# HIBERNATE

Contents

[HIBERNATE 1](#_Toc465762704)

[ORM? 1](#_Toc465762705)

[Configuration Object: 1](#_Toc465762706)

[SessionFactory Object: 1](#_Toc465762707)

[Session Object: 1](#_Toc465762708)

[Transaction Object: 1](#_Toc465762709)

[Query Object: 1](#_Toc465762710)

[Criteria Object: 1](#_Toc465762711)

[You can build a Maven project and download hibernate dependencies from the download website. 1](#_Toc465762712)

[Installing MySQL Community to create Database 1](#_Toc465762713)

[Sample queries 1](#_Toc465762714)

[I use a sample database from tutorial , please copy the scripts to create the “simplehr” database schema. ( click to drop down) 1](#_Toc465762715)

[Create hibernate.cfg.xml 1](#_Toc465762716)

[Create class Entity for queries ( just like for database ) 1](#_Toc465762717)

[Create SessionFactory package and run demo 1](#_Toc465762718)

[Data Utils 1](#_Toc465762719)

[Transient -> Persistent 1](#_Toc465762720)

[Persistent -> Detached and Detached -> Persistent 1](#_Toc465762721)

## ORM?

ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc. An ORM system has following advantages over plain JDBC

|  |  |
| --- | --- |
| **S.N.** | **Advantages** |
| 1 | Lets business code access objects rather than DB tables. |
| 2 | Hides details of SQL queries from OO logic. |
| 3 | Based on JDBC 'under the hood' |
| 4 | No need to deal with the database implementation. |
| 5 | Entities based on business concepts rather than database structure. |
| 6 | Transaction management and automatic key generation. |
| 7 | Fast development of application. |

An ORM solution consists of the following four entities:

|  |  |
| --- | --- |
| **S.N.** | **Solutions** |
| 1 | An API to perform basic CRUD operations on objects of persistent classes. |
| 2 | A language or API to specify queries that refer to classes and properties of classes. |
| 3 | A configurable facility for specifying mapping metadata. |
| 4 | A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions. |

Hibernate framework

Hibernate is an Object-Relational Mapping(ORM) solution for JAVA. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.



Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.

Provides simple APIs for storing and retrieving Java objects directly to and from the database.

If there is change in Database or in any table then the only need to change XML file properties.

Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.

Hibernate does not require an application server to operate.

Manipulates Complex associations of objects of your database.

Minimize database access with smart fetching strategies.

Provides Simple querying of data.

Hibernate supports almost all the major RDBMS. Following is list of few of the database engines supported by Hibernate.

HSQL Database Engine

DB2/NT

MySQL

PostgreSQL

FrontBase

Oracle

Microsoft SQL Server Database

Sybase SQL Server

Informix Dynamic Server

Architecture of Hibernate





### Configuration Object:

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys components:

* **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
* **Class Mapping Setup**

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

### SessionFactory Object:

Configuration object is used to create a SessionFactory object which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

### Session Object:

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

### Transaction Object:

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

This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

### Query Object:

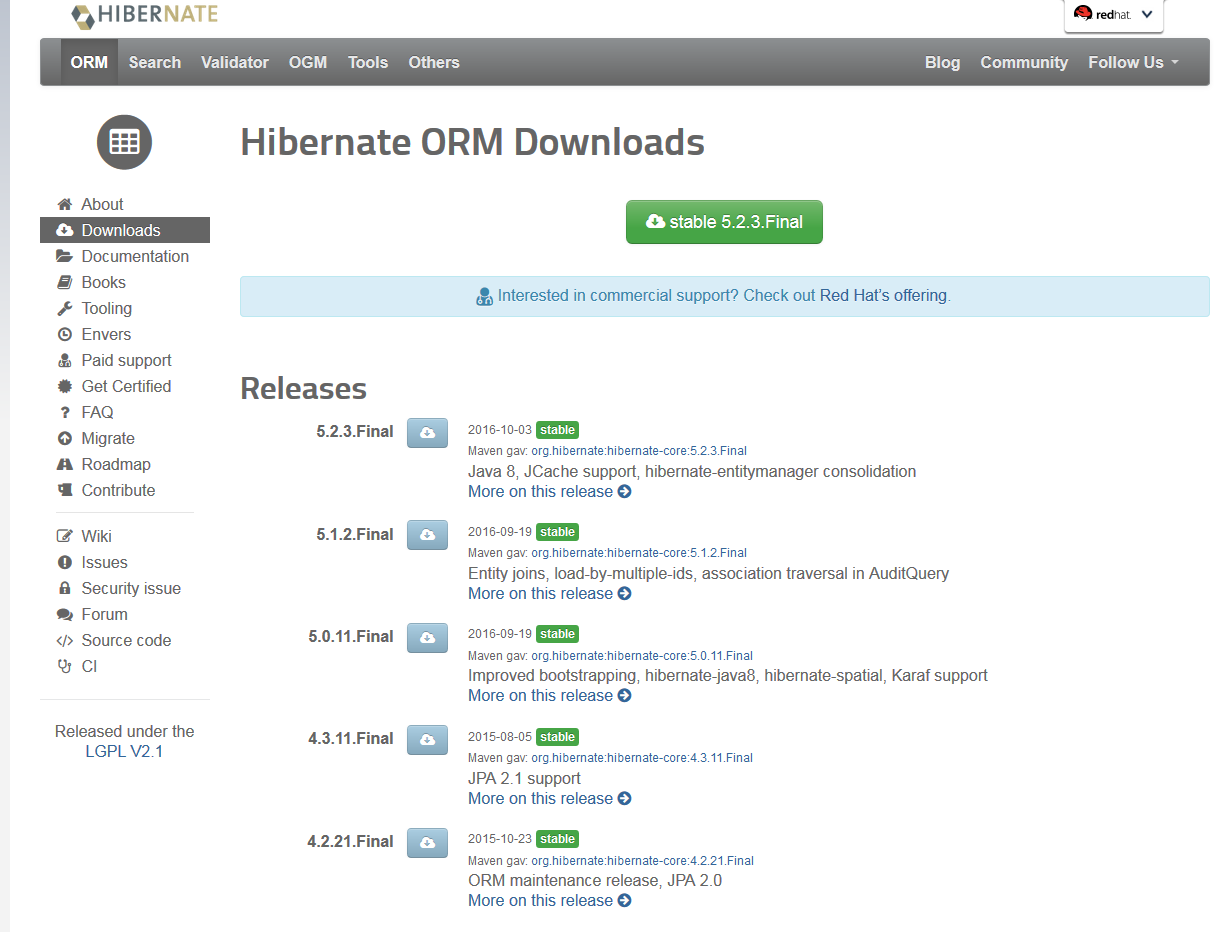
Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

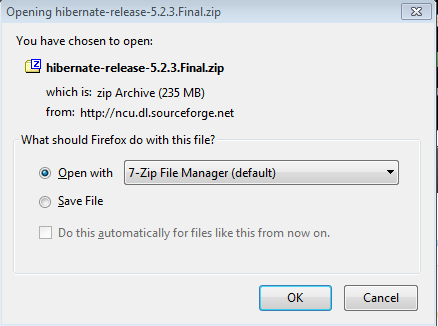
### Criteria Object:

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

**Installing Hibernate**

**Download page : http://hibernate.org/orm/downloads/**

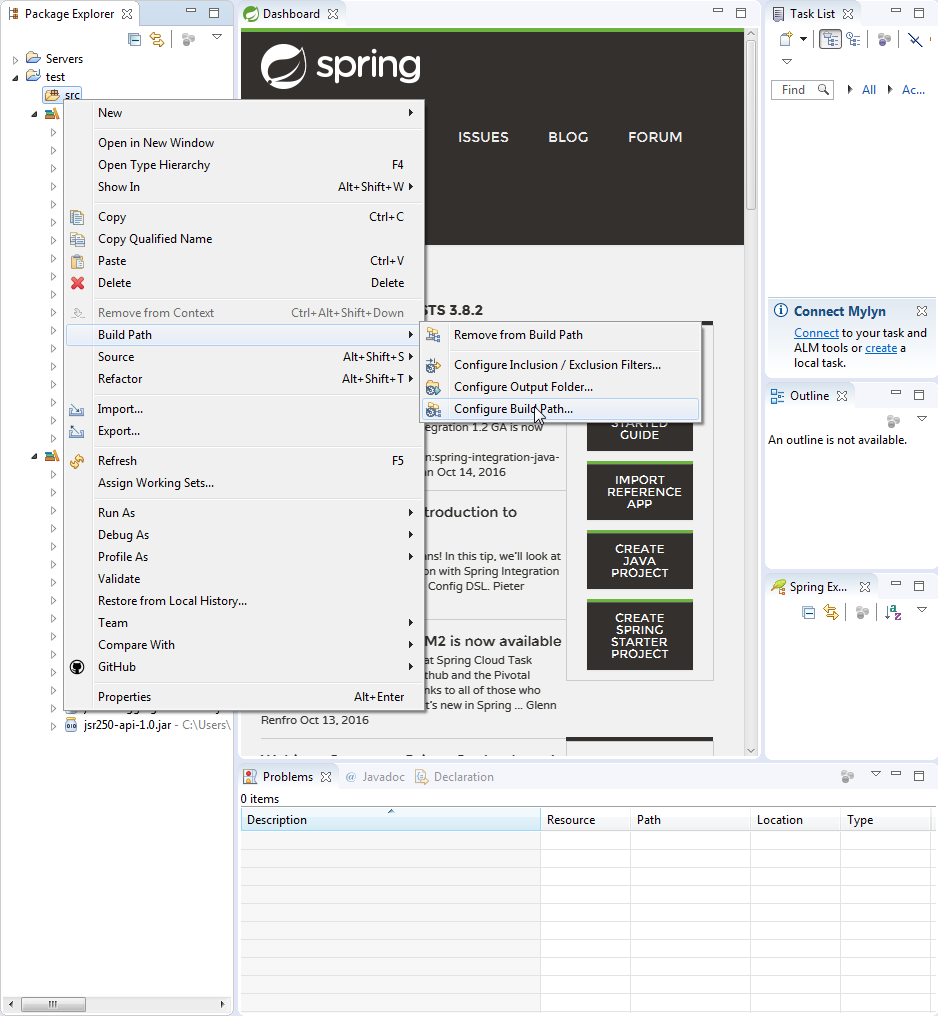




After unzipped to your destination folder.

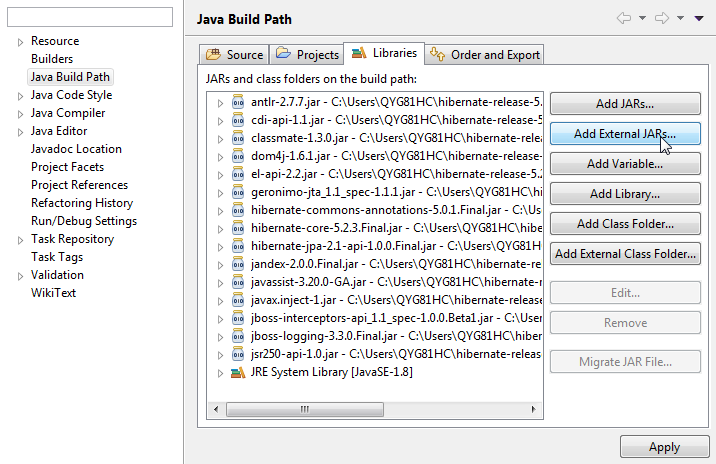
Make sure you are setting your CLASSPATH variable properly otherwise you will face problem while compiling your application.

