**Guidance of the joints in sql using spring and hibernate project**

## **📂 Project Title: springApiJoint – Spring Boot One-to-Many with SQL Join (Customer + Orders)**

### **🛠 1. Tools & Technologies Used**

* **IDE: IntelliJ IDEA**
* **Java: Version 17 (OpenJDK)**
* **Spring Boot: 3.5.0**
* **Hibernate (JPA): ORM for database mapping**
* **MySQL: 8.x (local and Docker)**
* **Maven: For dependency management and packaging**
* **Docker: For containerizing the app and database**
* **Postman: For API testing**

### **🧱 2. Initial Setup (Spring Boot Project)**

1. **Created Maven-based Spring Boot project in IntelliJ IDEA**
2. **Set package name: com.vwaves.curdoparation.springApiJoint**
3. **Configured pom.xml with these dependencies:**
   * **spring-boot-starter-data-jpa**
   * **spring-boot-starter-web**
   * **mysql-connector-j**
   * **Spring-boot-starter-test**

**📜 3. application.properties**

**Configured DB connection:**

**properties**

**CopyEdit**

**spring.datasource.url=jdbc:mysql://localhost:3306/curdapi**

**spring.datasource.username=root**

**spring.datasource.password=your\_password**

**spring.jpa.hibernate.ddl-auto=update**

**spring.jpa.show-sql=true**

**spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect**

### **👨‍💻 4. Model Files Created**

#### **✅ Customer.java**

* **Annotated with @Entity**
* **One-to-many relationship: List<Orders> orders**

#### **✅ Orders.java**

* **Annotated with @Entity**
* **Many-to-one relationship: Customer customer**

### **📦 5. DTO Created**

#### **✅ CustomerOrdersDTO.java**

* **Custom DTO to combine data from both Customer and Orders**

### 

### 

### **🗃 6. Repositories Created**

* **CustomerRepository extends JpaRepository<Customer, Long>**
* **OrdersRepository extends JpaRepository<Orders, Long>**

### **🎮 7. Controller Created**

#### **✅ MainController.java with endpoints:**

* **GET /api/customers – Get all customers**
* **GET /api/orders – Get all orders**
* **GET /api/joined – Get joined customer + order data (DTO)**
* **GET /api/customers/{id} – Get customer by ID**
* **POST /api/customers – Create customer**
* **POST /api/orders – Create order**
* **PUT /api/customers/{id} – Update customer**
* **DELETE /api/customers/{id} – Delete customer**

### **🧪 8. API Testing with Postman**

* **Tested all CRUD operations**
* **Validated JSON request and response**
* **Handled errors (404, 400, etc.)**

### 

### 

### **🛠 9. Maven Errors Solved**

* **Fixed version mismatch**
* **Resolved missing dependencies**
* **Packaged project using:**

**bash**

**CopyEdit**

**mvn clean package -DskipTests**

### **🐳 10. Dockerization**

#### **✅ Dockerfile created:**

**dockerfile**

**CopyEdit**

**FROM openjdk:17-jdk-alpine**

**WORKDIR /app**

**COPY target/springApiJoint-0.0.1-SNAPSHOT.jar app.jar**

**ENTRYPOINT ["java", "-jar", "app.jar"]**

#### **✅ docker-compose.yml created:**

* **Defined 2 services:**
  + **mysql**
  + **springboot-app**
* **Mapped ports 3306 and 8080**
* **Set environment variables**

#### **✅ Updated application.properties for Docker:**

**properties**

**CopyEdit**

**spring.datasource.url=jdbc:mysql://mysql:3306/curdapi**

#### 

#### **✅ Docker Commands Used:**

**bash**

**CopyEdit**

**docker-compose up --build**

**docker ps**

**docker logs -f springboot-app-container**

### **🧼 11. Clean Up and Final Testing**

* **Verified API via Postman**
* **Ensured DB connection in container**
* **Ensured data persistence via Docker volumes**