JDBC

Autor: José Guillermo Sandoval Huerta

Fecha: 02 Noviembre 2023

Índice

¿Por qué JDBC (Java Database Connectivity)? 2
Driver Manager 2
Connections 3
Statements 4
- Inserting Rows 5
- Updating Rows 5
- Deleting Rows 6
- Other Modifying Statements 6

Result Set 7
Mapping Between SQL & Java Data Types 8
SQLException 8

¿Por qué JDBC (Java Database Connectivity)?

El JDBC (Java Database Connectivity) es una API de Java que permite interactuar con bases de datos relacionales. En esencia, JDBC proporciona una forma estándar y coherente de acceder y manipular datos en bases de datos SQL desde aplicaciones Java [1].

Las principales funciones de JDBC incluyen:

- **Establecer una conexión**: JDBC permite a las aplicaciones Java establecer una conexión con una base de datos utilizando controladores o drivers específicos proporcionados por el fabricante de la base de datos.

- **Enviar consultas SQL**: Los desarrolladores pueden utilizar JDBC para enviar consultas SQL a la base de datos. Esto incluye la ejecución de consultas SELECT para recuperar datos, así como consultas INSERT, UPDATE y DELETE para modificar datos en la base de datos.

- **Procesar resultados**: JDBC proporciona métodos para recuperar y procesar los resultados de las consultas SQL. Los resultados se devuelven en forma de conjuntos de resultados (result sets), que pueden ser iterados para obtener los datos necesarios.

- **Controlar transacciones**: JDBC permite a los desarrolladores administrar transacciones en la base de datos, lo que incluye la capacidad de confirmar transacciones exitosas o revertir transacciones en caso de errores.

- **Manejar excepciones**: JDBC gestiona excepciones y errores que puedan ocurrir durante la interacción con la base de datos, lo que permite a los desarrolladores tomar medidas apropiadas en caso de problemas.

- **Gestión de conexiones**: JDBC permite administrar las conexiones a la base de datos, lo que incluye abrir y cerrar conexiones de manera eficiente para evitar problemas de rendimiento.

Es importante destacar que JDBC es una especificación estándar de Java y, por lo tanto, no está limitado a una base de datos específica. Los fabricantes de bases de datos proporcionan controladores JDBC específicos para sus sistemas, lo que permite a las aplicaciones Java conectarse a una amplia gama de bases de datos relacionales, como Oracle, MySQL, PostgreSQL, Microsoft SQL Server y muchos otros.

Driver Manager

El DriverManager es una clase del API JDBC. Esta clase es parte del paquete java.sql y se utiliza para administrar los controladores de bases de datos JDBC y establecer conexiones con bases de datos [2].

Entre sus funcionalidades principales están:

- **Gestión de controladores**: El DriverManager es responsable de administrar los controladores de bases de datos JDBC. Los controladores son implementaciones específicas proporcionadas por los fabricantes de bases de datos que permiten la comunicación entre una aplicación Java y una base de datos específica. El DriverManager carga y registra estos controladores para que estén disponibles para su uso en la aplicación.

- **Establecimiento de conexiones**: A través del DriverManager, los desarrolladores pueden utilizar la URL de conexión, el nombre de usuario y la contraseña para establecer conexiones con una base de datos. El DriverManager selecciona el controlador apropiado para la base de datos en función de la URL de conexión proporcionada y establece una conexión con esa base de datos.

- **Administración de conexiones**: El DriverManager también puede ayudar en la administración de conexiones a la base de datos. Puede proporcionar una gestión básica de conexiones, como la obtención y liberación de conexiones, lo que facilita la gestión de múltiples conexiones en una aplicación.

Connections

Una "Connection" (conexión) es un objeto que representa una conexión activa y en tiempo de ejecución entre una aplicación Java y una base de datos relacional. La conexión se utiliza para enviar consultas SQL a la base de datos, recuperar resultados y realizar transacciones [3].

Entre sus funcionalidades esta:

- Establecimiento y cierre de conexiones.
- Envío de consultas SQL.
- Recuperación de resultados.
- Administración de transacciones.
- Administración de excepciones.

Entre las diferentes conexiones que se pueden realizar, están siguientes:

- **Conexión básica**: En una conexión JDBC básica, se establece una conexión directa con una base de datos utilizando un controlador específico para la base de datos en cuestión. Esta es la forma más común de conectarse a una base de datos.

- **DataSource**: JDBC DataSource es una interfaz que proporciona una forma más eficiente de gestionar conexiones a la base de datos, especialmente en aplicaciones de servidor. Un DataSource proporciona conexiones de manera eficiente y puede administrar recursos de conexión, como pools de conexiones, de una manera más eficaz que la conexión básica.

