CRUD

CRUD es el conjunto de operaciones básicas utilizadas para manejar la persistencia de datos en bases de datos. Estas operaciones corresponden a:

- **Create (Crear)**: Insertar nuevos datos en la base de datos.
- **Read (Leer)**: Consultar y recuperar datos desde la base de datos.
- **Update (Actualizar)**: Modificar datos existentes en la base de datos.
- **Delete (Eliminar)**: Eliminar datos de la base de datos.

A continuación, se detalla cómo implementar estas operaciones utilizando JDBC en Java.

1. Configuración Inicial (Conexión a la Base de Datos)

Antes de realizar cualquier operación CRUD, es necesario establecer una conexión con la base de datos:

```
import java.sql.*;
public class ConexionDB {
    public static Connection conectar() throws SQLException {
        String url = "jdbc:mysql://localhost:3306/mi_base_de_datos";
        String usuario = "usuario";
        String contrasena = "contrasena";
        return DriverManager.getConnection(url, usuario, contrasena);
    }
}
```

2. Crear (Create)

Para **insertar** datos en la base de datos, se utiliza el comando SQL `INSERT`. Usamos `PreparedStatement` para evitar vulnerabilidades como la **inyección SQL**.

```
public class CRUD {
    public void crear(String nombre, int edad) {
    String sql = "INSERT INTO alumnos (nombre, edad) VALUES (?, ?)";
try (Connection conn = ConexionDB.conectar();a
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
pstmt.setString(1, nombre);
        pstmt.setInt(2, edad);
        pstmt.executeUpdate();
        System.out.println("Alumno creado con éxito.");
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
        }
    }
```

3. Leer (Read)

Para **consultar** datos desde la base de datos, usamos el comando SQL `SELECT`. El resultado se obtiene en un `ResultSet` que permite iterar sobre las filas devueltas.

```
public class CRUD {
    public void leer() {
        String sql = "SELECT * FROM alumnos";
        try (Connection conn = ConexionDB.conectar();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                String nombre = rs.getString("nombre");
                int edad = rs.getInt("edad");
                System.out.println("Nombre: " + nombre + ", Edad: " + edad);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Para realizar una consulta con parámetros, utilizamos `PreparedStatement`:

```
public class CRUD {
    public void leerPorId(int id) {
        String sql = "SELECT * FROM alumnos WHERE id = ?";
        try (Connection conn = ConexionDB.conectar();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()) {
                String nombre = rs.getString("nombre");
                int edad = rs.getInt("edad");
                System.out.println("Nombre: " + nombre + ", Edad: " + edad);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

4. Actualizar (Update)

Para **modificar** datos en la base de datos, utilizamos el comando SQL `UPDATE`. Nuevamente, usamos `PreparedStatement` para proteger la consulta de inyecciones SQL.

```
public class CRUD {
    public void actualizar(int id, String nuevoNombre, int nuevaEdad) {
        String sql = "UPDATE alumnos SET nombre = ?, edad = ? WHERE id = ?";
        try (Connection conn = ConexionDB.conectar();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setString(1, nuevoNombre);
                pstmt.setInt(2, nuevaEdad);
                pstmt.setInt(3, id);
                int filasAfectadas = pstmt.executeUpdate();
                if (filasAfectadas > 0) {
                        System.out.println("Alumno actualizado con éxito.");
                } else {
                    System.out.println("No se encontró el alumno");
                }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

5. Eliminar (Delete)

Para **eliminar** datos de la base de datos, utilizamos el comando SQL `DELETE`. Este comando puede eliminar registros específicos si se proporciona un identificador único.

```
public class CRUD {
    public void eliminar(int id) {
        String sql = "DELETE FROM alumnos WHERE id = ?";
        try (Connection conn = ConexionDB.conectar();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            int filasAfectadas = pstmt.executeUpdate();
            if (filasAfectadas > 0) {
                System.out.println("Alumno eliminado con éxito.");
            } else {
                System.out.println("No se encontró el alumno");
```

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

6. Flujo Completo CRUD

A continuación, se muestra cómo se integran las operaciones CRUD en un flujo general:

```
public class CRUD {
    public static void main(String[] args) {
        CRUD crud = new CRUD();                // Crear un nuevo alumno
        crud.crear("Juan", 25);                // Leer todos los alumnos
        crud.leer();                           // Actualizar un alumno con ID
1
        crud.actualizar(1, "Juan Pérez", 26);  // Eliminar un alumno con ID 1
        crud.eliminar(1);
    }
}
```

7. Consideraciones Adicionales

- **Manejo de excepciones**: En todos los métodos se maneja la excepción `SQLException`, que es la excepción base para errores relacionados con SQL en JDBC.
- **Cierre de recursos**: Es importante siempre cerrar las conexiones, `PreparedStatement` y `ResultSet` para evitar fugas de memoria. Esto se puede hacer utilizando el bloque `try-with-resources`, que se encarga de cerrarlos automáticamente.
- **Uso de `PreparedStatement`**: Siempre que sea posible, usa `PreparedStatement` en lugar de `Statement` para proteger las consultas contra **inyección SQL**.