

Bases de datos embebidas y Transacciones:

SQLite

Crear: primero creamos la conexión con la base de datos SQLite ubicada en la ruta especificada. Posteriormente, verificamos si existe una tabla llamada contadores y, si no, la crea con dos columnas: nombre como clave primaria y cuenta con un valor por defecto de 0. Luego, insertamos los primeros datos. Si ocurre algún error, muestra un mensaje con la excepción.

```
public class CrearTablaSQLite {  
  
    public static void main(String[] args) {  
        final String urlBD = "jdbc:sqlite:/home/alumno/bbdd/sqlite/contadores.db"; // Ruta válida a la base d  
  
        try (Connection connection = DriverManager.getConnection(urlBD);  
            Statement statement = connection.createStatement()) {  
  
            // Crear la tabla 'contadores' si no existe  
            String crearTablaSQL = "CREATE TABLE IF NOT EXISTS contadores (" +  
                                    "nombre TEXT PRIMARY KEY, " +  
                                    "cuenta INTEGER DEFAULT 0);";  
            statement.execute(crearTablaSQL);  
  
            // Insertar el contador inicial solo si no existe  
            String insertarSQL = "INSERT OR IGNORE INTO contadores (nombre, cuenta) VALUES ('contador1', 0);";  
            statement.executeUpdate(insertarSQL);  
            System.out.println("Creado");  
  
        } catch (SQLException e) {  
            System.out.println("Error al crear la tabla o insertar el contador: " + e.getMessage());  
        }  
    }  
}
```

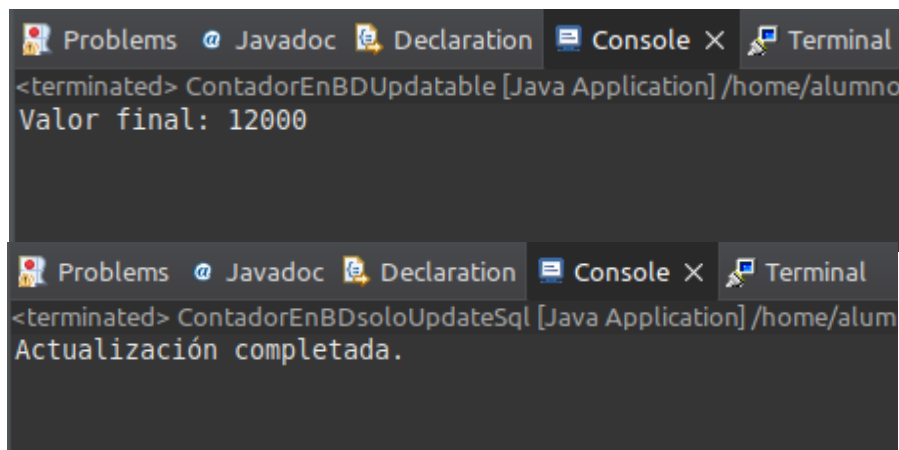
Buggy: incrementamos 1000 veces el contador contador1 en una base de datos SQLite, leyendo el valor con SQL_CONSULTA y actualizándose con SQL_ACTUALIZA. Al final, muestra el valor actualizado del contador.

```
public class ContadorBuggy_SQLITE {  
    static final String SQL_CONSULTA=  
        "select cuenta from contadores where nombre='contador1';"  
    static final String SQL_ACTUALIZA=  
        "update contadores set cuenta=? where nombre='contador1';"  
  
    public static void main(String[] args) {  
        try (Connection con =  
            DriverManager.getConnection("jdbc:sqlite:/home/alumno/bbdd/sqlite/contadores.db");  
            )  
        {  
            int cuenta = 0;  
            for (int i=1; i<=1000; i++) {  
                Statement consulta = con.createStatement();  
                PreparedStatement actualiza = con.prepareStatement(SQL_ACTUALIZA);  
                ResultSet res = consulta.executeQuery(SQL_CONSULTA);  
                if (res.next()) cuenta = res.getInt(1) + 1;  
                actualiza.setInt(1, cuenta);  
                actualiza.executeUpdate();  
            }  
            System.out.println("Valor final: " + cuenta);  
        } catch (SQLException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
    }  
}
```

