**Private static** SessionFactory *factory*;

*factory* = **new** Configuration().configure().buildSessionFactory();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Employee employee = **new** Employee(fname, lname, salary);

employeeID = (Integer) session.save(employee);

tx.commit();

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

session.close();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Employee employee = (Employee)session.get(Employee.**class**, EmployeeID);

employee.setSalary( salary );

session.update(employee);

tx.commit();

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

session.close();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Employee employee = (Employee)session.get(Employee.**class**, EmployeeID);

session.delete(employee);

tx.commit();

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

session.close();

**Query-Section**

**1Minute session:**

**Query**query = session.createQuery(“HQL”);

* **List** results = query.list(); ->if HQL is **select.**
* **int**result = query.executeUpdate();->if HQL is **insert, update, delete.**

**Named parameter**- Insert, update and Delete uses named parameter.

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();

HQL works with **persistent objects** and their properties not on database table like SQL.

**Criteria Queries**

Criteriacr = session.createCriteria(Employee.class);

cr.add(Restrictions.eq("salary", 2000));

List results = cr.list();

createCriteria() method used to create a Criteria object that returns **instances** of the persistence object's where you can apply filtration rules and logical conditions.

**Native SQL**

SQLQuery query = session.createSQLQuery(sql);

* query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP); ->Resultset
* query.addEntity(Employee.class); ->to get entity objects

List results = query.list();

You can use native SQL with hibernet to utilize database-specific features.

**Hibernate Batch Processing**

Set hibernate.jdbc.batch\_size as batch size(say 50)in .cfg file.

<propertyname=*"hibernate.jdbc.batch\_size"*> 50 </property>

To implement this in your code we would need to do little modification.

***if****(i % 50 == 0 ) { // Same as batch size*

*session.flush(); //flush a batch of inserts*

*session.clear(); //and release memory*

*}*

Consider a situation when you need to upload a large number of records(say 100000) into your database using Hibernate.

**Caching**

Hibernate allows a single cache provider for the whole application.

Specify a cache provider in hibernate.cfg.xml configuration file.

<propertyname=*"hibernate.cache.provider\_class"*>

org.hibernate.cache.EhCacheProvider

</property>

Specify the properties of the cache regions, for EHCachethis is ehcache.xml, which should be in the CLASSPATH of the application.

For each persistent class, configure concurrency strategy(responsible for storing/retrieving datain/from the cache) in respective mapping files(.hbm.xml) :

<cacheusage=*"read-write"*/>

Now the second-level caching is enabled for the particular class.

The first-level cache is the Session cache which is default cache.

**Collections Mappings**

**Hibernate Set Mappings**

Each employee can have one or more certificate(unique).

We will have Employee\_table, Employee class and corresponding hbm file.

Similarly we will have CERT\_table, Cert class and corresponding hbm file.

In RDMS CERT\_table will have emp\_id(PK) field as foreign key. So both table can be associated.

At class level this foreign key will not present at Cert class, but Cert class will be the property of Employee class.

And this property will be the part of Employee section of hbm file.

<classname=*"Employee"* table=*"EMPLOYEE"*>

<metaattribute=*"class-description"*>

This class contains the employee detail.

</meta>

<id name=*"id"* type=*"int"* column=*"id"*>

<generatorclass=*"native"*/>

</id>

<set name=*"certificates"* cascade=*"all"*>

<key column=*"employee\_id"*/>

<one-to-many class=*"Certificate"*/>

</set>

<property name=*"firstName"* column=*"first\_name"* type=*"string"*/>

<property name=*"lastName"* column=*"last\_name"* type=*"string"*/>

<property name=*"salary"* column=*"salary"* type=*"int"*/>

</class>

Note this property is as set

The <key> element is the column in the CERTIFICATE\_table that holds the foreign key to the parent object(Employee).

<one-to-many> element - indicates that one Employee object relates to many Certificate objects and, as such, the Certificate object must have a Employee parent associated with it.