Explore More

Subcription : Premium CDAC NOTES & MATERIAL @99

Contact to Join
Premium Group

Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Hibernate Interview Questions & Answers

Q. How to integrate hibernate with spring boot?

Step 01: Maven Dependencies

```
<!-- pom.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.5.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.javaexample.demo</groupId>
    <artifactId>SpringBoot2Demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>SpringBoot2Demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

- **spring-boot-starter-data-jpa**(required): It includes spring data, hibernate, HikariCP, JPA API, JPA Implementation (default is hibernate), JDBC and other required libraries.

Step 02: Create JPA entity classes

```
/** EmployeeEntity.java **/

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="TBL_EMPLOYEES")
public class EmployeeEntity {

    @Id
    @GeneratedValue
    private Long id;

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

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

    @Column(name="email", nullable=false, length=200)
    private String email;

    //Setters and getters left out for brevity.

    @Override
```

```
    public String toString() {
        return "EmployeeEntity [id=" + id + ", firstName=" + firstName
+
                ", lastName=" + lastName + ", email=" + email   + "]";
    }
}
```

Step 03: Create JPA Repository

Extend `JpaRepository` interface to allows to create repository implementations automatically, at runtime, for any given entity class. The types of entity class and it's ID field are specified in the generic parameters on JpaRepository.

```
/** EmployeeRepository.java **/
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.javaexample.demo.entity.EmployeeEntity;

@Repository
public interface EmployeeRepository extends
JpaRepository<EmployeeEntity, Long> {

}
```

By this simple extension, EmployeeRepository inherits several methods for working with Employee persistence, including methods for saving, deleting, and finding Employee entities.

Step 04: Properties Configuration

application.properties

```
spring.datasource.url=jdbc:h2:file:~/test
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Enabling H2 Console
spring.h2.console.enabled=true

# Custom H2 Console URL
spring.h2.console.path=/h2-console


#Turn Statistics on and log SQL stmts

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

```
#If want to see very extensive logging
spring.jpa.properties.hibernate.generate_statistics=true
logging.level.org.hibernate.type=trace
logging.level.org.hibernate.stat=debug

#Schema will be created using schema.sql and data.sql files

spring.jpa.hibernate.ddl-auto=none
```

```
## schama.sql
DROP TABLE IF EXISTS TBL_EMPLOYEES;

CREATE TABLE TBL_EMPLOYEES (
    id INT AUTO_INCREMENT  PRIMARY KEY,
    first_name VARCHAR(250) NOT NULL,
    last_name VARCHAR(250) NOT NULL,
    email VARCHAR(250) DEFAULT NULL
);

## data.sql
INSERT INTO TBL_EMPLOYEES
    (first_name, last_name, email)
VALUES
    ('Lokesh', 'Gupta', 'abc@gmail.com'),
    ('Deja', 'Vu', 'xyz@email.com'),
    ('Caption', 'America', 'cap@marvel.com');
```

Step 05. Spring boot hibernate demo

To test hibernate configuration with Spring boot, we need to autowire the EmployeeRepository dependency in a class and use it's method to save or fetch employee entities.

SpringBoot2DemoApplication.java

```
import java.util.Optional;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import com.javaexample.demo.entity.EmployeeEntity;
import com.javaexample.demo.repository.EmployeeRepository;

@SpringBootApplication
public class SpringBoot2DemoApplication implements CommandLineRunner {

    private Logger logger = LoggerFactory.getLogger(this.getClass());

    @Autowired
```

```
    EmployeeRepository repository;

    public static void main(String[] args) {
        SpringApplication.run(SpringBoot2DemoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception
    {
        Optional<EmployeeEntity> emp = repository.findById(2L);

        logger.info("Employee id 2 -> {}", emp.get());
    }
}
```

Output

```
Tomcat initialized with port(s): 8080 (http)
Starting service [Tomcat]
Starting Servlet engine: [Apache Tomcat/9.0.19]
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in 5748 ms

HikariPool-1 - Starting...
HikariPool-1 - Start completed.
HHH000204: Processing PersistenceUnitInfo [
    name: default
    ...]
HHH000412: Hibernate Core {5.3.10.Final}
HHH000206: hibernate.properties not found
HCANN000001: Hibernate Commons Annotations {5.0.4.Final}
HHH000400: Using dialect: org.hibernate.dialect.H2Dialect

Initialized JPA EntityManagerFactory for persistence unit 'default'
Initializing ExecutorService 'applicationTaskExecutor'
spring.jpa.open-in-view is enabled by default. Therefore, database
queries may be performed during view rendering.
Explicitly configure spring.jpa.open-in-view to disable this warning
Tomcat started on port(s): 8080 (http) with context path ''
Started SpringBoot2DemoApplication in 17.638 seconds (JVM running for
19.1)

Hibernate:
    select
        employeeen0_.id as id1_0_0_,
        employeeen0_.email as email2_0_0_,
        employeeen0_.first_name as first_na3_0_0_,
        employeeen0_.last_name as last_nam4_0_0_
    from
        tbl_employees employeeen0_
    where
```

```
        employeeeen0_.id=?

Employee id 2 -> EmployeeEntity [id=2, firstName=Deja, lastName=Vu,
email=xyz@email.com]

<b><a href="#">↥ back to top</a></b>
```

Q. Mention the differences between JPA and Hibernate?

JPA is a specification for accessing, persisting and managing the data between Java objects and the relational database.

Where as, Hibernate is the actual implementation of JPA guidelines. When hibernate implements the JPA specification, this will be certified by the JPA group upon following all the standards mentioned in the specification. For example, JPA guidelines would provide information of mandatory and optional features to be implemented as part of the JPA implementation.

Hibernate	JPA
Hibernate is the object-relational mapping framework which helps to deal with the data persistence.	It is the Java specification to manage the java application with relational data.
It's is one of the best JPA providers.	It is the only specification which doesn't deal with any implementation.
In this, we use Session for handling the persistence in an application.	In this, we use the Entity manager.
It is used to map Java data types with database tables and SQL data types.	It is the standard API which allows developers to perform database operations smoothly.
The Query language in this is Hibernate Query Language.	The query language of JPA is JPQL (Java Persistence Query Language)

Q. What is HQL and what are its benefits?

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries, which in turns perform action on database.

