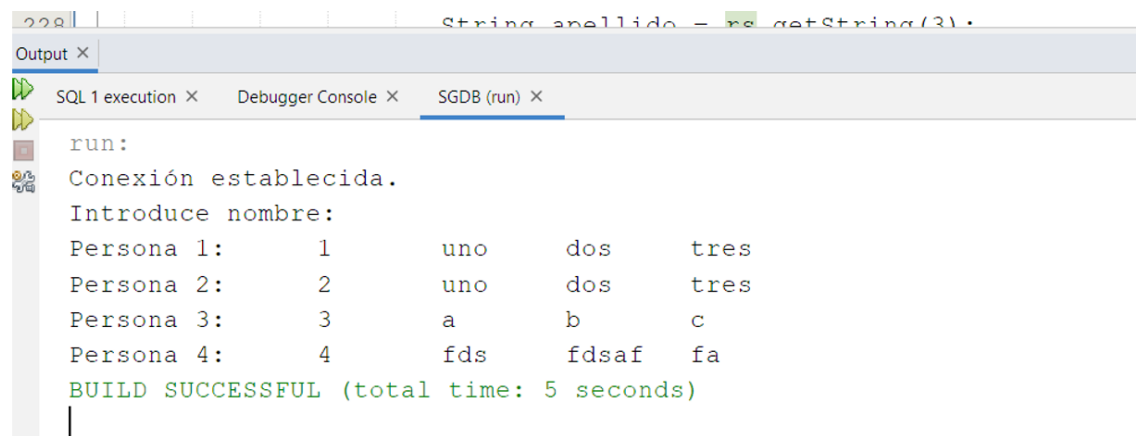


# Eliminación y Actualización de Datos

Finalizamos el tema de gestión de SGBD relacionales con la eliminación y actualización de datos. Sería lo que en SQL ejecutamos con las sentencias DELETE y UPDATE, pero vamos a ver dos maneras diferentes de realizarlo.

## 1) Con sentencias UPDATE de SQL.

Vamos a cambiar el nombre del registro dos a “otro” en el supuesto de que el que tenga sea “uno”, en otro caso lo dejamos sin modificar.



The screenshot shows a Java IDE with a debugger console. The console has tabs for 'Output', 'SQL 1 execution', 'Debugger Console', and 'SGDB (run)'. The 'SGDB (run)' tab is active, displaying the following output:

```
run:
Conexión establecida.
Introduce nombre:
Persona 1:      1      uno      dos      tres
Persona 2:      2      uno      dos      tres
Persona 3:      3      a       b       c
Persona 4:      4      fds     fdsaf   fa
BUILD SUCCESSFUL (total time: 5 seconds)
```

El método podría quedar algo así como:

```

/**
 * Actualiza datos en una tabla de la Base de Datos
 *
 * @param tabla Nombre de la tabla en la que se van a actualizar los datos
 * @param datos array de dos dimensiones <ol><li>primera dimensión: nombres de
 * los campos</li><li>segunda dimensión: valores de los campos. Entre '' si
 * son de algún tipo caracter</li></ol>
 * @param where Condición que deben cumplir los datos para ser actualizados.
 * null si no se desea condición
 * @return true si se pudo realizar la actualización o false en caso contrario
 */
public static boolean bdActualizarDatos(String tabla, String[][] datos,
    String where) {
    String sql = "UPDATE " + tabla + " SET ";
    for (String[] dato : datos) {
        sql += dato[0] + " = " + dato[1] + ", ";
    }
    sql = sql.substring(0, sql.length() - 2)
        + (where == null ? "" : " WHERE " + where);
    try {
        sentencia = conexion.createStatement();
        sentencia.executeUpdate(sql);
    } catch (SQLException ex) {
        LOG.log(Level.SEVERE, null, ex);
        return false;
    }
    try {
        sentencia.close();
    } catch (SQLException ex) {
        LOG.log(Level.SEVERE, null, ex);
    }
    return true;
}

```

Llamamos al método y comprobamos el resultado:

```

245     }
246     bdActualizarDatos(tabla, new String[][]{{"nombre", "'otro'"}}, "id = 2 and nombre = 'uno'");
247     ResultSet rs = bdRecuperarTodo(tabla);
248     if (rs != null) {

```

Output X

Debugger Console X Debugger Console X SGDB (run) X

```

run:
Conexión establecida.
Introduce nombre:
Persona 1:      1      uno      dos      tres
Persona 2:      2      otro     dos      tres
Persona 3:      3      a        b        c
Persona 4:      4      fds      fdsaf    fa
BUILD SUCCESSFUL (total time: 2 seconds)

