Answer the following:

1. **Explain the steps to establish MySQL connectivity in a project.**

Ans-

1. Add MySQL Dependency

Include the MySQL JDBC driver dependency in your pom.xml (for Maven) or build.gradle (for Gradle).

For Maven:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

For Gradle:

```
implementation 'mysql:mysql-connector-java'
```

---

2. Configure the application.properties File

Set up the database connection details in the application.properties or application.yml file.

Example (application.properties):

```
spring.datasource.url=jdbc:mysql://localhost:3306/your_database_name
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Key Properties:

- spring.datasource.url: The JDBC URL for the MySQL database.

- spring.datasource.username and spring.datasource.password: MySQL credentials.
- spring.datasource.driver-class-name: MySQL driver class.
- spring.jpa.hibernate.ddl-auto: Configures how Hibernate should handle database schema changes (update, create, create-drop, or none).
- spring.jpa.properties.hibernate.dialect: Specifies the SQL dialect to optimize queries for MySQL.

---

3. Create the Database

Create the database in MySQL using a SQL command or a database management tool:

CREATE DATABASE your_database_name;

---

4. Test the Connection

Run the Spring Boot application. Spring Boot will automatically configure a DataSource and test the connection to MySQL using the details provided in the application.properties.

2. **Explain the role of the application.properties file in MySQL connectivity.**

Ans-

Role of the application.properties File in MySQL Connectivity

The application.properties file plays a crucial role in connecting a Spring Boot application to a MySQL database. Here's how:

1. **Centralized Configuration:**
    - It acts as the central location for specifying database connection details, such as the JDBC URL, username, password, and driver class.
2. **Simplifies Database Integration:**
    - Spring Boot uses the properties to auto-configure the database connection and eliminate the need for boilerplate code in the application.
3. **Hibernate Integration:**
    - Properties like spring.jpa.hibernate.ddl-auto and spring.jpa.properties.hibernate.dialect integrate the Hibernate ORM framework, making it easier to interact with the database.
4. **Flexibility:**
    - By externalizing configuration, you can easily switch between different environments (development, testing, production) by providing different property files or profiles.
5. **Error Handling:**
    - Spring Boot validates the provided configurations and throws descriptive errors if any property is missing or incorrect, ensuring smooth connectivity.