Advantages Of HQL:

- HQL is database independent, means if we write any program using HQL commands then our program will be able to execute in all the databases with out doing any further changes to it
- HQL supports object oriented features like Inheritance, polymorphism,Associations(Relation ships)
- HQL is initially given for selecting object from database and in hibernate 3.x we can doDML operations (insert, update…) too

Different Ways Of Construction HQL Select

FROM Clause

```
/** In SQL **/
sql> select * from Employee

/** In HQL **/
String hql = "FROM Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

AS Clause

```
String hql = "FROM Employee AS E";
Query query = session.createQuery(hql);
List results = query.list();
```

SELECT Clause

```
String hql = "SELECT E.firstName FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

WHERE Clause

```
String hql = "FROM Employee E WHERE E.id = 10";
Query query = session.createQuery(hql);
List results = query.list();
```

ORDER BY Clause

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
Query query = session.createQuery(hql);
List results = query.list();
```

GROUP BY Clause

```
String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E " +
             "GROUP BY E.firstName";
Query query = session.createQuery(hql);
List results = query.list();
```

Using Named Parameters

Hibernate supports named parameters in its HQL queries. This makes writing HQL queries that accept input from the user easy and you do not have to defend against SQL injection attacks. Following is the simple syntax of using named parameters –

```
String hql = "FROM Employee E WHERE E.id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id",10);
List results = query.list();
```

UPDATE Clause

```
String hql = "UPDATE Employee set salary = :salary "  +
             "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("salary", 1000);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

DELETE Clause

```
String hql = "DELETE FROM Employee "  +
             "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

INSERT Clause

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)"  +
             "SELECT firstName, lastName, salary FROM old_employee";
Query query = session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

Pagination using Query

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
query.setFirstResult(1);
query.setMaxResults(10);
List results = query.list();
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. Mention the Key components of Hibernate?

hibernate.cfg.xml: This file has database connection details

hbm.xml or Annotation: Defines the database table mapping with POJO. Also defines the relation between tables in java way.
SessionFactory:
* There will be a session factory per database. * The SessionFacory is built once at start-up * It is a thread safe class * SessionFactory will create a new Session object when requested
Session:
* The Session object will get physical connection to the database. * Session is the Java object used for any DB operations. * Session is not thread safe. Hence do not share hibernate session between threads * Session represents unit of work with database * Session should be closed once the task is completed

Q. Explain Session object in Hibernate?

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. Persistent objects are saved and retrieved through a Session object.

The lifecycle of a Session is bounded by the beginning and end of a logical transaction. The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of three states:

- **transient** – A new instance of a persistent class, which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.
- **persistent** – You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.
- **detached** – Once we close the Hibernate Session, the persistent instance will become a detached instance.

```
Session session = factory.openSession();
Transaction tx = null;

try {
   tx = session.beginTransaction();
   // do some work
   ...
   tx.commit();
}

catch (Exception e) {
   if (tx!=null) tx.rollback();
   e.printStackTrace();
} finally {
   session.close();
}

<b><a href="#">↥ back to top</a></b>
```


Q. How transaction management works in Hibernate?

A **Transaction** is a sequence of operation which works as an atomic unit. A transaction only completes if all the operations completed successfully. A transaction has the Atomicity, Consistency, Isolation, and Durability properties (ACID).

In hibernate framework, **Transaction interface** that defines the unit of work. It maintains abstraction from the transaction implementation (JTA, JDBC). A transaction is associated with Session and instantiated by calling **session.beginTransaction()**.

Transaction interface	Description
void begin()	starts a new transaction.
void commit()	ends the unit of work unless we are in FlushMode.NEVER.
void rollback()	forces this transaction to rollback.
void setTimeout(int seconds)	It sets a transaction timeout for any transaction started by a subsequent call to begin on this
boolean isAlive()	checks if the transaction is still alive.
void registerSynchronization(Synchronization s)	registers a user synchronization callback for this transaction.
boolean wasCommited()	checks if the transaction is commited successfully.
boolean wasRolledBack()	checks if the transaction is rolledback successfully.

Example:

```
Transaction transObj = null;
Session sessionObj = null;
try {
    sessionObj = HibernateUtil.buildSessionFactory().openSession();
    transObj = sessionObj.beginTransaction();

    //Perform Some Operation Here
    transObj.commit();
} catch (HibernateException exObj) {
    if(transObj!=null){
        transObj.rollback();
    }
    exObj.printStackTrace();
} finally {
    sessionObj.close();
}
```

Q. Explain the Criteria object in Hibernate?

The Criteria API allows to build up a criteria query object programmatically; the `org.hibernate.Criteria` interface defines the available methods for one of these objects. The Hibernate Session interface contains several overloaded `createCriteria()` methods.

1. Restrictions with Criteria

```
Criteria cr = session.createCriteria(Employee.class);

// To get records having salary is equal to 2000
```

```
cr.add(Restrictions.eq("salary", 2000));
List results = cr.list();

// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To get records having salary less than 2000
cr.add(Restrictions.lt("salary", 2000));

// To get records having fistName starting with zara
cr.add(Restrictions.like("firstName", "zara%"));

// Case sensitive form of the above restriction.
cr.add(Restrictions.ilike("firstName", "zara%"));

// To get records having salary in between 1000 and 2000
cr.add(Restrictions.between("salary", 1000, 2000));

// To check if the given property is null
cr.add(Restrictions.isNull("salary"));

// To check if the given property is not null
cr.add(Restrictions.isNotNull("salary"));

// To check if the given property is empty
cr.add(Restrictions.isEmpty("salary"));

// To check if the given property is not empty
cr.add(Restrictions.isNotEmpty("salary"));
```

2. Logical Expression Restrictions

```
Criteria cr = session.createCriteria(Employee.class);

Criterion salary = Restrictions.gt("salary", 2000);
Criterion name = Restrictions.ilike("firstNname","zara%");

// To get records matching with OR conditions
LogicalExpression orExp = Restrictions.or(salary, name);
cr.add( orExp );

// To get records matching with AND conditions
LogicalExpression andExp = Restrictions.and(salary, name);
cr.add( andExp );

List results = cr.list();
```

3. Pagination Using Criteria

```
Criteria cr = session.createCriteria(Employee.class);
cr.setFirstResult(1);
cr.setMaxResults(10);
List results = cr.list();
```

4. Sorting the Results

The Criteria API provides the **org.hibernate.criterion.Order** class to sort your result set in either ascending or descending order, according to one of your object's properties.

