HIBERNATE - Relational Persistence for Idiomatic Java

Hibernate Reference Documentation

3.3.0.GA

HIBERNATE - Relational Persistence for Idiomatic Java

Legal Notice

1801 Varsity Drive Raleigh, NC27606-2072USA Phone: +1 919 754 3700 Phone: 888 733 4281 Fax: +1 919 754 3701 PO Box 13588Researc

The GPG fingerprint of the security@redhat.com key is:

CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

###

##
################# Hibernate#Java#########/###################
######/#############ORM####
###########SQL#########################
##############################

Hibernate#Java####################Java######SQL######
##
######SQL#JDBC#######################################

Hibernate##################################95################
Hibernate###########################
##
#################Java################################
###Hibernate#######SQL##################
##################################### ##################

Hibernate#######/############## ####Java#########################

1. # 1. *Hibernate*### ########## ##############
   ############################## `doc/reference/tutorial`
   ################

2. # 2. ####### #####Hibernate###################

3. Hibernate############ `eg/` ###############
   ############################# JDBC##### `lib/` ##############
   ########################## `etc/hibernate.properties` ##########
   ################################# `ant eg` ###########Ant#######
   ##Windows#### `build eg` ###########

4. ###################################
   ## *Java Persistence with Hibernate*#http://www.manning.com/bauer2# #########
   ##http://caveatemptor.hibernate.org## Java Persistence with Hibernate##########################

5. FAQ#Hibernate############

6. ######################Hibernate################

7. Hibernate### #Tomcat, JBoss AS, Struts, EJB, ################

######Hibernate##########################
#######################JIRA##################
Hibernate##################################
############################ ###############################

###################Hibernate########JBoss Inc.########
#http://www.hibernate.org/SupportTraining/#########
Hibernate#Professional Open Source####### ###JBoss Enterprise
Middleware System#JEMS############################

#1# Hibernate###

1.1.

This chapter is an introduction to Hibernate by way of a tutorial, intended for new users of Hibernate. We start with a simple application using an in-memory database. We build the application in small, easy to understand steps. The tutorial is based on another, earlier one developed by Michael Gloegl. All code is contained in the `tutorials/web` directory of the project source.

####

This tutorial expects the user have knowledge of both Java and SQL. If you are new or uncomfortable with either, it is advised that you start with a good introduction to that technology prior to attempting to learn Hibernate. It will save time and effort in the long run.

##

There is another tutorial/example application in the `/tutorials/eg` directory of the project source. That example is console based and as such would not have the dependency on a servlet container to execute. The basic setup is the same as the instructions below.

1.2. ###1 - ####Hibernate########

Let's assume we need a small database application that can store events we want to attend, and information about the host(s) of these events. We will use an in-memory, Java database named HSQLDB to avoid describing installation/setup of any particular database servers. Feel free to tweak this tutorial to use whatever database you feel comfortable using.

The first thing we need to do is set up our development environment, and specifically to setup all the required dependencies to Hibernate as well as other libraries. Hibernate is built using Maven which amongst other features provides `dependecy management`; moreover it provides *transitive* `dependecy management` which simply means that to use Hibernate we can simply define our dependency on Hibernate, Hibernate itself defines the dependencies it needs which then become transitive dependencies of our project.

```
.
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    ...

    <dependencies>
        <dependency>
            <groupId
>${groupId}</groupId>
            <artifactId
>hibernate-core</artifactId>
        </dependency>

        <!-- Because this is a web app, we also have a dependency on
 the servlet api. -->
        <dependency>
            <groupId
>javax.servlet</groupId>
            <artifactId
>servlet-api</artifactId>
        </dependency>
    </dependencies>

</project
>
```

##

Essentially we are describing here the `/tutorials/web/pom.xml` file. See the Maven [http://maven.org] site for more information.

####

While not strictly necessary, most IDEs have integration with Maven to read these POM files and automatically set up a project for you which can save lots of time and effort.

###############################

1.2.1.

############################JavaBean###

```
package org.hibernate.tutorial.domain;

import java.util.Date;

public class Event {
    private Long id;

    private String title;
    private Date date;

    public Event() {}
```

```
    public Long getId() {
        return id;
    }

    private void setId(Long id) {
        this.id = id;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

###################private################ JavaBean############################## ############################ #################################### Hibernate######################## ## #####

id ###################################### Hibernate############################## ############################ ######################## ###################web############ ## ############ID################# ##############private######## Hibernate################################# Hibernate##public, private, protected###################### public, private, protected################################ ###

############################# ###Hibernate#Java##################################### ########private############ ##################################### package#########

####### src #################### ##Java################# #############################

```
.
+lib
  <Hibernate and third-party libraries>
+src
  +events
    Event.java
```

#########Hibernate#################

1.2.2.

Hibernate##
###Hibernate#################
##
####################Hibernate######

##############################

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
[...]
</hibernate-mapping>
```

Hibernate DTD#########################
##DTD#######IDE##XML###################################
##DTD######################### ######################
################################# Hibernate##web##DTD############
##
DTD#####Hibernate############ `src/` ##########`hibernate3.jar`
##########

##################DTD########## #######################

2## `hibernate-mapping` ##### `class` ###########
##########################
##
#SQL###################################

```
<hibernate-mapping>

    <class name="events.Event" table="EVENTS">

    </class>
```

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

#########Event ########### EVENTS ################################# Hibernate## ### ############################### #############Hibernate################

```
<hibernate-mapping>

    <class name="events.Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
    </class>

</hibernate-mapping>
```

The id element is the declaration of the identifier property, name="id" declares the name of the Java property - Hibernate will use the getter and setter methods to access the property. The column attribute tells Hibernate which column of the EVENTS table we use for this primary key. The nested generator element specifies the identifier generation strategy, in this case we used native, which picks the best strategy depending on the configured database (dialect). Hibernate supports database generated, globally unique, as well as application assigned identifiers (or any strategy you have written an extension for).

################################### ###############################

```
<hibernate-mapping>

    <class name="events.Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```

id ########## property ### name ############################Hibernate###### ######Hibernate# getDate()/setDate() # getTitle()/setTitle() # #####

date ############# column ######title ################ column ########Hibernate#################### ### title ########## ### date ####################

######## title ###### type ############# ################type#############Java############# SQL################ ### *Hibernate*###### ###### Java##SQL#########SQL##Java################# ###########Hibernate# type ################### ################################ #Java######################## ## date ############### Hibernate#### java.util.Date ######## SQL# date , timestamp , time ############################## timestamp ####################################

#############Event.hbm.xml ### Event Java######################### #####################hbm.xml #######Hibernate######################## ##########################

```
.
+lib
  <Hibernate and third-party libraries>
+src
  +events
    Event.java
    Event.hbm.xml
```

Hibernate############

1.2.3. Hibernate###

################################Hibernate########## ################ HSQL DB#Java#########SQL DBMS####HSQL DB################### ############### hsqldb.jar ######## ############## lib/ ################

data ################################# HSQL DB################## ############### java -classpath lib/hsqldb.jar org.hsqldb.Server ##### ################# #######TCP/IP################# ######################### ############################### HSQL DB################## CTRL + C ###### data/ ##################### HSQL DB########

Hibernate########################## ################## #######JDBC####################################### Hibernate##########################JDBC######################

###########Hibernate######################
##################JDBC########################
#################

Hibernate######### `hibernate.properties` ##### ############## `hibernate.cfg.xml` ##### ################################### #############XML#########

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->
        <property
 name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property
 name="connection.url">jdbc:hsqldb:hsql://localhost</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property
 name="dialect">org.hibernate.dialect.HSQLDialect</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.NoCacheProvider</
property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create</property>

        <mapping resource="events/Event.hbm.xml"/>

    </session-factory>
```

```
</hibernate-configuration>
```

##XML#######DTD###############
############################Hibernate# SessionFactory #######
################### ################################## #####
<session-factory> #############

###4## property ###JDBC####################
dialect###### property ####Hibernate########SQL#########
##############Hibernate##
hbm2ddl.auto ######################on#####
###################### ###config##########off#####
SchemaExport ###Ant#############################
################################

########################### ###########################
Hibernate#################### hibernate.cfg.xml #################

1.2.4. Ant######

####Ant###################### ####Ant#####################
Ant######### [http://ant.apache.org/bindownload.cgi]
############### Ant####################### Ant#####
[http://ant.apache.org/manual/index.html] ##########
Ant############################## ####### build.xml
######################

######################

```
<project name="hibernate-tutorial" default="compile">

    <property name="sourcedir" value="${basedir}/src"/>
    <property name="targetdir" value="${basedir}/bin"/>
    <property name="librarydir" value="${basedir}/lib"/>

    <path id="libraries">
        <fileset dir="${librarydir}">
            <include name="*.jar"/>
        </fileset>
    </path>

    <target name="clean">
        <delete dir="${targetdir}"/>
        <mkdir dir="${targetdir}"/>
    </target>

    <target name="compile" depends="clean, copy-resources">
      <javac srcdir="${sourcedir}"
```

```
              destdir="${targetdir}"
              classpathref="libraries"/>
    </target>

    <target name="copy-resources">
        <copy todir="${targetdir}">
            <fileset dir="${sourcedir}">
                <exclude name="**/*.java"/>
            </fileset>
        </copy>
    </target>

</project>
```

.jar ####lib#################
#########################Ant######
###Java###
##########Hibernate############## #Ant######################

```
C:\hibernateTutorial\>ant
Buildfile: build.xml

copy-resources:
     [copy] Copying 2 files to C:\hibernateTutorial\bin

compile:
    [javac] Compiling 1 source file to C:\hibernateTutorial\bin

BUILD SUCCESSFUL
Total time: 1 second
```

1.2.5.

Event
##
###Hibernate################### ################### SessionFactory
############ ##
SessionFactory #### Session ################ Session
###############(Unit of Work)####### #####SessionFactory
######################### #################

############## ### SessionFactory ####### HibernateUtil
############# ###########

```
package util;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateUtil {
```

```
    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            sessionFactory = new
 Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be
 swallowed
            System.err.println("Initial SessionFactory creation
 failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

}
```

##############################JVM############# ####### `SessionFactory` ############ ################### #############JNDI## `SessionFactory` # ################

######## `SessionFactory` ######### Hibernate# `SessionFactory` ####JNDI################# ################### JMX############ JMX#################### `HibernateService` #JNDI############### ###############Hibernate########################

`HibernateUtil.java` ############### `events` ###### ###########

```
.
+lib
  <Hibernate and third-party libraries>
+src
  +events
    Event.java
    Event.hbm.xml
  +util
    HibernateUtil.java
  hibernate.cfg.xml
+data
build.xml
```

###################### ######################### Hibernate#commons logging######Log4j#JDK 1.4 logging############### #############Log4j### Hibernate############`etc/` #########

`log4j.properties` ##### `src` ######## `hibernate.cfg.xml` ############## ############################## #######Hibernate############################

################## Hibernate#####################

1.2.6.