- **Pool de conexiones**: Un pool de conexiones (Connection Pool) es una técnica que implica la creación y administración de un conjunto de conexiones preestablecidas a la base de datos. En lugar de abrir y cerrar una conexión cada vez que se necesita una operación en la base de datos, se reutilizan conexiones del pool. Esto puede mejorar el rendimiento y la eficiencia de una aplicación.

- **Conexión en memoria (in-memory)**: Algunas bases de datos, como H2 o HSQLDB, permiten la creación de bases de datos en memoria, lo que significa que los datos se almacenan temporalmente en la memoria RAM en lugar de en disco. Las conexiones a estas bases de datos son generalmente más rápidas y se utilizan en aplicaciones donde la persistencia de datos no es crucial.

- **Conexiones en red (Network Connections)**: En entornos de bases de datos distribuidas, las conexiones en red se utilizan para acceder a bases de datos remotas o bases de datos en clúster.

Statements

Un "Statement" es una interfaz que se utiliza para ejecutar instrucciones SQL en una base de datos desde una aplicación Java. Los objetos de tipo Statement permiten a los programadores enviar consultas y comandos SQL a la base de datos, lo que les permite interactuar con los datos almacenados en dicha base de datos [4]. Hay tres tipos principales de Statement en JDBC.

(1) **Statement**: La interfaz Statement se utiliza para ejecutar sentencias SQL simples sin parámetros. Puedes crear objetos Statement para ejecutar consultas de tipo SELECT, INSERT, UPDATE, DELETE, entre otras.

Código 1. Statement.

```
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery("SELECT * FROM tabla");
```

Nota: el uso de este Statement puede ser propenso a ataques de inyección SQL si no se utilizan correctamente, ya que no se manejan de forma segura los parámetros de entrada del usuario.

(2) **PreparedStatement**: La interfaz PreparedStatement se utiliza para ejecutar sentencias SQL pre compiladas que pueden contener parámetros. Estos parámetros se representan como marcadores de posición en la consulta y se pueden establecer de manera segura para evitar la inyección de SQL. El uso de PreparedStatement es preferible cuando se manejan datos de usuario.

Código 2. PreparedStatement.

```
PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO tabla (columna1,
columna2) VALUES (?, ?)");
preparedStatement.setString(1, valor1);
preparedStatement.setInt(2, valor2);
preparedStatement.executeUpdate();
```

(3) **CallableStatement**: La interfaz CallableStatement se utiliza para ejecutar llamadas a procedimientos almacenados en una base de datos.

Código 3. CallableStatement.

```
CallableStatement callableStatement = connection.prepareCall("{call nombre_procedimiento(?, ?)}");
callableStatement.setInt(1, valor1);
callableStatement.registerOutParameter(2, Types.INTEGER);
callableStatement.execute();
int resultado = callableStatement.getInt(2);
```

Inserting Rows

Por medio de los Statement, se puede realizar inserciones de registro en las bases de datos, un ejemplo de este proceso es por medio del siguiente fragmento de código.

Código 4. Inserción de registro.

```
// Definir la sentencia SQL INSERT con parámetros
    String sqlInsert = "INSERT INTO tu_tabla (nombre, edad) VALUES (?, ?)";

    try (Connection connection = DriverManager.getConnection(jdbcUrl, usuario, contraseña);
         PreparedStatement preparedStatement = connection.prepareStatement(sqlInsert)) {

        // Establecer los valores de los parámetros
        preparedStatement.setString(1, nombre);
        preparedStatement.setInt(2, edad);

        // Ejecutar la inserción
        int filasAfectadas = preparedStatement.executeUpdate();

        if (filasAfectadas > 0) {
            System.out.println("Se ha insertado la fila exitosamente.");
        } else {
            System.out.println("No se ha podido insertar la fila.");
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
```

Updating Rows

Además del proceso anterior, también se puede realizar actualizaciones de registro en las bases de datos, un ejemplo de este proceso es muestra en el siguiente fragmento de código.

Código 5. Actualización de registro.

```
// Definir la sentencia SQL UPDATE con parámetros
    String sqlUpdate = "UPDATE tu_tabla SET edad = ? WHERE nombre = ?";

    try (Connection connection = DriverManager.getConnection(jdbcUrl, usuario, contraseña);
         PreparedStatement preparedStatement = connection.prepareStatement(sqlUpdate)) {

        // Establecer los valores de los parámetros
        preparedStatement.setInt(1, nuevaEdad);
        preparedStatement.setString(2, nombre);

        // Ejecutar la actualización
        int filasAfectadas = preparedStatement.executeUpdate();

        if (filasAfectadas > 0) {
            System.out.println("Se ha actualizado la fila exitosamente.");
        } else {
            System.out.println("No se ha podido actualizar la fila.");
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
```

Deleting Rows

Otro Statement realizado comúnmente es la eliminación de registro en las bases de datos, un ejemplo de este proceso es muestra en el siguiente fragmento de código.

