Marwadi University
Marwadi Chandarana Group

Unit 5
Hibernate

PROF. RAVIKUMAR NATARAJAN

ASSISTANT PROFESSOR

DEPT. OF CE

Contents

- Introduction to Hibernate
- Exploring Architecture of Hibernate
- Object Relation Mapping(ORM) with Hibernate
- Hibernate Annotation
- Hibernate Query Language (HQL)
- CRUD Operation using Hibernate API

Hibernate: Introduction

- Hibernate is used to convert object data in JAVA to relational database tables.
- It is an open source Object-Relational Mapping (ORM) for Java.
- Hibernate is responsible for making data persistent by storing it in a database.

Object-Relational Mapping (ORM)

- It is a programming technique for converting object-type data of an object oriented programming language into database tables.
- Hibernate is used to convert object data in JAVA to relational database tables.

JDBC v/s Hibernate

Marwadi University
Marwadi Chandarana Group

JDBC	Hibernate
JDBC maps Java classes to database tables (and from Java data types to SQL data types)	Hibernate automatically generates the queries.
With JDBC, developer has to write code to map an object model's data to a relational data model.	Hibernate is flexible and powerful ORM to map Java classes to database tables.
With JDBC, it is developer's responsibility to handle JDBC result set and convert it to Java. So with JDBC, mapping between Java objects and database tables is done manually.	Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.

JDBC vs Hibernate

Marwadi University
Marwadi Chandarana Group

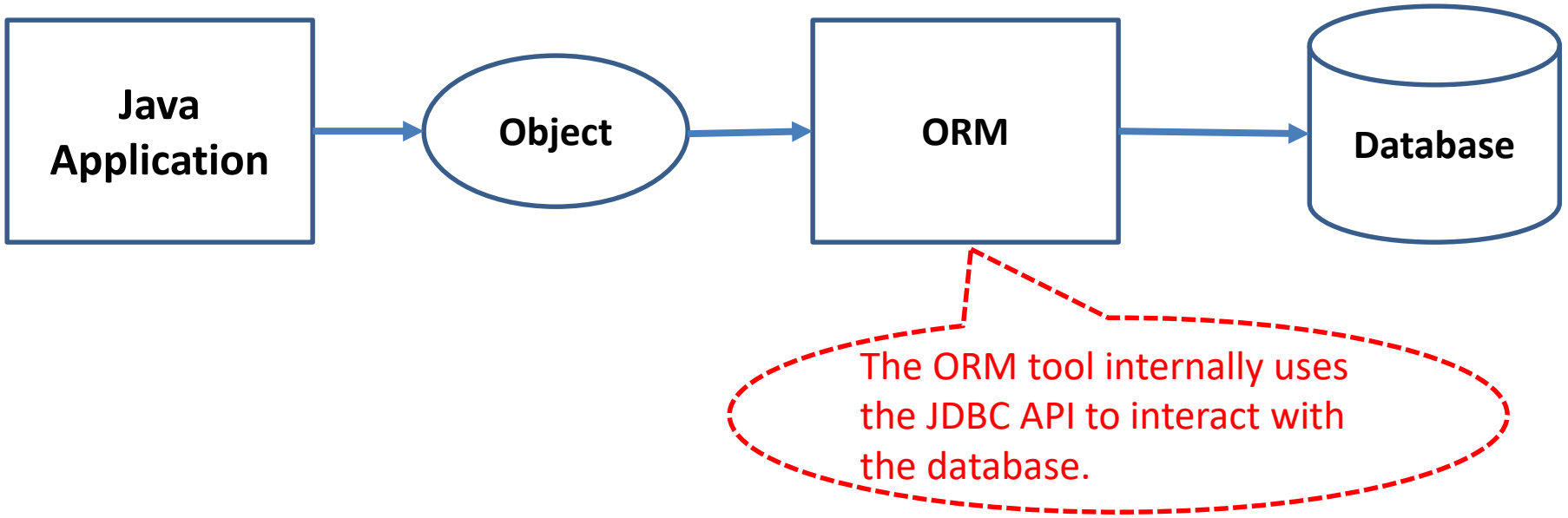
JDBC	Hibernate
Require JDBC Driver for different types of database.	Makes an application portable to all SQL databases.
Handles all create-read-update-delete (CRUD) operations using SQL Queries.	Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
Working with both Object-Oriented software and Relational Database is complicated task with JDBC.	Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.
JDBC supports only native Structured Query Language (SQL)	Hibernate provides a powerful query language Hibernate Query Language-HQL (independent from type of database)

Overview of Hibernate

Hibernate Framework

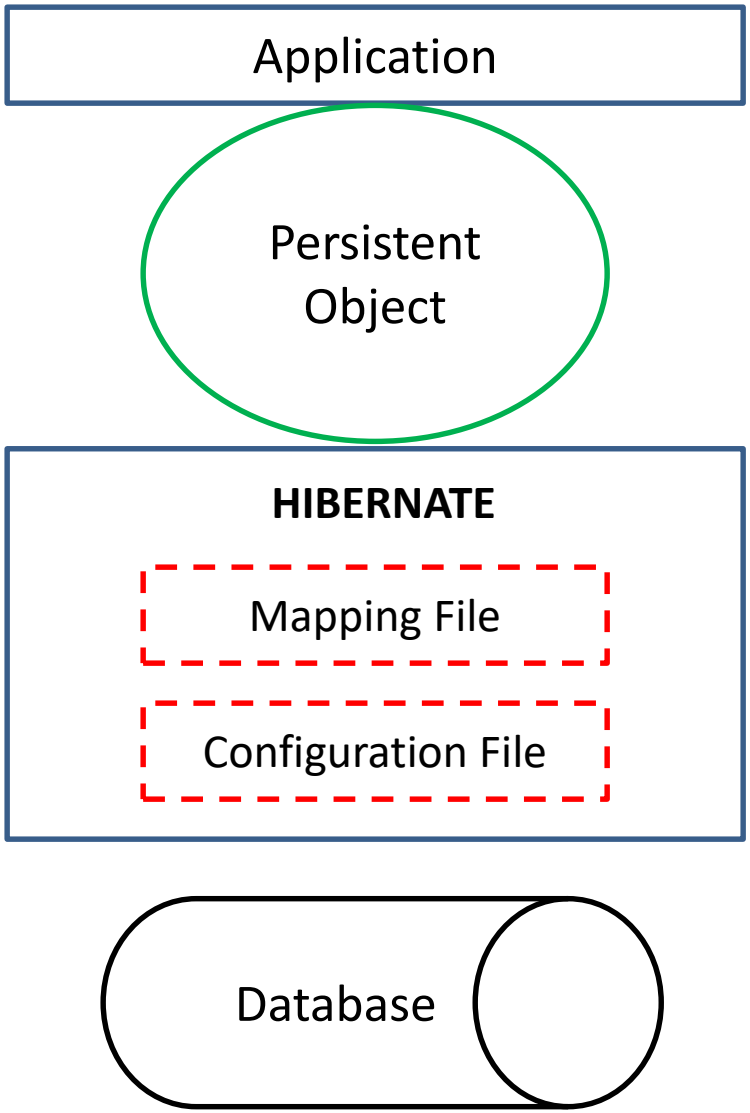
- Hibernate framework simplifies the development of java application to interact with the database.
- Hibernate is an open source, lightweight, ORM (Object Relational Mapping) tool.
- An ORM tool simplifies the data creation, data manipulation and data access.
- Hibernate is a programming technique that maps the object to the data stored in the database.