#################Hibernate########### `main()` ####### `EventManager` #########

```
package events;
import org.hibernate.Session;

import java.util.Date;

import util.HibernateUtil;

public class EventManager {

    public static void main(String[] args) {
        EventManager mgr = new EventManager();

        if (args[0].equals("store")) {
            mgr.createAndStoreEvent("My Event", new Date());
        }

        HibernateUtil.getSessionFactory().close();
    }

    private void createAndStoreEvent(String title, Date theDate) {

        Session session =
 HibernateUtil.getSessionFactory().getCurrentSession();

        session.beginTransaction();

        Event theEvent = new Event();
        theEvent.setTitle(title);
        theEvent.setDate(theDate);

        session.save(theEvent);

        session.getTransaction().commit();
    }

}
```

`Event` ##############Hibernate###### ###Hibernate#SQL############ `INSERT` ####### #### `Session` # `Transaction`

`Session` #1######(Unit of Work)### ################# `Session` # `Transaction` ############### ############################JTA#########JDBC################ Hibernate `Session` ####### `Transaction` API#######

`sessionFactory.getCurrentSession()` ############## ####### `SessionFactory` ######### # `HibernateUtil` ######################## ################### `getCurrentSession()` #################(Unit of Work)###### `hibernate.cfg.xml` ##########"thread"################### ######################################Java####### #################

`Session` #################### `getCurrentSession()` # ############## ####Hibernate##################### ################################## Hibernate####### `Session` ############### ## `getCurrentSession()` ##### ### `Session` ####################### ## *thread-bound* #############Hibernate##################

Related to the unit of work scope, should the Hibernate `Session` be used to execute one or several database operations? The above example uses one `Session` for one operation. This is pure coincidence, the example is just not complex enough to show any other approach. The scope of a Hibernate `Session` is flexible but you should never design your application to use a new Hibernate `Session` for *every* database operation. So even if you see it a few more times in the following (very trivial) examples, consider *session-per-operation* an anti-pattern. A real (web) application is shown later in this tutorial.

############################ # 11. ############# ######## ##########################

##################Ant####################### ##############

```
<target name="run" depends="compile">
    <java fork="true" classname="events.EventManager"
 classpathref="libraries">
        <classpath path="${targetdir}"/>
        <arg value="${action}"/>
    </java>
</target>
```

`action` #################################

```
C:\hibernateTutorial\>ant run -Daction=store
```

#########Hibernate################### ###############
##################

```
[java] Hibernate: insert into EVENTS (EVENT_DATE, title, EVENT_ID)
 values (?, ?, ?)
```

###Hibernate##### `INSERT` ## ##########JDBC################# ## `log4j.properties` ############

############################### #######################

```
if (args[0].equals("store")) {
    mgr.createAndStoreEvent("My Event", new Date());
}
else if (args[0].equals("list")) {
    List events = mgr.listEvents();
    for (int i = 0; i < events.size(); i++) {
        Event theEvent = (Event) events.get(i);
        System.out.println("Event: " + theEvent.getTitle() +
                           " Time: " + theEvent.getDate());
    }
}
```

`listEvents()`####

```
private List listEvents() {

    Session session =
 HibernateUtil.getSessionFactory().getCurrentSession();

    session.beginTransaction();

    List result = session.createQuery("from Event").list();

    session.getTransaction().commit();

    return result;
}
```

########################## `Event` ############HQL (Hibernate Query Language) ########### Hibernate####SQL################## ######### `Event` ############## ##HQL##################

##########################

- hbm2ddl## `ant run -Daction=store` ##########
- Now disable hbm2ddl by commenting out the property in your `hibernate.cfg.xml` file. Usually you only leave it turned on in continuous

unit testing, but another run of hbm2ddl would *drop* everything you have stored - the `create` configuration setting actually translates into "drop all tables from the schema, then re-create all tables, when the SessionFactory is build".

`-Daction=list` #####Ant##### ###################### `store`

######Hibernate####################*Table not found* ######### ################# ###############hbm2ddl############################ ## ##############################hbm2ddl###########

1.3. ###2 -

############################ ############################ ###

1.3.1. Person#########

`Person`

```
package events;

public class Person {

    private Long id;
    private int age;
    private String firstname;
    private String lastname;

    public Person() {}

    // Accessor methods for all properties, private setter for 'id'

}
```

`Person.hbm.xml` ######################### #########DTD##################

```
<hibernate-mapping>

    <class name="events.Person" table="PERSON">
        <id name="id" column="PERSON_ID">
            <generator class="native"/>
        </id>
        <property name="age"/>
        <property name="firstname"/>
```

```
        <property name="lastname"/>
    </class>

</hibernate-mapping>
```

###Hibernate######################

```
<mapping resource="events/Event.hbm.xml"/>
<mapping resource="events/Person.hbm.xml"/>
```

#######2################### ##################################
##

1.3.2. ###Set#####

############ Person ##########
#######################aPerson.getEvents()########
################################# ####Java########Set ######
######################################

Set ##################### Java############################

```
public class Person {

    private Set events = new HashSet();

    public Set getEvents() {
        return events;
    }

    public void setEvents(Set events) {
        this.events = events;
    }
}
```

################################ ###########################
Event ####################### ### anEvent.getParticipants()
####################
##
###
########## ####Hibernate#many-to-many###########

```
<class name="events.Person" table="PERSON">
    <id name="id" column="PERSON_ID">
        <generator class="native"/>
    </id>
    <property name="age"/>
    <property name="firstname"/>
    <property name="lastname"/>
```

```
    <set name="events" table="PERSON_EVENT">
        <key column="PERSON_ID"/>
        <many-to-many column="EVENT_ID" class="events.Event"/>
    </set>

</class>
```

Hibernate#################################### #########
`<set>` ### ######### *n:m* ################### ############
################################### ###### `set` ###
`table` ######### ############### `<key>` #### ############
`<many-to-many>` # `column` ######### Hibernate##################
#######################################

#####################################

```
    _____________        __________________
   |             |      |                  |       _____________
   |   EVENTS    |      |   PERSON_EVENT   |      |             |
   |_____________|      |__________________|      |    PERSON   |
   |             |      |                  |      |_____________|
   | *EVENT_ID   | <--> | *EVENT_ID        |      |             |
   |  EVENT_DATE |      | *PERSON_ID       | <--> | *PERSON_ID  |
   |  TITLE      |      |__________________|      |  AGE        |
   |_____________|                                |  FIRSTNAME  |
                                                  |  LASTNAME   |
                                                  |_____________|
```

1.3.3.

`EventManager` ############################

```
private void addPersonToEvent(Long personId, Long eventId) {

    Session session =
 HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    Person aPerson = (Person) session.load(Person.class, personId);
    Event anEvent = (Event) session.load(Event.class, eventId);

    aPerson.getEvents().add(anEvent);

    session.getTransaction().commit();
}
```

`Person` # `Event` ######## ##
`update()` # `save()`

Hibernate## ###
########## ##### ##########date########################
######Hibernate `Session`
########(Unit of Work)###################
Hibernate##################(write-behind)#SQL#######
#######(Unit of Work)##################################
############(Unit of
Work)## ####
`CurrentSessionContext` ####### `thread` ##########

#######(Unit of Work)########################
################################### ###*detached*# ####### `Session`
################# ############################

```
private void addPersonToEvent(Long personId, Long eventId) {

    Session session =
 HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    Person aPerson = (Person) session
            .createQuery("select p from Person p left join fetch
 p.events where p.id = :pid")
            .setParameter("pid", personId)
            .uniqueResult(); // Eager fetch the collection so we can
 use it detached

    Event anEvent = (Event) session.load(Event.class, eventId);

    session.getTransaction().commit();

    // End of first unit of work

    aPerson.getEvents().add(anEvent); // aPerson (and its
 collection) is detached

    // Begin second unit of work

    Session session2 =
 HibernateUtil.getSessionFactory().getCurrentSession();
    session2.beginTransaction();

    session2.update(aPerson); // Reattachment of aPerson

    session2.getTransaction().commit();
}
```

`update` ######################## ###########(Unit of
Work)################ ##
##

##################### ##################################### ################## EventManager ####################### ##################### ################# save() ############# ##

```
else if (args[0].equals("addpersontoevent")) {
    Long eventId = mgr.createAndStoreEvent("My Event", new Date());
    Long personId = mgr.createAndStorePerson("Foo", "Bar");
    mgr.addPersonToEvent(personId, eventId);
    System.out.println("Added person " + personId + " to event " +
 eventId);
}
```

###########2#########2################## ### ########### int # String ########## ########## ## ###### #################### ## #### ###############ID######## ###################### #####################2###### firstname ################ #######JDK################### ####Hibernate################JDK############### ### Address # MonetaryAmount ########################

###################### ## Java#################

1.3.4.

################# Person ############## E##################String ########## ########## Set ###

```
private Set emailAddresses = new HashSet();

public Set getEmailAddresses() {
    return emailAddresses;
}

public void setEmailAddresses(Set emailAddresses) {
    this.emailAddresses = emailAddresses;
}
```

Set

```
<set name="emailAddresses" table="PERSON_EMAIL_ADDR">
    <key column="PERSON_ID"/>
    <element type="string" column="EMAIL_ADDR"/>
</set>
```

################ element ####### Hibernate############################

String ######################
#######(string)#Hibernate###########################
########set ### table #### ######################## key
############################### element ### column ### String
########################

################

```
  _____________        __________________
 |             |      |                  |       _____________
 |   EVENTS    |      |   PERSON_EVENT   |      |             |
  ___________________
 |_____________|      |__________________|      |    PERSON   |
 |                   |
 |             |      |                  |      |_____________|
 | PERSON_EMAIL_ADDR |
 | *EVENT_ID   | <--> | *EVENT_ID        |      |             |
 |___________________|
 |  EVENT_DATE |      | *PERSON_ID       | <--> | *PERSON_ID  | <-->
 |  *PERSON_ID       |
 |  TITLE      |      |__________________|      |  AGE        |
 |  *EMAIL_ADDR      |
 |_____________|                                |  FIRSTNAME  |
 |___________________|
                                                |  LASTNAME   |
                                                |_____________|
```

##
#######E##################### Java#set####################

######################### ####################################
####Java##########

```
private void addEmailToPerson(Long personId, String emailAddress) {

    Session session =
 HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    Person aPerson = (Person) session.load(Person.class, personId);

    // The getEmailAddresses() might trigger a lazy load of the
 collection
    aPerson.getEmailAddresses().add(emailAddress);

    session.getTransaction().commit();
}
```

This time we didn't use a *fetch* query to initialize the collection. Hence, the call to its getter method will trigger an additional select to initialize it, so we can add an element to it. Monitor the SQL log and try to optimize this with an eager fetch.

1.3.5.

################# Java######################
#####################################
######################################
######################### ############################

Event

```
private Set participants = new HashSet();

public Set getParticipants() {
    return participants;
}

public void setParticipants(Set participants) {
    this.participants = participants;
}
```

Event.hbm.xml

```
<set name="participants" table="PERSON_EVENT" inverse="true">
    <key column="EVENT_ID"/>
    <many-to-many column="PERSON_ID" class="events.Person"/>
</set>
```

######################(XML####)###### set ############# key # many-to-many ####### ####################################### ############## Event ############# set ##### inverse="true" #####

#########2##################################
Hibernate############### Person ###############
##2#################################### ##################

1.3.6.

###Hibernate####Java#################################
################## Person # Event #################
Person ######################## Event ###############
#################### ############################
Event ######## Person ## ###############
##

Many developers program defensively and create link management methods to correctly set both sides, e.g. in `Person`:

```
protected Set getEvents() {
    return events;
}

protected void setEvents(Set events) {
    this.events = events;
}

public void addToEvent(Event event) {
    this.getEvents().add(event);
    event.getParticipants().add(this);
}

public void removeFromEvent(Event event) {
    this.getEvents().remove(event);
    event.getParticipants().remove(this);
}
```

#####################protected##################
##
##
##################################

`inverse` #####################
####Java#######################################
###Hibernate##############SQL# `INSERT` # `UPDATE`
############################# ################################
`inverse` #########Hibernate################# #### #
######Hibernate#################SQL###################
################## ####################### ###########
`inverse` ################# #########################
###################################

1.4. ###3 - EventManager Web########

Let's turn the following discussion into a small web application...

