Hibernate Example using JPA and MySQL

Last Updated : 29 Aug, 2025

When we build Java applications, we often need to store data in a database. Doing this with plain JDBC means writing long SQL queries, handling connections and managing transactions manually which can get complicated.

This is where Hibernate and JPA help:

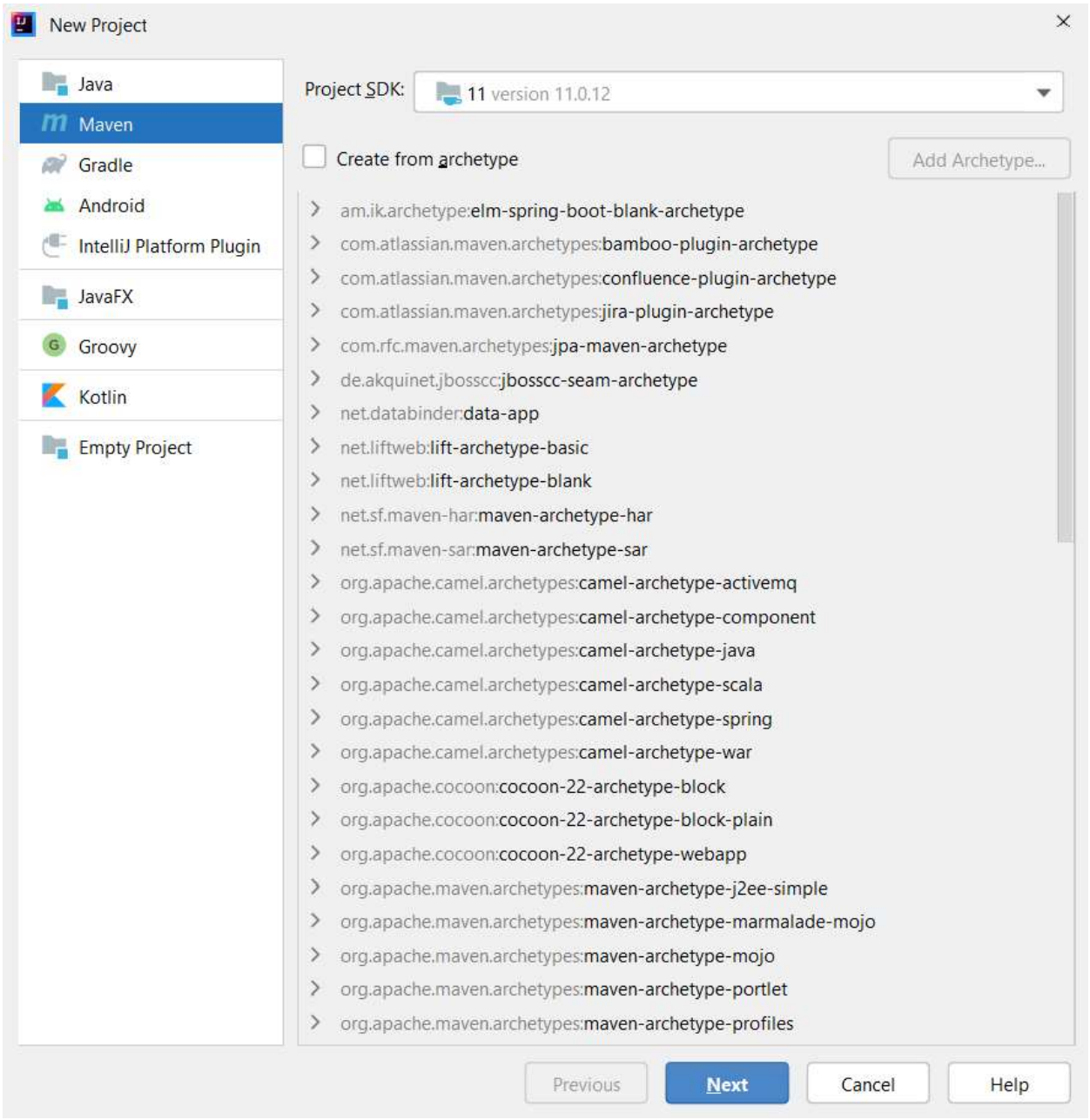
- Hibernate is a popular framework that makes working with databases easy. It lets you work with Java objects instead of writing a lot of SQL queries.
- JPA (Java Persistence API) is the standard specification in Java for managing data between objects and databases. Hibernate follows this standard and also adds extra features.

Step-by-Step Implementation

Step 1: Create a Maven Project

Open your IDE (IntelliJ IDEA, Eclipse or STS).
Create a New Maven Project.

- **GroupId:** org.example
- **ArtifactId:** hibernateapp
- **Version:** 1.0-SNAPSHOT



Step 2: Add Dependencies in pom.xml

We need Hibernate ORM and MySQL connector dependencies.

```
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.2.7.Final</version>
</dependency>
<dependency>
```

```
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
    </dependency>
```

Example: pom.xml File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>org.example</groupId>
    <artifactId>hibernateapp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.hibernate.orm</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.2.7.Final</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
        </dependency>
    </dependencies>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

</project>
```

Step 3: Create Entity Class (Song.java)

Create a simple POJO class and name the class as Song.