Update: incrementamos el contador contador1 en SQLite 1000 veces usando una actualización: SQL (UPDATE contador SET cuenta=cuenta+1 WHERE nombre=?). Conectándose a la base de datos, ejecutando la sentencia en un bucle, y verificando que se actualice una fila en cada iteración. Si no encuentra el contador, muestra un error y detiene el proceso. Finalmente, confirma la actualización completada.

```
public static void main(String[] args) {
    final String claveContador = "contador1";
    final String sqlConsulta = "SELECT nombre,cuenta FROM contadores WHERE nombre=?";
    try{
        Class.forName("org.sqlite.JDBC");
        Connection connection = DriverManager.getConnection("jdbc:sqlite:/home/alumno/bbdd/sqlite/contadores.db");
        PreparedStatement stmt = connection.prepareStatement("UPDATE contador SET cuenta=cuenta+1 WHERE nombre=?");
        stmt.setString(1, claveContador);
        int cuenta = 0;

        for (int i=0; i<1000;i++) {
            //ResultSet res = consulta.executeQuery("SELECT nombre,cuenta FROM contador WHERE nombre='" + claveContador + "';");
            //ResultSet res = consulta.executeQuery();
            if (consulta.executeUpdate() == 0) { // por ver para qué sirve esto del boolean devuelvo por el execute ¿?
                ResultSet res = consulta.getResultSet();
                if (res.next()) {
                    cuenta = res.getInt(2)+1;
                    res.updateInt(2, cuenta);
                    res.updateRow();
                }
                //else break;
                else System.out.println("Error");
            }
            //if (i%10==0) System.out.println(i/10 + "%");
        } //
        System.out.println("Valor final: " + cuenta);
    } // try
    catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```



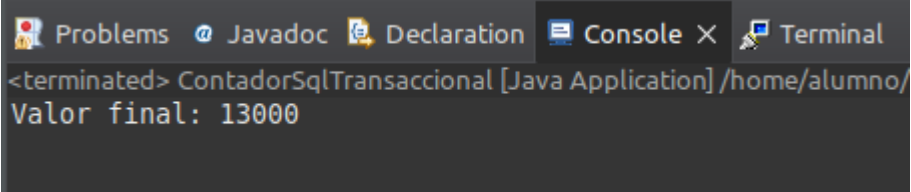
Transaccional: incrementa el contador contador1 en SQLite 1000 veces utilizando transacciones. Usa SELECT ... FOR UPDATE para bloquear la fila mientras se lee y una sentencia UPDATE para incrementar el valor. Ejecuta las operaciones dentro de una transacción, y luego hace un commit. Finalmente, imprime el valor final del contador.

```
public static void main(String[] args) throws ClassNotFoundException {
    // Prueba de concepto de transacción con bloqueo de fila para lectura
    // Será más fácil en el propio sql poner un set cuenta=cuenta+1 pero ilustramos
    // aquí el problema de concurrencia entre varios procesos.
    // con el for update + transacción conseguimos el bloque de fila y atomicidad
    String sqlConsulta = "select nombre,cuenta from contadores where nombre='contador1' for update;";
    String sqlActualizacion = "update contadores set cuenta=? where nombre='contador1';";

    Class.forName("org.sqlite.JDBC");

    try (Connection connection = DriverManager.getConnection("jdbc:sqlite:/home/alumno/bbdd/sqlite/contadores.db"))
    {
        PreparedStatement consulta = connection.prepareStatement(sqlConsulta);
        PreparedStatement actualizacion = connection.prepareStatement(sqlActualizacion);
        int cuenta = 0;

        for (int i=0; i<1000; i++) {
            connection.setAutoCommit(false);
            ResultSet res = consulta.executeQuery();
            if (res.next()) {
                cuenta = res.getInt(2);
                cuenta++;
                actualizacion.setInt(1, cuenta);
                actualizacion.executeUpdate();
            }
            else break;
            connection.commit();
            connection.setAutoCommit(false);
        } // for
        System.out.println("Valor final: " + cuenta);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



The screenshot shows an IDE interface with a dark theme. At the top, there are tabs for 'Problems', 'Javadoc', 'Declaration', 'Console', and 'Terminal'. The 'Console' tab is active, displaying the output of the Java application. The text in the console is: '<terminated> ContadorSqlTransaccional [Java Application] /home/alumno/ Valor final: 13000'. The text is white on a dark background.