Hibernate Framework

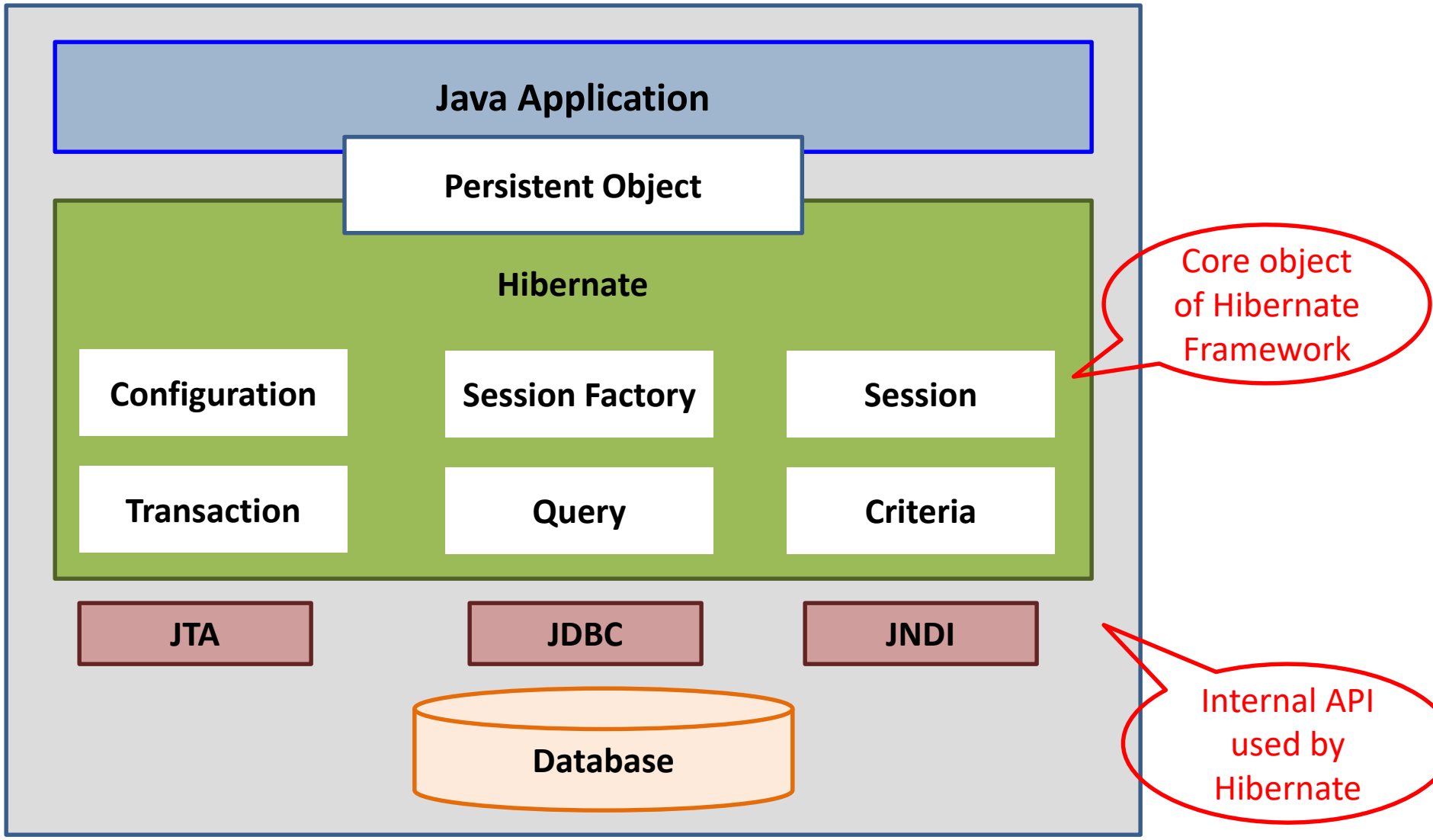


Hibernate Architecture

- There are 4 layers in hibernate architecture
  1. Java application layer
  2. Hibernate framework layer
  3. Backend API layer
  4. Database layer.



Hibernate Architecture



Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

What do you mean by *Persistence*?

Persistence simply means that we would like our application's data to outlive the applications process. In Java terms, we would like the state of (some of) our objects to live beyond the scope of the JVM so that the same state is available later.

Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

Internal API used by Hibernate

1. JDBC (Java Database Connectivity)
2. JTA (Java Transaction API)
3. JNDI (Java Naming Directory Interface)

https://www.youtube.com/watch?v=hfv9ZXUzjhk

Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

- For creating the first hibernate application, we must know the objects/elements of Hibernate architecture.
- They are as follows:
  1. Configuration
  2. Session factory
  3. Session
  4. Transaction factory
  5. Query
  6. Criteria

Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

[1] Configuration Object

- The Configuration object is the first Hibernate object you create in any Hibernate application.
- It is usually created only once during application initialization.
- The Configuration object provides two keys components:

1. **Database Connection:**

   This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.

2. **Class Mapping Setup**:

   This component creates the connection between the Java classes and database tables.

Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

[2] SessionFactory Object

- The SessionFactory is a thread safe object and used by all the threads of an application.
- Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application.
- You would need one SessionFactory object per database using a separate configuration file.
- So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.

Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

[3] Session Object

- A Session is used to get a physical connection with a database.
- The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database.
- The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed as needed.

Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

[4] Transaction Object

- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality.
- Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

[5] Query Object

- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
- A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

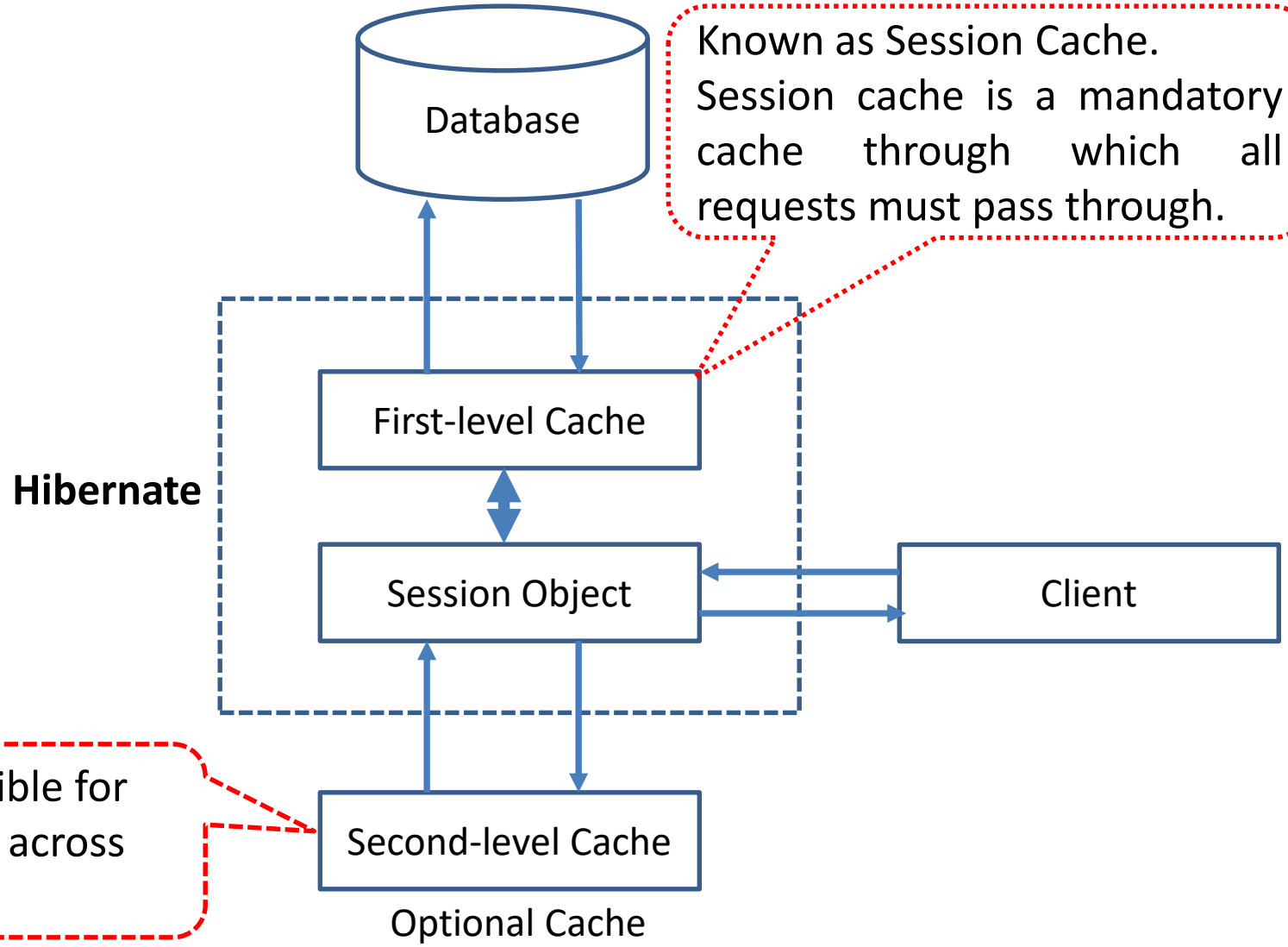
Hibernate Architecture

Marwadi University
Marwadi Chandarana Group

[6] Criteria Object

Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Hibernate Cache Architecture



Why Cache Architecture?

Marwadi University
Marwadi Chandarana Group

- Caching is all about application performance optimization.
- It is situated between your application and the database to avoid the number of database hits as many as possible.
- To give a better performance for critical applications.

Hibernate Cache Architecture

First-level cache:

- The first-level cache is the Session cache.
- The Session object keeps an object under its own control before committing it to the database.
- If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued.
- If you close the session, all the objects being cached are lost.

Hibernate Cache Architecture

Second-level cache:

- It is responsible for caching objects across sessions.
- Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache.
- Any third-party cache can be used with Hibernate. An **org.hibernate.cache.CacheProvider** interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.

Hibernate Mapping Types

- While preparing a Hibernate mapping document, we map the Java data types into RDBMS data types.
- The types declared and used in the mapping files are not Java data types; they are not SQL database types either.
- These types are called ***Hibernate mapping types***, which can translate from Java to SQL data types and vice versa.

Hibernate Mapping Types:

Marwadi University
Marwadi Chandarana Group

Primitive Types

Mapping type	Java type	SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
character	java.lang.String	CHAR(1)
byte	byte	TINYINT
boolean	boolean	BIT
true/false	boolean	CHAR(1) ('T' or 'F')

Hibernate O/R Mapping

Marwadi University
Marwadi Chandarana Group

Three most important mapping are as follows:

1. Collections Mappings
2. Association Mappings
3. Component Mappings

Hibernate O/R Mapping

Collections Mappings

- If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java.
- Hibernate can persist instances of **java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List**, and any **array** of persistent entities or values.

Hibernate O/R Mapping:

Marwadi University
Marwadi Chandarana Group

Collection type	Mapping and Description
java.util.Set	This is mapped with a **<set>** element and initialized with java.util.HashSet
java.util.SortedSet	This is mapped with a **<set>** element. The sort attribute can be set to either a comparator or natural ordering.
java.util.List	This is mapped with a <**list**> element and initialized with java.util.ArrayList
java.util.Collection	This is mapped with a <**bag**> or <**ibag**> element and initialized with java.util.ArrayList
java.util.Map	This is mapped with a <**map**> element and initialized with java.util.HashMap
java.util.SortedMap	This is mapped with a <**map**> element. The sort attribute can be set to either a comparator or natural ordering.

Hibernate O/R Mapping:

Association Mappings:

- The mapping of associations between entity classes and the relationships between tables is the soul of ORM.
- There are the four ways in which the cardinality of the relationship between the objects can be expressed.
- An association mapping can be unidirectional as well as bidirectional.

Hibernate O/R Mapping:

Marwadi University
Marwadi Chandarana Group

Association Mappings:

Mapping type	Description
Many-to-One	Mapping many-to-one relationship using Hibernate
One-to-One	Mapping one-to-one relationship using Hibernate
One-to-Many	Mapping one-to-many relationship using Hibernate
Many-to-Many	Mapping many-to-many relationship using Hibernate

Hibernate O/R Mapping:

Marwadi University
Marwadi Chandarana Group

Component Mappings:

- If the referred class does not have it's own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence therefore is called as the Component class.
- The mapping of Collection of Components is also possible in a similar way just as the mapping of regular Collections with minor configuration differences.

Mapping type	Description
Component Mappings	Mapping for a class having a reference to another class as a member variable.

