1. What is core concept of ORM?

Ans: An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.

1. What is Hibernate and why it is necessary?

Ans: Because It helps developers get rid of writing complex and tedious SQL statement, no more need of JDBC APIs for result set or data handling. It makes developers more concentrate on the business logic and increase the project’s productivity.

1. Hibernate is Object Oriented or Procedure Oriented?

Ans: Hibernate is object oriented.

4. What are the Advantages of Hibernate?

Ans: Advantages of hibernate:

1. Hibernate is database independent, same code will work for all data bases like ORACLE,MySQL ,SQLServer etc.

2. As Hibernate is set of Objects , you don't need to learn SQL language. You can treat TABLE as a Object . In case of JDBC you need to learn SQL.

3. Don't need Query tuning in case of Hibernate. If you use Criteria Quires in Hibernate then hibernate automatically tuned your query and return best result with performance.

4. You will get benefit of Cache. Hibernate support two level of cache. First level and 2nd level. So you can store your data into Cache for better performance.

5. Development fast in case of Hibernate because you don't need to write queries. 6. In the xml file you can see all the relations between tables in case of Hibernate. Easy readability.

5. Define Annotations and it’s usage in hibernate?

Ans: Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Annotations have a number of uses, among them:

 Information for the compiler — Annotations can be used by the compiler to detect errors or suppress warnings.

 Compile-time and deployment-time processing — Software tools can process annotation information to generate code, XML files, and so forth.

 Runtime processing — Some annotations are available to be examined at runtime.



6. Describe the Advantages of ORM

Ans: There are a number of benefits to using an ORM for development of database applications and here’s four: 1. 1.Productivity: When using an ORM tool, the amount of code is unlikely to be reduced—in fact, it might even go up—but the ORM tool generates 100% of the data access code automatically based on the data model you define, in mere moments.

2. Application design: A good ORM tool designed by very experienced software architects will implement effective design patterns that almost force you to use good programming practices in an application.

3. Code Reuse: If you create a class library to generate a separate DLL for the ORM-generated data access code, you can easily reuse the data objects in a variety of applications.

4. Application Maintainability: All of the code generated by the ORM is presumably well-tested, so you usually don’t need to worry about testing it extensively. Obviously you need to make sure that the code does what you need, but a widely used ORM is likely to have code banged on by many developers at all skill levels.

7. Describe Architecture or ORM

Ans:



Phase 1:

In first phase Object data contains pojo(plain old java object) classes, interfaces . It is the main business components layer.

o Phase 2:

In second phase names as mapping or persistence phase which contains JPA provider, mapping files, JPA loader.

o Phase 3:

It is relational database phase. It contains the relational data which is logically connected to business components

8. What is sessionfactory object?

Ans: Session factory objects are to be implemented using the singleton design pattern. Instances of SessionFactory are thread-safe and typically shared throughout an application. As these objects are heavy weight because they contains the connection information, hibernate configuration information and mapping files,location path. So creating number of instances will make our application heavy weight.

9. What are the types of cascadetypes?

1. Ans: CascadeType.PERSIST: Save operations to related entities.

2. CascadeType.MERGE: Related entities are merged.

3. CascadeType.DETACH: detach all entities if manual detach occurs

4. CascadeType.All: It is shorthand for all the above operations

10. List the methods while using the transaction concepts?

• Ans: Methods Of Transaction Interface are as follows

• Void begin() -> Starts a new transaction.

• Void commit() -> ends the unit of work.

• Void rollback() -> forces this transaction to rollback.

• Void setTimeout(int secionds) -> sets the time for transaction.

• Boolean isAlive() ->checks if the transaction is still alive.

• Void registerSynchronization(Sychronization s) -> registers the user synchronization callback for this transaction. Boolean wasCommited() -> checks if the transaction is committed successfully.

• Boolean wasRollback() -> checks if the transaction is rolled back successfully.

11. How the cache memory increases the performance?

Ans: In hibernate we have two levels of caching

 First Level Cache [ or ] Session Cache

 Second Level Cache [ or ] Session Factory Cache [ or ] JVM Level Cache  Every fresh session having its own cache memory, Caching is a mechanism for storing the loaded objects into a cache memory. The advantage of cache mechanism is, whenever again we want to load the same object from the database then instead of hitting the database once again, it loads from the local cache memory only, so that the no. of round trips between an application and a database server got decreased. It means caching mechanism increases the performance of the application.

12. Define Hit ratio and miss ratio.

Ans: The performance of memory is frequently measured in terms of quantity is called hit ratio. When the hibernate needs to find the word in the cache, if the word is found in the cache the then its produces a hit. If the word is not found in the cache, it is in main memory as counted miss. the ratio of number of hits is divided by the total hibernate reference of memory is called hit ratio.

13. Create one application with using Hibernate ORM

Ans : Crud operations using hibernate. package com.sdnext.hibernate.tutorial.dto; import java.io.Serializable; import javax.persistence.Column; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id; @Entity

@Table(name="STUDENT")

public class Student implements Serializable {

serialVersionUID = 8633415090390966715L;

@Id @Column(name="ID") @GeneratedValue(strategy=GenerationType.AUTO) private int id;

@Column(name="STUDENT\_NAME")

private String studentName; @Column(name="ROLL\_NUMBER")

private int rollNumber;

@Column(name="COURSE")

private String course;

public int getId() { return id; }

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

public String getStudentName() { returnstudentName;}

public void setStudentName(String studentName) { this.studentName = studentName; }

public int getRollNumber() { return rollNumber; } public void setRollNumber(int rollNumber) { this.rollNumber = rollNumber; }

public String getCourse() { return course; }

public void setCourse(String course) {

this.course = course; } }

Step 2: Create Hibernate Configuration file hibernate.cfg.xml

<hibernate-configuration>

<session-factory> <!-- Database connection settings --> <property name="connection.driver\_class">com.mysql.jdbc.Driver</property>

