Instituto Politécnico Nacional

Escuela Superior de Cómputo

Web Client and Backend Development Frameworks

HW2 : Reporte de Lectura Acceso a Datos con JDBC

Profesor: M. en C. José Asunción Enríquez Zárate

Alumno: Leonardo Daniel Dominguez Olvera

ldominguezo1801@alumno.ipn.mx

7CM1

March 7, 2025

Contents

1 Introducción

Java Database Connectivity (JDBC) es una API de Java que permite a las aplicaciones interactuar con bases de datos relacionales. Su objetivo es proporcionar un estándar que permita a los desarrolladores ejecutar consultas, actualizar registros y administrar la conexión entre una aplicación Java y una base de datos sin depender de una implementación específica.

JDBC se basa en el uso de controladores (drivers) que traducen las solicitudes de Java a instrucciones que la base de datos pueda entender. Entre los principales controladores se encuentran:

- **Driver JDBC-ODBC Bridge:** Un puente entre JDBC y ODBC.
- **Driver Nativo:** Utiliza APIs específicas del proveedor de la base de datos.
- **Driver de Protocolo de Red:** Comunica la aplicación Java con un servidor de base de datos a través de la red.
- **Driver 100% Java (Thin Driver):** Un puente entre JDBC y ODBC.

El uso de JDBC es crucial en el desarrollo de aplicaciones empresariales, permitiendo a los sistemas gestionar grandes volúmenes de información de manera segura y eficiente.

2 Desarrollo

Para que una aplicación Java pueda interactuar con una base de datos, es necesario seguir los siguientes pasos:

2.1 Conexión a una Base de Datos con JDBC

2.1.1 *Cargar el Controlador JDBC*

Antes de establecer una conexión, se debe cargar el controlador JDBC correspondiente a la base de datos que se va a utilizar, este proceso se realizaba manualmente con la siguiente línea de código:

```
1 Class.forName("com.mysql.cj.jdbc.Driver");
```

Codigo: Se carga el controlador JDBC de MySQL en memoria. El método **Class.forName()** permite que Java reconozca el controlador y lo registre automáticamente para su uso. A partir de JDBC 4.0, este paso no es obligatorio si el controlador está en el classpath.

2.1.2 *Establecer la Conexión con la Base de Datos*

```
1  String url = "jdbc:mysql://localhost:3306/nombre_base_datos";
2  String user = "user";
3  String password = "password";
4
5  try {
6      Connection conexion = DriverManager.getConnection(url, user, password);
7      System.out.println("Conexion exitosa");
8  } catch (SQLException e) {
9      e.printStackTrace();
10 }
```

Codigo: En este fragmento de código se define la URL de conexión, que contiene el tipo de base de datos (mysql), la ubicación (localhost), el puerto (3306) y el nombre de la base de datos. Luego, **DriverManager.getConnection()** establece la conexión con las credenciales proporcionadas. Si la conexión es exitosa, se imprime un mensaje, y si hay un error, se captura la excepción **SQLException** para mostrar información del problema.

Si la conexión es exitosa, se pueden ejecutar consultas y manipular los datos almacenados.

2.1.3 *Crear y Ejecutar una Consulta SQL*

```
1 Statement stmt = conexion.createStatement();
2 ResultSet rs = stmt.executeQuery("SELECT * FROM empleados");
3
4 while (rs.next()) {
5     System.out.println("ID: " + rs.getInt("id") + ", Nombre: " + rs.getString("nombre"));
6 }
```

Codigo: Aquí se crea un objeto **Statement**, que permite ejecutar consultas SQL. La consulta **SELECT * FROM** empleados obtiene todos los registros de la tabla empleados, y **executeQuery()** devuelve un objeto ResultSet con los resultados. Luego, se recorre cada fila del **ResultSet** con **rs.next()**, extrayendo los valores de las columnas id y nombre con **getInt()** y **getString()**, respectivamente.

2.1.4 *Cerrar la Conexión*

```
1 conexion.close();
```

Codigo: El método **close()** cierra la conexión con la base de datos, liberando los recursos utilizados. Es una buena práctica cerrar la conexión cuando ya no se necesite, para evitar problemas de rendimiento o bloqueos en la base de datos.

2.2 Operaciones SQL con JDBC

2.2.1 *Consulta de Datos (SELECT)*

```
String consulta = "SELECT id, nombre, salario FROM empleados";
Statement stmt = conexion.createStatement();
ResultSet rs = stmt.executeQuery(consulta);

while (rs.next()) {
    System.out.println("ID: " + rs.getInt("id") + ", Nombre: " +
    rs.getString("nombre")
    + ", Salario: " + rs.getDouble("salario"));
}
```

Codigo: Se define una consulta SQL para obtener el **id**, **nombre** y **salario** de la tabla **empleados**. Se ejecuta la consulta con **executeQuery()**, y se recorren los resultados con **rs.next()**, imprimiendo cada fila obtenida.

