Prueba de Argoti

Carlos Argoti Patiño

Octubre 2024

1 Introducción

Este documento detalla los pasos necesarios para ejecutar un proceso de inserción de datos en MySQL utilizando un programa Java. El objetivo es llevar a cabo la inserción de datos de forma programática, desde la creación de tablas hasta la ejecución de un programa Java que realiza dicha inserción.

2 Paso 1: Crear las tablas necesarias

El primer paso es crear las tablas en la base de datos MySQL. Estas tablas fueron creadas para almacenar información sobre empresas, aplicaciones, versiones y asociaciones entre empresas y versiones.

Aquí está el SQL utilizado para crear las tablas:

```sql
CREATE TABLE company (
    id_company          INT PRIMARY KEY,
    codigo_company      VARCHAR(50) NOT NULL UNIQUE,
    name_company        VARCHAR(255),
    description_company TEXT
);

CREATE TABLE application (
    app_id              INT PRIMARY KEY,
    app_code            VARCHAR(50) NOT NULL UNIQUE,
    app_name            VARCHAR(255),
    app_description     TEXT
);

CREATE TABLE version (
    version_id          INT PRIMARY KEY,
    app_id              INT NOT NULL,
    version             VARCHAR(50),
    version_description TEXT,
    FOREIGN KEY (app_id) REFERENCES application(app_id)
);

CREATE TABLE version_company (
    version_company_id          INT PRIMARY KEY,
    company_id                  INT NOT NULL,
    version_id                  INT NOT NULL,
    version_company_description TEXT,
    FOREIGN KEY (company_id) REFERENCES company(id_company),
    FOREIGN KEY (version_id) REFERENCES version(version_id)
);
```

Listing 1: Creación de tablas en MySQL

3 Paso 2: Instalación del conector MySQL JDBC en Ubuntu

Para interactuar con MySQL desde un programa Java, es necesario instalar el conector JDBC de MySQL. El archivo del conector en formato `.deb` fue instalado de la siguiente manera:

Listing 2: Instalación del conector MySQL JDBC

```
sudo dpkg -i mysql-connector-j_9.1.0-1ubuntu24.10_all.deb
```

Posteriormente, se verificó si había dependencias faltantes con el comando:

Listing 3: Verificación de dependencias

```
sudo apt --fix-broken install
```

El archivo `mysql-connector-java.jar` se ubicó en el directorio `/usr/share/java/`, el cual será utilizado para compilar y ejecutar el programa Java.

4 Paso 3: Compilación y ejecución del programa Java

Con el conector MySQL JDBC instalado, el siguiente paso fue compilar y ejecutar el programa Java. A continuación, se describe el código fuente del programa Java que se encargó de insertar los datos en las tablas de la base de datos MySQL:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

public class InsertarDatos {
    private static Object[][] datosTmp = {
        {1, "COMP001", "Compa a Alpha", "Descripci n Alpha", 1, 1, "v1.0",
            "Versi n inicial", 1, "Versi n personalizada para Alpha",
            "APP001", "Aplicaci n Uno", "Descripci n de la aplicaci n uno"},
        {2, "COMP002", "Compa a Beta", "Descripci n Beta", 2, 2, "v2.0",
            "Segunda versi n", 2, "Versi n personalizada para Beta",
            "APP002", "Aplicaci n Dos", "Descripci n de la aplicaci n dos"}
    };

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduce el host (localhost por defecto): ");
        String host = scanner.nextLine();
        if (host.isEmpty()) {
            host = "localhost";
        }

        System.out.print("Introduce el usuario: ");
        String user = scanner.nextLine();

        System.out.print("Introduce la contrase a: ");
        String password = scanner.nextLine();

        System.out.print("Introduce el nombre de la base de datos: ");
        String database = scanner.nextLine();