Create Project from IDE -> Config Build Path when right-clicked to the project.

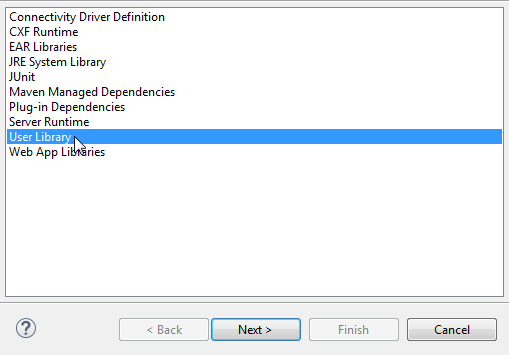


Add external JARS . The Hibernate library are where : <hibernate directory>/lib/required

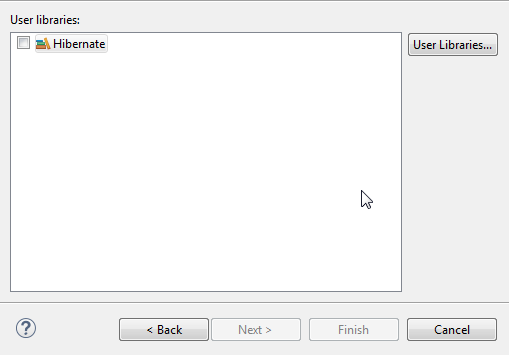
The required folder is a compulsory folder for Hibernate to run . Other folders are Hibernate optional features like envers , tools , etc.

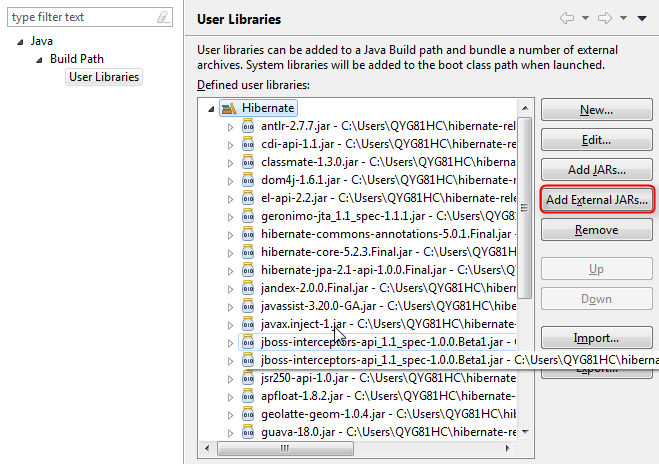


You can also define user library to package all the required jars into one Library Named “Hibernate”.



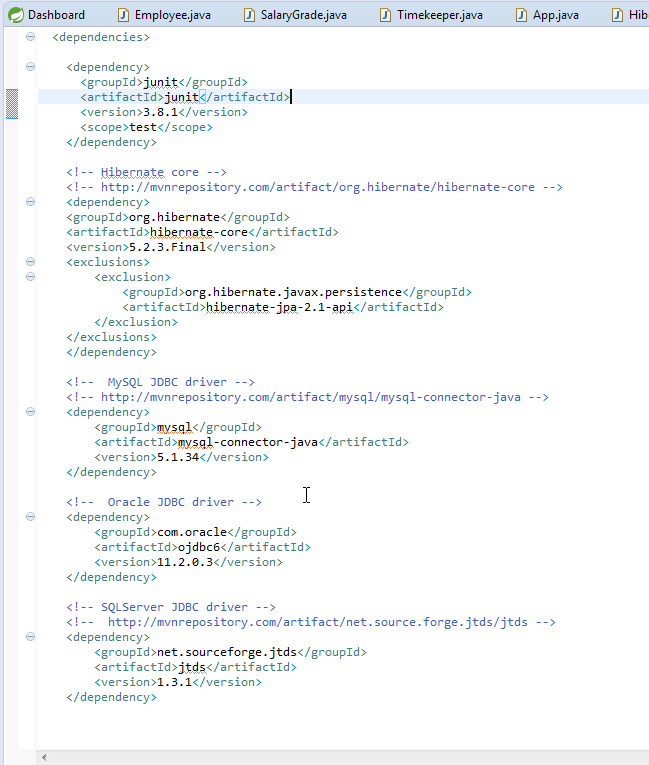
Click User Libraries to define new one.



Add External JARS to create customized “Hibernate” library.

**ELSE**

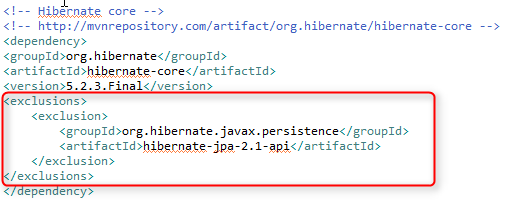
## You can build a Maven project and download hibernate dependencies from the download website.



**Please include JPA 2.1 and mysql connector java**



**Please eliminate the JPA from the Hibernate-core dependency**



**The external library supports more function than Hibernate-core version. (even their versions are both 2.1 ) – you can add JPA 2.1 from entitymanger dependency ( from hibernate website , this the same with the external library JPA 2.1 )**

## Installing MySQL Community to create Database

[**http://o7planning.org/vi/10221/huong-dan-cai-dat-va-cau-hinh-mysql-community**](http://o7planning.org/vi/10221/huong-dan-cai-dat-va-cau-hinh-mysql-community)

## Sample queries

[**http://o7planning.org/vi/10237/huong-dan-hoc-sql-cho-nguoi-moi-bat-dau-voi-mysql**](http://o7planning.org/vi/10237/huong-dan-hoc-sql-cho-nguoi-moi-bat-dau-voi-mysql)

### I use a sample database from tutorial , please copy the scripts to create the “simplehr” database schema. ( click to drop down)

**create** **table** DEPARTMENT **(**

DEPT\_ID **integer** **not** **null,**

DEPT\_NAME **varchar(**255**)** **not** **null,**

DEPT\_NO **varchar(**20**)** **not** **null,**

LOCATION **varchar(**255**),**

**primary** **key** **(**DEPT\_ID**),**

**unique** **(**DEPT\_NO**)**

**);**

**create** **table** EMPLOYEE **(**

EMP\_ID **bigint** **not** **null,**

EMP\_NAME **varchar(**50**)** **not** **null,**

EMP\_NO **varchar(**20**)** **not** **null,**

HIRE\_DATE **date** **not** **null,**

IMAGE longblob**,**

JOB **varchar(**30**)** **not** **null,**

SALARY **float** **not** **null,**

DEPT\_ID **integer** **not** **null,**

MNG\_ID **bigint,**

**primary** **key** **(**EMP\_ID**),**

**unique** **(**EMP\_NO**)**

**);**

**create** **table** SALARY\_GRADE **(**

GRADE **integer** **not** **null,**

HIGH\_SALARY **float** **not** **null,**

LOW\_SALARY **float** **not** **null,**

**primary** **key** **(**GRADE**)**

**);**

**create** **table** TIMEKEEPER **(**

Timekeeper\_Id **varchar(**36**)** **not** **null,**

Date\_Time **datetime** **not** **null,**

In\_Out **char(**1**)** **not** **null,**

EMP\_ID **bigint** **not** **null,**

**primary** **key** **(**Timekeeper\_Id**)**

**);**

**alter** **table** EMPLOYEE

**add** **index** FK75C8D6AE269A3C9 **(**DEPT\_ID**),**

**add** **constraint** FK75C8D6AE269A3C9

**foreign** **key** **(**DEPT\_ID**)**

**references** DEPARTMENT **(**DEPT\_ID**);**

**alter** **table** EMPLOYEE

**add** **index** FK75C8D6AE6106A42 **(**EMP\_ID**),**

**add** **constraint** FK75C8D6AE6106A42

**foreign** **key** **(**EMP\_ID**)**

**references** EMPLOYEE **(**EMP\_ID**);**

**alter** **table** EMPLOYEE

**add** **index** FK75C8D6AE13C12F64 **(**MNG\_ID**),**

**add** **constraint** FK75C8D6AE13C12F64

**foreign** **key** **(**MNG\_ID**)**

**references** EMPLOYEE **(**EMP\_ID**);**

**alter** **table** TIMEKEEPER

**add** **index** FK744D9BFF6106A42 **(**EMP\_ID**),**

**add** **constraint** FK744D9BFF6106A42

**foreign** **key** **(**EMP\_ID**)**

**references** EMPLOYEE **(**EMP\_ID**);**

**insert** **into** Department **(**DEPT\_ID**,** DEPT\_NAME**,** DEPT\_NO**,** LOCATION**)**

**values** **(**10**,** 'ACCOUNTING'**,** 'D10'**,** 'NEW YORK'**);**

**insert** **into** Department **(**DEPT\_ID**,** DEPT\_NAME**,** DEPT\_NO**,** LOCATION**)**

**values** **(**20**,** 'RESEARCH'**,** 'D20'**,** 'DALLAS'**);**

**insert** **into** Department **(**DEPT\_ID**,** DEPT\_NAME**,** DEPT\_NO**,** LOCATION**)**

**values** **(**30**,** 'SALES'**,** 'D30'**,** 'CHICAGO'**);**

**insert** **into** Department **(**DEPT\_ID**,** DEPT\_NAME**,** DEPT\_NO**,** LOCATION**)**

**values** **(**40**,** 'OPERATIONS'**,** 'D40'**,** 'BOSTON'**);**

-------------------------------------------------------------------------------------------------