```

Y ahora vamos a borrar los elementos cuyo apellido sea “dos”. Para ello utilizamos la sentencia DELETE de SQL

```

/**
 * Elimina registros en una tabla de la Base de Datos
 * @param tabla Nombre de la tabla en la que se van a actualizar los datos
 * @param where Condición que deben cumplir los registros para ser eliminados.
 * null si no se desea condición.
 * @return true si se pudo realizar la eliminación o false en caso contrario
 */
public static boolean bdEliminarRegistros(String tabla, String where) {
    String sql = "DELETE FROM " + tabla
        + (where == null ? "" : " WHERE " + where);
    try {
        sentencia = conexion.createStatement();
        sentencia.executeUpdate(sql);
    } catch (SQLException ex) {
        LOG.log(Level.SEVERE, null, ex);
        return false;
    }
    try {
        sentencia.close();
    } catch (SQLException ex) {
        LOG.log(Level.SEVERE, null, ex);
    }
    return true;
}

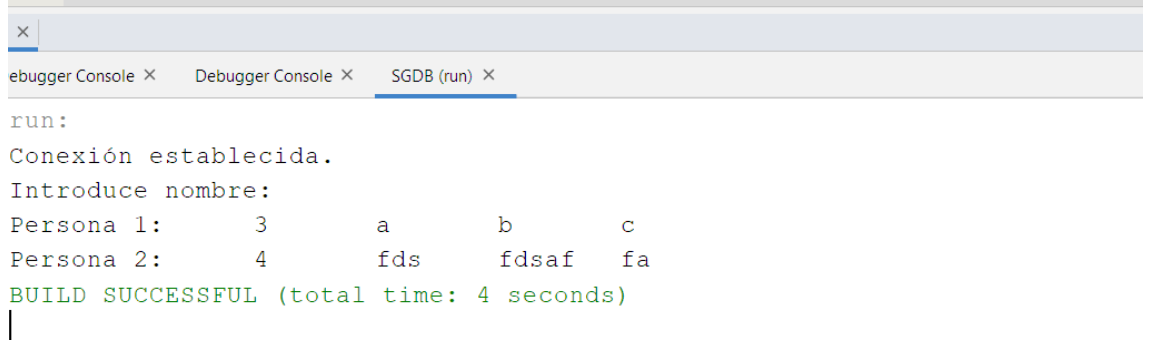
```

Lo ejecutamos:

```

bdActualizarDatos(tabla, new String[][]{{"nombre", "otro"}});
bdEliminarRegistro(tabla, "apellido = 'dos'");
ResultSet rs = bdRecuperarTodo(tabla);
if (rs.isFirst()) {

```



```

run:
Conexión establecida.
Introduce nombre:
Persona 1:      3      a      b      c
Persona 2:      4     fds     fdsaf   fa
BUILD SUCCESSFUL (total time: 4 seconds)

```

## 2) Mediante sentencias preparadas.

Cuando recuperamos los datos de una BD mediante un **ResultSet** podemos hacer que este sea de lectura escritura, de modo que, si modificamos algo en dicho conjunto, sus modificaciones se van a reflejar automáticamente en la BD. Esto se extiende también a las inserciones, así en lugar de utilizar sentencias **INSERT** o **UPDATE**, lo que haremos será posicionarnos convenientemente sobre el **ResultSet** e introducir/modificar los datos.

- Nos conectamos al comienzo del programa y nos desconectamos al finalizar.
- Nos conectamos y desconectamos cada vez que sea necesario, o sea en cada iteración que realicemos con la BD, y el resto del tiempo trabajamos sin conexión, liberando así recursos.

Vamos ahora con el manejo en modo lectura/escritura de los **ResultSet**. Para ello cambiamos el modo en el que abrimos el **createStatement**:

En lugar de:

```
stmt = conn.createStatement();
```

Usaremos:

```
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                              ResultSet.CONCUR_UPDATABLE);
```

El primer parámetro puede ser uno de los siguientes valores:

- TYPE\_FORWARD\_ONLY** Hoja de datos no desplazable. Solo se puede ir hacia adelante. (por defecto)
- TYPE\_SCROLL\_INSENSITIVE** Hoja de datos desplazable. No refleja los cambios mientras está abierta.
- TYPE\_SCROLL\_SENSITIVE** Hoja de datos desplazable. Sí refleja los cambios aun estando abierta.

El segundo argumento es una de las dos constantes de ResultSet para especificar si la hoja de resultados es de sólo lectura o actualizable:

- **CONCUR\_READ\_ONLY** De solo lectura (por defecto)
- **CONCUR\_UPDATABLE** Actualizable

Así que lo hemos abierto de modo que permita moverse en ambas direcciones, permita la actualización de los datos y esta se refleje de modo automático.

Para recuperar los datos en principio bastaría con cambiar esa línea en nuestros métodos. Pero ahora para actualizar, eliminar o insertar, ya no utilizaríamos sentencias SQL sino que actuaríamos directamente sobre el ResultSet.

Recuperamos todos los datos:

```
ResultSet rs = bd.recuperarTodo(tabla, "");
```

Insertamos un nuevo registro. Para ello nos colocamos en una fila especial, llamada **InsertRow**, y actualizamos los valores de cada campo. Importante, como en esta tabla de ejemplo el Id es auto incrementable, no hay que poner nada. Si no fuese auto incrementable, descomentaríamos la línea correspondiente, dándole el valor adecuado. Y por último ejecutamos la inserción llamando al método **insertRow**

```
Scanner tec = new Scanner(System.in);
while (true) {
    System.out.print("Introduce nombre: ");
    String nombre = tec.nextLine();
    if (nombre.isEmpty()) {
        break;
    }
    System.out.print("Introduce apellido: ");
    String apellido = tec.nextLine();
    System.out.print("Introduce email: ");
    String email = tec.nextLine();

    try {
        rs.moveToInsertRow();
        //rs.updateInt(1, );
        rs.updateString(2, nombre);
        rs.updateString(3, apellido);
        rs.updateString(4, email);
        rs.insertRow();
    } catch (SQLException ex) {
        LOG.log(Level.SEVERE, null, ex);
    }
}
```

Dejamos como ejercicio la eliminación de registros (**deleteRow()**), y la actualización (**updateRow()**). El esquema podría ser colocarse al comienzo del ResultSet y avanzar hasta encontrar un registro que cumpla los requerimientos, por ej. Buscamos una determinad id:

If (rs.getInt(1) == id) ....

Y entonces ejecutamos un **deleteRow()** o, para actualizar realizamos los **updateXXX()** para los campos necesarios y ejecutamos la actualización con **updateRow()**