        String url = "jdbc:mysql://" + host + "/" + database + "?useSSL=false";

        try (Connection connection = DriverManager.getConnection(url, user,
            password)) {
            System.out.println("Conexi n exitosa a la base de datos");

            for (Object[] registro : datosTmp) {
                insertarRegistro(connection, registro);
            }

            System.out.println("Inserciones completadas correctamente.");

        } catch (SQLException e) {
```

```
            System.err.println("Error al conectar o insertar los datos: " +
                e.getMessage());
        }
    }

    private static void insertarRegistro(Connection connection, Object[]
        registro) throws SQLException {
        String sqlCompany = "INSERT INTO company (id_company, codigo_company,
            name_company, description_company) "
                          + "VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE "
                          + "codigo_company = VALUES(codigo_company), "
                          + "name_company = VALUES(name_company), "
                          + "description_company = VALUES(description_company)";

        try (PreparedStatement stmt = connection.prepareStatement(sqlCompany)) {
            stmt.setInt(1, (Integer) registro[0]);
            stmt.setString(2, (String) registro[1]);
            stmt.setString(3, (String) registro[2]);
            stmt.setString(4, (String) registro[3]);
            stmt.executeUpdate();
        }

        String sqlApplication = "INSERT INTO application (app_id, app_code,
            app_name, app_description) "
                              + "VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE "
                              + "app_code = VALUES(app_code), "
                              + "app_name = VALUES(app_name), "
                              + "app_description = VALUES(app_description)";

        try (PreparedStatement stmt =
            connection.prepareStatement(sqlApplication)) {
            stmt.setInt(1, (Integer) registro[5]);
            stmt.setString(2, (String) registro[10]);
            stmt.setString(3, (String) registro[11]);
            stmt.setString(4, (String) registro[12]);
            stmt.executeUpdate();
        }

        String sqlVersion = "INSERT INTO version (version_id, app_id, version,
            version_description) "
                          + "VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE "
                          + "version = VALUES(version), "
                          + "version_description = VALUES(version_description)";

        try (PreparedStatement stmt = connection.prepareStatement(sqlVersion)) {
            stmt.setInt(1, (Integer) registro[4]);
            stmt.setInt(2, (Integer) registro[5]);
            stmt.setString(3, (String) registro[6]);
            stmt.setString(4, (String) registro[7]);
            stmt.executeUpdate();
        }

        String sqlVersionCompany = "INSERT INTO version_company
            (version_company_id, company_id, version_id,
            version_company_description) "
                                 + "VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE
                                     "
                                 + "version_company_description =
                                     VALUES(version_company_description)";

        try (PreparedStatement stmt =
            connection.prepareStatement(sqlVersionCompany)) {
            stmt.setInt(1, (Integer) registro[8]);
```

```
94              stmt.setInt(2, (Integer) registro[0]);
95              stmt.setInt(3, (Integer) registro[4]);
96              stmt.setString(4, (String) registro[9]);
97              stmt.executeUpdate();
98          }
99      }
100 }
```

Listing 4: Código fuente de InsertarDatos.java

Una vez que el archivo `InsertarDatos.java` estuvo listo, fue compilado utilizando el conector JDBC de MySQL instalado previamente:

Listing 5: Compilación del programa Java

```
javac -cp .:/usr/share/java/mysql-connector-java-9.1.0.jar InsertarDatos.java
```

Posteriormente, se ejecutó el programa:

Listing 6: Ejecución del programa Java