```
Criteria cr = session.createCriteria(Employee.class);

// To get records having salary more than 2000
cr.add(Restrictions.gt("salary", 2000));

// To sort records in descening order
cr.addOrder(Order.desc("salary"));

// To sort records in ascending order
cr.addOrder(Order.asc("salary"));

List results = cr.list();
```

5. Projections & Aggregations

The Criteria API provides the **org.hibernate.criterion.Projections** class, which can be used to get average, maximum, or minimum of the property values. The Projections class is similar to the Restrictions class, in that it provides several static factory methods for obtaining **Projection** instances.

```
Criteria cr = session.createCriteria(Employee.class);

// To get total row count.
cr.setProjection(Projections.rowCount());

// To get average of a property.
cr.setProjection(Projections.avg("salary"));

// To get distinct count of a property.
cr.setProjection(Projections.countDistinct("firstName"));

// To get maximum of a property.
cr.setProjection(Projections.max("salary"));

// To get minimum of a property.
cr.setProjection(Projections.min("salary"));

// To get sum of a property.
cr.setProjection(Projections.sum("salary"));
```

Criteria Queries Example

Employee.java

```java
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}

   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }

   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;
   }
}
```

EMPLOYEE.sql

```
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)
);
```

Hibernate Mapping File

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

<hibernate-mapping>
   <class name = "Employee" table = "EMPLOYEE">

      <meta attribute = "class-description">
         This class contains the employee detail.
      </meta>

      <id name = "id" type = "int" column = "id">
         <generator class="native"/>
      </id>

      <property name = "firstName" column = "first_name" type =
"string"/>
      <property name = "lastName" column = "last_name" type =
"string"/>
      <property name = "salary" column = "salary" type = "int"/>

   </class>
</hibernate-mapping>
```

ManageEmployee.java

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.Criteria;
import org.hibernate.criterion.Restrictions;
import org.hibernate.criterion.Projections;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
```

```
   public static void main(String[] args) {

      try {
         factory = new
Configuration().configure().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("Zara", "Ali", 2000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 5000);
      Integer empID4 = ME.addEmployee("Mohd", "Yasee", 3000);

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

      /* Print Total employee's count */
      ME.countEmployee();

      /* Print Total salary */
      ME.totalSalary();
   }

   /* Method to CREATE an employee in the database */
   public Integer addEmployee(String fname, String lname, int salary){
      Session session = factory.openSession();
      Transaction tx = null;
      Integer employeeID = null;

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

   /* Method to  READ all the employees having salary more than 2000
```

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

      try {
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);
         // Add restriction.
         cr.add(Restrictions.gt("salary", 2000));
         List employees = cr.list();

         for (Iterator iterator = employees.iterator();
iterator.hasNext();){
            Employee employee = (Employee) 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();
      }
   }

   /* Method to print total number of records */
   public void countEmployee(){
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);

         // To get total row count.
         cr.setProjection(Projections.rowCount());
         List rowCount = cr.list();

         System.out.println("Total Coint: " + rowCount.get(0) );
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
```

```
      }
   }

   /* Method to print sum of salaries */
   public void totalSalary(){
      Session session = factory.openSession();
      Transaction tx = null;

      try {
         tx = session.beginTransaction();
         Criteria cr = session.createCriteria(Employee.class);

         // To get total salary.
         cr.setProjection(Projections.sum("salary"));
         List totalSalary = cr.list();

         System.out.println("Total Salary: " + totalSalary.get(0) );
         tx.commit();
      } catch (HibernateException e) {
         if (tx!=null) tx.rollback();
         e.printStackTrace();
      } finally {
         session.close();
      }
   }
}
```

Output

```
cmd> java ManageEmployee
.......VARIOUS LOG MESSAGES WILL DISPLAY HERE........

First Name: Daisy  Last Name: Das  Salary: 5000
First Name: John  Last Name: Paul  Salary: 5000
First Name: Mohd  Last Name: Yasee  Salary: 3000
Total Coint: 4
Total Salary: 15000
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is a One-to-One association in Hibernate?

A **One-to-One** Association is similar to Many-to-One association with a difference that the column will be set as unique i.e. Two entities are said to be in a One-to-One relationship if one entity has only one occurrence in the other entity. For example, an address object can be associated with a single employee object. However, these relationships are rarely used in the relational table models and therefore, we won't need this mapping too often.

In One-to-One association, the source entity has a field that references another target entity. The `@OneToOne` JPA annotation is used to map the source entity with the target entity.

Example: Hibernate One to One Mapping Annotation

hibernate-annotation.cfg.xml

```
<?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>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</proper
ty>
        <property
name="hibernate.connection.password">dbpassword</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost/TestDB</propert
y>
        <property
name="hibernate.connection.username">dbusername</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <property
name="hibernate.current_session_context_class">thread</property>
        <property name="hibernate.show_sql">true</property>

        <mapping class="com.example.hibernate.model.Txn1"/>
        <mapping class="com.example.hibernate.model.Customer1"/>
    </session-factory>
</hibernate-configuration>
```

For hibernate one to one mapping annotation configuration, model classes are the most important part.