Código 6. Eliminación de registro.

```
// Definir la sentencia SQL DELETE con parámetros
      String sqlDelete = "DELETE FROM tu_tabla WHERE nombre = ?";

      try (Connection connection = DriverManager.getConnection(jdbcUrl, usuario, contraseña);
           PreparedStatement preparedStatement = connection.prepareStatement(sqlDelete)) {

          // Establecer los valores de los parámetros
          preparedStatement.setString(1, nombre);

          // Ejecutar la eliminación
          int filasAfectadas = preparedStatement.executeUpdate();

          if (filasAfectadas > 0) {
              System.out.println("Se ha eliminado la fila exitosamente.");
          } else {
              System.out.println("No se ha podido eliminar la fila.");
          }

      } catch (SQLException e) {
          e.printStackTrace();
      }
```

Other Modifying Statements

Además de los statements básicos en JDBC (Statement, PreparedStatement y CallableStatement) que se utilizan para insertar, actualizar y eliminar datos en una base de datos, existen otros tipos de declaraciones que se utilizan en situaciones específicas o para tareas más avanzadas, entre los que podemos encontrar:

- **Batch Statement**: La interfaz Statement permite la ejecución de lotes de instrucciones SQL utilizando el método addBatch para agregar múltiples sentencias a un lote y executeBatch para ejecutar todas las sentencias en el lote de una vez. Esto puede ser útil para mejorar el rendimiento cuando se realizan múltiples operaciones en lotes.

Código 7. Batch Statement.

```
String sqlInsert1 = "INSERT INTO tabla (nombre, edad) VALUES (?, ?)";
String sqlInsert2 = "INSERT INTO tabla (nombre, edad) VALUES (?, ?)";
PreparedStatement preparedStatement = connection.prepareStatement(sqlInsert1);
preparedStatement.setString(1, nombre1);
preparedStatement.setInt(2, edad1);
preparedStatement.addBatch();

preparedStatement = connection.prepareStatement(sqlInsert2);
preparedStatement.setString(1, nombre2);
preparedStatement.setInt(2, edad2);
preparedStatement.addBatch();

int[] filasAfectadas = preparedStatement.executeBatch();
```

- **DatabaseMetaData**: La interfaz DatabaseMetaData se utiliza para recuperar información sobre la base de datos, como las tablas, columnas, índices, esquemas, procedimientos almacenados, entre otros. Aunque no es una sentencia, se utiliza para obtener metadatos sobre la base de datos.

Código 8. DatabaseMetaData.

```
try (Connection connection = DriverManager.getConnection(jdbcUrl, usuario, contraseña)) {
    DatabaseMetaData metaData = connection.getMetaData();

    System.out.println("Nombre de la base de datos: " + metaData.getDatabaseProductName());
    System.out.println("Versión de la base de datos: " + metaData.getDatabaseProductVersion());
    // ... otras consultas a DatabaseMetaData
} catch (SQLException e) {
    e.printStackTrace();
}
```

Result Set

Un ResultSet en JDBC es una estructura de datos que representa los resultados de una consulta SQL realizada a una base de datos. El ResultSet es esencialmente una tabla virtual que contiene filas y columnas, y cada fila representa un registro de la base de datos que cumple con los criterios de la consulta. Cada columna del ResultSet representa un campo específico de los registros devueltos por la consulta [5].

Para obtener un ResultSet, se debe ejecutar una consulta SQL utilizando una conexión a la base de datos y un objeto Statement. El siguiente fragmento de código muestra un ejemplo de ResultSet.

Código 9. ResultSet.

```
try (Connection connection = DriverManager.getConnection(jdbcUrl, usuario, contraseña);
     Statement statement = connection.createStatement()) {

    String sqlQuery = "SELECT nombre, edad FROM tu_tabla";
    ResultSet resultSet = statement.executeQuery(sqlQuery);

    while (resultSet.next()) {
        String nombre = resultSet.getString("nombre");
        int edad = resultSet.getInt("edad");
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

- **ResultSetMetaData**: La interfaz ResultSetMetaData se utiliza para obtener información sobre las columnas de un conjunto de resultados (ResultSet). Puede ser útil cuando se necesita conocer detalles sobre las columnas devueltas por una consulta.

Código 10. ResultSetMetaData.