<propertyname="connection.url">jdbc:mysql://localhost:3306/hibernateDB2</property>

<propertyname="connection.username">root</property>

<propertyname="connection.password">root</property>

<!-- JDBC connection pool (use the built-in) --> <propertyname="connection.pool\_size">1</property>

<!-- SQL dialect -->

<propertyname="dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Enable Hibernate's automatic session context management -->

<propertyname="current\_session\_context\_class">thread</property>

<!-- Disable the second-level cache -->

<propertyname="cache.provider\_class">org.hibernate.cache.NoCacheProvider</property> <!-- Echo all executed SQL to stdout --> <property name="show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hbm2ddl.auto">update</property>

<mappingclass="com.sdnext.hibernate.tutorial.dto.Student"> </mapping>

</session-factory> </hibernate-configuration>

Step 3: Create Hibernate Utility Class HibernateUtil.java

package com.sdnext.hibernate.tutorial.utility;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.AnnotationConfiguration; public class HibernateUtil {

private static final SessionFactory sessionFactory; static {

try{

sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory(); } catch(Throwable th){ System.err.println("Enitial SessionFactory creation failed"+th); throw new ExceptionInInitializerError(th); } }

public static SessionFactory getSessionFactory(){ return sessionFactory; } }

Step 4: Create Student on the database. CreateStudent.java

package com.sdnext.hibernate.tutorial;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import com.sdnext.hibernate.tutorial.dto.Student; importcom.sdnext.hibernate.tutorial.utility.HibernateUtil;

public class CreateStudent {

/\*\* \* @param args \*/

public static void main(String[] args) {

//Create student entity object

Student student = new Student(); student.setStudentName("Dinesh Rajput"); student.setRollNumber(01); student.setCourse("MCA");

//Create session factory object

SessionFactory sessionFactory = HibernateUtil.getSessionFactory();

//getting session object from session factory Session session = sessionFactory.openSession(); //getting transaction object from session object session.beginTransaction();

session.save(student); System.out.println("Inserted Successfully"); session.getTransaction().commit(); session.close(); sessionFactory.close(); } }

Step 5: Reading the Student data from the database table STUDENT ReadStudent.java

package com.sdnext.hibernate.tutorial;

import java.util.List; import org.hibernate.Query; import org.hibernate.Session;

import org.hibernate.SessionFactory; import com.sdnext.hibernate.tutorial.dto.Student;

import com.sdnext.hibernate.tutorial.utility.HibernateUtil;

public class ReadStudent {

/\*\* \* @param args \*/

public static void main(String[] args) {

//Create session factory object SessionFactory sessionFactory = HibernateUtil.getSessionFactory(); //getting session object from session factory Session session = sessionFactory.openSession();

//getting transaction object from session object session.beginTransaction(); Query query = session.createQuery("from Student"); List students = query.list(); for(Student student : students) { System.out.println("Roll Number: "+student.getRollNumber()+", Student Name: "+student.getStudentName()+", Course: "+student.getCourse()); } session.getTransaction().commit(); sessionFactory.close(); }

}

Step 6: Update the Student Record in the Database. UpdateStudent.java

package com.sdnext.hibernate.tutorial;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import com.sdnext.hibernate.tutorial.dto.Student; import com.sdnext.hibernate.tutorial.utility.HibernateUtil; public class UpdateStudent { /\*\* \* @param args \*/ public static void main(String[] args) {

//Create session factory object SessionFactory sessionFactory = HibernateUtil.getSessionFactory();

//getting session object from session factory Session session = sessionFactory.openSession();

//getting transaction object from session object session.beginTransaction(); Student student = (Student)session.get(Student.class, 2); student.setStudentName("Sweety Rajput"); System.out.println("Updated Successfully"); session.getTransaction().commit(); sessionFactory.close(); } }

Step 7: Delete the student data from the database. DeleteStudent.java package com.sdnext.hibernate.tutorial;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import com.sdnext.hibernate.tutorial.dto.Student;

import com.sdnext.hibernate.tutorial.utility.HibernateUtil; public class DeleteStudent { /\*\* \* @param args \*/ public static void main(String[] args) {

//Create session factory object SessionFactory sessionFactory = HibernateUtil.getSessionFactory(); //getting session object from session factory Session session = sessionFactory.openSession();

//getting transaction object from session object session.beginTransaction();

Student student = (Student)session.load(Student.class, 4); session.delete(student);

System.out.println("Deleted Successfully"); session.getTransaction().commit(); sessionFactory.close(); } }

**14.What are the Core interfaces are of Hibernate framework?**

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

* Session interface
* SessionFactory interface
* Configuration interface
* Transaction interface
* Query and Criteria interfaces

### 15.How to implement Joins in Hibernate?

There are various ways to implement joins in hibernate.Using associations such as one-to-one, one-to-many etc.

Using JOIN in the HQL query. There is another form “join fetch” to load associated data simultaneously, no lazy loading.

use join keyword.

### 16. How transaction management works in Hibernate?

Transaction management is very easy in hibernate because most of the operations are not permitted outside of a transaction. So after getting the session from SessionFactory, we can call session beginTransaction() to start the transaction. This method returns the Transaction reference that we can use later on to either commit or rollback the transaction.

Overall hibernate transaction management is better than JDBC transaction management because we don’t need to rely on exceptions for rollback. Any exception thrown by session methods automatically rollback the transaction.