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7839**,** 'KING'**,** 'E7839'**,** Str\_To\_Date**(**'17-11-1981'**,** '%d-%m-%Y'**),** 'PRESIDENT'**,** 5000**,** 10**,** **null);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7566**,** 'JONES'**,** 'E7566'**,** Str\_To\_Date**(**'02-04-1981'**,** '%d-%m-%Y'**),** 'MANAGER'**,** 2975**,** 20**,** 7839**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7902**,** 'FORD'**,** 'E7902'**,** Str\_To\_Date**(**'03-12-1981'**,** '%d-%m-%Y'**),** 'ANALYST'**,** 3000**,** 20**,** 7566**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7369**,** 'SMITH'**,** 'E7369'**,** Str\_To\_Date**(**'17-12-1980'**,** '%d-%m-%Y'**),** 'CLERK'**,** 800**,** 20**,** 7902**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7698**,** 'BLAKE'**,** 'E7698'**,** Str\_To\_Date**(**'01-05-1981'**,** '%d-%m-%Y'**),** 'MANAGER'**,** 2850**,** 30**,** 7839**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7499**,** 'ALLEN'**,** 'E7499'**,** Str\_To\_Date**(**'20-02-1981'**,** '%d-%m-%Y'**),** 'SALESMAN'**,** 1600**,** 30**,** 7698**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7521**,** 'WARD'**,** 'E7521'**,** Str\_To\_Date**(**'22-02-1981'**,** '%d-%m-%Y'**),** 'SALESMAN'**,** 1250**,** 30**,** 7698**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7654**,** 'MARTIN'**,** 'E7654'**,** Str\_To\_Date**(**'28-09-1981'**,** '%d-%m-%Y'**),** 'SALESMAN'**,** 1250**,** 30**,** 7698**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7782**,** 'CLARK'**,** 'E7782'**,** Str\_To\_Date**(**'09-06-1981'**,** '%d-%m-%Y'**),** 'MANAGER'**,** 2450**,** 30**,** 7839**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7788**,** 'SCOTT'**,** 'E7788'**,** Str\_To\_Date**(**'19-04-1987'**,** '%d-%m-%Y'**),** 'ANALYST'**,** 3000**,** 20**,** 7566**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7844**,** 'TURNER'**,** 'E7844'**,** Str\_To\_Date**(**'08-09-1981'**,** '%d-%m-%Y'**),** 'SALESMAN'**,** 1500**,** 30**,** 7698**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7876**,** 'ADAMS'**,** 'E7876'**,** Str\_To\_Date**(**'23-05-1987'**,** '%d-%m-%Y'**),** 'CLERK'**,** 1100**,** 20**,** 7698**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7900**,** 'ADAMS'**,** 'E7900'**,** Str\_To\_Date**(**'03-12-1981'**,** '%d-%m-%Y'**),** 'CLERK'**,** 950**,** 30**,** 7698**);**

**insert** **into** Employee **(**EMP\_ID**,** EMP\_NAME**,** EMP\_NO**,** HIRE\_DATE**,** JOB**,** SALARY**,** DEPT\_ID**,** MNG\_ID**)**

**values** **(**7934**,** 'MILLER'**,** 'E7934'**,** Str\_To\_Date**(**'23-01-1982'**,** '%d-%m-%Y'**),** 'CLERK'**,** 1300**,** 10**,** 7698**);**

-------------------------------------------------------------------------------------------------

**insert** **into** Salary\_Grade **(**GRADE**,** HIGH\_SALARY**,** LOW\_SALARY**)**

**values** **(**1**,** 9999**,** 3001**);**

### Create hibernate.cfg.xml

<?xml version=*'1.0'* encoding=*'utf-8'*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/simplehr?serverTimezone=UTC</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>chorangho</property>

<!-- JDBC connection pool (use the built-in) -->

<property name=*"connection.pool\_size"*>1</property>

<!-- SQL dialect -->

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

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

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

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

<property name=*"cache.provider\_class"*>org.hibernate.cache.internal.NoCacheProvider</property>

<!-- Echo all executed SQL to stdout -->

<property name=*"show\_sql"*>true</property>

<mapping class=*"test.entities.Department"* />

<mapping class=*"test.entities.Employee"* />

<mapping class=*"test.entities.SalaryGrade"* />

<mapping class=*"test.entities.Timekeeper"* />

</session-factory>

</hibernate-configuration>

Dialect is a class that call Hibernate to converse Database data types into Java and vice versa. It defines Hibernate SQL functions the same as Database functions.

### Create class Entity for queries ( just like for database )

#### Department.java(copy all – there’s more.. )

**package** test.entities;

**import** java.util.HashSet;

**import** java.util.Set;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.Id;

**import** javax.persistence.OneToMany;

**import** javax.persistence.Table;

**import** javax.persistence.UniqueConstraint;

@Entity

@Table(name = "DEPARTMENT", uniqueConstraints = { @UniqueConstraint(columnNames = {"DEPT\_NO"}) })

**public** **class** Department {

**private** Integer deptId;

**private** String deptNo;

**private** String deptName;

**private** String location;

**private** Set<Employee> employees = **new** HashSet<Employee>(0);

**public** Department() {

}

**public** Department(Integer deptId, String deptName, String location){

**this**.deptId = deptId;

**this**.deptNo = "D" + **this**.deptId;

**this**.deptName = deptName;

**this**.location = location;

}

@Id

@Column(name = "DEPT\_ID")

**public** Integer getDeptId() {

**return** deptId;

}

**public** **void** setDeptId(Integer deptId) {

**this**.deptId = deptId;

}

@Column(name = "DEPT\_NO", length = 20, nullable = **false**)

**public** String getDeptNo() {

**return** deptNo;

}

**public** **void** setDeptNo(String deptNo) {

**this**.deptNo = deptNo;

}

@Column(name = "DEPT\_NAME", nullable = **false**)

**public** String getDeptName() {

**return** deptName;

}

**public** **void** setDeptName(String deptName) {

**this**.deptName = deptName;

}

@Column(name = "LOCATION")

**public** String getLocation() {

**return** location;

}

**public** **void** setLocation(String location) {

**this**.location = location;

}

@OneToMany(fetch = FetchType.***LAZY***, mappedBy = "department")

**public** Set<Employee> getEmployees() {

**return** employees;

}

**public** **void** setEmployees(Set<Employee> employees) {

**this**.employees = employees;

}

}

#### Employee.java

**package** test.entities;

**import** java.util.Date;

**import** java.util.HashSet;

**import** java.util.Set;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.Lob;

**import** javax.persistence.ManyToOne;

**import** javax.persistence.OneToMany;

**import** javax.persistence.Table;

**import** javax.persistence.Temporal;

**import** javax.persistence.TemporalType;

**import** javax.persistence.UniqueConstraint;

@Entity

@Table(name = "EMPLOYEE",

uniqueConstraints = { @UniqueConstraint(columnNames = { "EMP\_NO" }) })

**public** **class** Employee {

**private** Long empId;

**private** String empNo;

**private** String empName;

**private** String job;

**private** Employee manager;

**private** Date hideDate;

**private** Float salary;

**private** **byte**[] image;

**private** Department department;

**private** Set<Employee> employees = **new** HashSet<Employee>(0);

**public** Employee() {

}

**public** Employee(Long empId, String empName, String job, Employee manager,

Date hideDate, Float salary, Float comm, Department department) {

**this**.empId = empId;

**this**.empNo = "E" + **this**.empId;

**this**.empName = empName;

**this**.job = job;

**this**.manager = manager;

**this**.hideDate = hideDate;

**this**.salary = salary;

**this**.department = department;

}

@Id

@Column(name = "EMP\_ID")

**public** Long getEmpId() {

**return** empId;

}

**public** **void** setEmpId(Long empId) {

**this**.empId = empId;

}

@Column(name = "EMP\_NO", length = 20, nullable = **false**)

**public** String getEmpNo() {

**return** empNo;

}

**public** **void** setEmpNo(String empNo) {

**this**.empNo = empNo;

}

@Column(name = "EMP\_NAME", length = 50, nullable = **false**)

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

@Column(name = "JOB", length = 30, nullable = **false**)

**public** String getJob() {

**return** job;

}

**public** **void** setJob(String job) {

**this**.job = job;

}

@ManyToOne(fetch = FetchType.***LAZY***)

@JoinColumn(name = "MNG\_ID")

**public** Employee getManager() {

**return** manager;

}

**public** **void** setManager(Employee manager) {

**this**.manager = manager;

}

@Column(name = "HIRE\_DATE", nullable = **false**)

@Temporal(TemporalType.***DATE***)

**public** Date getHideDate() {

**return** hideDate;

}

**public** **void** setHideDate(Date hideDate) {

**this**.hideDate = hideDate;

}

@Column(name = "SALARY", nullable = **false**)

**public** Float getSalary() {

**return** salary;

}

**public** **void** setSalary(Float salary) {

**this**.salary = salary;

}

@Column(name = "IMAGE", length = 1111111, nullable = **true**)

@Lob

**public** **byte**[] getImage() {

**return** image;

}

**public** **void** setImage(**byte**[] image) {

**this**.image = image;

}

@ManyToOne(fetch = FetchType.***LAZY***)

@JoinColumn(name = "DEPT\_ID", nullable = **false**)

**public** Department getDepartment() {

**return** department;

}

**public** **void** setDepartment(Department department) {

**this**.department = department;

}

@OneToMany(fetch = FetchType.***LAZY***, mappedBy = "empId")

**public** Set<Employee> getEmployees() {

**return** employees;

}

**public** **void** setEmployees(Set<Employee> employees) {

**this**.employees = employees;

}

}

#### SalaryGrade.java

**package** test.entities;

**import** javax.persistence.\*;

@Entity

@Table(name = "SALARY\_GRADE")

**public** **class** SalaryGrade {

**private** Integer grade;

**private** Float lowSalary;

**private** Float highSalary;

**public** SalaryGrade() {

}

**public** SalaryGrade(Integer grade, Float lowSalary, Float highSalary) {

**this**.grade = grade;

**this**.lowSalary = lowSalary;

**this**.highSalary = highSalary;

}

@Id

@Column(name = "GRADE")

**public** Integer getGrade() {

**return** grade;

}

**public** **void** setGrade(Integer grade) {

**this**.grade = grade;

}

@Column(name = "LOW\_SALARY", nullable = **false**)

**public** Float getLowSalary() {

**return** lowSalary;

}

**public** **void** setLowSalary(Float lowSalary) {

**this**.lowSalary = lowSalary;

}

@Column(name = "HIGH\_SALARY", nullable = **false**)

**public** Float getHighSalary() {

**return** highSalary;

}

**public** **void** setHighSalary(Float highSalary) {

**this**.highSalary = highSalary;

}

}

#### Timekeeper.java

**package** test.entities;

**import** java.util.Date;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.ManyToOne;

**import** javax.persistence.Table;

**import** javax.persistence.Temporal;

**import** javax.persistence.TemporalType;

**import** org.hibernate.annotations.GenericGenerator;

@Entity

@Table(name = "TIMEKEEPER")

**public** **class** Timekeeper {

**public** **static** **final** **char** ***IN*** = 'I';

**public** **static** **final** **char** ***OUT*** = 'O';

**private** String timekeeperId;

**private** Date dateTime;

**private** Employee employee;

// 'I' or 'O'

**private** **char** inOut;

@Id

@GeneratedValue(generator = "uuid")

@GenericGenerator(name = "uuid", strategy = "uuid2")

@Column(name = "Timekeeper\_Id", length = 36)

**public** String getTimekeeperId() {

**return** timekeeperId;

}

**public** **void** setTimekeeperId(String timekeeperId) {

**this**.timekeeperId = timekeeperId;

}

@Column(name = "Date\_Time", nullable = **false**)

@Temporal(TemporalType.***TIMESTAMP***)

**public** Date getDateTime() {

**return** dateTime;

}

**public** **void** setDateTime(Date dateTime) {

**this**.dateTime = dateTime;

}

@ManyToOne(fetch = FetchType.***LAZY***)

@JoinColumn(name = "EMP\_ID", nullable = **false**)

**public** Employee getEmployee() {

**return** employee;

}

**public** **void** setEmployee(Employee employee) {

**this**.employee = employee;

}

@Column(name = "In\_Out", nullable = **false**, length = 1)

**public** **char** getInOut() {

**return** inOut;

}

**public** **void** setInOut(**char** inOut) {

**this**.inOut = inOut;

}

}

### Create SessionFactory package and run demo

#### Create HibernateUtils.java

**package** test.factory;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.boot.Metadata;

**import** org.hibernate.boot.MetadataSources;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**import** org.hibernate.service.ServiceRegistry;

**public** **class** HibernateUtils {

**private** **static** **final** SessionFactory ***sessionFactory*** = *buildSessionFactory*();

// Hibernate 5:

**private** **static** SessionFactory buildSessionFactory() {

**try** {

// Tạo đối tượng ServiceRegistry từ hibernate.cfg.xml

ServiceRegistry serviceRegistry = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

// Tạo nguồn siêu dữ liệu (metadata) từ ServiceRegistry

Metadata metadata = **new** MetadataSources(serviceRegistry).getMetadataBuilder().build();

**return** metadata.getSessionFactoryBuilder().build();

} **catch** (Throwable ex) {

System.***err***.println("Initial SessionFactory creation failed." + ex);

**throw** **new** ExceptionInInitializerError(ex);

}

}

**public** **static** SessionFactory getSessionFactory() {

**return** ***sessionFactory***;

}

**public** **static** **void** shutdown() {

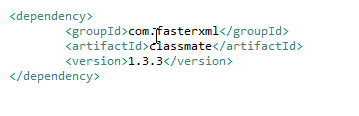
// Giải phóng cache và Connection Pools.