Advantages of Hibernate Framework

1. **Open source and Lightweight:** Hibernate framework is open source under the LGPL (GNU Lesser General Public License) license and lightweight.
2. **Fast performance:** The performance of hibernate framework is fast because cache is internally used in hibernate framework.
3. **Database Independent query:** HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

Advantages of Hibernate Framework

4. **Automatic table creation:** Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.
5. **Simplifies complex join:** To fetch data from multiple tables is easy in hibernate framework.
6. **Provides query statistics and database status:** Hibernate supports Query cache and provide statistics about query and database status.

Steps to run hibernate example

Steps to run first hibernate example with MySQL in Netbeans IDE 8.2

Step-1: Create the database

```
CREATE DATABASE retailer;
```

Step-2: Create table result

```
CREATE TABLE customers(
    name varchar(20),
    C_ID int NOT NULL AUTO_INCREMENT,
    address  varchar(20),
    email varchar(50),
    PRIMARY KEY(C_ID)
);
```

Steps to run hibernate example

Step-3: Create new java application.

- File > New project > Java > Java Application > Next Name it as **HibernateTest.**
- Then click Finish to create the project.

Steps to run hibernate example

Step-4: Create a POJO(Plain Old Java Objects) class

- We create this class to use variables to map with the database columns.
- Right click the package (hibernatetest) & select New > Java Class Name it as Customer.
- Click Finish to create the class.

Steps to run hibernate example: Step-4


```
1. package hibernatetest;
2. public class Customer {
3.     public String customerName;
4.     public int customerID;
5.     public String customerAddress;
6.     public String customerEmail;
7. public void setCustomerAddress(String
                                    customerAddress) {
            this.customerAddress = customerAddress;}
8. public void setCustomerEmail(String customerEmail) {
9.          this.customerEmail = customerEmail;}
10.public void setCustomerID(int customerID) {
11.         this.customerID = customerID; }
```

Steps to run hibernate example: Step-4


```
12. public void setCustomerName(String customerName) {
13.         this.customerName = customerName;     }
14. public String getCustomerAddress() {
15.         return customerAddress;     }
16. public String getCustomerEmail() {
17.         return customerEmail;     }
18. public int getCustomerID() {
19.         return customerID;     }
20. public String getCustomerName() {
21.         return customerName;     }
22.}
```

Steps to run hibernate example: Step-4

Marwadi University
Marwadi Chandarana Group

Step-4: Create a POJO(Plain Old Java Objects) class

- To generate getters and setters easily in NetBeans, right click on the code and select Insert Code Then choose Getter... or Setter...
- Variable **customerName** will map with the name column of the **customers** table.
- Variable **customerID** will map with the **C_ID** column of the customers table. It is integer & auto incremented. So POJO class variable also should be int.
- Variable **customerAddress** will map with the **address** column of the customers table.
- Variable **customerEmail** will map with the **email** column of the customers table.

Steps to run hibernate example

Step-5: Connect to the database we have already created. [retailer]

- Select Services tab lying next to the Projects tab.
- Expand Databases.
- Expand MySQL Server. There we can see the all databases on MySQL sever
- Right click the database retailer. Select Connect.

Steps to run hibernate example

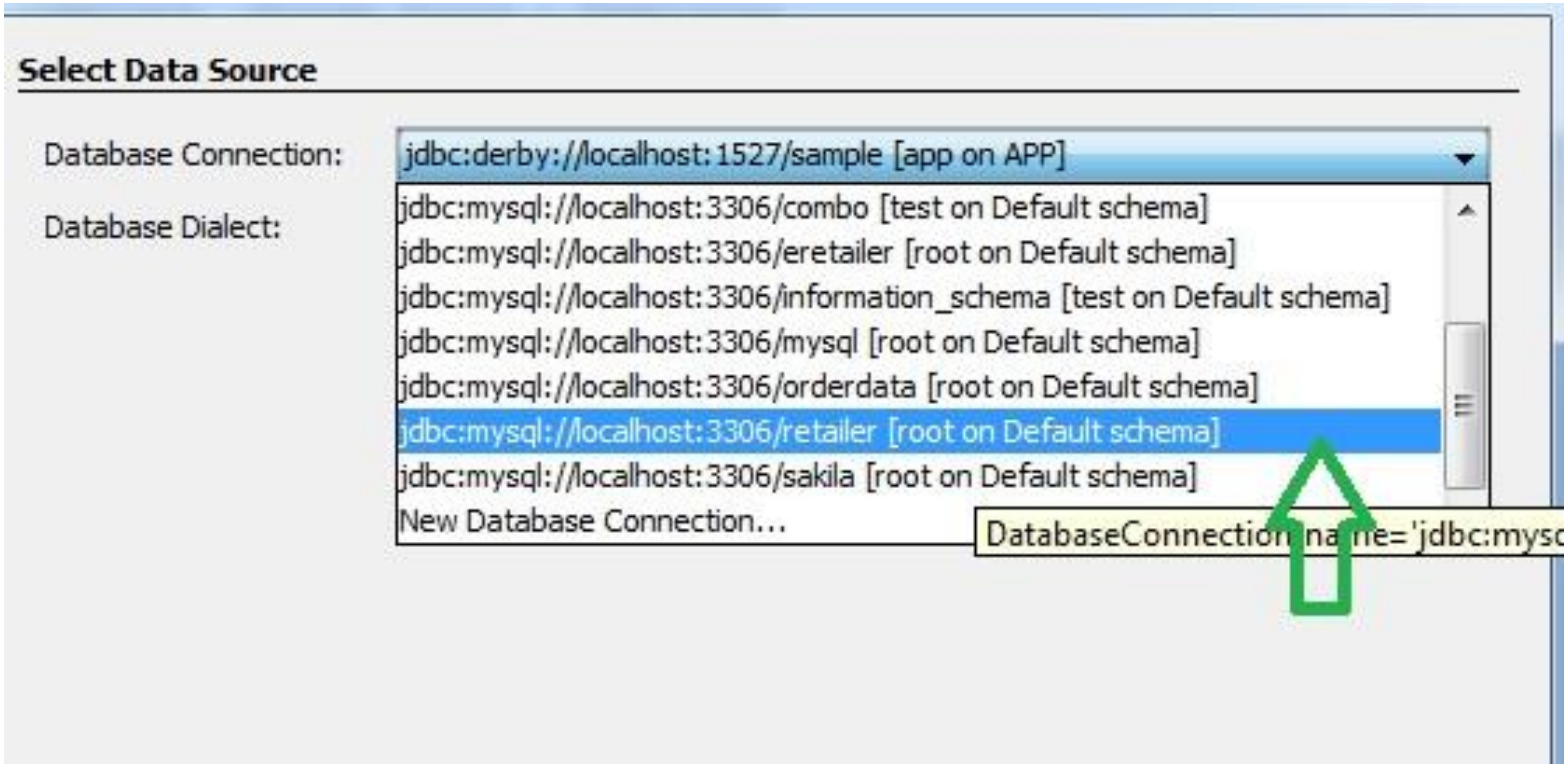
Step-6: Creating the configuration XML

- Hibernate need a configuration file to create the connection.
- Right click package hibernatetest select New > Other > **Hibernate > Hibernate Configuration Wizard**
- Click Next >
- In next window click the drop down menu of Database Connection and select retailer database connection.

Steps to run hibernate example: Step-6

Marwadi University
Marwadi Chandarana Group

Step-6: Creating the configuration XML



- Click Finish to create the file.

Steps to run hibernate example

Marwadi University
Marwadi Chandarana Group

hibernate.cfg.xml

```
1. <hibernate-configuration>
2.   <session-factory>
3.     <property name="hibernate.connection.driver_class">
                                        com.mysql.jdbc.Driver
   </property>
4.     <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/retailer</property>
5.     <property name="hibernate.connection.username">
                                                root</property>
6.     <property name="hibernate.connection.password">
                                                root</property>
7.     <property name="hibernate.connection.pool_size">
                                                  10</property>
8.     <property name="hibernate.dialect">
                org.hibernate.dialect.MySQLDialect</property>
```

Steps to run hibernate example

Marwadi University
Marwadi Chandarana Group

```
9. <property name="current_session_context_class">
                                        thread</property>
10.<property name="cache.provider_class">
        org.hibernate.cache.NoCacheProvider</property>
11.<property name="show_sql">true</property>
12.<property name="hibernate.hbm2ddl.auto">
                                        update</property>
13.<mapping resource="hibernate.hbm.xml"></mapping>
14.</session-factory>
15.</hibernate-configuration>
```

Steps to run hibernate example

Step-7: Creating the mapping file [hibernate.hbm]

- Mapping file will map relevant java object with relevant database table column.
- Right click project select New > Other > Hibernate **> Hibernate Mapping Wizard**
- click Next name it as hibernate.hbm
- click Next> In next window we have to select Class to Map and Database Table.
- After selecting correct class click OK
  Select Database Table
  Click drop down list and select the table you want to map.
  Code for mapping file.

Steps to run hibernate example: Step-7

Marwadi University
Marwadi Chandarana Group

```
1. <hibernate-mapping>
2.    <class name="hibernatetest.Customer" table="customers">
3.    <id column="C_ID" name="customerID" type="int">
4.    <generator class="native">
5.    </generator></id>
6.    <property name="customerName">
7.    <column name="name">
8.    </column></property>
9.    <property name="customerAddress">
10.   <column name="address">
11.   </column></property>
12.   <property name="customerEmail">
13.   <column name="email">
14.   </column></property>
15.   </class></hibernate-mapping>
```

hibernate.hbm.xml

Steps to run hibernate example: Step-7

Step-7: Creating the mapping file [hibernate.hbm]

- property name = variable name of the POJO class
- column name = database column that maps with previous variable

Steps to run hibernate example

Step-8: Now java program to insert record into the database

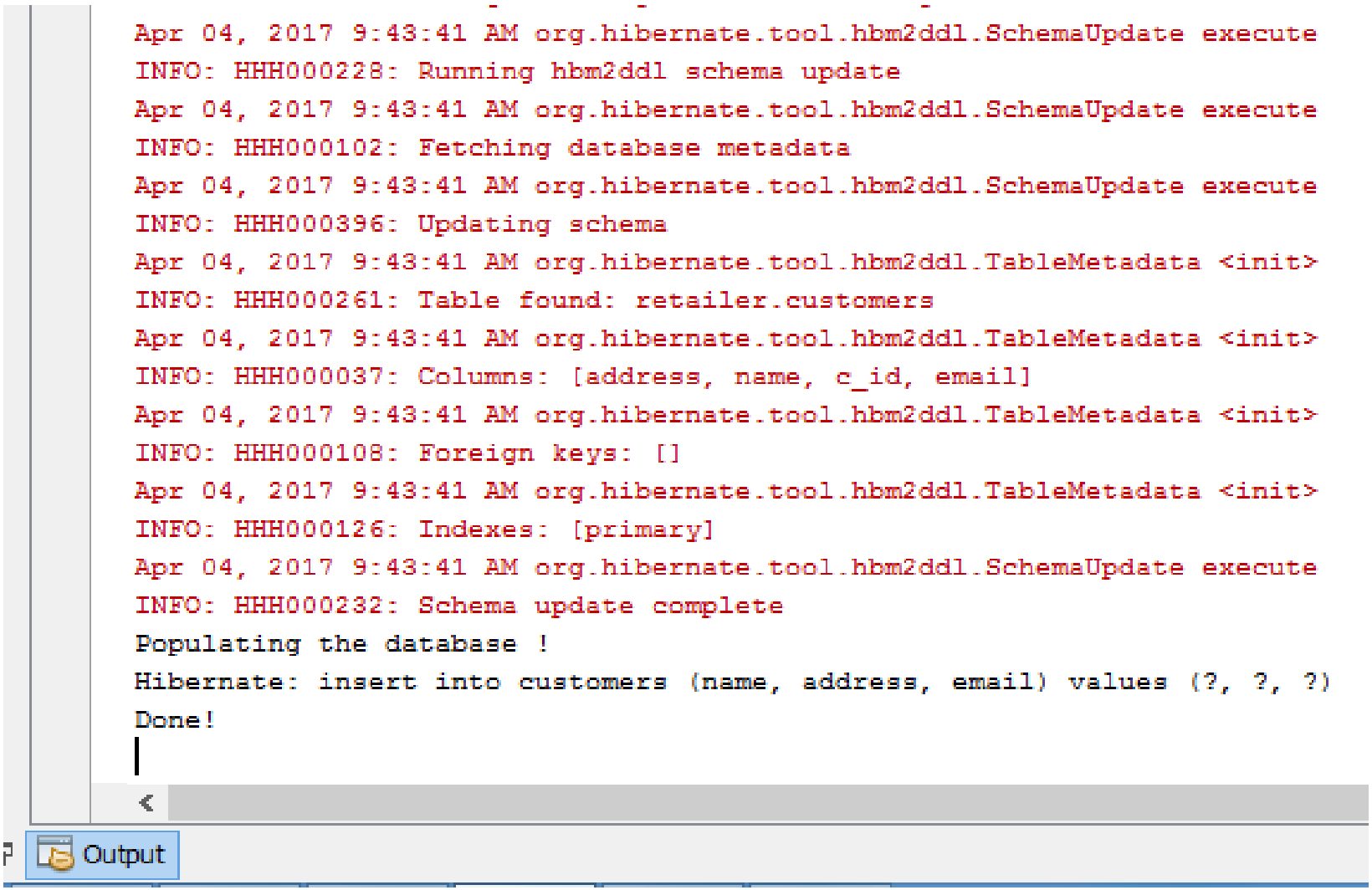
```
1. package hibernatetest;
2. import org.hibernate.Session;
3. import org.hibernate.SessionFactory;
4. public class HibernateTest {
5.       public static void main(String[] args) {
6.          Session session = null;
7. try
8. {
9. SessionFactory sessionFactory = new
   org.hibernate.cfg.Configuration().configure().buildSessi
   onFactory();
```

Steps to run hibernate example

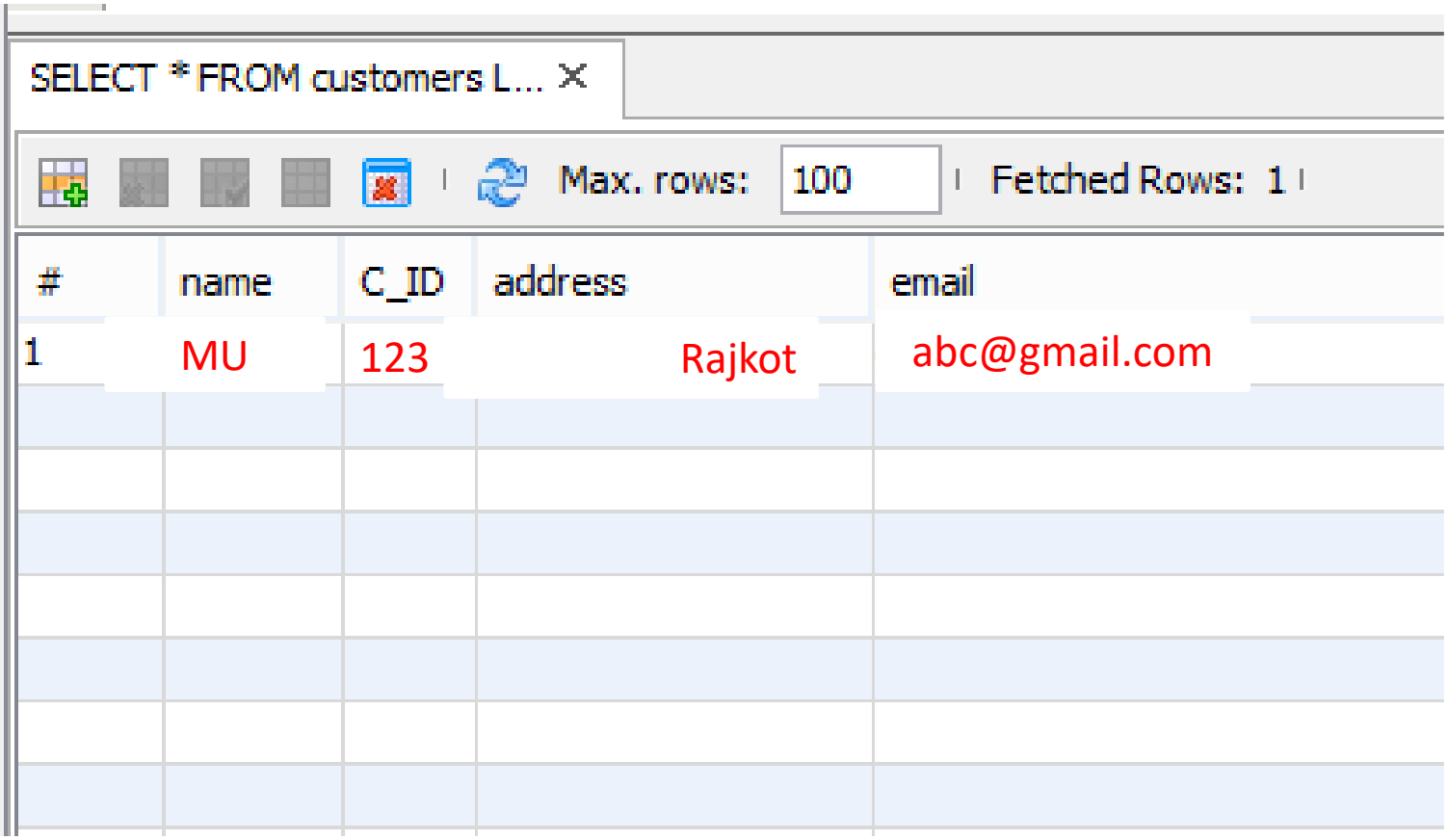
Marwadi University
Marwadi Chandarana Group

```
10.session =sessionFactory.openSession();
11.         session.beginTransaction();
12.         System.out.println("Populating the database !");
13.         Customer customer = new Customer();
14.         customer.setCustomerName("MU");
15.         customer.setCustomerAddress("Rajkot);
16.         customer.setCustomerEmail("abc@gmail.com");
17.         session.save(customer);
18.         session.getTransaction().commit();
19.         System.out.println("Done!");
20.         session.flush();
21.         session.close();
22.       }catch(Exception e)
      {System.out.println(e.getMessage());     } } }
```

Steps to run hibernate example:output

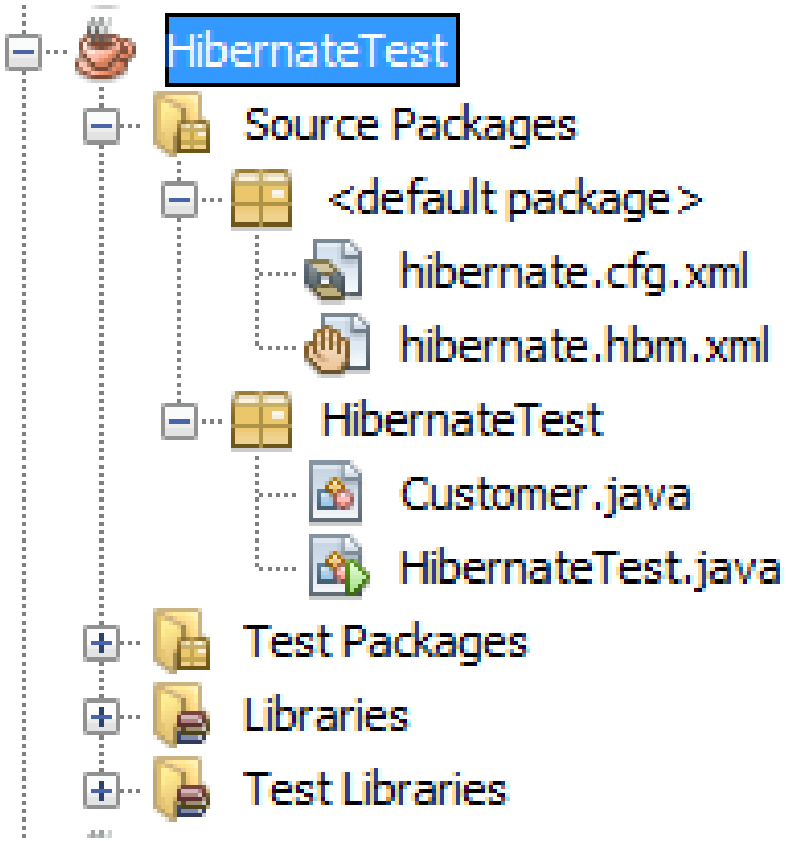


Steps to run hibernate example: output



Hibernate Program Hierarchy

Marwadi University
Marwadi Chandarana Group



HIBERNATE O/R MAPPING

Marwadi University
Marwadi Chandarana Group

- O/R Mapping is usually defined in **XML document**.
- The mapping language is java-centric, meaning that mapping is constructed around **persistence class declaration**.