```
package com.example.hibernate.model;

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import org.hibernate.annotations.Cascade;
```

```java
@Entity
@Table(name="TRANSACTION")
public class Txn1 {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="txn_id")
    private long id;

    @Column(name="txn_date")
    private Date date;

    @Column(name="txn_total")
    private double total;

    @OneToOne(mappedBy="txn")
    @Cascade(value=org.hibernate.annotations.CascadeType.SAVE_UPDATE)
    private Customer1 customer;

    @Override
    public String toString(){
        return id+", "+total+", "+customer.getName()+",
"+customer.getEmail()+", "+customer.getAddress();
    }

        //Getter-Setter methods, omitted for clarity
}


package com.example.hibernate.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name="CUSTOMER")
public class Customer1 {

    @Id
    @Column(name="txn_id", unique=true, nullable=false)
    @GeneratedValue(generator="gen")
    @GenericGenerator(name="gen", strategy="foreign",
parameters={@Parameter(name="property", value="txn")})
```

```
    private long id;

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

    @Column(name="cust_email")
    private String email;

    @Column(name="cust_address")
    private String address;

    @OneToOne
    @PrimaryKeyJoinColumn
    private Txn1 txn;

        //Getter-Setter methods
}
```

Hibernate SessionFactory Utility class

```
package com.example.hibernate.util;

import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HibernateAnnotationUtil {

    private static SessionFactory sessionFactory;

    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate-
annotation.cfg.xml
            Configuration configuration = new Configuration();
            configuration.configure("hibernate-annotation.cfg.xml");
            System.out.println("Hibernate Annotation Configuration
loaded");

            ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(configuration.getProper
ties()).build();
            System.out.println("Hibernate Annotation serviceRegistry
created");

            SessionFactory sessionFactory =
configuration.buildSessionFactory(serviceRegistry);
```

```
            return sessionFactory;
        }
        catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation
failed." + ex);
            ex.printStackTrace();
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        if(sessionFactory == null) sessionFactory =
buildSessionFactory();
        return sessionFactory;
    }
}
```

Hibernate One to One Mapping Annotation Example Test Program

```
package com.example.hibernate.main;

import java.util.Date;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.example.hibernate.model.Customer1;
import com.example.hibernate.model.Txn1;
import com.example.hibernate.util.HibernateAnnotationUtil;

public class HibernateOneToOneAnnotationMain {

    public static void main(String[] args) {

        Txn1 txn = buildDemoTransaction();

        SessionFactory sessionFactory = null;
        Session session = null;
        Transaction tx = null;
        try{
        //Get Session
        sessionFactory = HibernateAnnotationUtil.getSessionFactory();
        session = sessionFactory.getCurrentSession();
        System.out.println("Session created using annotations
configuration");
        //start transaction
        tx = session.beginTransaction();
        //Save the Model object
```

```
        session.save(txn);
        //Commit transaction
        tx.commit();
        System.out.println("Annotation Example. Transaction
ID="+txn.getId());

        //Get Saved Trasaction Data
        printTransactionData(txn.getId(), sessionFactory);
        }catch(Exception e){
            System.out.println("Exception occured. "+e.getMessage());
            e.printStackTrace();
        }finally{
            if(sessionFactory != null && !sessionFactory.isClosed()){
                System.out.println("Closing SessionFactory");
                sessionFactory.close();
            }
        }
    }

    private static void printTransactionData(long id, SessionFactory
sessionFactory) {
        Session session = null;
        Transaction tx = null;
        try{
            //Get Session
            sessionFactory =
HibernateAnnotationUtil.getSessionFactory();
            session = sessionFactory.getCurrentSession();
            //start transaction
            tx = session.beginTransaction();
            //Save the Model object
            Txn1 txn = (Txn1) session.get(Txn1.class, id);
            //Commit transaction
            tx.commit();
            System.out.println("Annotation Example. Transaction
Details=\n"+txn);

            }catch(Exception e){
                System.out.println("Exception occured.
"+e.getMessage());
                e.printStackTrace();
            }
    }

    private static Txn1 buildDemoTransaction() {
        Txn1 txn = new Txn1();
        txn.setDate(new Date());
        txn.setTotal(100);

        Customer1 cust = new Customer1();
```

```
        cust.setAddress("San Jose, USA");
        cust.setEmail("Alex@yahoo.com");
        cust.setName("Alex Kr");

        txn.setCustomer(cust);

        cust.setTxn(txn);
        return txn;
    }
}
```

Output

```
Hibernate Annotation Configuration loaded
Hibernate Annotation serviceRegistry created
Session created using annotations configuration
Hibernate: insert into TRANSACTION (txn_date, txn_total) values (?, ?)
Hibernate: insert into CUSTOMER (cust_address, cust_email, cust_name,
txn_id) values (?, ?, ?, ?)
Annotation Example. Transaction ID=20
Hibernate: select txn1x0_.txn_id as txn_id1_1_0_, txn1x0_.txn_date as
txn_date2_1_0_, txn1x0_.txn_total as txn_tota3_1_0_,
customer1x1_.txn_id as txn_id1_0_1_, customer1x1_.cust_address as
cust_add2_0_1_, customer1x1_.cust_email as cust_ema3_0_1_,
customer1x1_.cust_name as cust_nam4_0_1_ from TRANSACTION txn1x0_ left
outer join CUSTOMER customer1x1_ on
txn1x0_.txn_id=customer1x1_.txn_id where txn1x0_.txn_id=?
Annotation Example. Transaction Details=
20, 100.0, Alex Kr, Alex@yahoo.com, San Jose, USA
Closing SessionFactory
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is hibernate caching? Explain Hibernate first level cache?

Hibernate Cache can be very useful in gaining fast application performance if used correctly. The idea behind cache is to reduce the number of database queries, hence reducing the throughput time of the application.

Hibernate comes with different types of Cache:

First Level Cache: Hibernate first level cache is associated with the **Session object**. Hibernate uses this cache by default. Here, it processes one transaction after another one, means wont process one transaction many times. Mainly it reduces the number of SQL queries it needs to generate within a given transaction. That is instead of updating after every modification done in the transaction, it updates the transaction only at the end of the transaction.

Second Level Cache: Second-level cache always associates with the **Session Factory object**. While running the transactions, in between it loads the objects at the Session Factory level, so that those objects will be available to the entire application, not bound to

single user. Since the objects are already loaded in the cache, whenever an object is returned by the query, at that time no need to go for a database transaction. In this way the second level cache works. Here we can use query level cache also.

Hibernate Second Level cache is disabled by default but we can enable it through configuration. Currently EHCache and Infinispan provides implementation for Hibernate Second level cache and we can use them. We will look into this in the next tutorial for hibernate caching.

Query Cache: Hibernate can also cache result set of a query. Hibernate Query Cache doesn't cache the state of the actual entities in the cache; it caches only identifier values and results of value type. So it should always be used in conjunction with the second-level cache.

Example: Hibernate Caching – First Level Cache

```
package com.example.hibernate.main;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.example.hibernate.model.Employee;
import com.example.hibernate.util.HibernateUtil;

public class HibernateCacheExample {

    public static void main(String[] args) throws InterruptedException
{

        SessionFactory sessionFactory =
HibernateUtil.getSessionFactory();
        Session session = sessionFactory.getCurrentSession();
        Transaction tx = session.beginTransaction();

        //Get employee with id=1
        Employee emp = (Employee) session.load(Employee.class, new
Long(1));
        printData(emp,1);

        //waiting for sometime to change the data in backend
        Thread.sleep(10000);

        //Fetch same data again, check logs that no query fired
        Employee emp1 = (Employee) session.load(Employee.class, new
Long(1));
        printData(emp1,2);

        //Create new session
```

```
        Session newSession = sessionFactory.openSession();
        //Get employee with id=1, notice the logs for query
        Employee emp2 = (Employee) newSession.load(Employee.class, new
Long(1));
        printData(emp2,3);

        //START: evict example to remove specific object from
hibernate first level cache
        //Get employee with id=2, first time hence query in logs
        Employee emp3 = (Employee) session.load(Employee.class, new
Long(2));
        printData(emp3,4);

        //evict the employee object with id=1
        session.evict(emp);
        System.out.println("Session Contains Employee with
id=1?"+session.contains(emp));

        //since object is removed from first level cache, you will see
query in logs
        Employee emp4 = (Employee) session.load(Employee.class, new
Long(1));
        printData(emp4,5);

        //this object is still present, so you won't see query in logs
        Employee emp5 = (Employee) session.load(Employee.class, new
Long(2));
        printData(emp5,6);
        //END: evict example

        //START: clear example to remove everything from first level
cache
        session.clear();
        Employee emp6 = (Employee) session.load(Employee.class, new
Long(1));
        printData(emp6,7);
        Employee emp7 = (Employee) session.load(Employee.class, new
Long(2));
        printData(emp7,8);

        System.out.println("Session Contains Employee with
id=2?"+session.contains(emp7));

        tx.commit();
        sessionFactory.close();
    }

    private static void printData(Employee emp, int count) {
        System.out.println(count+":: Name="+emp.getName()+",
Zipcode="+emp.getAddress().getZipcode());
```

```
    }
}
```

Output

```
Hibernate Configuration loaded
Hibernate serviceRegistry created
Hibernate: select employee0_.emp_id as emp_id1_1_0_,
employee0_.emp_name as emp_name2_1_0_, employee0_.emp_salary as
emp_sala3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE
employee0_ left outer join ADDRESS address1_ on
employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
1:: Name=Alex, Zipcode=95129
2:: Name=Alex, Zipcode=95129
Hibernate: select employee0_.emp_id as emp_id1_1_0_,
employee0_.emp_name as emp_name2_1_0_, employee0_.emp_salary as
emp_sala3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE
employee0_ left outer join ADDRESS address1_ on
employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
3:: Name=AlexK, Zipcode=95129
Hibernate: select employee0_.emp_id as emp_id1_1_0_,
employee0_.emp_name as emp_name2_1_0_, employee0_.emp_salary as
emp_sala3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE
employee0_ left outer join ADDRESS address1_ on
employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
4:: Name=David, Zipcode=95051
Session Contains Employee with id=1?false
Hibernate: select employee0_.emp_id as emp_id1_1_0_,
employee0_.emp_name as emp_name2_1_0_, employee0_.emp_salary as
emp_sala3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE
employee0_ left outer join ADDRESS address1_ on
employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
5:: Name=Alex, Zipcode=95129
6:: Name=David, Zipcode=95051
Hibernate: select employee0_.emp_id as emp_id1_1_0_,
employee0_.emp_name as emp_name2_1_0_, employee0_.emp_salary as
emp_sala3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE
employee0_ left outer join ADDRESS address1_ on
employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
7:: Name=Alex, Zipcode=95129
Hibernate: select employee0_.emp_id as emp_id1_1_0_,
```

```
employee0_.emp_name as emp_name2_1_0_, employee0_.emp_salary as
emp_sala3_1_0_, address1_.emp_id as emp_id1_0_1_,
address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE
employee0_ left outer join ADDRESS address1_ on
employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
8:: Name=David, Zipcode=95051
Session Contains Employee with id=2?true
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is second level cache in Hibernate?

Hibernate second level cache uses a common cache for all the session object of a session factory. It is useful if you have multiple session objects from a session factory. **SessionFactory** holds the second level cache data. It is global for all the session objects and not enabled by default.

Different vendors have provided the implementation of Second Level Cache.

- EH Cache
- OS Cache
- Swarm Cache
- JBoss Cache

Each implementation provides different cache usage functionality. There are four ways to use second level cache.

- **read-only**: caching will work for read only operation.
- **nonstrict-read-write**: caching will work for read and write but one at a time.
- **read-write**: caching will work for read and write, can be used simultaneously.
- **transactional**: caching will work for transaction.

The cache-usage property can be applied to class or collection level in hbm.xml file.

```
<cache usage="read-only" />
```

Example: Hibernate Second Level Cache

Step 01: Create the persistent class using Maven

```
package com.example;
import javax.persistence.*;
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
@Entity
@Table(name="emp1012")
@Cacheable
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
public class Employee {
    @Id
```

```
    private int id;
    private String name;
    private float salary;

    public Employee() {}
    public Employee(String name, float salary) {
        super();
        this.name = name;
        this.salary = salary;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public float getSalary() {
        return salary;
    }
    public void setSalary(float salary) {
        this.salary = salary;
    }
}
```

Step 02: Add project information and configuration in pom.xml file.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.16.Final</version>
</dependency>

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>

<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.10.3</version>
</dependency>
```

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
    <version>5.2.16.Final</version>
</dependency>
```

Step 03: Create the Configuration file (hibernate.cfg.xml)

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 5.2.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-
5.2.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>

        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</proper
ty>

         <property name="cache.use_second_level_cache">true</property>

         <property
name="cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheR
egionFactory</property>
         <mapping class="com.example.Employee"/>
    </session-factory>
</hibernate-configuration>
```