TransaccionalUpdate: incrementa el contador contador1 en SQLite 1000 veces utilizando transacciones. Lee el valor con SELECT, lo incrementa y actualiza con UPDATE. Después de cada actualización, hace un commit para guardar los cambios y asegura la consistencia.

```
public static void main(String[] args) throws ClassNotFoundException {
    // Prueba de concepto de transacción con bloqueo de fila para lectura
    // Será a más fácil en el propio sql poner un set cuenta=cuenta+1 pero ilustramos
    // aquí el problema de concurrencia entre varios procesos.
    // con el for update + transacción conseguimos el bloque de fila y atomicidad
    String sqlConsulta = "SELECT nombre, cuenta FROM contadores WHERE nombre='contador1'";
    String sqlActualiza = "UPDATE contadores SET cuenta = cuenta + 1 WHERE nombre='contador1'";

    Class.forName("org.sqlite.JDBC");

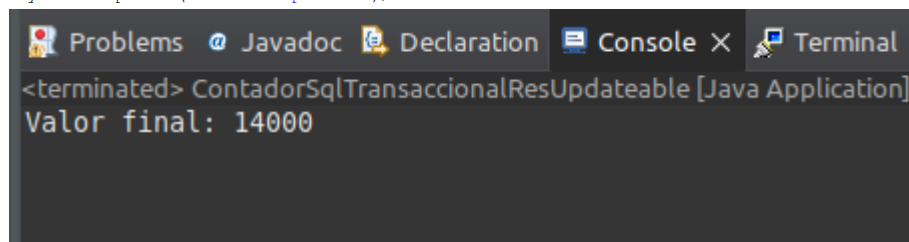
    try (Connection connection = DriverManager.getConnection("jdbc:sqlite:/home/alumno/bbdd/sqlite/contadores.db")) {
        connection.setAutoCommit(false); // Iniciar la transacción

        // Aquí usamos solo la consulta SELECT para leer el valor
        PreparedStatement consulta = connection.prepareStatement(sqlConsulta);
        PreparedStatement actualiza = connection.prepareStatement(sqlActualiza);

        for (int i = 0; i < 1000; i++) {
            // Comenzar la transacción
            connection.setAutoCommit(false);
            // Ejecutar la consulta para leer el valor
            var resultSet = consulta.executeQuery();
            if (resultSet.next()) {
                int cuenta = resultSet.getInt("cuenta");
                cuenta++; // Incrementar el valor
            } else {
                System.out.println("No se encontró el contador con nombre 'contador1'");
                break;
            }

            // Ejecutar la actualización para incrementar la cuenta
            actualiza.executeUpdate();
            connection.commit(); // Hacer commit para hacer efectiva la actualización
        }

        System.out.println("Proceso completado.");
    }
}
```



H2

Crear: primero conectamos a una base de datos H2, creamos la tabla contadores si no existe, e insertamos un contador inicial con nombre='contador1' y cuenta=0. Utilizamos PreparedStatement para la inserción y manejamos la conexión y recursos. Posteriormente imprime el número de filas insertadas.

```
public class Crear {
    public static void main(String[] args) throws SQLException {
        //
        Connection connection = DriverManager.getConnection("jdbc:h2:/home/alumno/bbdd/h2/contadores");
        Statement statement = connection.createStatement();
        try {
            statement.execute("CREATE TABLE IF NOT EXISTS contadores(nombre varchar(10) primary key, cuenta int)");

            String insertSQL = "insert into contadores(nombre,cuenta) values (?, ?)";
            PreparedStatement insertStatement = connection.prepareStatement(insertSQL);

            insertStatement.setString(1, "contador1");
            insertStatement.setInt(2, 0);
            int rowInserted = insertStatement.executeUpdate();
            System.out.println("Filas: " + rowInserted);

            insertStatement.close();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            statement.close();
            connection.close();
        }
    }
}
```

Eliminar: nos conectamos a una base de datos H2 y eliminamos la tabla contadores si existe. Usa DROP TABLE IF EXISTS y cierra la conexión. Imprime un mensaje confirmando la eliminación.

```
public class Eliminar {  
    public static void main(String[] args) throws SQLException {  
        // Conexión a la base de datos H2 en el archivo especificado  
        Connection connection = DriverManager.getConnection("jdbc:h2:/home/alumno/bbdd/h2/contadores");  
        Statement statement = connection.createStatement();  
  
        try {  
            // Eliminar la tabla 'contadores' si ya existe  
            statement.execute("DROP TABLE IF EXISTS contadores;");  
            System.out.println("Tabla 'contadores' eliminada.");  
  
        } catch (SQLException e) {  
            e.printStackTrace();  
        } finally {  
            // Cerrar los recursos para evitar fugas de memoria  
            statement.close();  
            connection.close();  
        }  
    }  
}
```

Posteriormente realizaremos los mismos métodos que hemos realizado en el apartado de SQLite pero simplemente cambiaremos la conexión a H2

DriverManager.getConnection("jdbc:h2:/home/alumno/bbdd/h2/contadores");

- conexión utilizada para conectar a h2.

DriverManager.getConnection("jdbc:sqlite:/home/alumno/bbdd/sqlite/contadores.db");

- conexión utilizada para conectar a SQLite.