*getSessionFactory*().close();

}

}

If you have errors saying you need classmate.jar then please add dependencies of classmate into pom.xml



#### QueryObjectDemo.java

This demo shows employees id and employees name. Hibernate calls out query without jdbc configuration.

**package** test.query;

**import** java.util.List;

**import** org.hibernate.query.Query;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** test.factory.HibernateUtils;

**import** test.entities.Employee;

**public** **class** QueryObjectDemo {

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

SessionFactory factory = HibernateUtils.*getSessionFactory*();

Session session = factory.getCurrentSession();

**try** {

// Every actions with database run through Hibernate

// Actions always in a Transaction

// Transaction Begin.

session.getTransaction().begin();

// Create a HQL query object

// The same with Native SQL

// Select e.\* from EMPLOYEE e order by e.EMP\_NAME, e.EMP\_NO

String sql = "Select e from " + Employee.**class**.getName() + " e "

+ " order by e.empName, e.empNo ";

// Query Object.

@SuppressWarnings("unchecked")

Query<Employee> query = session.createQuery(sql);

// Execute query

List<Employee> employees = query.getResultList();

**for** (Employee emp : employees) {

System.***out***.println("Emp: " + emp.getEmpNo() + " : "

+ emp.getEmpName());

}

// Commit data

session.getTransaction().commit();

} **catch** (Exception e) {

e.printStackTrace();

// Rollback in case of errors.

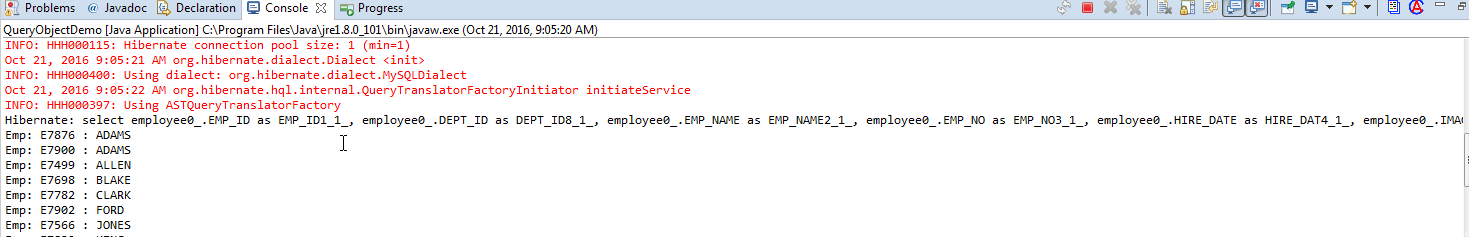
session.getTransaction().rollback();

}

}

}

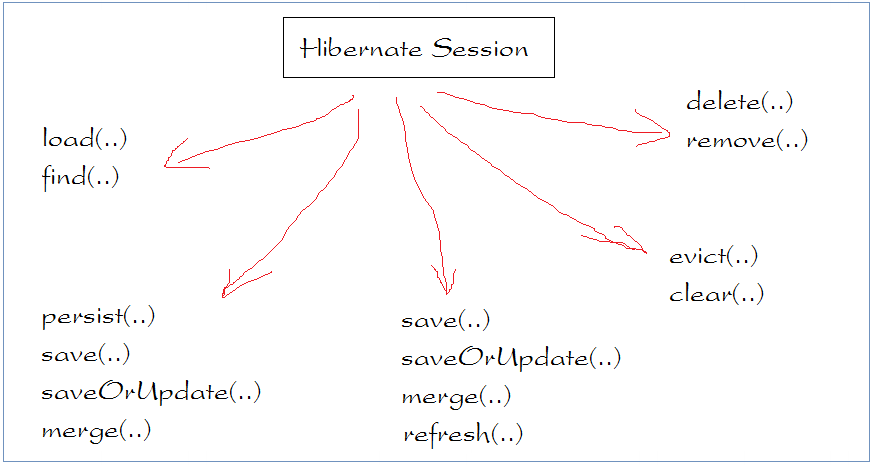
Results :



**Reference about Hibernate Query Language**

**http://docs.jboss.org/hibernate/orm/3.6/reference/en-US/html/queryhql.html**

**CRUD  
Hibernate lifecycle**



Class **Session** is very important method. The method divided into groups.

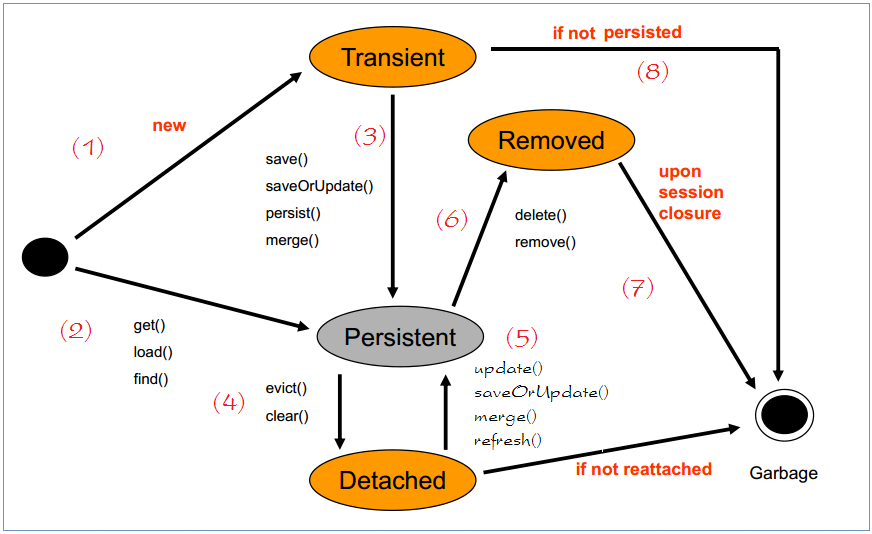
An object can have 4 status :

Transient - temporary

Persistent – firm

Removed - removed in dtabase

Detached - out of session



1. Trường hợp bạn tạo mới một đối tượng java từ một Entity, đối tượng đó có tình trạng là ***Transient***. Hibernate không biết về sự tồn tại của nó. Nó nằm ngoài sự quản lý của Hibernate.
2. Trường hợp bạn lấy ra đối tượng Entity bằng method get, load hoặc find, bạn có được một đối tượng nó tương ứng với 1 record dưới database. Đối tượng này có trạng thái ***Persistent***. Nó được quản lý bởi Hibernate.
3. Session gọi một trong các method save,saveOrUpdate, persist, merge sẽ đẩy đối tượng ***Transient*** vào sự quản lý của Hibernate và đối tượng này chuyển sang trạng thái ***Persistent***. Tùy tình huống nó sẽ insert hoặc update dữ liệu vào DB.
4. Session gọi evict(..) hoặc clear() để đuổi các đối tượng có trạng thái persistent (bền vững) ra khỏi sự quản lý của Hibernate, giờ các đối tượng này sẽ có trạng thái mới là Detected (Bị tách ra).  Nếu nó không được đính (Attached) trở lại, nó sẽ bị bộ gom rác của Java quét đi theo cơ chế thông thường.
5. Sử dụng update(..), saveOrUpdate(..), merge(..) sẽ đính trở lại các đối tượng Detached vào lại. Tùy tình huống nó sẽ tạo ra dưới DB câu lệnh update hoặc insert. Các đối tượng sẽ trở về trạng thái Persistent (bền vững).
6. Session gọi method remove(..), delete(..) để xóa một bản ghi, đối tượng persistent giờ chuyển sang trạng thái Removed (Đã bị xóa).
   1. Persistent

### Data Utils

package test**.**factory**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**query**.**Query**;**

**import** test**.**entities**.**Department**;**

**import** test**.**entities**.**Employee**;**

public class DataUtils **{**

public static Department findDepartment**(**Session session**,** String deptNo**)** **{**

String sql **=** "Select d from " **+** Department**.**class**.**getName**()** **+** " d "//

**+** " Where d.deptNo = :deptNo"**;**

Query**<**Department**>** query **=** session**.**createQuery**(**sql**);**

query**.**setParameter**(**"deptNo"**,** deptNo**);**

**return** query**.**getSingleResult**();**

**}**

public static Long getMaxEmpId**(**Session session**)** **{**

String sql **=** "Select max(e.empId) from " **+** Employee**.**class**.**getName**()** **+** " e "**;**

Query**<**Number**>** query **=** session**.**createQuery**(**sql**);**

Number value **=** query**.**getSingleResult**();**

**if** **(**value **==** **null)** **{**

**return** 0L**;**

**}**

**return** value**.**longValue**();**

**}**

public static Employee findEmployee**(**Session session**,** String empNo**)** **{**

String sql **=** "Select e from " **+** Employee**.**class**.**getName**()** **+** " e "//

**+** " Where e.empNo = :empNo"**;**

Query**<**Employee**>** query **=** session**.**createQuery**(**sql**);**

query**.**setParameter**(**"empNo"**,** empNo**);**

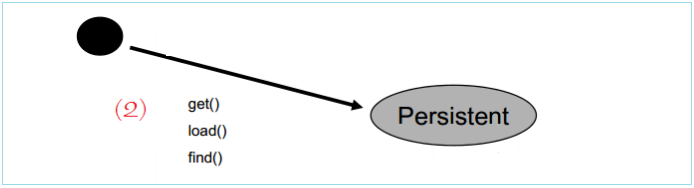
**return** query**.**getSingleResult**();**

**}**

**}**

This example show how to use Session.persist(object) to insert an Trasient object into Database.

#### This example also give a detail explaination on Transient , Persistent , and Detached.



#### PersistDemo.java

package test**.**query**;**

**import** java**.**util**.**Date**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**entities**.**Department**;**

**import** test**.**entities**.**Employee**;**

public class PersistDemo **{**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Department department **=** **null;**

Employee emp **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

Long maxEmpId **=** DataUtils**.**getMaxEmpId**(**session**);**

Long empId **=** maxEmpId **+** 1**;**

// Phòng ban với mã số D10.

