How to Make a Project Using Spring Boot, MySQL, Spring Data JPA, and Maven?

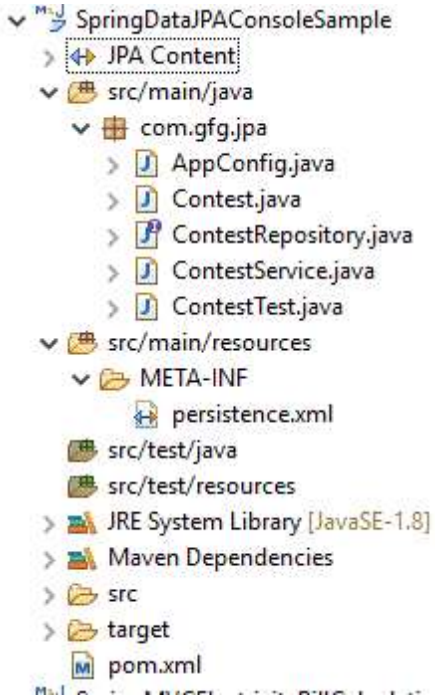
Last Updated : 23 Jul, 2025

For the sample project, below mentioned tools got used

- Java 8
- Eclipse IDE for development
- Hibernate ORM, Spring framework with Spring Data JPA
- MySQL database, MySQL Connector Java as JDBC driver.

Example Project Using Spring Boot, MySQL, Spring Data JPA, and Maven

Project Structure:



As this is getting prepared as a maven project, all dependencies are specified in pom.xml

pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.gfg</groupId>
    <artifactId>SpringDataJPAConsoleSample</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <properties>
     <maven.compiler.source>1.8</maven.compiler.source>
     <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <!--  Spring framework with support for Spring Data JPA -->
         <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.4.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>5.1.4.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.1.Final</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-jpa</artifactId>
            <version>2.1.4.RELEASE</version>
        </dependency>
        <!--  Spring framework with support for Spring Data JPA -->
        <!-- MySQL dependency -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.14</version>
        </dependency>
    </dependencies>
</project>
```

As we are connecting with MySQL, we need to create the database and table for use in the project

```
create database geeksforgeeks;  # Creation
use geeksforgeeks #Make the database active

CREATE TABLE `Contest` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `contestName` varchar(45) NOT NULL,
  `contestDescription` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;
```

Database connection properties are set in src/main/resources/persistence.xml

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
xmlns="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/persistence/index.html"
    xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/persistence/index.html

http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/persistence/index.html/persistence_2_1.xsd"
    version="2.1">
    <!-- Change the username and password with appropriate values -->
    <persistence-unit name="GeeksDB">
        <properties>
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/geeksforgeeks?useSSL=false" />
            <property name="javax.persistence.jdbc.user" value="***" />
            <property name="javax.persistence.jdbc.password" value="***" />
            <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
        </properties>
    </persistence-unit>

</persistence>
```

Configuration of EntityManagerFactory and TransactionManager

ContestAppConfig.java

```
import javax.persistence.EntityManagerFactory;
```

```
import javax.persistence.EntityManagerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalEntityManagerFactoryBean;

@Configuration
// Scan the repositories that is present under com.gfg.jpa
@EnableJpaRepositories(basePackages = {"com.gfg.jpa"})
public class ContestAppConfig {
    @Bean
    // LocalEntityManagerFactoryBean sets up an EntityManagerFactory to work
with GeeksDB
    public LocalEntityManagerFactoryBean entityManagerFactory() {
        LocalEntityManagerFactoryBean factoryBean = new
LocalEntityManagerFactoryBean();
        factoryBean.setPersistenceUnitName("GeeksDB");
        return factoryBean;
    }

    @Bean
    // Transaction manager for the configured EntityManagerFactory,
    public JpaTransactionManager transactionManager(EntityManagerFactory
entityManagerFactory) {
        JpaTransactionManager transactionManager = new
JpaTransactionManager();
        transactionManager.setEntityManagerFactory(entityManagerFactory);
        return transactionManager;
    }
}
```

Let's implement the model class.

Contest.java

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

// It maps with db table Contest
@Entity
public class Contest {
    // The @Id and @GeneratedValue annotations map
      // the field id to the primary key column of the table.
      // Suppose that all the fields of the class have
      // same name as the column names in the database table.
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String contestName;
    private String contestDescription;

    protected Contest() {
```

```
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getContestName() {
        return contestName;
    }

    public void setContestName(String contestName) {
        this.contestName = contestName;
    }

    public String getContestDescription() {
        return contestDescription;
    }

    public void setContestDescription(String contestDescription) {
        this.contestDescription = contestDescription;
    }

    @Override
    public String toString() {
        return "Contest [contestName=" + contestName + ",
contestDescription=" + contestDescription + "]";
    }

}
```

ContestRepository.java

Instead of writing a generic DAO class, a simple interface like below is enough. It extends CrudRepository defined by Spring Data JPA. Common CRUD operations like save(), findAll(), findById(), delete(), count() etc., are defined by the interface.

```
import java.util.List;
import org.springframework.data.repository.CrudRepository;

public interface ContestRepository extends CrudRepository<Contest, Long> {
    // We can add the required methods here
    List<Contest> findByContestName(String contestName);
}
```

ContestService.java

```
import java.util.List;
```

```
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service("contestService")
public class ContestService {

    @Autowired
    private ContestRepository contestRepository;

    public void test() {
        // Save a new contest
        Contest geekContest = new Contest();
        geekContest.setContestName("PremierLeague");
        geekContest.setContestDescription("Inviting Geeks To submit articles
in plenty");

        contestRepository.save(geekContest);

        // Find a contest by ID
        Optional<Contest> result = contestRepository.findById(1L);
        result.ifPresent(contest -> System.out.println(contest));

        // Find contest by contest name
        List<Contest> contests =
contestRepository.findByContestName("PremierLeague");
        contests.forEach(contest -> System.out.println(contest));

        // List all contests
        Iterable<Contest> iterator = contestRepository.findAll();
        iterator.forEach(contest -> System.out.println(contest));

        // Count number of contest
        long countOfContest = contestRepository.count();
        System.out.println("Number of contest held: " + countOfContest);
    }
}
```

We can test the same via a test file

ContestTest.java

```
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class ContestTest {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext appContext = new
AnnotationConfigApplicationContext();
        appContext.scan("com.gfg.jpa");
        appContext.refresh();
```

```
        ContestService contestService = (ContestService)
appContext.getBean("contestService");
        contestService.test();

        appContext.close();

    }

}
```

We can run the test file as an ordinary Java application

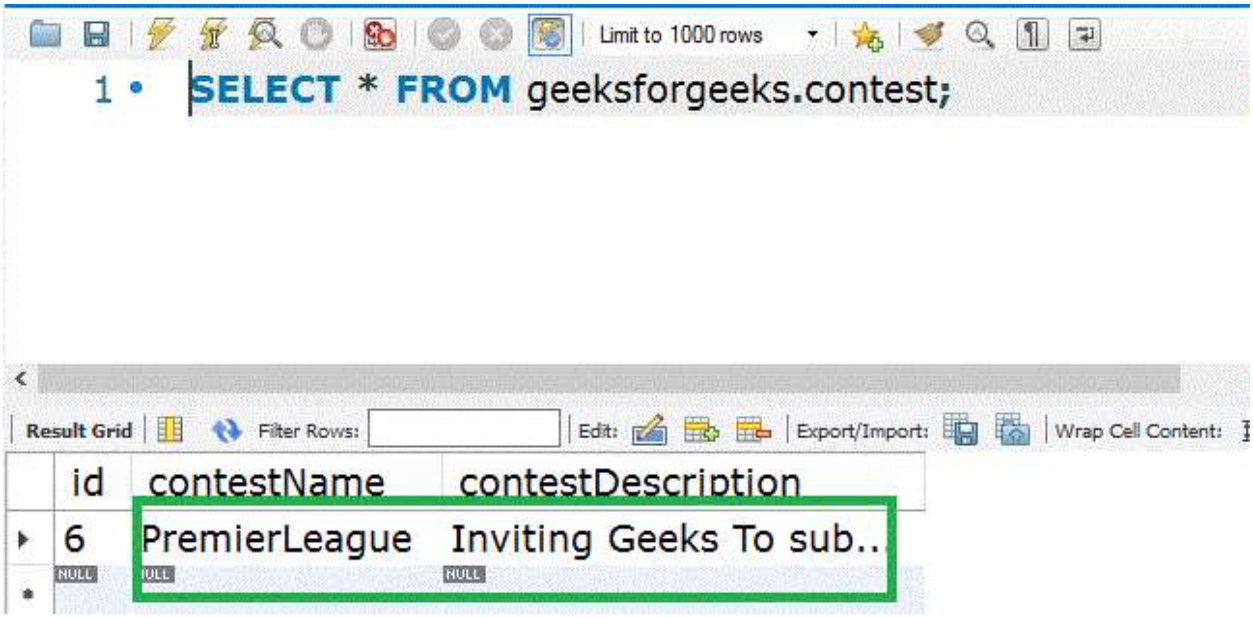
Output:

```
Hibernate:
    insert
    into
        Contest
        (contestDescription, contestName)
    values
        (?, ?)
Hibernate:
    select
        contest0_.id as id1_0_0_,
        contest0_.contestDescription as contestD2_0_0_,
        contest0_.contestName as contestN3_0_0_
    from
        Contest contest0_
    where
        contest0_.id=?
Hibernate:
    select
        contest0_.id as id1_0_,
        contest0_.contestDescription as contestD2_0_,
        contest0_.contestName as contestN3_0_
    from
        Contest contest0_
    where
        contest0_.contestName=?
Contest [contestName=PremierLeague, contestDescription=Inviting Geeks To submit articles in plenty]
Hibernate:
```

As we have tested via contestname as well, in the first output, we saw that one contest got inserted and the same is getting used to test while testing via contestname

```
Contest [contestName=PremierLeague, contestDescription=Inviting Geeks To submit articles in plenty]
Hibernate:
    select
        count(*) as col_0_0_
    from
        Contest contest0_
Number of contest held: 1
```

We can check the DB data also



***Note**: id field is auto-generated.*

Comment More info

Campus Training Program

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305