**Hibernate Insert Query Tutorial**

Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. This means you are not required to build and execute SQL queries for interaction with database. You just have to instruct hibernate for that by calling hibernate APIs and hibernate will create and execute those SQL queries on behalf of you.

In this tutorial, I am giving example of inserting data in a single table. We have an object in application i.e. Employee. We want to store it in database. It has following attributes:

1) employee id – Integer  
2) email – String  
3) first name – String  
4) last name – String

It can have more fields, but I am taking only four to make example concrete.

Hibernate talks with java POJO classes which are marked as hibernate entities. To convert a java class into entity, we need to apply **@Entity** annotation on it. There are other annotations like **@Table**, **@Column** and **@Id** etc. which help in mapping entity fields to database table and columns.

Employee entity class looks like below after applying the annotations.

|  |
| --- |
| @Entity  @Table(name = "EMPLOYEE", uniqueConstraints = {          @UniqueConstraint(columnNames = "ID"),          @UniqueConstraint(columnNames = "EMAIL") })  public class EmployeeEntity implements Serializable  {      private static final long serialVersionUID = -1798070786993154676L;        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "ID", unique = true, nullable = false)      private Integer id;        @Column(name = "EMAIL", unique = true, nullable = false, length = 100)      private String email;        @Column(name = "FIRST\_NAME", unique = false, nullable = false, length = 100)      private String firstName;        @Column(name = "LAST\_NAME", unique = false, nullable = false, length = 100)      private String lastName;        //Getters and setters  } |

Next step is to configure hibernate in “**hibernate.cgf.xml**” file. This file contains all available entities in system and database connection metadata.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd>">  <hibernate-configuration>      <session-factory>          <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>          <propertyname="hibernate.connection.url">jdbc:<mysql://localhost:3306/hibernatedemo></property>          <property name="hibernate.connection.password">password</property>          <property name="hibernate.connection.username">root</property>          <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>          <property name="show\_sql">true</property>          <property name="hbm2ddl.auto">create</property>          <mapping class="hibernate.test.dto.EmployeeEntity"></mapping>      </session-factory>  </hibernate-configuration> |

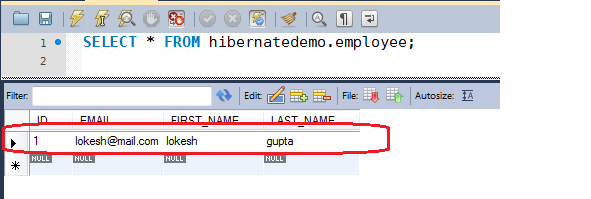
Now hibernate configuration is on place, we have to build the **hibernate session factory**. Session factory is used to obtain the connection of database and various activities like commit and rollback.

|  |
| --- |
| public class HibernateUtil  {      private static final SessionFactory sessionFactory = buildSessionFactory();        private static SessionFactory buildSessionFactory()      {          try          {              // Create the SessionFactory from hibernate.cfg.xml              return new AnnotationConfiguration().configure(newFile("hibernate.cgf.xml")).buildSessionFactory();          }          catch (Throwable ex) {              // Make sure you log the exception, as it might be swallowed              System.err.println("Initial SessionFactory creation failed." + ex);              throw new ExceptionInInitializerError(ex);          }      }        public static SessionFactory getSessionFactory() {          return sessionFactory;      }        public static void shutdown() {          // Close caches and connection pools          getSessionFactory().close();      }  } |

Finally we will use this hibernate session factory to execute insert query to save employee in database.

|  |
| --- |
| public class TestHibernateInsert {        public static void main(String[] args)      {          Session session = HibernateUtil.getSessionFactory().openSession();          session.beginTransaction();            //Add new Employee object          EmployeeEntity emp = new EmployeeEntity();          emp.setEmail("lokesh@mail.com");          emp.setFirstName("lokesh");          emp.setLastName("gupta");            //Save the employee in database          session.save(emp);            //Commit the transaction          session.getTransaction().commit();          HibernateUtil.shutdown();      }  }    Output:    Hibernate: insert into EMPLOYEE (EMAIL, FIRST\_NAME, LAST\_NAME) values (?, ?, ?) |

Lets verify the data in database.

Hibernate insert query example

**Note:**

1) Please ensure to add hibernate and mysql driver dependencies in project. In maven these are as below:

|  |
| --- |
| <dependency>      <groupId>org.hibernate</groupId>      <artifactId>hibernate-commons-annotations</artifactId>      <version>3.0.0.ga</version>  </dependency>  <dependency>      <groupId>org.hibernate</groupId>      <artifactId>hibernate-annotations</artifactId>      <version>3.3.0.ga</version>  </dependency>  <dependency>      <groupId>mysql</groupId>      <artifactId>mysql-connector-java</artifactId>      <version>5.1.6</version>  </dependency> |

2) Verify that hibernate.cgf.xml file is present and has valid database metadata.