Step 04: Create the class that retrieves the persistent object

```
package com.example;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchTest {
```

```
   public static void main(String[] args) {

       StandardServiceRegistry ssr = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build(
);
       Metadata meta = new
MetadataSources(ssr).getMetadataBuilder().build();

       SessionFactory factory =
meta.getSessionFactoryBuilder().build();

       Session session1 = factory.openSession();
       Employee emp1 = (Employee)session1.load(Employee.class,121);

       System.out.println(emp1.getId()+" "+emp1.getName()+"
"+emp1.getSalary());
       session1.close();

       Session session2 = factory.openSession();
       Employee emp2 = (Employee)session2.load(Employee.class,121);

       System.out.println(emp2.getId()+" "+emp2.getName()+"
"+emp2.getSalary());
       session2.close();

   }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What are concurrency strategies?

The READ_WRITE strategy is an asynchronous cache concurrency mechanism and to prevent data integrity issues (e.g. stale cache entries), it uses a locking mechanism that provides unit-of-work isolation guarantees.

In hibernate, cache concurrency strategy can be set globally using the property hibernate.cache. default_cache_concurrency_strategy. The allowed values are,

- **read-only**: caching will work for read only operation. supported by ConcurrentHashMap, EHCache, Infinispan
- **nonstrict-read-write**: caching will work for read and write but one at a time. supported by ConcurrentHashMap, EHCache.
- **read-write**: caching will work for read and write, can be used simultaneously. supported by ConcurrentHashMap, EHCache.
- **transactional**: caching will work for transaction. supported by EHCache, Infinispan.

Example: Inserting data (READ_WRITE strategy)

```
@Override
public boolean afterInsert(
    Object key, Object value, Object version)
        throws CacheException {
    region().writeLock( key );
    try {
        final Lockable item =
            (Lockable) region().get( key );
        if ( item == null ) {
            region().put( key,
                new Item( value, version,
                    region().nextTimestamp()
                )
            );
            return true;
        }
        else {
            return false;
        }
    }
    finally {
        region().writeUnlock( key );
    }
}
```

For an entity to be cached upon insertion, it must use a SEQUENCE generator, the cache being populated by the EntityInsertAction:

```
@Override
public void doAfterTransactionCompletion(boolean success,
    SessionImplementor session)
    throws HibernateException {

    final EntityPersister persister = getPersister();
    if ( success && isCachePutEnabled( persister,
        getSession() ) ) {
            final CacheKey ck = getSession()
               .generateCacheKey(
                    getId(),
                    persister.getIdentifierType(),
                    persister.getRootEntityName() );

            final boolean put = cacheAfterInsert(
                persister, ck );
        }
    }
    postCommitInsert( success );
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is Lazy loading in hibernate?

Hibernate defaults to a lazy fetching strategy for all entities and collections. Lazy loading in hibernate improves the performance. It loads the child objects on demand. To enable lazy loading explicitly you must use **fetch = FetchType.LAZY** on a association which you want to lazy load when you are using hibernate annotations.

Example:

```
@OneToMany( mappedBy = "category", fetch = FetchType.LAZY )
private Set<ProductEntity> products;
```

Q. Explain the persistent classes in Hibernate?

Persistence class are simple POJO classes in an application. It works as implementation of the business application for example Employee, department etc. It is not necessary that all instances of persistence class are defined persistence.

There are following main rules of persistent classes

- A persistence class should have a default constructor.
- A persistence class should have an id to uniquely identify the class objects.
- All attributes should be declared private.
- Public getter and setter methods should be defined to access the class attributes.

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import java.util.Date;

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

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

    @Column(name = "salary")
    private int salary;

    @Column(name = "date_of_join")
    private Date dateOfJoin;

    public int getId() {
```

```
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getSalary() {
        return salary;
    }

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

    public Date getDateOfJoin() {
        return dateOfJoin;
    }

    public void setDateOfJoin(Date dateOfJoin) {
        this.dateOfJoin = dateOfJoin;
    }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. Explain some of the elements of hbm.xml?

Hibernate mapping file is used by hibernate framework to get the information about the mapping of a POJO class and a database table.

It mainly contains the following mapping information:

- Mapping information of a POJO class name to a database table name.
- Mapping information of POJO class properties to database table columns.

Elements of the Hibernate mapping file:

- **hibernate-mapping**: It is the root element.
- **Class**: It defines the mapping of a POJO class to a database table.
- **Id**: It defines the unique key attribute or primary key of the table.

- **generator**: It is the sub element of the id element. It is used to automatically generate the id.
- **property**: It is used to define the mapping of a POJO class property to database table column.

Syntax

```
<hibernate-mapping>

 <class name="POJO class name" table="table name in database">
  <id name="propertyName" column="columnName" type="propertyType" >
    <generator class="generatorClass"/>
  </id>
  <property name="propertyName1" column="colName1" type="propertyType
" />
  <property name="propertyName2" column="colName2" type="propertyType
" />
    . . . . .
 </class>

</hibernate-mapping>
```

Q. What is Java Persistence API (JPA)?

Java Persistence API is a collection of classes and methods to persistently store the vast amounts of data into a database. The Java Persistence API (JPA) is one possible approach to ORM. Via JPA the developer can map, store, update and retrieve data from relational databases to Java objects and vice versa. JPA can be used in Java-EE and Java-SE applications.

JPA metadata is typically defined via annotations in the Java class. Alternatively, the metadata can be defined via XML or a combination of both. A XML configuration overwrites the annotations.

Relationship Mapping

JPA allows to define relationships between classes. Classes can have one to one, one to many, many to one, and many to many relationships with other classes. A relationship can be bidirectional or unidirectional, e.g. in a bidirectional relationship both classes store a reference to each other while in an unidirectional case only one class has a reference to the other class.