```java
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
```

```
@Entity
@Table(name = "song")

// POJO class
public class Song {

    @Id @Column(name = "songId") private int id;

    @Column(name = "songName") private String songName;

    @Column(name = "singer") private String artist;

    public int getId() { return id; }

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

    public String getSongName() { return songName; }

    public void setSongName(String songName)
    {
        this.songName = songName;
    }

    public String getArtist() { return artist; }

    public void setArtist(String artist)
    {
        this.artist = artist;
    }
}
```

Step 4: Configure Hibernate (hibernate.cfg.xml)

Create a hibernate configuration file (XML file) inside the src > main > resources folder. Here we have named the file hibernate.cfg.xml. In this file, we are going to configure all the properties for the MySQL Database.

hibernate.cfg.xml File

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>

        <!-- Database Connection -->
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernate_demo</
```

```
property>
        <property name="hibernate.connection.username">root</property>
        <property
name="hibernate.connection.password">your_password</property>
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>

        <!-- Hibernate Settings -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>

        <!-- Mapping Entity -->
        <mapping class="Song"/>
    </session-factory>
</hibernate-configuration>
```

Step 5: Create Main Class (App.java)

Create a class named App and inside the class write the main() method

Example:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class App {

    public static void main(String[] args)
    {

        // Create Configuration
        Configuration configuration = new Configuration();
        configuration.configure("hibernate.cfg.xml");
        configuration.addAnnotatedClass(Song.class);

        // Create Session Factory and auto-close with try-with-resources.
        try (SessionFactory sessionFactory
                = configuration.buildSessionFactory()) {

            // Initialize Session Object
            Session session = sessionFactory.openSession();

            Song song1 = new Song();

            song1.setId(1);
            song1.setSongName("Broken Angel");
            song1.setArtist("Akon");

            session.beginTransaction();

            // Here we have used persist() method of JPA
```

```
            session.persist(song1);

            session.getTransaction().commit();
        }
    }
}
```

Step 6: Create Database Schema in MySQL

Create a schema named **hibernate-demo** (you can choose your own) inside your MySQL Database. And run your application.

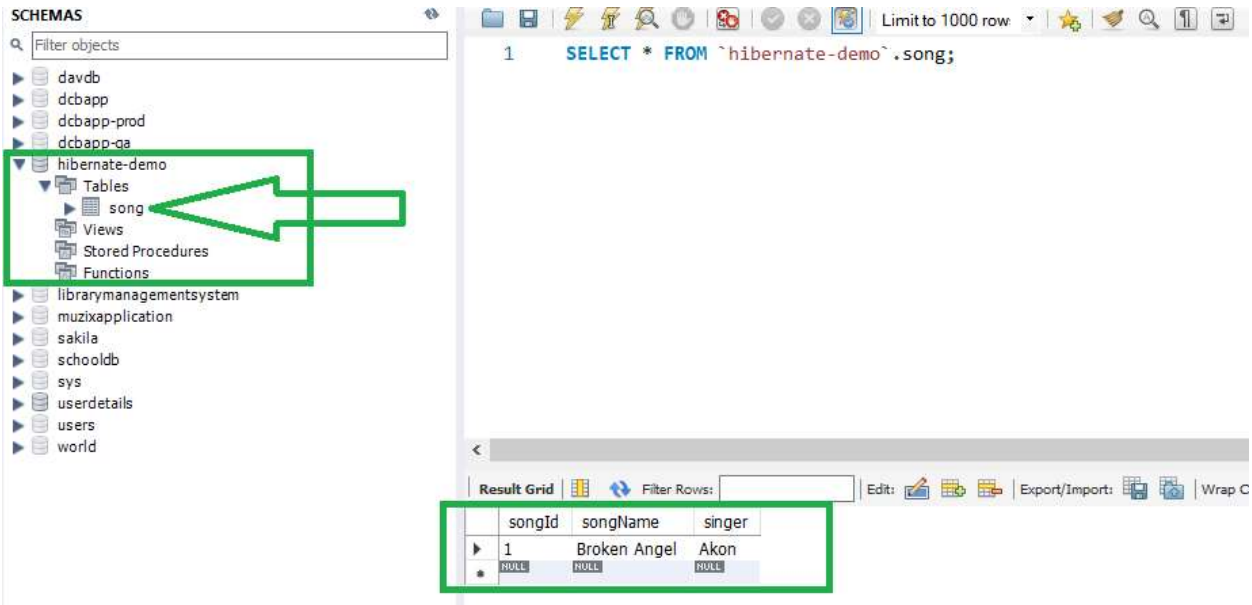
Run the following command in MySQL Workbench or CLI:

CREATE DATABASE hibernate-demo;

Output:

After running the project, check MySQL Workbench:

*SELECT * FROM song;*



You can see the data has been saved inside your MySQL workbench. And in the hibernate-demo schema, a table named song has been created and the corresponding values for each column that you have set in App.java class have been stored.