```
java -cp .:/usr/share/java/mysql-connector-java-9.1.0.jar InsertarDatos
```

Finalmente, se introdujeron las credenciales de la base de datos y los datos fueron insertados correctamente en las tablas.

5 Paso 4: Implementación de la API REST

En este paso, se desarrolló una API REST utilizando Spring Boot para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la tabla `company`. Además, se implementó un método GET adicional que, al recibir el código de una compañía, devuelve un JSON con los campos `codigo_company`, `name_company`, `app_name` y `version`.

5.1 Estructura del Proyecto

La estructura del proyecto Maven está organizada de la siguiente manera:

- `src/main/java/com/example/mi_maven_app/controller`: Contiene los controladores REST.
- `src/main/java/com/example/mi_maven_app/entity`: Define las entidades JPA que representan las tablas de la base de datos.
- `src/main/java/com/example/mi_maven_app/repository`: Contiene las interfaces de repositorio para interactuar con la base de datos.
- `src/main/resources`: Almacena archivos de configuración como `application.properties`.

5.2 Controlador REST: `CompanyController.java`

Se creó el controlador REST `CompanyController` que maneja las solicitudes HTTP para realizar operaciones CRUD en la entidad `Company`. Además, se agregó un método GET adicional para obtener detalles específicos de una compañía basada en su código.

```
1  package com.example.mi_maven_app.controller;
2
3  import com.example.mi_maven_app.entity.Company;
4  import com.example.mi_maven_app.entity.VersionCompany;
5  import com.example.mi_maven_app.repository.CompanyRepository;
6  import com.example.mi_maven_app.repository.VersionCompanyRepository;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.http.ResponseEntity;
9  import org.springframework.web.bind.annotation.*;
10 import java.util.*;
11
12 @RestController
```

```java
@RequestMapping("/api/company")
public class CompanyController {

    @Autowired
    private CompanyRepository companyRepository;

    @Autowired
    private VersionCompanyRepository versionCompanyRepository;

    // M todo para obtener todas las compa  as
    @GetMapping
    public List<Company> getAllCompanies() {
        return companyRepository.findAll();
    }

    // M todo para obtener una compa  a por su ID
    @GetMapping("/{id}")
    public ResponseEntity<Company> getCompanyById(@PathVariable Long id) {
        Optional<Company> company = companyRepository.findById(id);
        if (company.isPresent()) {
            return ResponseEntity.ok(company.get());
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    // M todo para crear una nueva compa  a
    @PostMapping
    public Company createCompany(@RequestBody Company company) {
        return companyRepository.save(company);
    }

    // M todo para actualizar una compa  a existente
    @PutMapping("/{id}")
    public ResponseEntity<Company> updateCompany(@PathVariable Long id,
        @RequestBody Company companyDetails) {
        Optional<Company> optionalCompany = companyRepository.findById(id);
        if (!optionalCompany.isPresent()) {
            return ResponseEntity.notFound().build();
        }

        Company company = optionalCompany.get();
        company.setCodigoCompany(companyDetails.getCodigoCompany());
        company.setNameCompany(companyDetails.getNameCompany());
        company.setDescriptionCompany(companyDetails.getDescriptionCompany());

        Company updatedCompany = companyRepository.save(company);
        return ResponseEntity.ok(updatedCompany);
    }

    // M todo para eliminar una compa  a
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteCompany(@PathVariable Long id) {
        Optional<Company> company = companyRepository.findById(id);
        if (!company.isPresent()) {
            return ResponseEntity.notFound().build();
        }

        companyRepository.delete(company.get());
        return ResponseEntity.noContent().build();
    }

    // M todo GET adicional para obtener detalles de la compa  a por c digo
```

```
    @GetMapping("/detalle/{codigoCompany}")
    public ResponseEntity<Map<String, Object>> getCompanyDetails(@PathVariable
        String codigoCompany) {
        Optional<Company> companyOptional =
            companyRepository.findByCodigoCompany(codigoCompany);
        if (!companyOptional.isPresent()) {
            return ResponseEntity.notFound().build();
        }

        Company company = companyOptional.get();
        Optional<VersionCompany> versionCompanyOptional =
            versionCompanyRepository.findByCompany(company);

        if (!versionCompanyOptional.isPresent()) {
            return ResponseEntity.notFound().build();
        }

        VersionCompany versionCompany = versionCompanyOptional.get();
        String appName =
            versionCompany.getVersion().getApplication().getAppName();
        String version = versionCompany.getVersion().getVersion();

        Map<String, Object> response = new HashMap<>();
        response.put("codigo_company", company.getCodigoCompany());
        response.put("name_company", company.getNameCompany());
        response.put("app_name", appName);
        response.put("version", version);

        return ResponseEntity.ok(response);
    }
}
```

Listing 7: Código fuente de `CompanyController.java`

5.3 Configuración de la Aplicación: `application.properties`

Se configuró el archivo `application.properties` para establecer la conexión con la base de datos MySQL.