// Nó là đối tượng chịu sự quản lý của session

// Và được gọi là đối tượng persistent.

department **=** DataUtils**.**findDepartment**(**session**,** "D10"**);**

// Tạo mới đối tượng Employee

// Đối tượng này chưa chịu sự quản lý của session.

// Nó được coi là đối tượng Transient.

emp **=** **new** Employee**();**

emp**.**setEmpId**(**empId**);**

emp**.**setEmpNo**(**"E" **+** empId**);**

emp**.**setEmpName**(**"Name " **+** empId**);**

emp**.**setJob**(**"Coder"**);**

emp**.**setSalary**(**1000f**);**

emp**.**setManager**(null);**

emp**.**setHideDate**(new** Date**());**

emp**.**setDepartment**(**department**);**

// Sử dụng persist(..)

// Lúc này 'emp' đã chịu sự quản lý của session.

// nó có trạng thái persistent.

// Chưa có hành động gì với DB tại đây.

session**.**persist**(**emp**);**

// Tại bước này dữ liệu mới được đẩy xuống DB.

// Câu lệnh Insert được tạo ra.

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

// Sau khi session bị đóng lại (commit, rollback, close)

// Đối tượng 'emp', 'dept' trở thành đối tượng Detached.

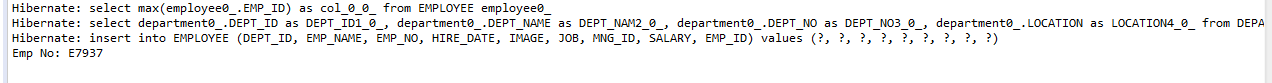
// Nó không còn trong sự quản lý của session nữa.

System**.**out**.**println**(**"Emp No: " **+** emp**.**getEmpNo**());**

**}**

**}**

The result.



#### PersistDemo2.java

package test**.**query**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Department**;**

public class PersistentDemo2 **{**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Department department **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

System**.**out**.**println**(**"- Finding Department deptNo = D10..."**);**

// Đây là một đối tượng có trạng thái Persistent.

department **=** DataUtils**.**findDepartment**(**session**,** "D10"**);**

System**.**out**.**println**(**"- First change Location"**);**

// Thay đổi gì đó trên đối tượng Persistent.

department**.**setLocation**(**"Chicago " **+** System**.**currentTimeMillis**());**

System**.**out**.**println**(**"- Location = " **+** department**.**getLocation**());**

System**.**out**.**println**(**"- Calling flush..."**);**

// Sử dụng session.flush() để chủ động đẩy các thay đổi xuống DB.

// Có tác dụng trên tất cả các đối tượng Persistent có thay đổi.

session**.**flush**();**

System**.**out**.**println**(**"- Flush OK"**);**

System**.**out**.**println**(**"- Second change Location"**);**

// Thay đổi gì đó trên đối tượng Persistent.

// Hình thành câu lệnh update, sẽ được thực thi

// sau khi session đóng lại (commit).

department**.**setLocation**(**"Chicago " **+** System**.**currentTimeMillis**());**

// In ra Location.

System**.**out**.**println**(**"- Location = " **+** department**.**getLocation**());**

System**.**out**.**println**(**"- Calling commit..."**);**

// Lệnh commit sẽ làm tất cả những sự thay đổi được đẩy xuống DB.

session**.**getTransaction**().**commit**();**

System**.**out**.**println**(**"- Commit OK"**);**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

// Tạo lại session sau khi nó đã bị đóng trước đó

// (Do commit hoặc rollback)

session **=** factory**.**getCurrentSession**();**

**try** **{**

session**.**getTransaction**().**begin**();**

System**.**out**.**println**(**"- Finding Department deptNo = D10..."**);**

// Query lại Department D10.

department **=** DataUtils**.**findDepartment**(**session**,** "D10"**);**

// In ra thông tin Location.

System**.**out**.**println**(**"- D10 Location = " **+** department**.**getLocation**());**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

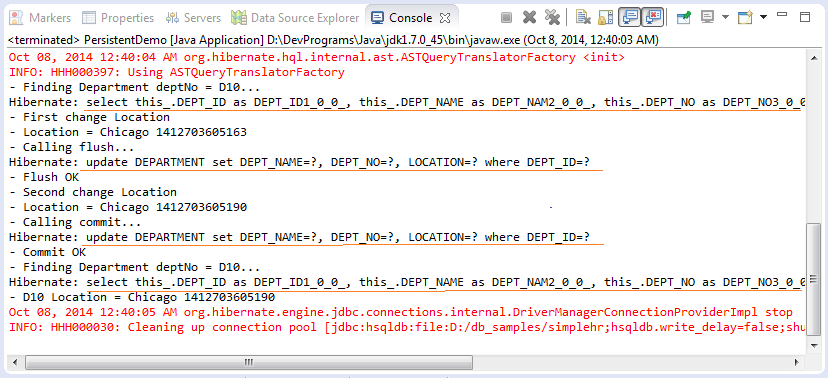
e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

**}**

**}**



### Transient -> Persistent

There are 4 ways to change transient to persistent with different style and purposes :

Persist(Object) : convert the object to persistent , but the function is void()

Save(Object) : convert the object to persistent , return the session ID

SaveOrUpdate(Object) : If have session Id , do nothing , if not , It give a session ID

Merge(Object): maintain the transient of the original variable , give another variable that value + the persistent status.

TransientPersistentDemo.java ( Using function persist(Object) )

package test**.**query**;**

**import** java**.**text**.**DateFormat**;**

**import** java**.**text**.**SimpleDateFormat**;**

**import** java**.**util**.**Date**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Employee**;**

**import** test**.**entities**.**Timekeeper**;**

public class TransientPersistentDemo **{**

private static DateFormat df **=** **new** SimpleDateFormat**(**"dd-MM-yyyy HH:mm:ss"**);**

private static Timekeeper persist\_Transient**(**Session session**,** Employee emp**)** **{**

// Hãy chú ý:

// timekeeperId cấu hình tự động được tạo ra bởi UUID.

// @GeneratedValue(generator = "uuid")

// @GenericGenerator(name = "uuid", strategy = "uuid2")

// Tạo một đối tượng, nó đang có tình trạng Transient.

Timekeeper tk1 **=** **new** Timekeeper**();**

tk1**.**setEmployee**(**emp**);**

tk1**.**setInOut**(**Timekeeper**.**IN**);**

tk1**.**setDateTime**(new** Date**());**

// Now, 'tk1' is transient object

System**.**out**.**println**(**"- tk1 Persistent? " **+** session**.**contains**(**tk1**));**

System**.**out**.**println**(**"====== CALL persist(tk).... ==========="**);**

// Hibernate gán Id vào 'tk1', sẽ chưa có insert gì cả

session**.**persist**(**tk1**);**

// 'tk1' đã được gắn ID

System**.**out

**.**println**(**"- tk1.getTimekeeperId() = " **+** tk1**.**getTimekeeperId**());**

// Lúc này 'tk1' đã có trạng thái Persistent

// Nó đã được quản lý trong Session.

// Nhưng chưa có hành động gì insert xuống DB.

// ==> true

System**.**out**.**println**(**"- tk1 Persistent? " **+** session**.**contains**(**tk1**));**

System**.**out**.**println**(**"- Call flush.."**);**

// Chủ động đẩy dữ liệu xuống DB, gọi flush().

// Nếu không gọi flush() dữ liệu sẽ được đẩy xuống tại lệnh commit().

// Lúc này mới có insert.

session**.**flush**();**

String timekeeperId **=** tk1**.**getTimekeeperId**();**

System**.**out**.**println**(**"- timekeeperId = " **+** timekeeperId**);**

System**.**out**.**println**(**"- inOut = " **+** tk1**.**getInOut**());**

System**.**out**.**println**(**"- dateTime = " **+** df**.**format**(**tk1**.**getDateTime**()));**

System**.**out**.**println**();**

**return** tk1**;**

**}**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

emp **=** DataUtils**.**findEmployee**(**session**,** "E7499"**);**

persist\_Transient**(**session**,** emp**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

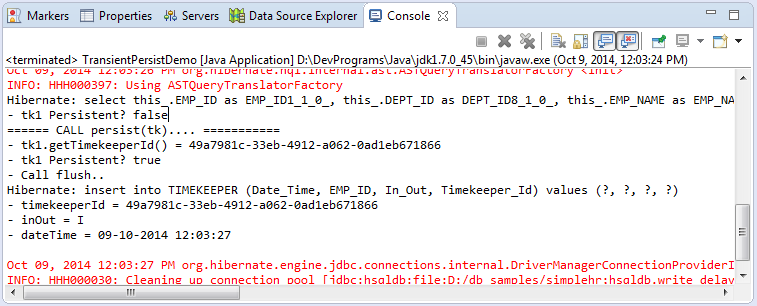
e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

**}**

**}**



Transient -> Persistent ( Using function save(Object) \* )

TransientPersistentDemo2.java

package test**.**query**;**

**import** java**.**io**.**Serializable**;**

**import** java**.**text**.**DateFormat**;**

**import** java**.**text**.**SimpleDateFormat**;**

**import** java**.**util**.**Date**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Employee**;**

**import** test**.**entities**.**Timekeeper**;**

public class TransientPersistentDemo2 **{**

private static DateFormat df **=** **new** SimpleDateFormat**(**"dd-MM-yyyy HH:mm:ss"**);**

private static Timekeeper persist\_Transient**(**Session session**,** Employee emp**)** **{**

// Hãy chú ý:

// timekeeperId cấu hình tự động được tạo ra bởi UUID.

// @GeneratedValue(generator = "uuid")

// @GenericGenerator(name = "uuid", strategy = "uuid2")

// Tạo một đối tượng, nó đang có tình trạng Transient.

Timekeeper tk2 **=** **new** Timekeeper**();**

tk2**.**setEmployee**(**emp**);**

tk2**.**setInOut**(**Timekeeper**.**IN**);**

tk2**.**setDateTime**(new** Date**());**

// Lúc này 'tk2' đang có tình trạng Transient.

System**.**out**.**println**(**"- tk2 Persistent? " **+** session**.**contains**(**tk2**));**

System**.**out**.**println**(**"====== CALL save(tk).... ==========="**);**

// save() rất giống với persist()

// save() trả về ID còn persist() là void.

// Hibernate gán Id vào 'tk2', sẽ chưa có insert gì cả

// Nó trả về ID của 'tk2'.

Serializable id **=** session**.**save**(**tk2**);**

System**.**out**.**println**(**"- id = " **+** id**);**

// 'tk2' đã được gắn ID

System**.**out

**.**println**(**"- tk2.getTimekeeperId() = " **+** tk2**.**getTimekeeperId**());**

// Lúc này 'tk2' đã có trạng thái Persistent

// Nó đã được quản lý trong Session.

// Nhưng chưa có hành động gì insert xuống DB.

// ==> true

System**.**out**.**println**(**"- tk2 Persistent? " **+** session**.**contains**(**tk2**));**

System**.**out**.**println**(**"- Call flush.."**);**

// Chủ động đẩy dữ liệu xuống DB, gọi flush().