Student.hbm.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
    <hibernate-mapping>
    <class name="com.mypackage.Student" table="STUDENT">
     <id name="roll">
      <generator class="assigned"></generator>
     </id>
      <property name="studname"></property>
      <property name="branch"></property>
    </class>
    </hibernate-mapping>
```

HIBERNATE O/R MAPPING

The mapping document is an XML document having **<hibernate-mapping> as the root element which contains <class> elements corresponding class.**

The **<class>** elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the **name** attribute of the class element and the database table name is specified using the **table** attribute.

The **<id>** element maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class.

HIBERNATE O/R MAPPING

Marwadi University
Marwadi Chandarana Group

The **<generator>** element within the id element is used to generate the primary key values automatically.

The **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity, sequence** to create primary key depending upon the capabilities of the underlying database.

The **<property>** element is used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table.

The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

HIBERNATE ANNOTATION

Marwadi University
Marwadi Chandarana Group

- **Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping.** All the metadata is clubbed into the POJO java file along with the code, this helps the user to understand the table structure and POJO simultaneously during the development.
- Annotations which are commonly used are **@Entity**, @Table, @Id, @GenerateValue, @Column etc.,

Confused when to use Annotation and XML Mapping???

HIBERNATE ANNOTATION

- If you going to make your application **portable** to other EJB 3 compliant ORM applications, you must use annotations to represent the mapping information.
- But still if you want greater flexibility, then you should go with XML-based mappings.

Environment Setup for Hibernate Annotation

First of all you would have to make sure that you are using JDK 5.0 otherwise you need to upgrade your JDK to JDK 5.0 to take advantage of the native support for annotations.

Second, you will need to install the Hibernate 3.x annotations distribution package and copy hibernate-annotations.jar, lib/hibernate-comons-annotations.jar and lib/ejb3-persistence.jar from the Hibernate Annotations distribution to your CLASSPATH.

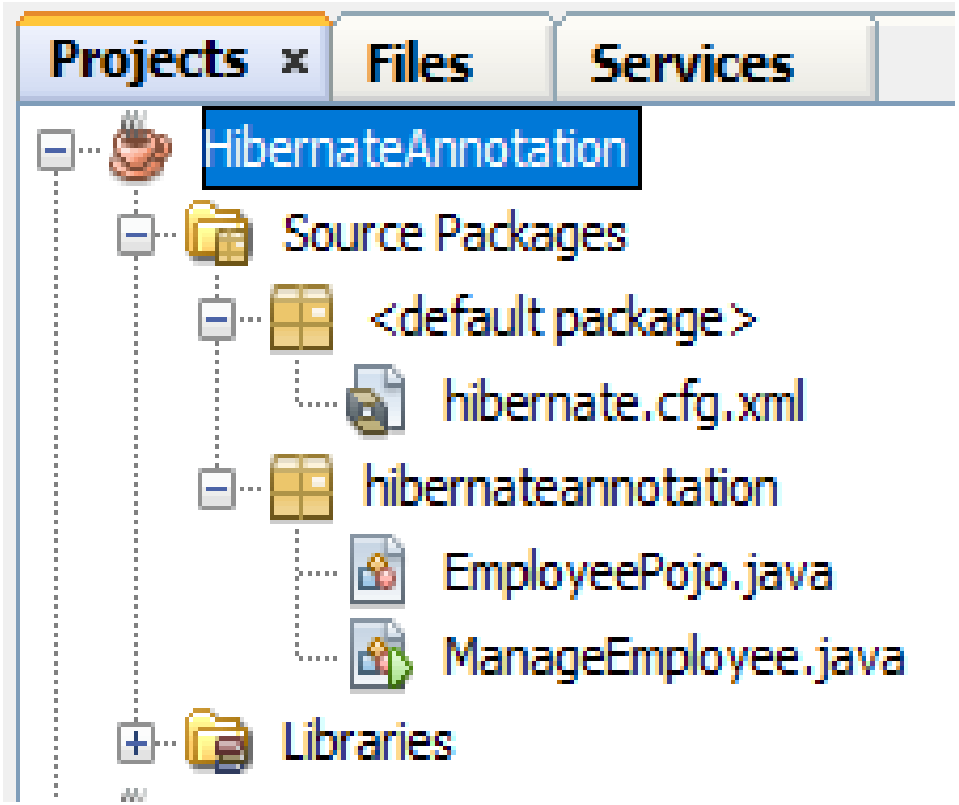
Annotated Class Example

Marwadi University
Marwadi Chandarana Group

In Hibernate Annotation, all the metadata is clubbed into the POJO java file along with the code, this helps the user to understand the table structure and POJO simultaneously during the development. Consider we are going to use the following EMPLOYEE table to store our objects –

```
create table EMPLOYEE (
id INT NOT NULL auto_increment,
first_name VARCHAR(20) default NULL,
last_name VARCHAR(20) default NULL,
salary INT default NULL,
PRIMARY KEY (id)
);
```

Project Hierarchy



Example Continue… mapping of Employee class with annotations to map objects with the defined EMPLOYEE table

EmployeePojo.java

```
import javax.persistence.*;
@Entity
@Table(name = "EMPLOYEE")

public class Employee
{
@Id   @GeneratedValue
@Column(name = "id")
private int id;

@Column(name = "first_name")
private String firstName;

@Column(name = "last_name")
private String lastName;

@Column(name = "salary")
private int salary;
public Employee() {}

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

public String getFirstName()
{ return firstName; }
public void setFirstName( String first_name )
 { this.firstName = first_name; }

public String getLastName()
 { return lastName; }
public void setLastName( String last_name )
{ this.lastName = last_name; }

 public int getSalary()
 { return salary; }
public void setSalary( int salary )
 { this.salary = salary; }
 }
```

```
package hibernateannotation;
import java.util.Iterator;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;
public class ManageEmployee {
   private static SessionFactory factory;
  public static void main(String[] args) {

    try {
      factory = new AnnotationConfiguration().
              configure().
              addPackage("com"). //add package if used.
              addAnnotatedClass(EmployeePojo.class).
              buildSessionFactory();
    } catch (Throwable ex) {
      System.err.println("Failed to create sessionFactory object." + ex);
      throw new ExceptionInInitializerError(ex);
    }

    ManageEmployee ME = new ManageEmployee();