Hibernate#Web#################################
`Session` # `Transaction` ####### ########################
`EventManagerServlet`
###############################HTML########################

1.4.1. ####Servlet###

################### `events` ###############

```
package events;

// Imports

public class EventManagerServlet extends HttpServlet {

    // Servlet code
}
```

Servlet#HTTP# `GET` ############### `doGet()` #######

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {

    SimpleDateFormat dateFormatter = new
 SimpleDateFormat("dd.MM.yyyy");

    try {
        // Begin unit of work
        HibernateUtil.getSessionFactory()
                .getCurrentSession().beginTransaction();

        // Process request and render page...

        // End unit of work
        HibernateUtil.getSessionFactory()
                .getCurrentSession().getTransaction().commit();

    } catch (Exception ex) {
        HibernateUtil.getSessionFactory()
                .getCurrentSession().getTransaction().rollback();
        throw new ServletException(ex);
    }

}
```

The pattern we are applying here is called *session-per-request*. When a request hits the servlet, a new Hibernate `Session` is opened through the first call to `getCurrentSession()` on the `SessionFactory`. Then a database transaction is started-all data access as to occur inside a transaction, no matter if data is read or written (we don't use the auto-commit mode in applications).

Do *not* use a new Hibernate `Session` for every database operation. Use one Hibernate `Session` that is scoped to the whole request. Use `getCurrentSession()`, so that it is automatically bound to the current Java thread.

################################HTML######## ################

Finally, the unit of work ends when processing and rendering is complete. If any problem occurred during processing or rendering, an exception will be thrown and the database transaction rolled back. This completes the `session-per-request` pattern. Instead of the transaction demarcation code in every servlet you could also write a servlet filter. See the Hibernate website and Wiki for more information about this pattern, called *Open Session in View*-you'll need it as soon as you consider rendering your view in JSP, not in a servlet.

1.4.2.

########################

```
// Write HTML header
PrintWriter out = response.getWriter();
out.println("<html><head><title>Event
 Manager</title></head><body>");

// Handle actions
if ( "store".equals(request.getParameter("action")) ) {

    String eventTitle = request.getParameter("eventTitle");
    String eventDate = request.getParameter("eventDate");

    if ( "".equals(eventTitle) || "".equals(eventDate) ) {
        out.println("<b><i>Please enter event title and
 date.</i></b>");
    } else {
        createAndStoreEvent(eventTitle,
 dateFormatter.parse(eventDate));
        out.println("<b><i>Added event.</i></b>");
    }
}

// Print page
printEventForm(out);
listEvents(out, dateFormatter);

// Write HTML footer
out.println("</body></html>");
out.flush();
out.close();
```

Granted, this coding style with a mix of Java and HTML would not scale in a more complex application-keep in mind that we are only illustrating basic Hibernate concepts in this tutorial. The code prints an HTML header and a footer. Inside this page, an HTML form for event entry and a list of all events in the database are printed. The first method is trivial and only outputs HTML:

```
private void printEventForm(PrintWriter out) {
```

```
    out.println("<h2>Add new event:</h2>");
    out.println("<form>");
    out.println("Title: <input name='eventTitle'
 length='50'/><br/>");
    out.println("Date (e.g. 24.12.2009): <input name='eventDate'
 length='10'/><br/>");
    out.println("<input type='submit' name='action'
 value='store'/>");
    out.println("</form>");
}
```

`listEvents()` ################## Hibernate# `Session` ################

```
private void listEvents(PrintWriter out, SimpleDateFormat
 dateFormatter) {

    List result = HibernateUtil.getSessionFactory()

 .getCurrentSession().createCriteria(Event.class).list();
    if (result.size() > 0) {
        out.println("<h2>Events in database:</h2>");
        out.println("<table border='1'>");
        out.println("<tr>");
        out.println("<th>Event title</th>");
        out.println("<th>Event date</th>");
        out.println("</tr>");
        for (Iterator it = result.iterator(); it.hasNext();) {
            Event event = (Event) it.next();
            out.println("<tr>");
            out.println("<td>" + event.getTitle() + "</td>");
            out.println("<td>" +
 dateFormatter.format(event.getDate()) + "</td>");
            out.println("</tr>");
        }
        out.println("</table>");
    }
}
```

`store` ###### `createAndStoreEvent()`
######################## `Session` #######

```
protected void createAndStoreEvent(String title, Date theDate) {
    Event theEvent = new Event();
    theEvent.setTitle(title);
    theEvent.setDate(theDate);

    HibernateUtil.getSessionFactory()
                    .getCurrentSession().save(theEvent);
}
```

That's it, the servlet is complete. A request to the servlet will be processed in a single `Session` and `Transaction`. As earlier in the standalone application, Hibernate can automatically bind these objects to the current thread of execution. This gives you the freedom to layer your code and access the `SessionFactory` in any way you like. Usually you'd use a more sophisticated design and move the data access code into data access objects (the DAO pattern). See the Hibernate Wiki for more examples.

1.4.3.

######################Web#######WAR############ ###Ant####### `build.xml` ##########

```
<target name="war" depends="compile">
    <war destfile="hibernate-tutorial.war" webxml="web.xml">
        <lib dir="${librarydir}">
          <exclude name="jsdk*.jar"/>
        </lib>

        <classes dir="${targetdir}"/>
    </war>
</target>
```

######### `hibernate-tutorial.war` ######### ##################### ################### `web.xml` ########### ###################################

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <servlet>
        <servlet-name>Event Manager</servlet-name>
        <servlet-class>events.EventManagerServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Event Manager</servlet-name>
        <url-pattern>/eventmanager</url-pattern>
    </servlet-mapping>
</web-app>
```

Before you compile and deploy the web application, note that an additional library is required: `jsdk.jar`. This is the Java servlet development kit, if you don't have this library already, get it from the Sun website and copy it to your library directory. However, it will be only used for compilation and excluded from the WAR package.

########################### ant war ######
hibernate-tutorial.war #####Tomcat# webapp #################
##Tomcat##
##########################Tomcat#################

########Tomcat#######
http://localhost:8080/hibernate-tutorial/eventmanager #
##################### ############################Tomcat####
Hibernate############### # HibernateUtil #####################
###exception###################

1.5.

#########################Hibernate#########
####Web#########################

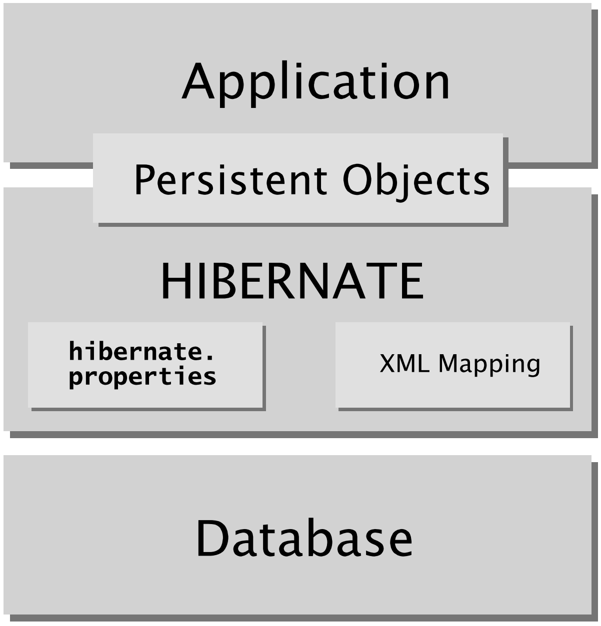
##Hibernate############################### #####################
########################## 11. ############### ############### 19.
############ API###### 10. ############### ##10.4. #########

######################Hibernate#######################

#2#

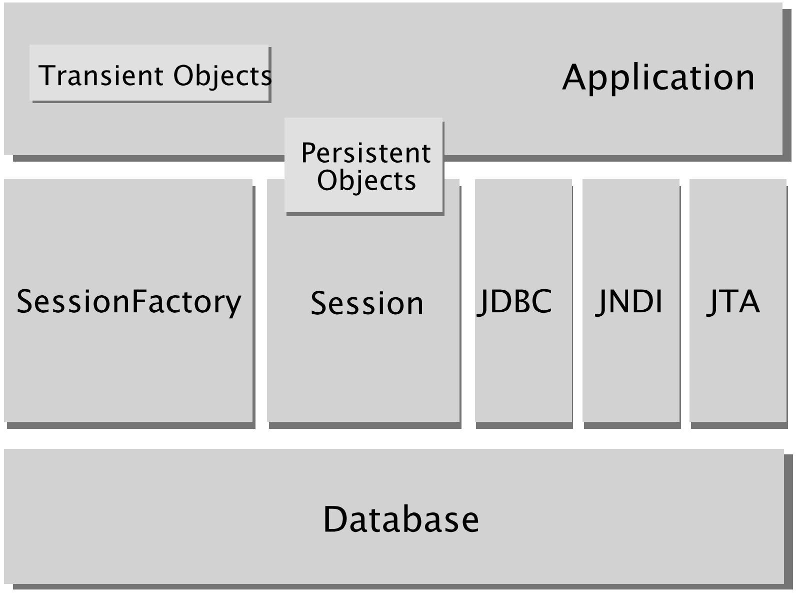
2.1.

Hibernate########################

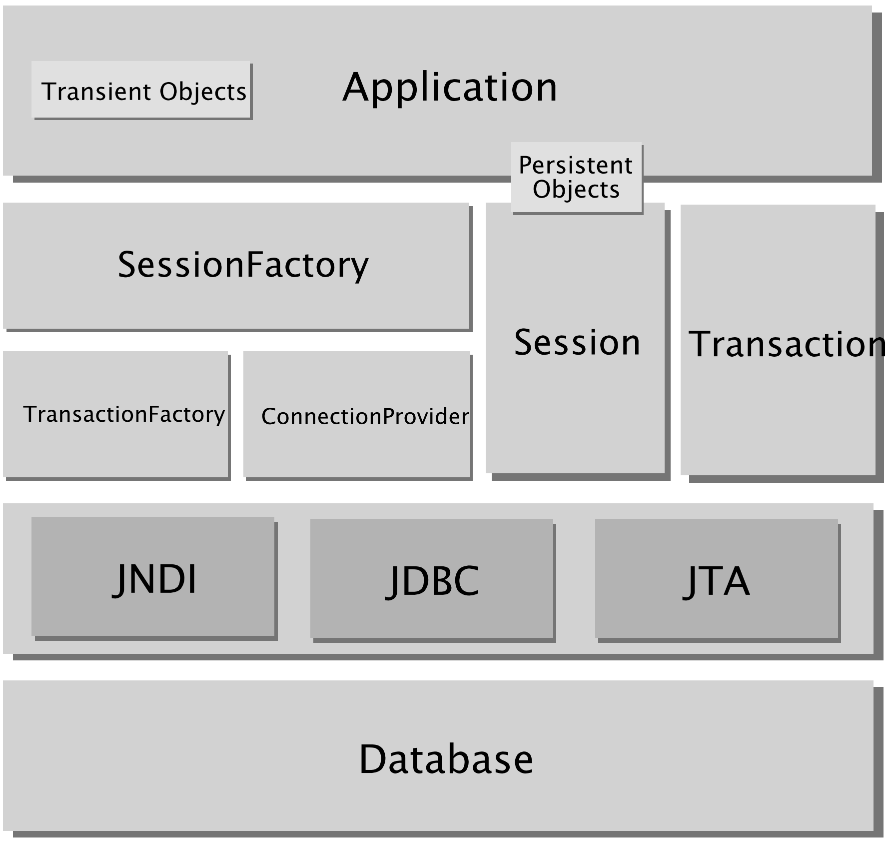


####Hibernate####################
#######

################################
#####Hibernate############################ #####2##############
###########################JDBC###########
########################### ######Hibernate API################