// Nếu không gọi flush() dữ liệu sẽ được đẩy xuống tại lệnh commit().

// Lúc này mới có insert.

session**.**flush**();**

String timekeeperId **=** tk2**.**getTimekeeperId**();**

System**.**out**.**println**(**"- timekeeperId = " **+** timekeeperId**);**

System**.**out**.**println**(**"- inOut = " **+** tk2**.**getInOut**());**

System**.**out**.**println**(**"- dateTime = " **+** df**.**format**(**tk2**.**getDateTime**()));**

System**.**out**.**println**();**

**return** tk2**;**

**}**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

emp **=** DataUtils**.**findEmployee**(**session**,** "E7499"**);**

persist\_Transient**(**session**,** emp**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

**}**

**}**

Transient -> Persistent ( SaveOrUpdate(object) )

package test**.**query**;**

**import** java**.**text**.**DateFormat**;**

**import** java**.**text**.**SimpleDateFormat**;**

**import** java**.**util**.**Date**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Employee**;**

**import** test**.**entities**.**Timekeeper**;**

public class SaveOrUpdateTransientDemo **{**

private static DateFormat df **=** **new** SimpleDateFormat**(**"dd-MM-yyyy HH:mm:ss"**);**

private static Timekeeper saveOrUpdate\_Transient**(**Session session**,**

Employee emp**)** **{**

// Hãy chú ý:

// timekeeperId cấu hình tự động được tạo ra bởi UUID.

// @GeneratedValue(generator = "uuid")

// @GenericGenerator(name = "uuid", strategy = "uuid2")

// Tạo một đối tượng, nó đang có tình trạng Transient.

Timekeeper tk3 **=** **new** Timekeeper**();**

tk3**.**setEmployee**(**emp**);**

tk3**.**setInOut**(**Timekeeper**.**IN**);**

tk3**.**setDateTime**(new** Date**());**

// Lúc này 'tk3' đang có tình trạng Transient.

System**.**out**.**println**(**"- tk3 Persistent? " **+** session**.**contains**(**tk3**));**

System**.**out**.**println**(**"====== CALL saveOrUpdate(tk).... ==========="**);**

// Tại đây Hibernate sẽ kiểm tra, tk3 có ID chưa (timekeeperId)

// Nếu chưa có nó tự gán ID vào.

session**.**saveOrUpdate**(**tk3**);**

System**.**out

**.**println**(**"- tk3.getTimekeeperId() = " **+** tk3**.**getTimekeeperId**());**

// Lúc này 'tk3' đã có trạng thái Persistent

// Nó đã được quản lý trong Session.

// Nhưng chưa có hành động gì insert, hoặc update xuống DB.

// ==> true

System**.**out**.**println**(**"- tk3 Persistent? " **+** session**.**contains**(**tk3**));**

System**.**out**.**println**(**"- Call flush.."**);**

// Chủ động đẩy dữ liệu xuống DB, gọi flush().

// Nếu không gọi flush() dữ liệu sẽ được đẩy xuống tại lệnh commit().

// Lúc này có thể có Insert hoặc Update xuống DB. (!!!)

// Tùy thuộc vào ID của 'tk3' có trên DB chưa.

session**.**flush**();**

String timekeeperId **=** tk3**.**getTimekeeperId**();**

System**.**out**.**println**(**"- timekeeperId = " **+** timekeeperId**);**

System**.**out**.**println**(**"- inOut = " **+** tk3**.**getInOut**());**

System**.**out**.**println**(**"- dateTime = " **+** df**.**format**(**tk3**.**getDateTime**()));**

System**.**out**.**println**();**

**return** tk3**;**

**}**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

emp **=** DataUtils**.**findEmployee**(**session**,** "E7499"**);**

saveOrUpdate\_Transient**(**session**,** emp**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

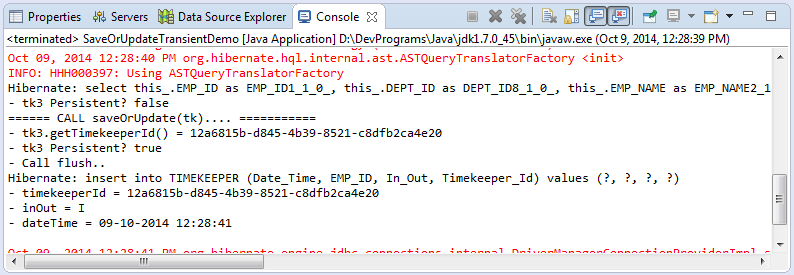
e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

**}**

**}**



Transient -> Persistent - MergeTransientDemo.java ( Merge(Object))

package test**.**query**;**

**import** java**.**text**.**DateFormat**;**

**import** java**.**text**.**SimpleDateFormat**;**

**import** java**.**util**.**Date**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Employee**;**

**import** test**.**entities**.**Timekeeper**;**

public class MergeTransientDemo **{**

private static DateFormat df **=** **new** SimpleDateFormat**(**"dd-MM-yyyy HH:mm:ss"**);**

private static Timekeeper saveOrUpdate\_Transient**(**Session session**,**

Employee emp**)** **{**

// Hãy chú ý:

// timekeeperId cấu hình tự động được tạo ra bởi UUID.

// @GeneratedValue(generator = "uuid")

// @GenericGenerator(name = "uuid", strategy = "uuid2")

// Tạo một đối tượng, nó đang có tình trạng Transient.

Timekeeper tk4 **=** **new** Timekeeper**();**

tk4**.**setEmployee**(**emp**);**

tk4**.**setInOut**(**Timekeeper**.**IN**);**

tk4**.**setDateTime**(new** Date**());**

// Lúc này 'tk4' đang có tình trạng Transient.

System**.**out**.**println**(**"- tk4 Persistent? " **+** session**.**contains**(**tk4**));**

System**.**out**.**println**(**"====== CALL merge(tk).... ==========="**);**

// Hibernate2 có method copyOrUpdateCopy

// phiên bản 3 trở nó đổi tên thành merge.

// (Vì vậy sẽ có sự tương đồng).

// Tại đây Hibernate sẽ kiểm tra, tk4 có ID (timekeeperId) chưa

// Nếu chưa có nó tự gán ID vào.

// Sau đó tạo ra một bản copy và trả về.

Timekeeper tk4Copy **=** **(**Timekeeper**)** session**.**merge**(**tk4**);**

System**.**out

**.**println**(**"- tk4.getTimekeeperId() = " **+** tk4**.**getTimekeeperId**());**

// Chú ý:

// Lúc này 'tk4' vẫn có trạng thái Transient.

// còn 'tk4Copy' thì có trạng thái Persistent.

// Nhưng chưa có hành động gì insert, hoặc update xuống DB.

// ==> false

System**.**out**.**println**(**"- tk4 Persistent? " **+** session**.**contains**(**tk4**));**

// 'tk4Copy' có trạng thái Persistent.

// ==> true

System**.**out

**.**println**(**"- tk4Copy Persistent? " **+** session**.**contains**(**tk4Copy**));**

System**.**out**.**println**(**"- Call flush.."**);**

// Chủ động đẩy dữ liệu xuống DB, gọi flush().

// Nếu không gọi flush() dữ liệu sẽ được đẩy xuống tại lệnh commit().

// Lúc này có thể có Insert hoặc Update xuống DB. (!!!)

// Tùy thuộc vào ID của 'tk4' có trên DB chưa.

session**.**flush**();**

// 'tk4' vẫn là Transient, sau khi flush().

// Nhận xét: merge(..) an toàn hơn so với saveOrUpdate().

System**.**out**.**println**(**"- tk4 Persistent? " **+** session**.**contains**(**tk4**));**

//

String timekeeperId **=** tk4**.**getTimekeeperId**();**

System**.**out**.**println**(**"- timekeeperId = " **+** timekeeperId**);**

System**.**out**.**println**(**"- inOut = " **+** tk4**.**getInOut**());**

System**.**out**.**println**(**"- dateTime = " **+** df**.**format**(**tk4**.**getDateTime**()));**

System**.**out**.**println**();**

**return** tk4**;**

**}**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

emp **=** DataUtils**.**findEmployee**(**session**,** "E7499"**);**

saveOrUpdate\_Transient**(**session**,** emp**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

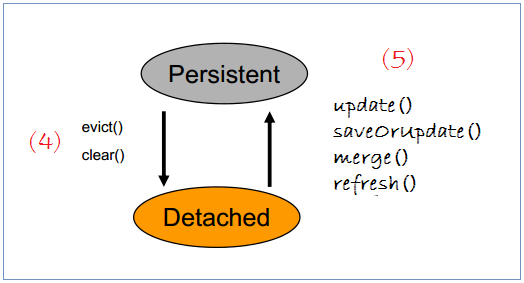
session**.**getTransaction**().**rollback**();**

**}**

**}**

**}**

### Persistent -> Detached and Detached -> Persistent



To detach Object , we can use function like evict() and clear()

Evict : Push out an object that is under Hibernate responsible.

Clear : Push out all objects that are under Hibernate responsible.

For object that are detached , it can be attach with the methods like : update() , saveorupdate() , merge() , refresh() , refresh() , lock()

Update() : Detached ( after commit() ) -> Persistent. But cannot not use this function with Transient status.

SaveOrUpdate() : Detached ( after commit() ) -> Persistent. Call Exception if aObject (Transient to Persistent) id is matched with bObject id( Is Persistent ).

Merge() : Create a copy of the detach object and make that copy persistent. Does have error like saveorupdate().

Refresh() : Any changes is detached object **is not saved** after using the function to turn the object to persistent.

EvictDemo.java

package test**.**query**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Employee**;**

public class EvictDemo **{**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

// Đây là một đối tượng có tình trạng Persistent.

emp **=** DataUtils**.**findEmployee**(**session**,** "E7499"**);**

// ==> true

System**.**out**.**println**(**"- emp Persistent? " **+** session**.**contains**(**emp**));**

// Sử dụng evict(Object) để đuổi đối tượng Persistent

// ra khỏi quản lý của Hibernate.

session**.**evict**(**emp**);**

// Lúc này 'emp' đang có trạng thái Detached.

// ==> false

System**.**out**.**println**(**"- emp Persistent? " **+** session**.**contains**(**emp**));**

// Tất cả các thay đổi trên 'emp' sẽ không được update

// nếu không đưa 'emp' trở lại trạng thái Persistent.

emp**.**setEmpNo**(**"NEW"**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

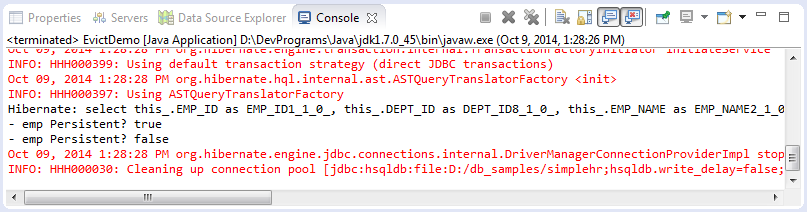
e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

**}**

**}**



ClearDemo.java

package test**.**query**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Department**;**

**import** test**.**entities**.**Employee**;**

public class ClearDemo **{**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

Department dept **=** **null;**

**try** **{**

session**.**getTransaction**().**begin**();**

// Đây là một đối tượng có tình trạng Persistent.

emp **=** DataUtils**.**findEmployee**(**session**,** "E7499"**);**

dept **=** DataUtils**.**findDepartment**(**session**,** "D10"**);**

// Sử dụng clear để đuổi hết tất cả các đối tượng có

// trạng thái Persistent ra khỏi sự quản lý của Hibernate.

session**.**clear**();**

// Lúc này 'emp' & 'dept' đang có trạng thái Detached.