    /* Add few employee records in database */
    Integer empID1 = ME.addEmployee("RK", "Keynotes", 1000);
    Integer empID2 = ME.addEmployee("Marwadi", "University", 5000);
    Integer empID3 = ME.addEmployee("Aarav", "Natarajan", 10000);

    /* List down all the employees */
    ME.listEmployees();
```

```
public Integer addEmployee(String fname, String lname,
int salary){
    Session session = factory.openSession();
    Transaction tx = null;
    Integer employeeID = null;

    try {
      tx = session.beginTransaction();
      EmployeePojo employee = new EmployeePojo();
      employee.setFirstName(fname);
      employee.setLastName(lname);
      employee.setSalary(salary);
      employeeID = (Integer) session.save(employee);
      tx.commit();
    } catch (HibernateException e) {
      if (tx!=null) {
          tx.rollback();
       }
      e.printStackTrace();
    } finally {
      session.close();
    }
    return employeeID;
  }
```

```
public void listEmployees( ){
    Session session = factory.openSession();
    Transaction tx = null;

    try {
      tx = session.beginTransaction();
      List employees = session.createQuery("FROM EmployeePojo").list();
      for (Iterator iterator = employees.iterator(); iterator.hasNext();){
        EmployeePojo employee = (EmployeePojo) iterator.next();
        System.out.print("First Name: " + employee.getFirstName());
        System.out.print("  Last Name: " + employee.getLastName());
        System.out.println("  Salary: " + employee.getSalary());
      }
      tx.commit();
    } catch (HibernateException e) {
      if (tx!=null) {
          tx.rollback();
       }
      e.printStackTrace();
    } finally {
      session.close();
    }
  }

  }
```

HIBERNATE ANNOTATION

Marwadi University
Marwadi Chandarana Group

@Entity Annotation

@Entity annotation to the Employee class, which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

@Table Annotation

The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.

The @Table annotation provides four attributes, allowing you to override the **name** of the table, its **catalogue**, and its **schema**, and enforce **unique constraints** on columns in the table. For now, we are using just table name, which is EMPLOYEE.

HIBERNATE ANNOTATION

Marwadi University
Marwadi Chandarana Group

@Id and @GeneratedValue Annotations

Each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure.

By default, the @Id annotation will automatically determine the most appropriate primary key generation strategy to be used but you can override this by applying the **@GeneratedValue** annotation, which takes two parameters **strategy** and **generator.**

HIBERNATE ANNOTATION

Marwadi University
Marwadi Chandarana Group

@Column Annotation

The @Column annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes –

name attribute permits the name of the column to be explicitly specified.

length attribute permits the size of the column used to map a value particularly for a String value.

nullable attribute permits the column to be marked NOT NULL when the schema is generated.

unique attribute permits the column to be marked as containing only unique values.

Hibernate Query Language (HQL)

- The Hibernate ORM framework provides its own query language called Hibernate Query Language .
- Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL.

  Therefore, it is database independent query language.

Hibernate Query Language (HQL)

Characteristics of HQL

1. **Similar to SQL**

   HQL's syntax is very similar to standard SQL. If you are familiar with SQL then writing HQL would be pretty easy.

2. **Fully object-oriented:** HQL doesn't use real names of table and columns. It uses class and property names instead. HQL can understand inheritance, polymorphism and association.

3. **Reduces the size of queries**

HQL vs SQL

Marwadi University
Marwadi Chandarana Group

SELECT QUERY

SQL

ResultSet rs=st.executeQuery("**select * from diet**");

HQL

Query query= session.createQuery("**from diet**");

//here persistent class name is diet

HQL vs SQL

Marwadi University
Marwadi Chandarana Group

SELECT with WHERE clause

SQL

ResultSet rs=st.executeQuery("**select * from diet where id=301**");

HQL

Query query= session.createQuery("**from diet where id=301** ");

//here persistent class name is diet

HQL vs SQL

Marwadi University
Marwadi Chandarana Group

UPDATE QUERY

SQL

1. String query = "**update User set name=? where id = ?";**
2. PreparedStatement preparedStmt = conn.prepareStatement(query);
3. preparedStmt.setString (1, “DIET_CE”);
4. preparedStmt.setInt(2, 054);
5. preparedStmt.executeUpdate();

HQL

1. Query q=session.createQuery("**update User set name=:n where id=:i**");
2. q.setParameter("n", "DIET_CE");
3. q.setParameter("i",054);
4. int status=q.executeUpdate();

HQL vs SQL

Marwadi University
Marwadi Chandarana Group

INSERT QUERY

SQL

String sql = "**INSERT INTO Stock VALUES (100, 'abc')**";

int result = stmt.executeUpdate(sql);

HQL

Query query = session.createQuery("**insert into Stock(stock_code, stock_name) select stock_code, stock_name from backup_stock**");

int result = query.executeUpdate();

HIBERNATE QUERY LANGUAGE(HQL)

Hibernate is supported by a very powerful query language which is just similar to SQL.

The Hibernate query language is an object oriented query language.

It works with persistent object and its property.

HQL does not depends upon table of database. Instead of table name we use class name in HQL.

The Hibernate query are converted into conventional query that in turn perform operation.

These query are case sensitive.

HIBERNATE QUERY LANGUAGE(HQL)

1. The HQL can perform bulk of operation at a time on Hibernate.
2. HQL supports object oriented feature such as inheritance, polymorphism, association and so on.
3. Instead of returning plain data HQL return object. These objects can be easily accessed or programmed.
4. HQL is database independent. The same HQL can be executed on different databases.

HIBERNATE QUERY LANGUAGE(HQL)

Marwadi University
Marwadi Chandarana Group

1. HQL stands for Hibernate Query Language and SQL stands for Structured Query Language

2. The structure of HQL is similar to SQL, but the main deference is that HQL makes use of class name instead of table and property name.

3. HQL is object oriented query language and sql is not.

4. HQL is database independent where as SQL is database dependent.

CRUD Operation using Hibernate API

- Demo

CRUD Operation using Hibernate API

Marwadi University
Marwadi Chandarana Group

- Demo

Summary

Marwadi University
Marwadi Chandarana Group

- Introduction to Hibernate
- Exploring Architecture of Hibernate
- Object Relation Mapping(ORM) with Hibernate
- Hibernate Annotation
- Hibernate Query Language (HQL)
- CRUD Operation using Hibernate API

Up Next?

Marwadi University
Marwadi Chandarana Group

- Spring: Introduction, Architecture,
- Spring MVC Module,
- Life Cycle of Bean Factory,
- Explore: Constructor Injection, Dependency Injection,
- Inner Beans, Aliases in Bean, Bean Scopes,
- Spring Annotations, Spring AOP Module,
- Spring DAO,
- Database Transaction Management,
- CRUD Operation using DAO and Spring API

END OF UNIT - 5