##################################JDBC#JTA#API#
##################Hibernate#######



Heres some definitions of the objects in the diagrams:

SessionFactory (`org.hibernate.SessionFactory`)
: 1########################### ######################
`Session` ########## `ConnectionProvider`
######## #############################
####################################

Session (`org.hibernate.Session`)
: ####################### ####################
JDBC############## `Transaction` #########
############################# ###############################
########################

Persistent objects # Collections
: ######################################
######JavaBeans/POJO##################
#########1### `Session` ############ `Session`
############# #####################################
##########################
######################################

Transient # detached # objects # Collections
: ##### `Session` ######### ##############
`Session`
#####################

Transaction (`org.hibernate.Transaction`)
: (#####)###########(Unit of Work)########################
####################
######JDBC#JTA#CORBA########################### `Session`
`Transaction` ############## ########API#######
`Transaction` ####### ######################################

ConnectionProvider (`org.hibernate.connection.ConnectionProvider`)
: (#####)JDBC###################### ########
`Datasource` # `DriverManager` ###################
##

TransactionFactory (`org.hibernate.TransactionFactory`)
: (#####) `Transaction` ##############
##

Extension Interfaces
: Hibernate######################## ###########################
###API#################

##########################JTA#JDBC######### `Transaction` # `TransactionFactory` # `ConnectionProvider` #########

2.2.

#################################### #### ######## ##########
Hibernate# `Session` #######################

transient
: #############################
  #########################ID######## ########

persistent
: ####################################### #####ID###########
  ############################## ####################ID#
  Java#ID########################### Hibernate# ## ####

detached
: The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process. It has a persistent identity and, perhaps, a corresponding row in the database. For detached instances, Hibernate makes no guarantees about the relationship between persistent identity and Java identity.

2.3. JMX####

JMX#Java##########J2EE##### JMX###########Hibernate######## ############### `org.hibernate.jmx.HibernateService` ### MBean###########

JBoss ###############Hibernate#JMX####################### JBoss ################# JBoss ################## JMX###########################

- *#######:* Hibernate# `Session` ########## ####JTA#################################### `Session` ################################### ####JBoss EJB ################# ################################# ### ######Hibernate# `Transaction` ############ `Session` ############# `HibernateContext` # ##########
- *HAR ####:* ###(EAR ### SAR ########JBoss #### ####################Hibernate JMX ############## ####Hibernate# `SessionFactory` ################# ### ############### #########HAR#################### JBoss#####HAR#############################

#####################JBoss################## ##########

JMX##################Hibernate############# #3.4.6. #Hibernate ### ########

2.4. JCA

Hibernate # JCA #######################Web########### Hibernate JCA #####################################

2.5.

Hibernate ##########################"########"############### ## ## ### #####3.0#### Hibernate ###### `ThreadLocal` ##################### ###### `HibernateUtil` ################### proxy/interception ######################### #Spring # Pico ####################################

3.0.1 ###Hibernate ## `SessionFactory.getCurrentSession()` # ####### #### `JTA` ###################### `JTA` ## Hibernate ################# ############ `JTA TransactionManager` ######### `J2EE` ######################### ############################ `JTA` ###################### ########### `JTA` #####################

######### 3.1 #### `SessionFactory.getCurrentSession()` ####### ############# ### ############# (`org.hibernate.context.CurrentSessionContext`) # ########### (`hibernate.current_session_context_class`) #########

`org.hibernate.context.CurrentSessionContext` ################ ###### Javadoc ########## ##### `currentSession()` ###1################ ### ###########Hibernate##############2#########

- `org.hibernate.context.JTASessionContext` - `JTA` ############################ ####################JTA################### ###Javadoc##########

- `org.hibernate.context.ThreadLocalSessionContext` - ############################# ###Javadoc##########

- `org.hibernate.context.ManagedSessionContext` - ############################# ##########static##### `Session` ####### ####/##################### ###### `Session` ######################

The first two implementations provide a "one session - one database transaction" programming model, also known and used as *session-per-request*. The beginning and end of a Hibernate session is defined by the duration of a database transaction. If you use programmatic transaction demarcation in plain JSE without JTA, you are advised to use the Hibernate `Transaction` API to hide the underlying transaction system from your code. If you use JTA, use the JTA interfaces to demarcate transactions. If you execute in an EJB container that supports CMT, transaction boundaries are defined declaratively and you don't need any transaction or session demarcation operations in your code. Refer to # 11. ############ for more information and code examples.

`hibernate.current_session_context_class` ######### `org.hibernate.context.CurrentSessionContext` ################# ####################### `org.hibernate.transaction.TransactionManagerLookup` ########### Hibernate# `org.hibernate.context.JTASessionContext` ################ ##############3############################### ####"jta", "thread", "managed"######################

#3#

Hibernate## #######Hibernate############### etc/ ##### hibernate.properties ################################## ## hibernate.properties ################################

3.1.

An instance of org.hibernate.cfg.Configuration represents an entire set of mappings of an application's Java types to an SQL database. The org.hibernate.cfg.Configuration is used to build an (immutable) org.hibernate.SessionFactory. The mappings are compiled from various XML mapping files.

You may obtain a org.hibernate.cfg.Configuration instance by instantiating it directly and specifying XML mapping documents. If the mapping files are in the classpath, use addResource():

```
Configuration cfg = new Configuration()
    .addResource("Item.hbm.xml")
    .addResource("Bid.hbm.xml");
```

###(################)########################Hibernate############ ############

```
Configuration cfg = new Configuration()
    .addClass(org.hibernate.auction.Item.class)
    .addClass(org.hibernate.auction.Bid.class);
```

Then Hibernate will look for mapping files named /org/hibernate/auction/Item.hbm.xml and /org/hibernate/auction/Bid.hbm.xml in the classpath. This approach eliminates any hardcoded filenames.

A org.hibernate.cfg.Configuration also allows you to specify configuration properties:

```
Configuration cfg = new Configuration()
    .addClass(org.hibernate.auction.Item.class)
    .addClass(org.hibernate.auction.Bid.class)
    .setProperty("hibernate.dialect",
 "org.hibernate.dialect.MySQLInnoDBDialect")
    .setProperty("hibernate.connection.datasource",
 "java:comp/env/jdbc/test")
    .setProperty("hibernate.order_updates", "true");
```

Hibernate##############1######### ####################

1. Pass an instance of `java.util.Properties` to `Configuration.setProperties()`.
2. Place a file named `hibernate.properties` in a root directory of the classpath.
3. `System` ###### `java -Dproperty=value` ############
4. `<property>` ### `hibernate.cfg.xml` ###########

`hibernate.properties` is the easiest approach if you want to get started quickly.

The `org.hibernate.cfg.Configuration` is intended as a startup-time object, to be discarded once a `SessionFactory` is created.

3.2. SessionFactory#####

When all mappings have been parsed by the `org.hibernate.cfg.Configuration`, the application must obtain a factory for `org.hibernate.Session` instances. This factory is intended to be shared by all application threads:

```
SessionFactory sessions = cfg.buildSessionFactory();
```

Hibernate does allow your application to instantiate more than one `org.hibernate.SessionFactory`. This is useful if you are using more than one database.

3.3. JDBC######

Usually, you want to have the `org.hibernate.SessionFactory` create and pool JDBC connections for you. If you take this approach, opening a `org.hibernate.Session` is as simple as:

```
Session session = sessions.openSession(); // open a new Session
```

############JDBC################## ###############

For this to work, we need to pass some JDBC connection properties to Hibernate. All Hibernate property names and semantics are defined on the class `org.hibernate.cfg.Environment`. We will now describe the most important settings for JDBC connection configuration.

Hibernate will obtain (and pool) connections using `java.sql.DriverManager` if you set the following properties:

3.1. Hibernate JDBC#####

######	##
hibernate.connection.driver_class	*JDBC#######*
hibernate.connection.url	*jdbc URL*
hibernate.connection.username	*database user*
hibernate.connection.password	*database user password*
hibernate.connection.pool_size	*###############*

Hibernate's own connection pooling algorithm is however quite rudimentary. It is intended to help you get started and is *not intended for use in a production system* or even for performance testing. You should use a third party pool for best performance and stability. Just replace the hibernate.connection.pool_size property with connection pool specific settings. This will turn off Hibernate's internal pool. For example, you might like to use C3P0.

C3P0 is an open source JDBC connection pool distributed along with Hibernate in the `lib` directory. Hibernate will use its `org.hibernate.connection.C3P0ConnectionProvider` for connection pooling if you set hibernate.c3p0.* properties. If you'd like to use Proxool refer to the packaged `hibernate.properties` and the Hibernate web site for more information.

Here is an example `hibernate.properties` file for C3P0:

```
hibernate.connection.driver_class = org.postgresql.Driver
hibernate.connection.url = jdbc:postgresql://localhost/mydatabase
hibernate.connection.username = myuser
hibernate.connection.password = secret
hibernate.c3p0.min_size=5
hibernate.c3p0.max_size=20
hibernate.c3p0.timeout=1800
hibernate.c3p0.max_statements=50
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

For use inside an application server, you should almost always configure Hibernate to obtain connections from an application server `javax.sql.Datasource` registered in JNDI. You'll need to set at least one of the following properties:

3.2. Hibernate

######	##
hibernate.connection.datasource	*#######JNDI#*
hibernate.jndi.url	*URL of the JNDI provider* (optional)
hibernate.jndi.class	*class of the JNDI* *`InitialContextFactory`* (optional)
hibernate.connection.username	*database user* (optional)
hibernate.connection.password	*database user password* (optional)

Here's an example `hibernate.properties` file for an application server provided JNDI datasource:

```
hibernate.connection.datasource = java:/comp/env/jdbc/test
hibernate.transaction.factory_class = \
    org.hibernate.transaction.JTATransactionFactory
hibernate.transaction.manager_lookup_class = \
    org.hibernate.transaction.JBossTransactionManagerLookup
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
```

JNDI############JDBC####################
##########################

Arbitrary connection properties may be given by prepending "`hibernate.connection`" to the connection property name. For example, you may specify a charSet connection property using hibernate.connection.charSet.

You may define your own plugin strategy for obtaining JDBC connections by implementing the interface `org.hibernate.connection.ConnectionProvider`, and specifying your custom implementation via the hibernate.connection.provider_class property.

3.4.

##########Hibernate#############
####################################

Warning: some of these properties are "system-level" only. System-level properties can be set only via `java -Dproperty=value` or `hibernate.properties`. They may *not* be set by the other techniques described above.

	eg. `SCHEMA_NAME`
hibernate.default_catalog	Qualify unqualified table names with the given catalog in generated SQL. **eg.** `CATALOG_NAME`
hibernate.session_factory_name	The `org.hibernate.SessionFactory` will be automatically bound to this name in JNDI after it has been created. **eg.** `jndi/composite/name`
hibernate.max_fetch_depth	Set a maximum "depth" for the outer join fetch tree for single-ended associations (one-to-one, many-to-one). A `0` disables default outer join fetching. **##** ###### `0` ## `3` ###
hibernate.default_batch_fetch_size	Set a default size for Hibernate batch fetching of associations. **##** ###### `4` , `8` , `16` ###
hibernate.default_entity_mode	Set a default mode for entity representation for all sessions opened from this `SessionFactory` `dynamic-map`, `dom4j`, `pojo`
hibernate.order_updates	Force Hibernate to order SQL updates by the primary key value of the items being updated. This will result in fewer transaction deadlocks in highly concurrent systems. **eg.** `true` | `false`
hibernate.generate_statistics	If enabled, Hibernate will collect statistics useful for performance tuning. **eg.** `true` | `false`
hibernate.use_identifer_rollback	If enabled, generated identifier properties will be reset to default values when objects are deleted. **eg.** `true` | `false`
hibernate.use_sql_comments	If turned on, Hibernate will generate comments inside the SQL, for easier debugging, defaults to `false`. **eg.** `true` | `false`

3.4. Hibernate JDBC

	eg. `classname.of.ConnectionProvider`
hibernate.connection.isolation	Set the JDBC transaction isolation level. Check `java.sql.Connection` for meaningful values but note that most databases do not support all isolation levels and some define additional, non-standard isolations. **eg.** `1, 2, 4, 8`
hibernate.connection.autocommit	Enables autocommit for JDBC pooled connections (not recommended). **eg.** `true` | `false`
hibernate.connection.release_mode	Specify when Hibernate should release JDBC connections. By default, a JDBC connection is held until the session is explicitly closed or disconnected. For an application server JTA datasource, you should use `after_statement` to aggressively release connections after every JDBC call. For a non-JTA connection, it often makes sense to release the connection at the end of each transaction, by using `after_transaction`. `auto` will choose `after_statement` for the JTA and CMT transaction strategies and `after_transaction` for the JDBC transaction strategy. **eg.** `auto` (default) | `on_close` | `after_transaction` | `after_statement` Note that this setting only affects `Session`s returned from `SessionFactory.openSession`. For `Session`s obtained through `SessionFactory.getCurrentSession`, the `CurrentSessionContext` implementation configured for use controls the connection release mode for those `Session`s. See #2.5. ################
hibernate.connection.*<propertyName>*	Pass the JDBC property *<propertyName>* to `DriverManager.getConnection()`.
hibernate.jndi.*<propertyName>*	Pass the property *<propertyName>* to the JNDI `InitialContextFactory`.

3.5. Hibernate

######	##
hibernate.cache.provider_class	The classname of a custom CacheProvider. **eg.** classname.of.CacheProvider
hibernate.cache.use_minimal_puts	Optimize second-level cache operation to minimize writes, at the cost of more frequent reads. This setting is most useful for clustered caches and, in Hibernate3, is enabled by default for clustered cache implementations. **eg.** true|false
hibernate.cache.use_query_cache	Enable the query cache, individual queries still have to be set cachable. **eg.** true|false
hibernate.cache.use_second_level_cache	May be used to completely disable the second level cache, which is enabled by default for classes which specify a <cache> mapping. **eg.** true|false
hibernate.cache.query_cache_factory	The classname of a custom QueryCache interface, defaults to the built-in StandardQueryCache. **eg.** classname.of.QueryCache
hibernate.cache.region_prefix	A prefix to use for second-level cache region names. **eg.** prefix
hibernate.cache.use_structured_entries	Forces Hibernate to store data in the second-level cache in a more human-friendly format. **eg.** true|false

3.6. Hibernate

######	##
hibernate.transaction.factory_class	The classname of a TransactionFactory to use with Hibernate Transaction API (defaults to JDBCTransactionFactory). **eg.** classname.of.TransactionFactory
jta.UserTransaction	A JNDI name used by JTATransactionFactory to obtain the JTA UserTransaction from the application server. **eg.** jndi/composite/name
hibernate.transaction.manager_lookup_class	The classname of a TransactionManagerLookup - required when JVM-level caching is enabled or when using hilo generator in a JTA environment. **eg.** classname.of.TransactionManagerLookup
hibernate.transaction.flush_before_completion	If enabled, the session will be automatically flushed during the before completion phase of the transaction. Built-in and automatic session context management is preferred, see #2.5. ################. **eg.** true | false
hibernate.transaction.auto_close_session	If enabled, the session will be automatically closed during the after completion phase of the transaction. Built-in and utomatic session context management is preferred, see #2.5. ################. **eg.** true | false

3.7.

######	##
hibernate.current_session_context_class	Supply a (custom) strategy for the scoping of the "current" Session. See #2.5. ############### for more information about the built-in strategies. **eg.** jta | thread | managed | custom.Class
hibernate.query.factory_class	Chooses the HQL parser implementation. **eg.** org.hibernate.hql.ast.ASTQueryTranslatorFactory or org.hibernate.hql.classic.ClassicQueryTranslatorF
hibernate.query.substitutions	Mapping from tokens in Hibernate queries to SQL tokens (tokens might be function or literal names, for example). **eg.** hqlLiteral=SQL_LITERAL, hqlFunction=SQLFUNC
hibernate.hbm2ddl.auto	Automatically validate or export schema DDL to the database when the SessionFactory is created. With create-drop, the database schema will be dropped when the SessionFactory is closed explicitly. **eg.** validate | update | create | create-drop
hibernate.cglib.use_reflection_optimizer	Enables use of CGLIB instead of runtime reflection (System-level property). Reflection can sometimes be useful when troubleshooting, note that Hibernate always requires CGLIB even if you turn off the optimizer. You can not set this property in hibernate.cfg.xml. **eg.** true | false

3.4.1. SQL ###Dialect#

`hibernate.dialect` ######## ############## `org.hibernate.dialect.Dialect` ######## ########## ###########Hibernate################### ################# #########################

3.8. Hibernate SQL Dialects (`hibernate.dialect`)

RDBMS	Dialect
DB2	`org.hibernate.dialect.DB2Dialect`
DB2 AS/400	`org.hibernate.dialect.DB2400Dialect`
DB2 OS390	`org.hibernate.dialect.DB2390Dialect`
PostgreSQL	`org.hibernate.dialect.PostgreSQLDialect`
MySQL	`org.hibernate.dialect.MySQLDialect`
MySQL with InnoDB	`org.hibernate.dialect.MySQLInnoDBDialect`
MySQL with MyISAM	`org.hibernate.dialect.MySQLMyISAMDialect`
Oracle (any version)	`org.hibernate.dialect.OracleDialect`
Oracle 9i/10g	`org.hibernate.dialect.Oracle9Dialect`
Sybase	`org.hibernate.dialect.SybaseDialect`
Sybase Anywhere	`org.hibernate.dialect.SybaseAnywhereDialect`
Microsoft SQL Server	`org.hibernate.dialect.SQLServerDialect`
SAP DB	`org.hibernate.dialect.SAPDBDialect`
Informix	`org.hibernate.dialect.InformixDialect`
HypersonicSQL	`org.hibernate.dialect.HSQLDialect`
Ingres	`org.hibernate.dialect.IngresDialect`
Progress	`org.hibernate.dialect.ProgressDialect`
Mckoi SQL	`org.hibernate.dialect.MckoiDialect`
Interbase	`org.hibernate.dialect.InterbaseDialect`
Pointbase	`org.hibernate.dialect.PointbaseDialect`
FrontBase	`org.hibernate.dialect.FrontbaseDialect`
Firebird	`org.hibernate.dialect.FirebirdDialect`

3.4.2.

##DB#ANSI##Oracle#Sybase##################### *outer join fetching* ##DB#SQL######## ###############DB###################

################################### ###########1##SQL# SELECT ####

hibernate.max_fetch_depth ######## 0 #### Outer join fetching# ### ############# 1 ############################# ########## fetch="join" #############

See #19.1. ######## for more information.

3.4.3.

Oracle#JDBC############### byte ############# binary # serializable ################### hibernate.jdbc.use_streams_for_binary ########### ### ################## #

3.4.4. 2#################

hibernate.cache ######### Hibernate################################ #19.2. ##2########################

3.4.5.

hibernate.query.substitutions ####### ###Hibernate############### ##

```
hibernate.query.substitutions true=1, false=0
```

####### true # false ## #####SQL#################

```
hibernate.query.substitutions toLowercase=LOWER
```

###SQL# LOWER ##################

3.4.6. Hibernate

hibernate.generate_statistics ######### ########################SessionFactory.getStatistics() ######Hibernate############### JMX###################### Javadoc# org.hibernate.stats ####### ########################

3.5.

Hibernate utilizes Simple Logging Facade for Java [http://www.slf4j.org/] (SLF4J) in order to log various system events. SLF4J can direct your logging output to several logging frameworks (NOP, Simple, log4j version 1.2, JDK 1.4 logging, JCL or logback) depending on your chosen binding. In order to setup logging properly you will need slf4j-api.jar in your classpath together with the jar file for your preferred binding - slf4j-log4j12.jar in the case of

Log4J. See the SLF4J documentation [http://www.slf4j.org/manual.html] for more detail. To use Log4j you will also need to place a `log4j.properties` file in your classpath, an example properties file is distributed with Hibernate in the `src/` directory.

Hibernate########################## Hibernate################################## ######################### ####################

3.9. Hibernate

####	##
org.hibernate.SQL	########SQL#DDL#################
org.hibernate.type	####JDBC###############
org.hibernate.tool.hbm2ddl	########SQL#DDL#################
org.hibernate.pretty	session##
org.hibernate.cache	########################
org.hibernate.transaction	##########################
org.hibernate.jdbc	JDBC################
org.hibernate.hql.ast.AST	HQL#SQL#AST##################
org.hibernate.secure	####JAAS############
org.hibernate	###

Hibernate##################`org.hibernate.SQL` ##### `debug` ###################### ########`hibernate.show_sql` ###############

3.6. `NamingStrategy`

######## `net.sf.hibernate.cfg.NamingStrategy` #### #######################################

Java################################# ################################ ### ################# `TBL_` ############### ############################## Hibernate############################

############ `Configuration.setNamingStrategy()` ###### #########################

```
SessionFactory sf = new Configuration()
    .setNamingStrategy(ImprovedNamingStrategy.INSTANCE)
    .addFile("Item.hbm.xml")
```

```
    .addFile("Bid.hbm.xml")
    .buildSessionFactory();
```

org.hibernate.cfg.ImprovedNamingStrategy ###########
###

3.7. XML######

##1##### hibernate.cfg.xml ########### ############### #######
hibernate.properties ############### ##############################

XML############ CLASSPATH ########## #######

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"

 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <!-- a SessionFactory instance listed as /jndi/name -->
    <session-factory
        name="java:hibernate/SessionFactory">

        <!-- properties -->
        <property
 name="connection.datasource">java:/comp/env/jdbc/MyDB</property>
        <property
 name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">false</property>
        <property name="transaction.factory_class">
            org.hibernate.transaction.JTATransactionFactory
        </property>
        <property
 name="jta.UserTransaction">java:comp/UserTransaction</property>

        <!-- mapping files -->
        <mapping resource="org/hibernate/auction/Item.hbm.xml"/>
        <mapping resource="org/hibernate/auction/Bid.hbm.xml"/>

        <!-- cache settings -->
        <class-cache class="org.hibernate.auction.Item"
 usage="read-write"/>
        <class-cache class="org.hibernate.auction.Bid"
 usage="read-only"/>
        <collection-cache
 collection="org.hibernate.auction.Item.bids" usage="read-write"/>

    </session-factory>

</hibernate-configuration>
```

##
Hibernate############################ hibernate.cfg.xml ########
hibernate.properties # hibernate.cfg.xml # #######################
###################XML################

XML#########Hibernate#################

```
SessionFactory sf = new
 Configuration().configure().buildSessionFactory();
```

##XML##################

```
SessionFactory sf = new Configuration()
    .configure("catdb.cfg.xml")
    .buildSessionFactory();
```

3.8. J2EE################

Hibernate#J2EE######################

- #############Hibernate# JNDI##############JDBC##############
###JTA### TransactionManager # ResourceManager
###########(CMT)# ##################################
##########################(BMT)# ############################
######Hibernate# Transaction API################

- ##*JNDI*########Hibernate#JNDI######### SessionFactory #######

- *JTA*############ # Hibernate Session
############JTA################## ###
SessionFactory #JNDI##lookup### ### Session #######
JTA#################Hibernate# Session #################
EJB#####################################

- *JMX####:* ##JMX######################JBOSS#
######Hibernate#MBean################## ### Configuration ##
SessionFactory # #################### ##### HibernateService
##########################Hibernate###########
#####################

##########################"connection containment"###########
######## hibernate.connection.aggressive_release #true########

3.8.1.