// ==> false

System**.**out**.**println**(**"- emp Persistent? " **+** session**.**contains**(**emp**));**

System**.**out**.**println**(**"- dept Persistent? " **+** session**.**contains**(**dept**));**

// Tất cả các thay đổi trên 'emp' sẽ không được update

// nếu không đưa 'emp' trở lại trạng thái Persistent.

emp**.**setEmpNo**(**"NEW"**);**

dept **=** DataUtils**.**findDepartment**(**session**,** "D20"**);**

System**.**out**.**println**(**"Dept Name = "**+** dept**.**getDeptName**());**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

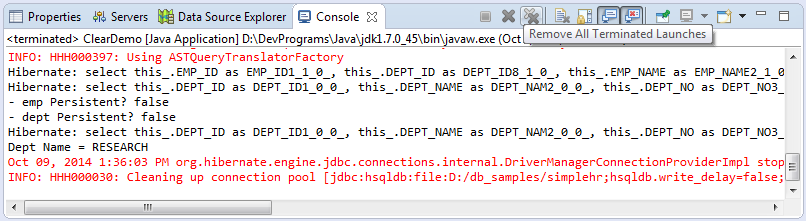
e**.**printStackTrace**();**

session**.**getTransaction**().**rollback**();**

**}**

**}**

**}**



UpdateDetachedDemo.java

package test**.**query**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Employee**;**

public class UpdateDetachedDemo **{**

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

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session1 **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

**try** **{**

session1**.**getTransaction**().**begin**();**

// Đây là đối tượng có trạng thái Persistent.

emp **=** DataUtils**.**findEmployee**(**session1**,** "E7499"**);**

// session1 đã bị đóng lại sau commit được gọi.

session1**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

session1**.**getTransaction**().**rollback**();**

**}**

// Mở một Session khác

Session session2 **=** factory**.**getCurrentSession**();**

**try** **{**

session2**.**getTransaction**().**begin**();**

// Kiểm tra trạng thái của emp:

// ==> false

System**.**out**.**println**(**"- emp Persistent? " **+** session2**.**contains**(**emp**));**

System**.**out**.**println**(**"Emp salary: " **+** emp**.**getSalary**());**

emp**.**setSalary**(**emp**.**getSalary**()** **+** 100**);**

// update(..) chỉ áp dụng cho đối tượng Detached.

// (Không dùng được với đối tượng Transient).

// Sử dụng update(emp) để đưa emp trở lại trạng thái Persistent.

session2**.**update**(**emp**);**

// Chủ động đẩy dữ liệu xuống DB.

// Câu lệnh update sẽ được gọi.

session2**.**flush**();**

System**.**out**.**println**(**"Emp salary after update: " **+** emp**.**getSalary**());**

// session2 đã bị đóng lại sau commit được gọi.

session2**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

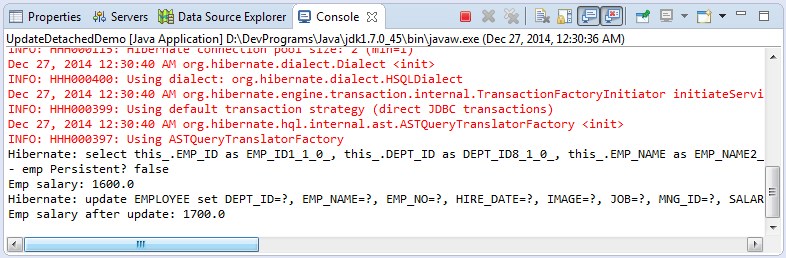
e**.**printStackTrace**();**

session2**.**getTransaction**().**rollback**();**

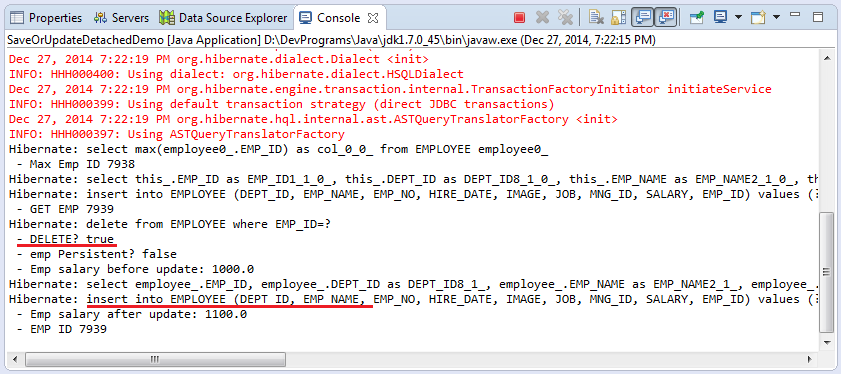
**}**

**}**

**}**



Saveorupdatedemo.java



**package** test.query;

**import** java.util.Random;

**import** org.hibernate.~~Query~~;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** test.factory.DataUtils;

**import** test.factory.HibernateUtils;

**import** test.entities.Employee;

**public** **class** SaveOrUpdateDetachedDemo {

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

// Một đối tượng có trạng thái Detached.

Employee emp = *getEmployee\_Detached*();

System.***out***.println(" - GET EMP " + emp.getEmpId());

// Ngẫu nhiên xóa hoặc không xóa Employee ứng với ID.

**boolean** delete = *deleteOrNotDelete*(emp.getEmpId());

System.***out***.println(" - DELETE? " + delete);

// saveOrUpdate đối tượng Detached.

*saveOrUpdate\_test*(emp);

// Sau khi gọi saveOrUpdate().

// Có thể ID của Entity sẽ khác đi trong trường hợp

// entity có ID tự tăng và saveOrUpdate tạo ra câu Insert.

System.***out***.println(" - EMP ID " + emp.getEmpId());

}

// Hàm trả về một đối tượng Employee đã

// nằm ngoài sự quản lý của Hibernate (Detached).

**private** **static** Employee getEmployee\_Detached() {

SessionFactory factory = HibernateUtils.*getSessionFactory*();

Session session1 = factory.getCurrentSession();

Employee emp = **null**;

**try** {

session1.getTransaction().begin();

Long maxEmpId = DataUtils.*getMaxEmpId*(session1);

System.***out***.println(" - Max Emp ID " + maxEmpId);

Employee emp2 = DataUtils.*findEmployee*(session1, "E7839");

Long empId = maxEmpId + 1;

emp = **new** Employee();

emp.setEmpId(empId);

emp.setEmpNo("E" + empId);

emp.setDepartment(emp2.getDepartment());

emp.setEmpName(emp2.getEmpName());

emp.setHideDate(emp2.getHideDate());

emp.setJob("Test");

emp.setSalary(1000F);

// emp đã được quản lý bởi Hibernate

session1.persist(emp);

// session1 đã bị đóng lại sau commit được gọi.

// Một bản ghi Employee đã được insert vào DB.

session1.getTransaction().commit();

} **catch** (Exception e) {

e.printStackTrace();

session1.getTransaction().rollback();

}

// session1 đã bị đóng 'emp' đã trở về trạng thái Detached.

**return** emp;

}

// Xóa Employee theo ID cho bởi tham số.

// Ngẫu nhiên xóa hoặc không xóa.

**private** **static** **boolean** deleteOrNotDelete(Long empId) {

// Một số ngẫu nhiên từ 0-9

**int** random = **new** Random().nextInt(10);

**if** (random < 5) {

**return** **false**;

}

SessionFactory factory = HibernateUtils.*getSessionFactory*();

Session session2 = factory.getCurrentSession();

**try** {

session2.getTransaction().begin();

String sql = "Delete " + Employee.**class**.getName() + " e "

+ " where e.empId =:empId ";

~~Query~~ query = session2.createQuery(sql);

query.~~setParameter~~("empId", empId);

query.executeUpdate();

session2.getTransaction().commit();

**return** **true**;

} **catch** (Exception e) {

e.printStackTrace();

session2.getTransaction().rollback();

**return** **false**;

}

}

**private** **static** **void** saveOrUpdate\_test(Employee emp) {

SessionFactory factory = HibernateUtils.*getSessionFactory*();

// Mở một Session khác

Session session3 = factory.getCurrentSession();

**try** {

session3.getTransaction().begin();

// Thực tế emp đang có trạng thái Detached

// Nó không được quản lý bởi Hibernate.

// Kiểm tra trạng thái của emp:

// ==> false

System.***out***.println(" - emp Persistent? " + session3.contains(emp));

System.***out***.println(" - Emp salary before update: "

+ emp.getSalary());

// Set salary mới cho đối tượng Detached emp.

// Bạn cũng có thể sét ID mới nếu muốn.

emp.setSalary(emp.getSalary() + 100);

// Sử dụng saveOrUpdate(emp) để đưa emp

// trở lại trạng thái Persistent.

// Chú ý: Nếu có một đối tượng có cùng ID

// đang được quản lý bởi Hibernate mà gọi hàm này sẽ bị Exception.

//

// Lúc này vẫn chưa có sử lý gì liên quan DB.

session3.saveOrUpdate(emp);

// Chủ động đẩy dữ liệu xuống DB.

// Tại đây có thể có thể tạo ra câu Insert hoặc Update vào DB.

// Nếu bản ghi tương ứng đã bị xóa bởi ai đó, câu lệnh Insert sẽ

// được tạo ra.

// Ngược lại sẽ là một câu lệnh Update.

session3.flush();

System.***out***

.println(" - Emp salary after update: " + emp.getSalary());

// session3 đã bị đóng lại sau commit được gọi.

session3.getTransaction().commit();

} **catch** (Exception e) {

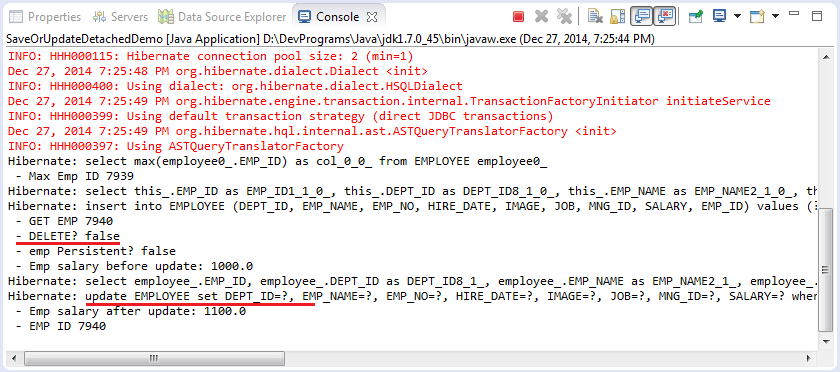
e.printStackTrace();

session3.getTransaction().rollback();

}

}

}



MergeDetachedDemo.java

package test**.**query**;**

**import** java**.**util**.**Random**;**

**import** org**.**hibernate**.**Query**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** test**.**factory**.**DataUtils**;**

**import** test**.**factory**.**HibernateUtils**;**

**import** test**.**entities**.**Employee**;**

public class MergeDetachedDemo **{**

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

// Một đối tượng có trạng thái Detached.