```
spring.datasource.url=jdbc:mysql://localhost:3306/tu_base_de_datos?useSSL=false&serverTimezon
spring.datasource.username=tu_usuario
spring.datasource.password=tu_contrase a
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Listing 8: Configuración de `application.properties`

5.4 Entidades y Repositorios

Las entidades `Company`, `Application`, `Version` y `VersionCompany` ya fueron definidas en pasos anteriores. Los repositorios `CompanyRepository` y `VersionCompanyRepository` permiten la interacción con la base de datos.

5.5 Pruebas de la API REST

Para verificar el correcto funcionamiento de la API REST, se realizaron pruebas utilizando herramientas como `Postman` o `cURL`. A continuación, se muestran ejemplos de cómo interactuar con la API.

5.5.1 Crear una Nueva Compañía

```
POST /api/company
Content-Type: application/json

{
    "codigoCompany": "COMP003",
    "nameCompany": "Compa  a Gamma",
    "descriptionCompany": "Descripci n Gamma"
}
```

Listing 9: Solicitud POST para crear una compañía

5.5.2 Obtener Todas las Compañías

```
GET /api/company
```

Listing 10: Solicitud GET para obtener todas las compañías

5.5.3 Obtener una Compañía por ID

```
GET /api/company/1
```

Listing 11: Solicitud GET para obtener una compañía por ID

5.5.4 Actualizar una Compañía

```
PUT /api/company/1
Content-Type: application/json

{
    "codigoCompany": "COMP001",
    "nameCompany": "Compa  a Alpha Actualizada",
    "descriptionCompany": "Descripci n Alpha Actualizada"
}
```

Listing 12: Solicitud PUT para actualizar una compañía

5.5.5 Eliminar una Compañía

```
DELETE /api/company/1
```

Listing 13: Solicitud DELETE para eliminar una compañía

5.5.6 Obtener Detalles de una Compañía por Código

```
GET /api/company/detalle/COMP001
```

Listing 14: Solicitud GET para obtener detalles de una compañía por código

Respuesta Esperada:

```
{
    "codigo_company": "COMP001",
    "name_company": "Compa  a Alpha",
    "app_name": "Aplicaci n Uno",
    "version": "v1.0"
}
```

Listing 15: Respuesta JSON

5.6 Consideraciones Finales

- **Validación de Datos:** Es recomendable agregar validaciones a las entidades y controlar posibles excepciones para mejorar la robustez de la API.

- **Seguridad:** Para entornos de producción, implementar medidas de seguridad como autenticación y autorización es esencial.

- **Documentación:** Utilizar herramientas como Swagger para generar documentación interactiva de la API facilita su uso y mantenimiento.

5.7 Compilación y Ejecución de la Aplicación

Una vez implementada la API REST, se compiló y ejecutó la aplicación utilizando Maven.

Listing 16: Compilación y ejecución de la aplicación

```
mvn clean install
mvn spring-boot:run
```

La aplicación se iniciará y estará disponible en `http://localhost:8080/api/company`.

6 Conclusiones

En este proyecto, se logró crear una API REST completa que permite realizar operaciones CRUD en la tabla `company` de una base de datos MySQL. Además, se implementó un método GET específico que, mediante el código de una compañía, devuelve información detallada relacionada con la aplicación y su versión asociada. Este enfoque modular facilita la gestión y expansión futura de la API, permitiendo integrar nuevas funcionalidades conforme se requiera.