Hibernate Session API###################### ######################
############JDBC##########JDBC API## #####################

###J2EE####################Bean############## ###### `UserTransaction` #JTA API##############

2## #######Hibernate `Transaction` API####### Hibernate######## `hibernate.transaction.factory_class` ######## ##### `Transaction` #####################

3####################

`org.hibernate.transaction.JDBCTransactionFactory`
: ######(JDBC)#####################

`org.hibernate.transaction.JTATransactionFactory`
: ################EJB#####Bean############################ ####################Bean#################### #####################

`org.hibernate.transaction.CMTTransactionFactory`
: ######JTA##############

###################CORBA#########################

Hibernate#####################JTA############## ### ##########JTA `TransactionManager` ############# J2EE####################### #################Hibernate# `TransactionManager` ####### #####################

3.10. JTA

Transaction Factory	Application Server
`org.hibernate.transaction.JBossTransactionManagerLookup`	JBoss
`org.hibernate.transaction.WeblogicTransactionManagerLookup`	Weblogic
`org.hibernate.transaction.WebSphereTransactionManagerLookup`	WebSphere
`org.hibernate.transaction.WebSphereExtendedJTATransactionLookup`	WebSphere 6
`org.hibernate.transaction.OrionTransactionManagerLookup`	Orion
`org.hibernate.transaction.ResinTransactionManagerLookup`	Resin
`org.hibernate.transaction.JOTMTransactionManagerLookup`	JOTM
`org.hibernate.transaction.JOnASTransactionManagerLookup`	JOnAS
`org.hibernate.transaction.JRun4TransactionManagerLookup`	JRun4
`org.hibernate.transaction.BESTransactionManagerLookup`	Borland ES

3.8.2. SessionFactory #JNDI####

JNDI#####Hibernate SessionFactory #### ################# Session ###### ###JNDI###### Datasource ####### ###############################

SessionFactory #JNDI################ ########### java:hibernate/SessionFactory ## hibernate.session_factory_name ############## ################## SessionFactory # JNDI#############Tomcat############JNDI######################

SessionFactory #JNDI########Hibernate# hibernate.jndi.url #######hibernate.jndi.class ####################### #################### InitialContext #######

cfg.buildSessionFactory() ##### Hibernate##### SessionFactory #JNDI####### HibernateService ####JMX################ ## ###############################

##JNDI SessionFactory ######EJB####### JNDI########## SessionFactory #######

######### SessionFactory #JNDI####### ####### static ################### ############################### HibernateUtil.getSessionFactory() ################ SessionFactory ######################### #############Hibernate################# ############

3.8.3. JTA#########################

####### Session ################# Hibernate########## Session ########## #2.5. ################ ############## ## #JTA# ################### ###JTA#########Hibernate Session ############ ### sessionFactory.getCurrentSession() ###### JTA######################## #JTA# ####### getCurrentSession() # ####### Session ################## ############################### #############JDBC################### #####JTA################## Session ########### ################################# UserTransaction ####JTA#################### ####################Hibernate Transaction API# ######################## EJB###########CMT#########################

3.8.4. JMX#######

SessionFactory #JNDI########## cfg.buildSessionFactory() ###################### ########static ######### HibernateUtil ###### *managed service* ###Hibernate#####################

JBOSS####JMX######################### org.hibernate.jmx.HibernateService ########### ######################## #######JBOSS 4.0.x## jboss-service.xml ######

```
<?xml version="1.0"?>
<server>

<mbean code="org.hibernate.jmx.HibernateService"
    name="jboss.jca:service=HibernateFactory,name=HibernateFactory">

    <!-- Required services -->
    <depends>jboss.jca:service=RARDeployer</depends>
    <depends>jboss.jca:service=LocalTxCM,name=HsqlDS</depends>

    <!-- Bind the Hibernate service to JNDI -->
    <attribute
 name="JndiName">java:/hibernate/SessionFactory</attribute>

    <!-- Datasource settings -->
    <attribute name="Datasource">java:HsqlDS</attribute>
    <attribute
 name="Dialect">org.hibernate.dialect.HSQLDialect</attribute>

    <!-- Transaction integration -->
    <attribute name="TransactionStrategy">
        org.hibernate.transaction.JTATransactionFactory</attribute>
    <attribute name="TransactionManagerLookupStrategy">

 org.hibernate.transaction.JBossTransactionManagerLookup</attribute>
    <attribute name="FlushBeforeCompletionEnabled">true</attribute>
    <attribute name="AutoCloseSessionEnabled">true</attribute>

    <!-- Fetching options -->
    <attribute name="MaximumFetchDepth">5</attribute>

    <!-- Second-level caching -->
    <attribute name="SecondLevelCacheEnabled">true</attribute>
    <attribute
 name="CacheProviderClass">org.hibernate.cache.EhCacheProvider</
attribute>
    <attribute name="QueryCacheEnabled">true</attribute>

    <!-- Logging -->
    <attribute name="ShowSqlEnabled">true</attribute>
```

```
    <!-- Mapping files -->
    <attribute
 name="MapResources">auction/Item.hbm.xml,auction/Category.hbm.xml</
attribute>

</mbean>

</server>
```

####### `META-INF` ############ JAR#########
`.sar` (service archive)############
###Hibernate##############Hibernate#####################
####################################.sar###########
########bean#########bean#####JAR############
1##EJB
JAR############# ########JBoss###################JXM#####
EJB#####################

#4#

###########################E###################
#################### ##
#############transient###########detached################

Plain Old Java Object (POJO)#################
#################Hibernate########## ########################
##Hibernate3##############################
######################## ### Map #####################

4.1. ###POJO##

#####Java################################

```
package eg;
import java.util.Set;
import java.util.Date;

public class Cat {
    private Long id; // identifier

    private Date birthdate;
    private Color color;
    private char sex;
    private float weight;
    private int litterId;

    private Cat mother;
    private Set kittens = new HashSet();

    private void setId(Long id) {
        this.id=id;
    }
    public Long getId() {
        return id;
    }

    void setBirthdate(Date date) {
        birthdate = date;
    }
    public Date getBirthdate() {
        return birthdate;
    }

    void setWeight(float weight) {
        this.weight = weight;
    }
    public float getWeight() {
        return weight;
    }
```

```
    public Color getColor() {
        return color;
    }
    void setColor(Color color) {
        this.color = color;
    }

    void setSex(char sex) {
        this.sex=sex;
    }
    public char getSex() {
        return sex;
    }

    void setLitterId(int id) {
        this.litterId = id;
    }
    public int getLitterId() {
        return litterId;
    }

    void setMother(Cat mother) {
        this.mother = mother;
    }
    public Cat getMother() {
        return mother;
    }
    void setKittens(Set kittens) {
        this.kittens = kittens;
    }
    public Set getKittens() {
        return kittens;
    }

    // addKitten not needed by Hibernate
    public void addKitten(Cat kitten) {
            kitten.setMother(this);
        kitten.setLitterId( kittens.size() );
        kittens.add(kitten);
    }
}
```

####4###########

4.1.1.

`Cat` #################### Hibernate# `Constructor.newInstance()` ########## ## #public###################### Hibernate#################### *package* ##############################

4.1.2.

`Cat` ## `id` ############## #####################################
#####################################
#################`java.lang.String` # `java.util.Date`
######## #############################
##################################### ##################

#####################
########Hibernate############################# #############

##

- Transitive reattachment for detached objects (cascade update or cascade merge) - see #10.11. #########
- `Session.saveOrUpdate()`
- `Session.merge()`

######################################
###null#####(##############)###############

4.1.3. final###############

Hibernate########## #### ##
######final################public#######
##########################

Hibernate################# `final` ##################
###############################
#################################

final######## `public final` #################### `public final` ################## `lazy="false"` ############################

4.1.4.

`Cat` ##################################
#####ORM########################
###
########Hibernate#JavaBean################
`getFoo`, `isFoo`, `setFoo` ###############
###

######public######## ##### # Hibernate#######`protected` #### `private` #get / set#########################

4.2.

######1###2################### ############## Cat ##################

```
package eg;

public class DomesticCat extends Cat {
        private String name;

        public String getName() {
                return name;
        }
        protected void setName(String name) {
                this.name=name;
        }
}
```

4.3. equals() # hashCode()

######### equals() # hashCode() #######################

- ############# Set ###### ######################### ######
- ########################

Hibernate####ID########################## #####JavaID################ ################################### Set ############################## ###equals() # hashCode() ###############

################################# equals()# hashCode() ########## ## #### Set ######## Set ##1################ ###################################### Hibernate################################ #################################### ######################## Set ####### ################################# ## equals() # hashCode() ################# ############### Set ########## #################Hibernate############### ###Hibernate############################## ###Java########################

########## ##### equals() # hashCode() ################ #############equals() ############## ############################## ##### ###########################

```
public class Cat {
```

```
    ...
    public boolean equals(Object other) {
        if (this == other) return true;
        if ( !(other instanceof Cat) ) return false;

        final Cat cat = (Cat) other;

        if ( !cat.getLitterId().equals( getLitterId() ) ) return
 false;
        if ( !cat.getMother().equals( getMother() ) ) return false;

        return true;
    }

    public int hashCode() {
        int result;
        result = getMother().hashCode();
        result = 29 * result + getLitterId();
        return result;
    }

}
```

##
##11.1.3. ############### #########
#######################################

4.4.

######################## ###########################

##################POJO####JavaBean####################
Hibernate###### `Map` # `Map` ###########
DOM4J########################
###

########Hibernate####POJO########## `default_entity_mode`
############ ### `SessionFactory` ##################################
3.3. #Hibernate########

`Map` #############
`entity-name` ##############

```
<hibernate-mapping>

    <class entity-name="Customer">

        <id name="id"
            type="long"
            column="ID">
```

```
            <generator class="sequence"/>
        </id>

        <property name="name"
            column="NAME"
            type="string"/>

        <property name="address"
            column="ADDRESS"
            type="string"/>

        <many-to-one name="organization"
            column="ORGANIZATION_ID"
            class="Organization"/>

        <bag name="orders"
            inverse="true"
            lazy="false"
            cascade="all">
            <key column="CUSTOMER_ID"/>
            <one-to-many class="Order"/>
        </bag>

    </class>

</hibernate-mapping>
```

################################
##########POJO##################################

SessionFactory ##################### dynamic-map ####### #### Map # Map ###########

```
Session s = openSession();
Transaction tx = s.beginTransaction();
Session s = openSession();

// Create a customer
Map david = new HashMap();
david.put("name", "David");

// Create an organization
Map foobar = new HashMap();
foobar.put("name", "Foobar Inc.");

// Link both
david.put("organization", foobar);

// Save both
s.save("Customer", david);
s.save("Organization", foobar);

tx.commit();
```

```
s.close();
```

####################################
###################################
###
Hibernate######################################
################################

############ Session ###############

```
Session dynamicSession = pojoSession.getSession(EntityMode.MAP);

// Create a customer
Map david = new HashMap();
david.put("name", "David");
dynamicSession.save("Customer", david);
...
dynamicSession.flush();
dynamicSession.close()
...
// Continue on pojoSession
```

`EntityMode` #### `getSession()` ###### `SessionFactory` #### `Session` API############### ########## `Session` ########JDBC####### ############################ ###2### `Session` ## `flush()` # `close()` ###################################### 1#######(Unit of Work)##################

XML################# # 18. *XML##### ########*

4.5. Tuplizer

`org.hibernate.tuple.Tuplizer` ############### ### `org.hibernate.EntityMode` ###### ####################### #########################Tuplizer############ ###################################### ############# ###POJO################Tuplizer############# POJO################################ POJO######################## Tuplizer################## #####`org.hibernate.tuple.EntityTuplizer` # `org.hibernate.tuple.ComponentTuplizer` ################ `EntityTuplizer` ################################## ###`ComponentTuplizer` ###########################

#######Tuplizer############## ####dynamic-map entity-mode### `java.util.HashMap` ######### `java.util.Map` ########### ##

##########Tuplizer################
Tuplizer##
###############

```
<hibernate-mapping>
    <class entity-name="Customer">
        <!--
            Override the dynamic-map entity-mode
            tuplizer for the customer entity
        -->
        <tuplizer entity-mode="dynamic-map"
                class="CustomMapTuplizerImpl"/>

        <id name="id" type="long" column="ID">
            <generator class="sequence"/>
        </id>

        <!-- other properties -->
        ...
    </class>
</hibernate-mapping>


public class CustomMapTuplizerImpl
        extends org.hibernate.tuple.entity.DynamicMapEntityTuplizer
 {
    // override the buildInstantiator() method to plug in our custom
 map...
    protected final Instantiator buildInstantiator(
            org.hibernate.mapping.PersistentClass mappingInfo) {
        return new CustomMapInstantiator( mappingInfo );
    }

    private static final class CustomMapInstantiator
            extends org.hibernate.tuple.DynamicMapInstantitor {
        // override the generateMap() method to return our custom
 map...
            protected final Map generateMap() {
                    return new CustomMap();
            }
    }
}
```

4.6. Extentsions

TODO: ######################################

#5# ####O/R#####

5.1.

######/###############XML############# ## ########Java##

###Hibernate####XML################### XDoclet, Middlegen, AndroMDA########################### ##########

####################

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="eg">

        <class name="Cat"
            table="cats"
            discriminator-value="C">

                <id name="id">
                        <generator class="native"/>
                </id>

                <discriminator column="subclass"
                     type="character"/>

                <property name="weight"/>

                <property name="birthdate"
                    type="date"
                    not-null="true"
                    update="false"/>

                <property name="color"
                    type="eg.types.ColorUserType"
                    not-null="true"
                    update="false"/>

                <property name="sex"
                    not-null="true"
                    update="false"/>

                <property name="litterId"
                    column="litterId"
                    update="false"/>
```

```
                <many-to-one name="mother"
                    column="mother_id"
                    update="false"/>

                <set name="kittens"
                    inverse="true"
                    order-by="litter_id">
                        <key column="mother_id"/>
                        <one-to-many class="Cat"/>
                </set>

                <subclass name="DomesticCat"
                    discriminator-value="D">

                        <property name="name"
                            type="string"/>

                </subclass>

        </class>

        <class name="Dog">
                <!-- mapping for Dog could go here -->
        </class>

</hibernate-mapping>
```

#########################
########Hibernate##################################
not-null
##

5.1.1. Doctype

XML##################################### ###DTD#####URL# hibernate-x.x.x/src/org/hibernate ####### ### hibernate.jar ####### Hibernate#############DTD######## ##########DTD############### ############DTD############

5.1.1.1.

########Hibernate##########DTD########### org.xml.sax.EntityResolver ######## XML############SAXReader############DTD####### ####### EntityResolver #2#########ID###########

- Hibernate#### ####### http://hibernate.sourceforge.net/ ########ID######### ####### #########Hibernate##################### ######################

- ####### ####### URL###### classpath:// ########ID######### ################(1)####################### ###(2)Hibernate##################### #####################

###################

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"
 [
    <!ENTITY types SYSTEM "classpath://your/domain/types.xml">
]>

<hibernate-mapping package="your.domain">
    <class name="MyEntity">
        <id name="id" type="my-custom-id-type">
            ...
        </id>
    <class>
    &types;
</hibernate-mapping>
```

types.xml # your.domain ############### #### #5.2.3. #######

5.1.2. hibernate-mapping

#######################schema ### catalog #### ####################################(###)################# ### ##############default-cascade #### cascade ######################## ######################## auto-import #### ####################################

```
<hibernate-mapping
         schema="schemaName"                          (1)
         catalog="catalogName"                        (2)
         default-cascade="cascade_style"              (3)
         default-access="field|property|ClassName"    (4)
         default-lazy="true|false"                    (5)
         auto-import="true|false"                     (6)
         package="package.name"                       (7)
 />
```

(1) schema######################

(2) catalog ######################

(3) default-cascade ###### - ###### none## ################

(4) `default-access` (##### - ###### `property` ## Hibernate###################### `PropertyAccessor` #################

(5) `default-lazy` (##### - ###### `true`)# `lazy` #####################################

(6) `auto-import` ###### - ###### `true`## ##

(7) `package` (#####): ################################### #########(prefix)#######

#####################2###### `auto-import="false"` ######### 2######”########”################Hibernate#########

`hibernate-mapping` ###################### `<class>` ############## ##############################(############)############# ########################(################################)# ############## `Cat.hbm.xml` , `Dog.hbm.xml` , ########## `Animal.hbm.xml` #

5.1.3. class

`class` ####################

```
<class
        name="ClassName"                              (1)
        table="tableName"                             (2)
        discriminator-value="discriminator_value"     (3)
        mutable="true|false"                          (4)
        schema="owner"                                (5)
        catalog="catalog"                             (6)
        proxy="ProxyInterface"                        (7)
        dynamic-update="true|false"                   (8)
        dynamic-insert="true|false"                   (9)
        select-before-update="true|false"             (10)
        polymorphism="implicit|explicit"              (11)
        where="arbitrary sql where condition"         (12)
        persister="PersisterClass"                    (13)
        batch-size="N"                                (14)
        optimistic-lock="none|version|dirty|all"      (15)
        lazy="true|false"                             (16)
        entity-name="EntityName"                      (17)
        check="arbitrary sql check condition"         (18)
        rowid="rowid"                                 (19)
        subselect="SQL expression"                    (20)
        abstract="true|false"                         (21)
        node="element-name"
/>
```

(1) `name` (#####)########################Java##### ###############POJO##############################

(2) `table` (##### - ##################)##############

(3) `discriminator-value` (##### - ##########)# ###################################### ## `null` # `not null` ###########

(4) `mutable` (###### ###### `true`)# #####################################

(5) `schema` (optional): Override the schema name specified by the root `<hibernate-mapping>` element.

(6) `catalog` (optional): Override the catalog name specified by the root `<hibernate-mapping>` element.

(7) `proxy` ################################### ##########################

(8) `dynamic-update` ####### ###### `false` ## ###############SQL# `UPDATE` ####################

(9) `dynamic-insert` ######, ###### `false` ## ##null############SQL# `INSERT` ####################

(10) `select-before-update` (#####, ###### `false`): #######################Hibernate#SQL# `UPDATE` # ######## ######### #######(############### `update()` #### ###################)#`UPDATE` ################### Hibernate####SQL# `SELECT` ###############

(11) (optional, ####### `implicit`): implicit#####explicit###### #########################

(12) `where` ######## ############################SQL# `WHERE` #########

(13) `persister` ############ `ClassPersister` #######

(14) `batch-size` ######, ###### `1` ## ##

(15) `optimistic-lock` ######,###### `version` ## ##############

(16) `lazy` ######## `lazy="false"` ######### ################

(17) `entity-name` ################### Hibernate3###################################### Java####Map#XML####################### ## #### #4.4. ####### # # 18. *XML*##### ##########

(18) `check` ######################### #### *check* #######SQL##

(19) `rowid` ########Hibernate####################ROWID# ################# ###Oracle################# `rowid` ####### Hiberante#update######### `rowid` #################### ROWID###################################

(20) `subselect` ############################ ############subselect########### ############################# #####################################

(21) `abstract` ######## `<union-subclass>` ###########################

##################################### ##### `<subclass>` ####### ############################## *static* ################# ############## `eg.Foo$Bar` ###################

`mutable="false"` ############# ############################### ######Hibernate#################

`proxy` #################################### Hibernate########################CGLIB########## #######################################

###################`<class>` ##### `<subclass>` # `<joined-subclass>` #################################### ################ `polymorphism="implicit"` ###### ###############2####################################

`persister` ################################### ### `org.hibernate.persister.EntityPersister` ################ ###################################### LDAP############## `org.hibernate.persister.ClassPersister` ################################### `org.hibernate.test.CustomPersister` ########### `Hashtable` ##########

`dynamic-update` # `dynamic-insert` ################### #### `<subclass>` # `<joined-subclass>` ################ ###

`select-before-update` ##################### ## `Session` ####################### ###

`dynamic-update` ############################

- `version` #####/###################
- `all` #################
- `dirty` ##############################

- none ############

Hibernate###################/################ ### #########
Session.marge()
######## #########################

Hibernate############################### ################
######################DBMS###### ######################
################################ ###################
###############################SQL###############

```
<class name="Summary">
    <subselect>
        select item.name, max(bid.amount), count(*)
        from item
        join bid on bid.item_id = item.id
        group by item.name
    </subselect>
    <synchronize table="item"/>
    <synchronize table="bid"/>
    <id name="name"/>
    ...
</class>
```

###############################
##############
<subselect> ##############################

5.1.4. id

############################### ########## #
###############################JavaBeans################ <id>
##################################

```
<id
        name="propertyName"
  (1)
        type="typename"
  (2)
        column="column_name"
  (3)
        unsaved-value="null|any|none|undefined|id_value"
  (4)
        access="field|property|ClassName">
  (5)
        node="element-name|@attribute-name|element/@attribute|."

        <generator class="generatorClass"/>
</id>
```

(1) name####################

(2) type########Hibernate########

(3) column###### - ############### ###########

(4) unsaved-value###### - ########sensible## ##################### ############################## ###Session################################ #####

(5) access (optional - defaults to property): The strategy Hibernate should use for accessing the property value.

name ####################################

unsaved-value ###Hibernate3####################

############################ <composite-id> #################### #########################

5.1.4.1.

