Experiment: 09	Exp.No: 09 Date: 29-09-2023

Aim: CONNECT JAVA TO A MYSQL DATABASE using HIBERNATE (ORM)

Write a java program to Connect with MySQL database using hibernate.
Connecting Using ORM
More typically, we'll connect to our MySQL database using an Object Relational Mapping (ORM) Framework.
Native Hibernate APIs

Step 1: We need to add the hibernate-core Maven dependency:

```
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.4.10.Final</version>
</dependency>
```

Step 2: Hibernate requires that an entity class must be created for each table. Let's go ahead and define the Person class:

```
@Entity
@Table(name = "Person")
public class Person {
@Id
Long id;
@Column(name = "FIRST_NAME")
String firstName;
@Column(name = "LAST_NAME")
String lastName;
// getters & setters
}
```

Step 3: Hibernate resource file, typically named hibernate.cfg.xml, where we'll define configuration information:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<!-- Database connection settings -->
<propertyname="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://localhost:3306/test </property>
<property name="connection.username">root</property>
<property name="connection.password">root</property>
<!-- SQL dialect -->
<property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
<!-- Validate the database schema on startup -->
<property name="hbm2ddl.auto">validate</property>
<!-- Names the annotated entity class -->
<mapping class="Person"/>
</session-factory>
```

</hibernate-configuration>

Hibernate has many configuration properties. Apart from the standard connection properties, it is worth mentioning the dialect property which allows us to specify the name of the SQL dialect for the database.

Finally, hibernate also needs to know the fully-qualified name of the entity class via the mapping tag. Once we comp lete the configuration, we'll use the SessionFactory class, which is the class responsible for creating and pooling JDBC connections**.**

Step 4: App class body:

SessionFactory sessionFactory;

```
// configures settings from hibernate.cfg.xml
StandardServiceRegistry registry = new
StandardServiceRegistryBuilder().configure().build();
try {
sessionFactory = new
MetadataSources(registry).buildMetadata().buildSessionFactory();
} catch (Exception e) {
System.out.println(e.getMessage());
}
```

Now that we have our connection set up, we can run a query to select all the people from our person table:

```
Session session = sessionFactory.openSession();
session.beginTransaction();
List<Person> result = session.createQuery("from Person", Person.class).list();
result.forEach(person -> {
//do something with Person instance...
});
session.getTransaction().commit();
session.close();
```

Output: