**Q : What is hibernate?**

Hibernate is an open-source and lightweight ORM tool that is used to store, manipulate, and retrieve data from the database.

Hibernate is a popular framework of Java which allows an efficient Object Relational mapping using configuration files in XML format. After java objects mapping to database tables, database is used and handled using Java objects without writing complex database queries.

**Q : What is ORM?**

ORM is an acronym for Object/Relational mapping. It is a programming strategy to map object with the data stored in the database. It simplifies data creation, data manipulation, and data access.

ORM (Object Relational Mapping) is the fundamental concept of Hibernate framework which maps database tables with Java Objects and then provides various API’s to perform different types of operations on the data tables.

### **Q :Explain hibernate architecture?**



Hibernate framework uses many objects such as session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.



**Q : What are the core interfaces of Hibernate?**

The core interfaces of Hibernate framework are:

Configuration

SessionFactory

Session

Query & Criteria

Transaction

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.

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

1. **Configuration**:  
The Configuration object is used to configure hibernate. The application uses a Configuration instance to specify the location of mapping documents and Hibernate-specific properties and then create the SessionFactory.

Configuration cfg = new Configuration();

cfg.addResource("hello/Message.hbm.xml");

cfg.setProperties( System.getProperties() );

1. **Transaction**:  
   A transaction represents a unit of work.

Application uses transactions to do some operations on DB. Within one transaction you can do several operations and can commit transaction once after successfully completed all operations.

The advantage here is you can rollback all previous operations if one operation is fail in your operation batch.  
The Transaction does not get committed when session gets flushed.  
The Transaction interface is an optional API.

Hibernate applications may choose not to use this interface, instead managing transactions in their own infrastructure code.

A Transaction abstracts application code from the underlying transaction implementation-which might be a JDBC transaction, a JTA UserTransaction, or even a Common Object Request Broker Architecture (CORBA) transaction-allowing the application to control transaction boundaries via a consistent API. This helps to keep Hibernate applications portable between different kinds of execution environments and containers.

Transaction transaction = session.beginTransaction();

3. **SessionFactory**:  
The application obtains Session instances from a **SessionFactory**. SessionFactory instances are not lightweight and typically one instance is created for the whole application. If the application accesses multiple databases, it needs one per database.  
SessionFactory can hold an optional (second-level) cache of data that is reusable between transactions at a process, or cluster, level.

SessionFactory sessionFactory =cfg.buildSessionFactory();

4. **Session**:  
The Session is a persistence manager that manages operation like storing and retrieving objects. Instances of Session are inexpensive to create and destroy. They are not thread safe.  
Session holds a mandatory first-level cache of persistent objects that are used when navigating the object graph or looking up objects by identifier.

Session session = sessionFactory.openSession();

1. **Query and Criteria interfaces:**The Query interface allows you to perform queries against the database and control how the query is executed.

Queries are written in HQL or in the native SQL dialect of your database. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

The Criteria interface is very similar; it allows you to create and execute object-oriented criteria queries.  
To help make application code less verbose, Hibernate provides some shortcut methods on the Session interface that let you invoke a query in one line of code. We won't use these shortcuts; instead, we'll always use the Query interface.  
A Query instance is lightweight and can't be used outside the Session that created it.

**Q : Mention some of the advantages of using ORM over JDBC.?**

ORM has the following advantages over JDBC:

* Application development is fast.
* Management of transaction.
* Generates key automatically.
* Details of SQL queries are hidden.

1. have you ever felt irritated by repeating the same lines of code over and over again in your application for fetching data from a database?
2. are you struggling to map objects to your database tables?
3. is it difficult for you to implement oops with your jdbc code?
4. does it takes too much rework while migrating from one database to another?
5. have you ever found it difficult to create associations between tables using jdbc?

there is an orm tool called [hibernate](http://hibernate.org/orm/" \t "https://dzone.com/articles/_blank)that you can use to overcome above 5 points this.

## **what is jdbc?**

jdbc stands for **java database connectivity** . it is a free open source application programming interface for java that enables applications to access databases. it enables developers to create queries and update data to a relational database using the **structured query language**(sql).

**Steps to connect JDBC :**

1. open a database connection
2. send sql queries to database using a jdbc driver
3. jdbc driver connects to the database
4. execute the queries to get the result set
5. send the data to the application via the driver manager
6. when results are returned, it processes the data
7. finally, the connection is closed.

**what is hibernate?**

hibernate is a free, open source object-relational mapping library for java designed to map objects to an RDBMS and to implement the object-oriented programming concepts in a relational database.

1. unlike jdbc, hibernate connects with the database itself and uses hql (hibernate query language) to execute the queries, then maps the results to java objects.
2. the results are mapped to objects based on the properties given in the hibernate configuration xml file.
3. the database connection from an application is created using the session, which also helps in saving and retrieving the persistent object.
4. session factory is an interface that helps to create an instance of a session. there must be only one session factory per database. for example, if you are using mysql and oracle in your application, one session factory for mysql and one session factory for oracle is maintained. there will not be more than one session factory for mysql alone.

## **object mapping**

in **jdbc**, you need to write code to map the object model’s data representation to a relational model and its corresponding schema. **hibernate** itself maps java classes to database tables using xml or by using annotations.

In jdbc, you need to set every property of an object upon fetching the data each and every time. but in hibernate, we need to map the table with the java class.

**In Jdbc :**

user user = new user();

user.setuserid(rs.getstring("userid"));

user.setemail(rs.getstring(“email”));

users.add(user);

**In Hibernate :**

@entity

@table(name = "user")

public class usermodel {

@id

@generatedvalue(strategy = generationtype.identity)

private biginteger id;

@notempty

@column(name = "email", unique = true)

private string email;

@notempty

@column(name = "name") private string name;

## **hql**

hibernate uses **hql** (hibernate query language), which is similar to sql, but hibernate’s hql provides full support for polymorphic queries . hql understands object-oriented concepts like **inheritance**, **polymorphism**, and **association**.

**database independent**

hibernate’s code is database independent because you do not need to change the hql queries (with a few exceptions) when you change databases like mysql, oracle, etc. hence, it is easy to migrate to a new database. it is achieved by using a friendly dialect to communicate with the database. the database can be specified using a dialect in the hibernate configuration xml as follows.

property name="dialect" ;org.hibernate.dialect.mysql;

## **minimize code changes**

hibernate minimizes code changes when we add a new column to a database table.

### jdbc

* you have to add a new field in your pojo class.
* change your jdbc method that performs the “select” to include the new column.
* change your jdbc method that performs the “insert” to add a new value into the new column.
* change your jdbc method that performs the “update” to update an existing value in your new column.

### hibernate

* add the new field into your pojo class.
* modify the hibernate xml mapping file to include the new column.

thus, database table changes can be easily implemented using hibernate with less effort.

## **reduce repeat code**

hibernate reduces the amount of repeating lines of code , which you can often find with jdbc.

in jdbc, for inserting a record, you will be creating a prepared statement and will be setting each column of a table. if the number of columns increases, the statements will also increase. but in hibernate, we just need to save the object with persist(object); .

**lazy loading**

we can achieve ****lazy loading****using hibernate.

consider an example where there is a list of users in the user table. the identity proof documents uploaded by the users are stored in the identity\_proof table. the user has a ‘one to many’ relationships with the identity\_proof. in this case, the user is the parent class and identity\_proof is the child class. if you fetch the parent class, i.e the user, all the documents associated with the user will also be fetched.

 imagine the size of each document. as the number of documents increases, the size of data to be processed also increases, and hence it will slow up the application.

with hibernate, you can specify the fetch type for data as lazy. if you do so when you fetch a user, documents will not be fetched. you can fetch the documents where you want using hibernate’s initialize() method.

// declaring fetch type for one to many association in your pojo

@onetomany(**cascade = cascadetype.all, fetch = fetchtype.laz**y)

private set&lt;proof&gt; prooflist = new hashset&lt;proof&gt;();

// to fetch user with document use initialize() method as follows

gethibernatetemplate().initialize(user.getprooflist());

**Lazy fetching** decides whether to load child objects while loading the Parent Object. You need to do this setting respective hibernate mapping file of the parent class. **Lazy = true**(means not to load child) By default the lazy loading of the child objects is true.

In the Employee.hbm.xml file

<set name="address" inverse="true" cascade="delete" lazy="false">

<key column="a\_id" />

<one-to-many class="beans Address"/>

</set>

In the above configuration. If lazy="false" : - when you load the Employee object that time child object Address is also loaded and set to setAddresss() method. If you call employee.getAdress() then loaded data returns.No fresh database call.

If lazy="true" :- This the default configuration. If you don?t mention then hibernate consider lazy=true. when you load the Employee object that time child object Adress is not loaded. You need extra call to data base to get address objects. If you call employee.getAdress() then that time database query fires and return results. Fresh database call.

## **avoiding try-catch blocks**

jdbc will throw sqlexception, which is a checked exception. so you will be writing “try-catch” blocks in your code. hibernate handles this by converting all jdbc exceptions to unchecked exceptions. therefore, you need not waste your time implementing try-catch blocks.

## **transaction management**

a [transaction](http://www.sitesbay.com/jdbc/jdbc-transaction-management" \t "https://dzone.com/articles/_blank)is a group of operations to be performed under one task. if all the operations in a group succeed, then the task is finished and the transaction is successfully completed. if one of the operations fails, then the whole task fails, and thus the transaction also fails. . in **jdbc**, if a transaction is a success you need to commit. otherwise, you need to perform a rollback with the following commands:

con.commit();

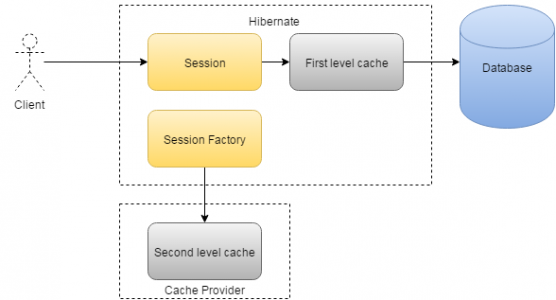
con.rollback();

in **hibernate**, you don’t have to commit and roll back these transactions, as it is implicitly provided.

## **associations**

it is easy to create an association between tables using hibernate. associations like one-to-one, one-to-many, many-to-one, and many-to-many can be achieved easily in hibernate by using annotations, mapping the object entity of the required table.

## **hibernate caching:**



hibernate provides a ****caching mechanism****, which helps reduce the number of hits, as much as possible, that your application makes to the database server.

This will have a considerable effect regarding the performance of your application. there is no such caching mechanism available in **jdbc**.

this is because **hibernate** stores the object in session, which is available until the transaction is active. when a particular query is executed repeatedly, the value stored in the session is used. when a new transaction begins, the object is fetched again from the database and is stored in the session. in fact, two levels of caches are provided by hibernate.

## **jpa annotation support:**

hibernate provides support to jpa annotations like @entity, @table, @column, etc. these annotations make the code portable to other orm frameworks.

## **connection pooling**

Connection pooling is a mechanism in which the database connections, when created are stored in the cache by an external tool so that these connections can be reused from the cache when the application tries to connect to the same database in future. connection pooling helps increase performance. we can achieve this connection pooling in hibernate. hibernate supports the following pools:

[c3p0](http://sourceforge.net/projects/c3p0" \t "https://dzone.com/articles/_blank)

[apache dbcp](http://jakarta.apache.org/commons/dbcp/" \t "https://dzone.com/articles/_blank)

[proxool](http://proxool.sourceforge.net/" \t "https://dzone.com/articles/_blank)

you need to set the properties in hibernate.cfg.xml as required to enable connection pooling.

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<property name="hibernate.c3p0.max\_statements">50</property>

<property name="hibernate.c3p0.idle\_test\_period">3000</property>

**Q : Disadvantages of Hibernate**

1) Steep learning curve.

2) Use of Hibernate is an overhead for the applications which are :

• simple and use one database that never change

• need to put data to database tables, no further SQL queries

• there are no objects which are mapped to two different tables

Hibernate increases extra layers and complexity. So for these types of applications JDBC is the best choice.

3) Support for Hibernate on Internet is not sufficient.

4) Anybody wanting to maintain application using Hibernate will need to know Hibernate.

5) For complex data, mapping from Object-to-tables and vise versa reduces performance and increases time of conversion.

6) Hibernate does not allow some type of queries which are supported by JDBC. For example It does not allow to insert multiple objects (persistent data) to same table using single query. Developer has to write separate query to insert each object.

### **Q : List some of the databases supported by Hibernate**.

Some of the databases supported by Hibernate are:

* DB2
* MySQL
* Oracle
* Sybase SQL Server
* Informix Dynamic Server
* HSQL
* PostgreSQL
* FrontBase

**Q: What’s the usage of Configuration Interface in hibernate?**

Configuration interface of hibernate framework is used to configure hibernate. It’s also used to [bootstrap](https://career.guru99.com/top-25-bootstrap-interview-questions/" \o "bootstrap) hibernate. Mapping documents of hibernate are located using this interface.

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

Mappings between class properties and table columns are specified in XML file & Annotations.

**Q :Should all the mapping files of hibernate have .hbm.xml extension to work properly?**

No, having .hbm.xml extension is a convention and not a requirement for hibernate mapping file names. We can have any extension for these mapping files.

**Q : How do we create session factory in hibernate?**

To create a session factory in hibernate, an object of configuration is created first which refers to the path of configuration file and then for that configuration, session factory is created as given in the example below:

Configuration config = new Configuration();

config.addResource(&amp;amp;quot;myinstance/configuration.hbm.xml&amp;amp;quot;);

config.setProperties( System.getProperties() );

SessionFactory sessions = config.buildSessionFactory();

**Q: What are POJOs and what’s their significance?**

POJOs( Plain Old Java Objects) are java beans with proper getter and setter methods for each and every properties.

Use of POJOs instead of simple java classes results in an efficient and well constructed code.

**Q: What’s HQL?**

HQL is the query language used in Hibernate which is an extension of SQL. HQL is very efficient, simple and flexible query language to do various type of operations on relational database without writing complex database queries.

The HQL query is created with the help of the following syntax:

Session.createQuery

A criterion is added to a SQL query by using the Session.createCriteria.

### **Q: Define persistent classes.**

Classes whose objects are stored in a database table are called as persistent classes.

**Q : How can we invoke stored procedures in hibernate?**

In hibernate we can execute stored procedures using code as below:



**Q: What is transient instance state in Hibernate?**If an instance is not associated with any persistent context and also, it has never been associated with any persistent context, then it’s said to be in transient state.

**Q: What’s general hibernate flow using RDBMS?**General hibernate flow involving RDBMS is as follows:  
a. Load configuration file and create object of configuration class.  
b. Using configuration object, create sessionFactory object.  
c. From sessionFactory, get one session.  
d. Create HQL query.  
e. Execute HQL query and get the results. Results will be in the form of a list.

**Q: How can we map the classes as immutable?**If we don’t want an application to update or delete objects of a class in hibernate, we can make the class as immutable by setting mutable=false

**Q: What the three inheritance models are of hibernate?**Hibernate has following three inheritance models:  
a. Tables Per Concrete Class  
b. Table per class hierarchy  
c. Table per sub-class

**Q: Does hibernate support polymorphism?**Yes, hibernate fully supports polymorphism. Polymorphism queries and polymorphism associations are supported in all mapping strategies of hibernate.

**Q: What’s the use of session.lock() in hibernate?**session.lock() method of session class is used to reattach an object which has been detached earlier. This method of reattaching doesn’t check for any data synchronization in database while reattaching the object and hence may lead to lack of synchronization in data.

**Q: In how many ways, objects can be fetched from database in hibernate?**Hibernate provides following four ways to fetch objects from database:  
a. Using HQL  
b. Using identifier  
c. Using Criteria API  
d. Using Standard SQL

**Q: What the two methods are of hibernate configuration?**We can use any of the following two methods of hibernate configuration:  
a. XML based configuration ( using hibernate.cfg.xml file)  
b. Programmatic configuration ( Using code logic)

**Q : What is the default cache service of hibernate?**Hibernate supports multiple cache services like EHCache, OSCache, SWARMCache and TreeCache and default cache service of hibernate is EHCache.

**Q: What are the two mapping associations used in hibernate?**In hibernate; we have following two types of mapping associations between entities:  
a. One-to-One Association  
b. Many-to-Many Association

**Q: How primary key is created by using hibernate?**Database primary key is specified in the configuration file hbm.xml. Generator can also be used to specify how primary key is being created in the database.

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

Hibernate second level cache can be disabled using any of the following ways:  
a. By setting use\_second\_level\_cache as false.  
b. By using CACHEMODE.IGNORE  
c. Using cache provider as org.hibernate.cache.NoCacheProvider

### **Q: Is SessionFactory a thread-safe object?**

Yes, SessionFactory is a thread-safe object, many threads cannot access it simultaneously.

### **Q: What is Session?**

It maintains a connection between the hibernate application and database.

It provides methods to store, update, delete or fetch data from the database such as persist(), update(), delete(), load(), get() etc.

It is a factory of Query, Criteria and Transaction i.e. it provides factory methods to return these instances.

### **Q: Is Session a thread-safe object?**

No, Session is not a thread-safe object, many threads can access it simultaneously. In other words, you can share it between threads.

### **Q: What are the states of the object in hibernate?**

There are 3 states of the object (instance) in hibernate.

1. **Transient**: The object is in a transient state if it is just created but has no primary key (identifier) and not associated with a session.
2. **Persistent**: The object is in a persistent state if a session is open, and you just saved the instance in the database or retrieved the instance from the database.
3. **Detached**: The object is in a detached state if a session is closed. After detached state, the object comes to persistent state if you call lock() or update() method.

### **Q: What are the inheritance mapping strategies?**

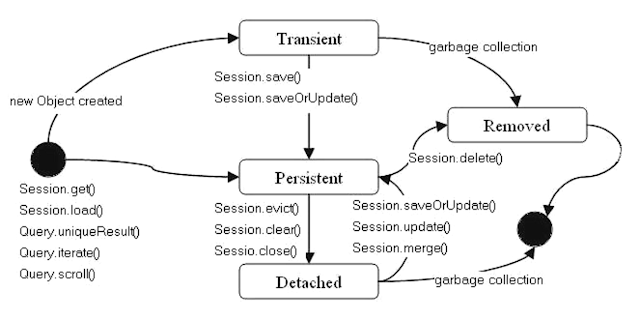
There are 3 ways of inheritance mapping in hibernate.

1. Table per hierarchy
2. Table per concrete class
3. Table per subclass

### **Q : What is the difference between session.save() and session.persist() method?**

the **save**() method saves records into the database by INSERT SQL query, Generates a new identifier, and returns the [Serializable](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) identifier back.

On the other hand  **saveOrUpdate**() method either INSERT or UPDATE based upon the existence of an object in the database. If a persistence object already exists in the database then UPDATE SQL will [execute](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html), and if there is no corresponding object in the database, then INSERT will run.  
  
The first difference between save and **persist** is there return type. Similar to save method, persist also INSERT records into the database, but return type of persist is void while return type of save is [Serializable](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html) Object.

Another difference between persisting and save is that both methods make a [transient](http://javarevisited.blogspot.sg/2012/03/difference-between-transient-and.html) instance persistent. However, persist() method doesn't guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time.  


First difference between save and persist is their return type. The return type of persist method is void while return type of save  
method is Serializable object.

The persist() method doesn’t guarantee that the identifier value will be assigned to the persistent state immediately, the assignment might happen at flush time.

The persist() method will not execute an insert query if it is called outside of transaction boundaries. While, the save() method returns an identifier so that an insert query is executed immediately to get the identifier, no matter if it are inside or outside of a transaction.

The persist method is called outside of transaction boundaries, it is useful in long-running conversations with an extended Session context. On the other hand save method is not good in a long-running conversation with an extended Session context.

Fifth difference between save and persist method in Hibernate: persist is supported by JPA, while save is only supported by Hibernate.

### **Q :What is the difference between get and load method?**

The differences between get() and load() methods are given below.

**get():**

Returns null if an object is not found.

get() method always hit the database.

It returns the real object, not the proxy.

It should be used if you are not sure about the existence of instance.

**load()**

Throws ObjectNotFoundException if an object is not found.

load() method doesn't hit the database.

It returns proxy object.

It should be used if you are sure that instance exists.

### **Q :What is the difference between update and merge method?**

**The update() method**

Update means to edit something.

update() should be used if the session doesn't contain an already persistent state with the same id. It means an update should be used inside the session only. After closing the session, it will throw the error.

**merge() method**

Merge means to combine something.

merge() should be used if you don't know the state of the session, means you want to make the modification at any time.

### **Q: How many types of association mapping are possible in hibernate?**

There can be 4 types of association mapping in hibernate.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

### **Q: What is lazy loading in hibernate?**

Lazy loading in hibernate improves the performance. It loads the child objects on demand.

Since Hibernate 3, lazy loading is enabled by default, and you don't need to do lazy="true". It means not to load the child objects when the parent is loaded.

### **Q: What is HQL (Hibernate Query Language)?**

Hibernate Query Language is known as an object-oriented query language. It is like a structured query language (SQL).

The main advantage of HQL over SQL is:

1. You don't need to learn SQL
2. Database independent
3. Simple to write a query

### **Q: What is the difference between first level cache and second level cache?**

**First Level Cache**

First Level Cache is associated with Session.

It is enabled by default.

**Second Level Cache**

Second Level Cache is associated with SessionFactory.

It is not enabled by default.

Note :

Below code will work . updating the existing records.

We can use save , update ,merge ,saveorupdate method

Configuration configuration = new Configuration();

configuration.configure("hibernate.cfg.xml");

SessionFactory factory = configuration.buildSessionFactory();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Student student = session.load(Student.class, 111);

student.setName("Johnson");

session.update(student);

tx.commit();

session.close();

Session sess = factory.openSession();

Transaction tx;

try {

tx = sess.beginTransaction();

//do some work

...

tx.commit();

}

catch (Exception e) {

if (tx!=null) tx.rollback();

throw e;

}

finally {

sess.close();

}

**What does flush()?**

**flush():** Forces the session to flush. It is used to synchronize session data with database.

public void flush() throws HibernateException

When you call session.flush(), the statements are executed in database but it will not committed.

If you dont call session.flush() and if you call session.commit() , internally commit() method executes the statement and commits.

So commit()= flush+commit.

So seesion.flush() just executes the statements in database (but not commits) and statements are NOT IN MEMORY anymore. It just forces the session to flush.

  public static void main(String[] args) {

        Session session = getSession();

        session.getTransaction().begin();

        try{

            Employee emp = (Employee)session.get(Employee.class, 1);

            emp.setEmpName("NEW SREE");

            emp.setEmpDept("NEW DEVELOPMENT");

            session.flush();

            //Put break point and observe console

            session.getTransaction().commit();

            session.close();

        }catch(HibernateException e){

            e.printStackTrace();

        }

    }

* After **session.flush(),**hibernate compares employee object data and corresponding record in database. If there is a difference it will execute update query to update object data in the database, but it will not commit.
* After **transaction.commit(),**Here also , hibernate compares employee object data and corresponding record in database. If there is a difference it will execute update query to update object data in the database, and commits transaction.
* **session.flush()**must be called before committing the transaction and closing the session.

**Batch Processing with flush() method**

Consider a requirement when you want to insert a large number of records in database using Hibernate. The code looks like as below.

Session session = SessionFactory.openSession();

Transaction tx = session.beginTransaction();

for ( int i=0; i<100000; i++ ) {

    Employee emp = new Employee(.....);

    session.save(emp);

}

tx.commit();

session.close();

This code may throw **OutOfMemoryError**somewhere around 50,000th row. Because Hibernate caches all the newly inserted Employee objects in  the session level cache. We can solve this problem using **hibernate batch processing.**

We need to set **hibernate.jdbc.batch\_size**in **hibernate.cfg.xml**as below

**<property name="hibernate.jdbc.batch\_size">40</property>**

So, Hibernate executes every 40 rows as a batch.

Session session = SessionFactory.openSession();

Transaction tx = session.beginTransaction();

for ( int i=0; i<100000; i++ ) {

    Employee emp = new Employee(.....);

    session.save(emp);

    if( i % 40 == 0 ) { // Same as the JDBC batch size

        //flush a batch of inserts and release memory:

        session.flush();

        session.clear();

    }

}

tx.commit();

session.close();

Batch processing helps to avoid **OutOfMemoryError**

 public void saveOrUpdate(Object object) throws HibernateException

* If the record is not present in the database, it will call save() method and inserts the record in the database.
* If the record is present in the database, it will call update() method and updates the record in the database.

So **saveOrUpdate()**method calls **save()** method if there is no record in database, and it calls **update()**method if there is a record in database.