Relationship annotations:

- @OneToOne
- @OneToMany
- @ManyToOne
- @ManyToMany

Java Exception

Java Exception

JPA - Architecture

Units	Description
EntityManagerFactory	This is a factory class of EntityManager. It creates and manages multiple EntityManager instances.
EntityManager	It is an Interface, it manages the persistence operations on objects. It works like factory for Query instance.
Entity	Entities are the persistence objects, stores as records in the database.
EntityTransaction	It has one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by EntityTransaction class.
Persistence	This class contain static methods to obtain EntityManagerFactory instance.
Query	This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria.

```
<b><a href="#">↥ back to top</a></b>
```

Q. Name some important interfaces of Hibernate framework?

- **Session Interface**: This is the primary interface used by hibernate applications The instances of this interface are lightweight and are inexpensive to create and destroy Hibernate sessions are not thread safe
- **Session Factory Interface**: This is a factory that delivers the session objects to hibernate application.
- **Configuration Interface**: This interface is used to configure and bootstrap hibernate. The instance of this interface is used by the application in order to specify the location of hbm documents
- **Transaction Interface**: This interface abstracts the code from any kind of transaction implementations such as JDBC transaction, JTA transaction
- **Query and Criteria Interface**: This interface allows the user to perform queries and also control the flow of the query execution

Q. What is Hibernate SessionFactory and how to configure it?

SessionFactory can be created by providing Configuration object, which will contain all DB related property details pulled from either hibernate.cfg.xml file or hibernate.properties file. SessionFactory is a factory for Session objects.

We can create one SessionFactory implementation per database in any application. If your application is referring to multiple databases, then we need to create one SessionFactory per database.

The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. The SessionFactory is a thread safe object and used by all the threads of an application.

```
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

import com.example.model.Employee;

public class HibernateUtil {

    private static SessionFactory sessionFactory = null;

    static {
        try {
            loadSessionFactory();
        } catch(Exception e){
            System.err.println("Exception while initializing hibernate
util.. ");
            e.printStackTrace();
        }
    }

    public static void loadSessionFactory() {

        Configuration configuration = new Configuration();
        configuration.configure("/j2n-hibernate.cfg.xml");
        configuration.addAnnotatedClass(Employee.class);
        ServiceRegistry srvcReg = new
StandardServiceRegistryBuilder().applySettings(configuration.getProper
ties()).build();
        sessionFactory = configuration.buildSessionFactory(srvcReg);
    }

    public static Session getSession() throws HibernateException {

        Session retSession=null;
            try {
                retSession = sessionFactory.openSession();
            } catch(Throwable t) {
                System.err.println("Exception while getting session..
");
```

```
            t.printStackTrace();
        }
        if(retSession == null) {
            System.err.println("session is discovered null");
        }
        return retSession;
    }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is the difference between opensession and getcurrentsession in hibernate?

Hibernate SessionFactory getCurrentSession() method returns the session bound to the context. Since this session object belongs to the hibernate context, we don't need to close it. Once the session factory is closed, this session object gets closed.

Hibernate SessionFactory openSession() method always opens a new session. We should close this session object once we are done with all the database operations.

Parameter	openSession	getCurrentSession
Session object	It always create new Session object	It creates a new Session if not exists , else use same session which is in current hibernate context
Flush and close	You need to explicitly flush and close session objects	You do not need to flush and close session objects, it will be automatically taken care by Hibernate internally
Performance	In single threaded environment , It is slower than getCurrentSession	In single threaded environment , It is faster than getOpenSession
Configuration	You do not need to configure any property to call this method	You need to configure additional property "hibernate.current_session_context_class" to call getCurrentSession method, otherwise it will throw exceptions.

Example: openSession()

```
Transaction transaction = null;
Session session = HibernateUtil.getSessionFactory().openSession();
try {
    transaction = session.beginTransaction();
    // do Some work

    session.flush(); // extra work
    transaction.commit();
```

```
} catch(Exception ex) {
    if(transaction != null) {
        transaction.rollback();
    }
    ex.printStackTrace();
} finally {
    if(session != null) {
        session.close(); // extra work
    }
}
```

Example: getCurrentSession()

```
Transaction transaction = null;
Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
try {
    transaction = session.beginTransaction();
    // do Some work

    // session.flush(); // no need
    transaction.commit();
} catch(Exception ex) {
    if(transaction != null) {
        transaction.rollback();
    }
    ex.printStackTrace();
} finally {
    if(session != null) {
        // session.close(); // no need
    }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is difference between Hibernate Session get() and load() method?

Hibernate Session class provides two method to access object e.g. `session.get()` and `session.load()`.The difference between get() vs load() method is that get() involves database hit if object doesn't exists in **Session Cache** and returns a fully initialized object which may involve several database call while load method can return proxy in place and only initialize the object or hit the database if any method other than getId() is called on persistent or entity object. This lazy initialization can save couple of database round-trip which result in better performance.

load()	get()
Only use the load() method if you are sure that the object exists.	If you are not sure that the object exists, then use one of the get() methods.
load() method will throw an exception if	get() method will return null if the unique id

load()	get()
the unique id is not found in the database.	is not found in the database.
load() just returns a proxy by default and database won't be hit until the proxy is first invoked.	get() will hit the database immediately.

Q. What are different states of an entity bean?

The Entity bean has three states:

1. Transient: Transient objects exist in heap memory. Hibernate does not manage transient objects or persist changes to transient objects. Whenever pojo class object created then it will be in the Transient state.

Transient state Object does not represent any row of the database, It means not associated with any Session object or no relation with the database its just an normal object.

2. Persistent: Persistent objects exist in the database, and Hibernate manages the persistence for persistent objects. If fields or properties change on a persistent object, Hibernate will keep the database representation up to date when the application marks the changes as to be committed.

Any instance returned by a **get()** or **load()** method is persistent state object.

3. Detached: Detached objects have a representation in the database, but changes to the object will not be reflected in the database, and vice-versa. A detached object can be created by closing the session that it was associated with, or by evicting it from the session with a call to the session's `evict()` method.