```
try (Connection connection = DriverManager.getConnection(jdbcUrl, usuario, contraseña;
     Statement statement = connection.createStatement();
     ResultSet resultSet = statement.executeQuery("SELECT * FROM tu_tabla")) {

    ResultSetMetaData metaData = resultSet.getMetaData();
```

```
        int columnCount = metaData.getColumnCount();

        for (int i = 1; i <= columnCount; i++) {
            System.out.println("Nombre de la columna " + i + ": " + metaData.getColumnName(i));
            System.out.println("Tipo de datos de la columna " + i + ": " + metaData.getColumnTypeName(i));
            // ... otras consultas a ResultSetMetaData
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

Mapping Between SQL & Java Data Types

Este tipo de mapeo se refiere a la relación entre los tipos de datos utilizados en SQL y los tipos de datos utilizados en el lenguaje de programación Java cuando se interactúa con una base de datos a través de JDBC. E mapeo de estos tipos de datos es fundamental para asegurarse de que los datos se puedan recuperar y manipular de manera efectiva entre la base de datos y la aplicación Java [6].

Entre estas relaciones podemos encontrar las siguiente:

Tipo de Datos SQL	Tipo de Datos Java	Objeto Mapeado Java
CHARACTER		String
VARCHAR		String
LONGVARCHAR		String
NUMERIC		java.math.BigDecimal
DECIMAL		java.math.BigDecimal
BIT	boolean	Boolean
TINYINT	byte	Integer
SMALLINT	short	Integer
INTEGER	int	Integer
BIGINT	long	Long
REAL	float	Float
FLOAT	double	Double
DOUBLE PRECISION	double	Double
BINARY		byte[]
VARBINARY		byte[]
LONGVARBINARY		byte[]
DATE		java.sql.Date
TIME		java.sql.Time
TIMESTAMP		java.sql.Timestamp

Tabla 1. Relación de los tipos de datos de SQL y Java.

El mapeo preciso entre los tipos de datos SQL y Java puede variar según el sistema de gestión de bases de datos (DBMS) específico que se esté utilizando, así como la configuración de JDBC. Por lo tanto, es importante consultar la documentación del DBMS y JDBC para conocer los mapeos exactos.

SQLException

SQLException se deriva de la clase java.sql.SQLException y es una subclase de la clase java.sql.Exception. Esta excepción se utiliza para capturar y manejar situaciones inesperadas o errores relacionados con la base de datos cuando se realizan operaciones de conexión, consulta o modificación de datos [7].

Código 11. Ejemplo ejecución de SQLException.

```
try {
    Connection connection = DriverManager.getConnection(jdbcUrl, usuario, contraseña);
    Statement statement = connection.createStatement();
    // Intenta ejecutar una consulta SQL incorrecta
    statement.executeQuery("SELECT * FROM tabla_inexistente");
} catch (SQLException e) {
    System.err.println("Error de base de datos: " + e.getMessage());
    System.err.println("Código de estado SQL: " + e.getSQLState());
    System.err.println("Código de error del DBMS: " + e.getErrorCode());
    e.printStackTrace();
}
```

Referencias

[1] «¿Qué es JDBC (Java Database Connectivity)?,» Keepcoding, 21 Abril 2023 . [En línea]. Available: https://keepcoding.io/blog/jdbc-java-database-connectivity/. [Último acceso: 2 Noviembre 2023].

[2] «Class DriverManager,» Oracle, [En línea]. Available: https://docs.oracle.com/javase/8/docs/api/java/sql/DriverManager.html. [Último acceso: 2 Noviembre 2023].

[3] «Establishing JDBC Connection in Java,» Geeksforgeeks, 28 Marzo 2023. [En línea]. Available: https://www.geeksforgeeks.org/establishing-jdbc-connection-in-java/?ref=ml_lbp. [Último acceso: 2 Noviembre 2023].

[4] «Types of Statements in JDBC,» Geeksforgeeks, 02 Septiembre 2022. [En línea]. Available: https://www.geeksforgeeks.org/types-of-statements-in-jdbc/. [Último acceso: 2 Noviembre 2023].

[5] «Interface ResultSet,» Oracle, [En línea]. Available: https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html. [Último acceso: 2 Noviembre 2023].

[6] D. K. Barry, «Mapping SQL and Java Data Types,» Service Architecture, [En línea]. Available: https://www.service-architecture.com/articles/database/mapping-sql-and-java-data-types.html. [Último acceso: 2 Noviembre 2023].

[7] «Handling SQLExceptions,» Oracle, [En línea]. Available: https://docs.oracle.com/javase/tutorial/jdbc/basics/sqlexception.html. [Último acceso: 2 Noviembre 2023].