Employee emp **=** getEmployee\_Detached**();**

System**.**out**.**println**(**" - GET EMP " **+** emp**.**getEmpId**());**

// Ngẫu nhiên xóa hoặc không xóa Employee ứng với ID.

boolean delete **=** deleteOrNotDelete**(**emp**.**getEmpId**());**

System**.**out**.**println**(**" - DELETE? " **+** delete**);**

// saveOrUpdate đối tượng Detached.

saveOrUpdate\_test**(**emp**);**

// Sau khi gọi saveOrUpdate().

// Có thể ID của Entity sẽ khác đi trong trường hợp

// entity có ID tự tăng và saveOrUpdate tạo ra câu Insert.

System**.**out**.**println**(**" - EMP ID " **+** emp**.**getEmpId**());**

**}**

// Hàm trả về một đối tượng Employee đã

// nằm ngoài sự quản lý của Hibernate (Detached).

private static Employee getEmployee\_Detached**()** **{**

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session1 **=** factory**.**getCurrentSession**();**

Employee emp **=** **null;**

**try** **{**

session1**.**getTransaction**().**begin**();**

Long maxEmpId **=** DataUtils**.**getMaxEmpId**(**session1**);**

System**.**out**.**println**(**" - Max Emp ID " **+** maxEmpId**);**

Employee emp2 **=** DataUtils**.**findEmployee**(**session1**,** "E7839"**);**

Long empId **=** maxEmpId **+** 1**;**

emp **=** **new** Employee**();**

emp**.**setEmpId**(**empId**);**

emp**.**setEmpNo**(**"E" **+** empId**);**

emp**.**setDepartment**(**emp2**.**getDepartment**());**

emp**.**setEmpName**(**emp2**.**getEmpName**());**

emp**.**setHideDate**(**emp2**.**getHideDate**());**

emp**.**setJob**(**"Test"**);**

emp**.**setSalary**(**1000F**);**

// emp đã được quản lý bởi Hibernate

session1**.**persist**(**emp**);**

// session1 đã bị đóng lại sau commit được gọi.

// Một bản ghi Employee đã được insert vào DB.

session1**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

session1**.**getTransaction**().**rollback**();**

**}**

// session1 đã bị đóng 'emp' đã trở về trạng thái Detached.

**return** emp**;**

**}**

// Xóa Employee theo ID cho bởi tham số.

// Ngẫu nhiên xóa hoặc không xóa.

private static boolean deleteOrNotDelete**(**Long empId**)** **{**

// Một số ngẫu nhiên từ 0-9

int random **=** **new** Random**().**nextInt**(**10**);**

**if** **(**random **<** 5**)** **{**

**return** **false;**

**}**

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

Session session2 **=** factory**.**getCurrentSession**();**

**try** **{**

session2**.**getTransaction**().**begin**();**

String sql **=** "Delete " **+** Employee**.**class**.**getName**()** **+** " e "

**+** " where e.empId =:empId "**;**

Query query **=** session2**.**createQuery**(**sql**);**

query**.**setParameter**(**"empId"**,** empId**);**

query**.**executeUpdate**();**

session2**.**getTransaction**().**commit**();**

**return** **true;**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

session2**.**getTransaction**().**rollback**();**

**return** **false;**

**}**

**}**

private static void saveOrUpdate\_test**(**Employee emp**)** **{**

SessionFactory factory **=** HibernateUtils**.**getSessionFactory**();**

// Mở một Session khác

Session session3 **=** factory**.**getCurrentSession**();**

**try** **{**

session3**.**getTransaction**().**begin**();**

// Thực tế emp đang có trạng thái Detached

// Nó không được quản lý bởi Hibernate.

// Kiểm tra trạng thái của emp:

// ==> false

System**.**out**.**println**(**" - emp Persistent? " **+** session3**.**contains**(**emp**));**

System**.**out**.**println**(**" - Emp salary before update: "

**+** emp**.**getSalary**());**

// Set salary mới cho đối tượng Detached emp.

// Bạn cũng có thể sét ID mới nếu muốn.

emp**.**setSalary**(**emp**.**getSalary**()** **+** 100**);**

// merge(emp) trả về empMerge, một bản copy của emp,

// empMerge được quản lý bởi Hibernate.

// Còn emp vẫn trong tình trạng Detached

//

// Lúc này vẫn chưa có sử lý gì liên quan DB.

Employee empMerge **=** **(**Employee**)** session3**.**merge**(**emp**);**

// ==> false

System**.**out**.**println**(**" - emp Persistent? " **+** session3**.**contains**(**emp**));**

// ==> true

System**.**out**.**println**(**" - empMerge Persistent? "

**+** session3**.**contains**(**empMerge**));**

// Chủ động đẩy dữ liệu xuống DB.

// Tại đây có thể có thể tạo ra câu Insert hoặc Update vào DB.

// Nếu bản ghi tương ứng đã bị xóa bởi ai đó, câu lệnh Insert sẽ

// được tạo ra.

// Ngược lại sẽ là một câu lệnh Update.

session3**.**flush**();**

System**.**out

**.**println**(**" - Emp salary after update: " **+** emp**.**getSalary**());**

// session3 đã bị đóng lại sau commit được gọi.

session3**.**getTransaction**().**commit**();**

**}** **catch** **(**Exception e**)** **{**

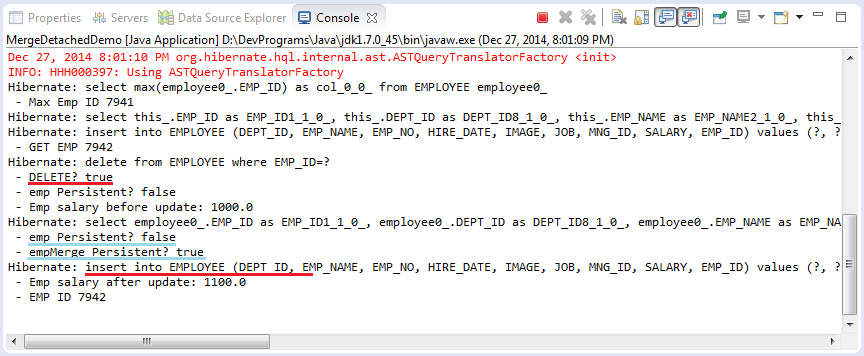
e**.**printStackTrace**();**

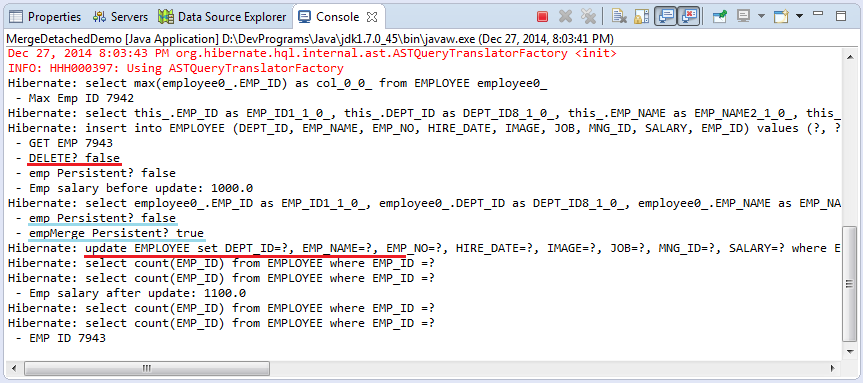
session3**.**getTransaction**().**rollback**();**

**}**

**}**

**}**





RefreshDetachedDemo.java

**package** test.query;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** test.factory.DataUtils;

**import** test.factory.HibernateUtils;

**import** test.entities.Employee;

**public** **class** RefreshDetachedDemo {

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

// Một đối tượng có trạng thái Detached.

Employee emp = *getEmployee\_Detached*();

System.***out***.println(" - GET EMP " + emp.getEmpId());

// refresh đối tượng Detached.

*refresh\_test*(emp);

}

// Hàm trả về một đối tượng Employee đã

// nằm ngoài sự quản lý của Hibernate (Detached).

**private** **static** Employee getEmployee\_Detached() {

SessionFactory factory = HibernateUtils.*getSessionFactory*();

Session session1 = factory.getCurrentSession();

Employee emp = **null**;

**try** {

session1.getTransaction().begin();

emp = DataUtils.*findEmployee*(session1, "E7839");

// session1 đã bị đóng lại sau commit được gọi.

// Một bản ghi Employee đã được insert vào DB.

session1.getTransaction().commit();

} **catch** (Exception e) {

e.printStackTrace();

session1.getTransaction().rollback();

}

// session1 đã bị đóng 'emp' đã trở về trạng thái Detached.

**return** emp;

}

**private** **static** **void** refresh\_test(Employee emp) {

SessionFactory factory = HibernateUtils.*getSessionFactory*();

// Mở một Session khác

Session session2 = factory.getCurrentSession();

**try** {

session2.getTransaction().begin();

// Thực tế emp đang có trạng thái Detached

// Nó không được quản lý bởi Hibernate.

// Kiểm tra trạng thái của emp:

// ==> false

System.***out***.println(" - emp Persistent? " + session2.contains(emp));

System.***out***.println(" - Emp salary before update: "

+ emp.getSalary());

// Set salary mới cho đối tượng Detached emp.

emp.setSalary(emp.getSalary() + 100);

// refresh tạo ra câu query.

// Và đưa đối tượng Detached về Persistent.

// Các thay đổi của emp không được lưu lại.O

session2.refresh(emp);

// ==> true

System.***out***.println(" - emp Persistent? " + session2.contains(emp));

System.***out***.println(" - Emp salary after refresh: "

+ emp.getSalary());

session2.getTransaction().commit();

} **catch** (Exception e) {

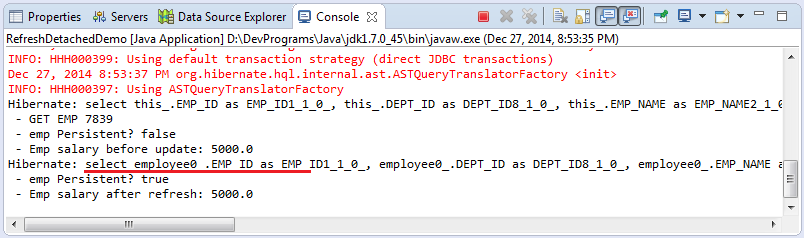
e.printStackTrace();

session2.getTransaction().rollback();

}

}

}



## Create tables from Entities of Hibernate

Generate schemas with JPA 2.1

<https://antoniogoncalves.org/2014/12/11/generating-database-schemas-with-jpa-2-1/>

Create table in hibernate ( guide and configuration ) :

<http://mrbool.com/how-to-create-database-table-using-hibernate/28269>

Please do take note this in hibernate-cfg.xml

**hbm2ddl.auto property :**

This property tell the hibernate to create or update database schema, while executing.