Example:

```
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.AnnotationConfiguration;

import com.example.hibernate.Student;

public class ObjectStatesDemo {

    public static void main(String[] args) {

        // Transient object state
        Student student = new Student();
        student.setId(101);
        student.setName("Alex");
        student.setRoll("10");
        student.setDegree("B.E");
        student.setPhone("9999");
```

```
        // Transient object state
        Session session = new AnnotationConfiguration().configure()
                .buildSessionFactory().openSession();
        Transaction t = session.beginTransaction();

        // Persistent object state
        session.save(student);
        t.commit();

        // Detached object state
        session.close();
    }
}
```

Output

```
Hibernate:
    insert
    into
        STUDENT
        (degree, name, phone, roll, id)
    values
        (?, ?, ?, ?, ?)
```

In The Database :

```
mysql> select * from student;
+-----+--------+--------+-------+------+
| id  | degree | name   | phone | roll |
+-----+--------+--------+-------+------+
| 101 | B.E    | Alex   | 9999  | 10   |
+-----+--------+--------+-------+------+
1 row in set (0.05 sec)
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is difference between merge() and update() methods in Hibernate?

Both update() and merge() methods in hibernate are used to convert the object which is in detached state into persistence state. But there are different situation where we should be used update() and where should be used merge() method in hibernate

```
Employee emp1 = new Employee();
emp1.setEmpId(100);
emp1.setEmpName("Alex");
//create session
Session session1 = createNewHibernateSession();
session1.saveOrUpdate(emp1);
session1.close();
//emp1 object in detached state now
```

```
emp1.setEmpName("Alex Rajput");// Updated Name
//Create session again
Session session2 = createNewHibernateSession();
Employee emp2 =(Employee)session2.get(Employee.class, 100);
//emp2 object in persistent state with id 100

/**
Try to make on detached object with id 100 to persistent state by
using update method of hibernate
It occurs the exception NonUniqueObjectException because emp2 object
is having employee with same
empid as 100. In order to avoid this exception we are using merge like
given below instead of
**/
session2.update(emp1);
session.update(emp1);

session2.merge(emp1); //it merge the object state with emp2
session2.update(emp1); //Now it will work with exception
```

In the hibernate session we can maintain only one employee object in persistent state with same primary key, while converting a detached object into persistent, if already that session has a persistent object with the same primary key then hibernate throws an Exception whenever update() method is called to reattach a detached object with a session. In this case we need to call **merge()** method instead of **update()** so that hibernate copies the state changes from detached object into persistent object and we can say a detached object is converted into a persistent object.

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is difference between Hibernate save(), saveOrUpdate() and persist() methods?

Q. What will happen if we don't have no-args constructor in Entity bean?

Q. What is difference between sorted collection and ordered collection, which one is better?

Q. What are the collection types in Hibernate?

Q. How to implement Joins in Hibernate?

Q. Why we should not make Entity Class final?

Q. What is the benefit of native sql query support in hibernate?

Q. What is Named SQL Query? What are the benefits of Named SQL Query?

Q. How to log hibernate generated sql queries in log files?

Q. What is cascading and what are different types of cascading?

Q. How to integrate log4j logging in hibernate application?

Q. What is HibernateTemplate class?

Q. How to integrate Hibernate with Servlet or Struts2 web applications?

Q. Which design patterns are used in Hibernate framework?

Q. What is Hibernate Validator Framework?

Q. What is the benefit of Hibernate Tools Eclipse plugin?

Q. What are the technologies that are supported by Hibernate?

Q. What is your understanding of Hibernate proxy?

Q. Can you explain Hibernate callback interfaces?

Q. How to create database applications in Java with the use of Hibernate?

Q. Can you share your views on mapping description files?

Q. What are your thoughts on file mapping in Hibernate?

Q. Can you explain version field?

Q. What are your views on the function addClass?

Q. Can you explain the role of addDirectory() and addjar() methods?

Q. What do you understand by Hibernate tuning?

Q. What is your understanding of Light Object Mapping?

Q. How does Hibernate create the database connection?

Q. What are possible ways to configure object-table mapping?

Q. Which annotation is used to declare a class as a hibernate bean?

Q. How do I specify table name linked to an entity using annotation?

Q. How does a variable in an entity connect to the database column?

Q. How do we specify a different column name for the variables mapping?

Q. How do we specify a variable to be primary key for the table?

Q. How do you configure the dialect in hibernate.cfg.xml?

Q. How to configure the database URL and credentials in hibernate.cfg.xml?

Q. How to configure the connection pool size?

Q. How do you configure folder scan for Hibernate beans?

Q. How to configure hibernate beans without Spring framework?

Q. Is it possible to connect multiple database in a single Java application using Hibernate?

Q. Does Hibernate support polymorphism?

Q. How many Hibernate sessions exist at any point of time in an application?

Q. What is N+1 SELECT problem in Hibernate? What are some strategies to solve the N+1 SELECT problem in Hibernate?

Q. What is the requirement for a Java object to become a Hibernate entity object?

Q. How do you log SQL queries issued by the Hibernate framework in Java application?

Q. What is the difference between the transient, persistent and detached state in Hibernate?

Q. How properties of a class are mapped to the columns of a database table in Hibernate?

Q. What is the usage of Configuration Interface in hibernate?

Q. How can we use new custom interfaces to enhance functionality of built-in interfaces of hibernate?

Q. What are POJOs and what is their significance?

Q. How can we invoke stored procedures in hibernate?

Q. What are the benefits of using Hibernate template?

Q. How can we get hibernate statistics?

Q. How can we reduce database write action times in Hibernate?

Q. When an instance goes in detached state in hibernate?

Q. What the four ORM levels are in hibernate?

Q. What is the default cache service of hibernate?

Q. What are the two mapping associations used in hibernate?

Q. What is the usage of Hibernate QBC API?

Q. In how many ways, objects can be fetched from database in hibernate?

Q. How primary key is created by using hibernate?

Q. How can we reattach any detached objects in Hibernate?

Q. What are different ways to disable hibernate second level cache?

Q. What is ORM metadata?

Q. Which one is the default transaction factory in hibernate?

Q. What is the role of JMX in hibernate?

Q. In how many ways objects can be identified in Hibernate?

Q. What different fetching strategies are of hibernate?

Q. How mapping of java objects is done with database tables?

Q. What are derived properties in hibernate?

Q. What is the use of version property in hibernate?

Q. What is attribute oriented programming?

Q. What is the use of session.lock() in hibernate?

Q. What the three inheritance models are of hibernate?

Q. What is general hibernate flow using RDBMS?

Q. What is difference between managed associations and hibernate associations?

Q. What are the inheritance mapping strategies?

Q. What is automatic dirty checking in hibernate?

Q. Explain Hibernate configuration file and Hibernate mapping file?

↥ back to top

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project