2.2.2 *Inserción de Datos (INSERT)*

```
String insert = "INSERT INTO empleados (nombre, salario) VALUES (?, ?)";
PreparedStatement pstmt = conexion.prepareStatement(insert);
pstmt.setString(1, "Juan Perez");
pstmt.setDouble(2, 45000);
pstmt.executeUpdate();
```

Codigo: El uso de **PreparedStatement** permite insertar datos en la base de datos de manera segura, evitando inyección SQL. Los signos de interrogación (**?**) son marcadores de posición que se reemplazan con los valores correspondientes mediante **setString()** y **setDouble()**. Finalmente, **executeUpdate()** ejecuta la inserción.

2.2.3 *Actualización de Datos (UPDATE))*

```
String update = "UPDATE empleados SET salario = ? WHERE nombre = ?";
PreparedStatement pstmt = conexion.prepareStatement(update);
pstmt.setDouble(1, 50000);
pstmt.setString(2, "Juan Perez");
pstmt.executeUpdate();
```

Codigo: Actualiza el salario de un empleado con un nombre específico. Se utiliza **PreparedStatement** para evitar errores de seguridad y optimizar el rendimiento. El método **executeUpdate()** ejecuta la consulta de actualización.

2.2.4 *Eliminación de Datos (DELETE)*

```
String delete = "DELETE FROM empleados WHERE nombre = ?";
PreparedStatement pstmt = conexion.prepareStatement(delete);
pstmt.setString(1, "Juan Perez");
pstmt.executeUpdate();
```

Codigo: En este fragmento de código se elimina un registro de la tabla empleados donde el **nombre** sea **"Juan Perez"**. El **PreparedStatement** asegura que el valor del parámetro se pase de manera segura.

2.3 Manejo de Transacciones en JDBC

2.3.1 *Desactivar el AutoCommit*

Por defecto, cada operación SQL se ejecuta y confirma automáticamente. Desactivar autoCommit permite agrupar varias operaciones en una transacción, garantizando que todas se completen correctamente antes de confirmarlas.

```
conexion.setAutoCommit(false);
```

Codigo: Desactiva la confirmación automática de cambios en la base de datos.

2.3.2 *Ejecutar Múltiples Operaciones en una Transacción*

```
try {
    PreparedStatement pstmt1 = conexion.prepareStatement("UPDATE empleados SET salario = ?
    WHERE id = ?");
    pstmt1.setDouble(1, 60000);
    pstmt1.setInt(2, 1);
    pstmt1.executeUpdate();

    PreparedStatement pstmt2 = conexion.prepareStatement("INSERT INTO empleados
    (nombre, salario) VALUES (?, ?)");
    pstmt2.setString(1, "Carlos Lopez");
    pstmt2.setDouble(2, 48000);
    pstmt2.executeUpdate();

    conexion.commit();
} catch (SQLException e) {
    conexion.rollback();
    e.printStackTrace();
}
```

Codigo: Este código agrupa una actualización y una inserción dentro de una misma transacción. Si ambas operaciones se ejecutan correctamente, se confirma la transacción con commit(). Si ocurre un error, rollback() deshace todos los cambios realizados antes del fallo, asegurando la consistencia de los datos.

3 Conclusión

JDBC es una API fundamental para la conexión entre Java y bases de datos relacionales. Permite manejar datos de forma segura y eficiente, ofreciendo herramientas para realizar consultas, inserciones, actualizaciones y eliminaciones con facilidad.

El uso de PreparedStatement mejora la seguridad y el rendimiento de las consultas SQL. Además, la gestión de transacciones con commit() y rollback() es crucial para asegurar la integridad de los datos en operaciones críticas.

Leonardo Daniel Dominguez Olvera

4 Referencias Bibliográficas

References

[Oracle, 2018] Oracle. *Web Component Development With Servlet and JSP Technologies* Oracle.

[Devlin, 2002] Edgar Martinez, Tom McGinn, Eduardo Moranchel, Anjana Shenoy, Michael Williams. *Java EE 7: Back-end Server Application Development* Oracle, 2016.

[Jendrock, 2014] Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. *Java Platform, Enterprise Edition (Java EE) 8 The Java EE Tutorial* Oracle, 2017.