<generator> ##### ##################################Java########## #######################################<param>

```
<id name="id" type="long" column="cat_id">
        <generator class="org.hibernate.id.TableHiLoGenerator">
                <param name="table">uid_table</param>
                <param name="column">next_hi_value_column</param>
        </generator>
</id>
```

All generators implement the interface org.hibernate.id.IdentifierGenerator. This is a very simple interface; some applications may choose to provide their own specialized implementations. However, Hibernate provides a range of built-in implementations. There are shortcut names for the built-in generators:

increment
: long , short , int ############ ## ################# #

identity
: DB2, MySQL, MS SQL Server, Sybase, HypersonicSQL######## ######## ########## long , short , int ########

sequence
: DB2, PostgreSQL, Oracle, SAP DB, McKoi########Interbase############### ########## long , short , int ########

`hilo`
: `long` , `short` , `int` ##############hi/lo############ hi######################(########### `hibernate_unique_key` # `next_hi`)# hi/lo#####################################

`seqhilo`
: `long` , `short` , `int` ##############hi/lo############ ######################

`uuid`
: (IP##########)############################### 128####UUID#############UUID###32#16####################

`guid`
: MS SQL####MySQL###########GUID##########

`native`
: ################ `identity` , `sequence` , `hilo` ############

`assigned`
: `save()` ######## ################################# `<generator>` ##############################

`select`
: ############################# #########################

`foreign`
: #################### ####`<one-to-one>` #################

`sequence-identity`
: a specialized sequence generation strategy which utilizes a database sequence for the actual value generation, but combines this with JDBC3 getGeneratedKeys to actually return the generated identifier value as part of the insert statement execution. This strategy is only known to be supported on Oracle 10g drivers targetted for JDK 1.4. Note comments on these insert statements are disabled due to a bug in the Oracle drivers.

5.1.4.2. Hi/lo

`hilo` # `seqhilo` ######## ##################hi/lo#######2############# 1##############"hi"##################### ####### 2#######Oracle#############################

```
<id name="id" type="long" column="cat_id">
        <generator class="hilo">
```

```
                <param name="table">hi_value</param>
                <param name="column">next_value</param>
                <param name="max_lo">100</param>
        </generator>
</id>
```

```
<id name="id" type="long" column="cat_id">
        <generator class="seqhilo">
                <param name="sequence">hi_value</param>
                <param name="max_lo">100</param>
        </generator>
</id>
```

#####Hibernate##### Connection ##########hilo
#######
Hibernate#JTA###
hibernate.transaction.manager_lookup_class #################

5.1.4.3. UUID

UUID############## IP#####JVM############4##1#######
########JVM################
Java#####MAC########################### JNI################

5.1.4.4.

######################DB2, MySQL, Sybase, MS SQL####
identity ########## ####################DB2, Oracle, PostgreSQL,
Interbase, McKoi, SAP DB#### sequence ###############
##########################SQL####2########

```
<id name="id" type="long" column="person_id">
        <generator class="sequence">
                <param name="sequence">person_id_sequence</param>
        </generator>
</id>
```

```
<id name="id" type="long" column="person_id" unsaved-value="0">
        <generator class="identity"/>
</id>
```

#################native ### identity , sequence , hilo
######1######### #########################

5.1.4.5.

############################(Hibernate############# assigned
################# ##

`<generator>`

`assigned` ############## Hibernate# `unsaved-value="undefined"` ####### ############################### `Interceptor.isUnsaved()` ########################(transient)######### ############(detached)#############################

5.1.4.6.

####################(Hibernate########DDL#######)#

```
<id name="id" type="long" column="person_id">
        <generator class="select">
                <param name="key">socialSecuritySNumber</param>
        </generator>
</id>
```

######################### `socialSecurityNumber` ##################### ############## `person_id` #################

5.1.5. Enhanced identifier generators

Starting with release 3.2.3, there are 2 new generators which represent a re-thinking of 2 different aspects of identifier generation. The first aspect is database portability; the second is optimization (not having to query the database for every request for a new identifier value). These two new generators are intended to take the place of some of the named generators described above (starting in 3.3.x); however, they are included in the current releases and can be referenced by FQN.

The first of these new generators is `org.hibernate.id.enhanced.SequenceStyleGenerator` which is intended firstly as a replacement for the `sequence` generator and secondly as a better portability generator than `native` (because `native` (generally) chooses between `identity` and `sequence` which have largely different semantics which can cause subtle isssues in applications eyeing portability). `org.hibernate.id.enhanced.SequenceStyleGenerator` however achieves portability in a different manner. It chooses between using a table or a sequence in the database to store its incrementing values depending on the capabilities of the dialect being used. The difference between this and `native` is that table-based and sequence-based storage have the same exact semantic (in fact sequences are exactly what Hibernate tries to emmulate with its table-based generators). This generator has a number of configuration parameters:

- `sequence_name` (optional, defaults to `hibernate_sequence`): The name of the sequence (or table) to be used.
- `initial_value` (optional, defaults to `1`): The initial value to be retrieved from the sequence/table. In sequence creation terms, this is analogous to the clause typical named "STARTS WITH".
- `increment_size` (optional, defaults to `1`): The value by which subsequent calls to the sequence/table should differ. In sequence creation terms, this is analogous to the clause typical named "INCREMENT BY".
- `force_table_use` (optional, defaults to `false`): Should we force the use of a table as the backing structure even though the dialect might support sequence?
- `value_column` (optional, defaults to `next_val`): Only relevant for table structures! The name of the column on the table which is used to hold the value.
- `optimizer` (optional, defaults to `none`): See #5.1.6. #Identifier generator optimization#

The second of these new generators is `org.hibernate.id.enhanced.TableGenerator` which is intended firstly as a replacement for the `table` generator (although it actually functions much more like `org.hibernate.id.MultipleHiLoPerTableGenerator`) and secondly as a re-implementation of `org.hibernate.id.MultipleHiLoPerTableGenerator` utilizing the notion of pluggable optimiziers. Essentially this generator defines a table capable of holding a number of different increment values simultaneously by using multiple distinctly keyed rows. This generator has a number of configuration parameters:

- `table_name` (optional, defaults to `hibernate_sequences`): The name of the table to be used.
- `value_column_name` (optional, defaults to `next_val`): The name of the column on the table which is used to hold the value.
- `segment_column_name` (optional, defaults to `sequence_name`): The name of the column on the table which is used to hold the "segement key". This is the value which distinctly identifies which increment value to use.
- `segment_value` (optional, defaults to `default`): The "segment key" value for the segment from which we want to pull increment values for this generator.
- `segment_value_length` (optional, defaults to `255`): Used for schema generation; the column size to create this segment key column.
- `initial_value` (optional, defaults to `1`): The initial value to be retrieved from the table.
- `increment_size` (optional, defaults to `1`): The value by which subsequent calls to the table should differ.

- `optimizer` (optional, defaults to): See #5.1.6. #Identifier generator optimization#

5.1.6. Identifier generator optimization

For identifier generators which store values in the database, it is inefficient for them to hit the database on each and every call to generate a new identifier value. Instead, you'd ideally want to group a bunch of them in memory and only hit the database when you have exhausted your in-memory value group. This is the role of the pluggable optimizers. Currently only the two enhanced generators (#5.1.5. #Enhanced identifier generators# support this notion.

- `none` (generally this is the default if no optimizer was specified): This says to not perform any optimizations, and hit the database each and every request.
- `hilo`: applies a hi/lo algorithm around the database retrieved values. The values from the database for this optimizer are expected to be sequential. The values retrieved from the database structure for this optimizer indicates the "group number"; the `increment_size` is multiplied by that value in memory to define a group "hi value".
- `pooled`: like was discussed for `hilo`, this optimizers attempts to minimize the number of hits to the database. Here, however, we simply store the starting value for the "next group" into the database structure rather than a sequential value in combination with an in-memory grouping algorithm. `increment_size` here refers to the values coming from the database.

5.1.7. composite-id

```
<composite-id
        name="propertyName"
        class="ClassName"
        mapped="true|false"
        access="field|property|ClassName">
        node="element-name|."

        <key-property name="propertyName" type="typename"
 column="column_name"/>
        <key-many-to-one name="propertyName class="ClassName"
 column="column_name"/>
        ......
</composite-id>
```

################ ##
`<composite-id>` ########## `<key-property>` ###########
`<key-many-to-one>` #############

```
<composite-id>
```

```
        <key-property name="medicareNumber"/>
        <key-property name="dependent"/>
</composite-id>
```

########################## equals() # hashCode() ######## ########## # ## Serializable ##############

##################### ############################## ################################### ############### load() ########### ## ##### #################### ######################

2###### ###### ############### <composite-id>################### ################################

```
<composite-id class="MedicareId" mapped="true">
        <key-property name="medicareNumber"/>
        <key-property name="dependent"/>
</composite-id>
```

################ MedicareId ################### medicareNumber # dependent ################# ########equals() # hashCode() ######### Serializable ############### #################################

#################################

- mapped (############ false): ## ####################################
- class (#####,###################): #################

3#####################8.4. ##################### ####################### #############################

- name (#####, ###########): ###########################(9##########).
- access (optional - defaults to property): The strategy Hibernate should use for accessing the property value.
- class ###### - ################################ ######################################

##3###### ########## #### ###############################

5.1.8. discriminator

<discriminator> #### table-per-class-hierarchy##############################

################## ##
################# ################ `string` , `character` , `integer`, `byte` , `short` , `boolean` , `yes_no` , `true_false`.

```
<discriminator
        column="discriminator_column"                        (1)
        type="discriminator_type"                            (2)
        force="true|false"                                   (3)
        insert="true|false"                                  (4)
        formula="arbitrary sql expression"                   (5)
/>
```

(1) `column`###### - ###### `class` ## #########

(2) `type` ###### - ###### `string` ##Hibernate########

(3) `force` ###### - ###### `false` ## ################################ Hibernate########################

(4) `insert` ###### - ###### `true` ## ##############################`false` ########### (Hibernate#SQL# `INSERT` ###############)

(5) `formula` (#####)###################SQL## ####################

############ `<class>` # `<subclass>` ### `discriminator-value` ##########

############################## ###################`force` ######## ####################

`formula` #####################SQL#########

```
<discriminator
    formula="case when CLASS_TYPE in ('a', 'b', 'c') then 0 else 1
 end"
    type="integer"/>
```

5.1.9. version#######

`<version>` ############ ######################### ### ########### #############################

```
<version
        column="version_column"
  (1)
        name="propertyName"
  (2)
        type="typename"
  (3)
        access="field|property|ClassName"
  (4)
        unsaved-value="null|negative|undefined"
  (5)
        generated="never|always"
  (6)
```

```
        insert="true|false"
  (7)
        node="element-name|@attribute-name|element/@attribute|."
/>
```

(1) `column`###### - ##############: ####################

(2) `name` #################

(3) `type` ###### - ###### `integer` ############

(4) `access` (optional - defaults to `property`): The strategy Hibernate should use for accessing the property value.

(5) `unsaved-value` ###### - ###### `undefined` ## ########################## ############################## ###Session################################## ##### # `undefined` ###########################

(6) `generated` (##### - ###### `never`): ### #5.6. ######### ############

(7) `insert` (##### - ###### `true`): SQL#insert############################## #################### 0 ########### `false` ##############

######## Hibernate# `long` , `integer` , `short` , `timestamp` , `calendar` #########

######################################null########### ######### `unsaved-value` ########## Hibernate#null################################ ################ *null*################################## ############################# ##################################

5.1.10. timestamp#######

`<timestamp>` #### ########################## ############################## ##################################

```
<timestamp
        column="timestamp_column"
  (1)
        name="propertyName"
  (2)
        access="field|property|ClassName"
  (3)
        unsaved-value="null|undefined"
  (4)
        source="vm|db"
  (5)
```

```
        generated="never|always"
  (6)
        node="element-name|@attribute-name|element/@attribute|."
/>
```

(1) column###### - ############### ####################

(2) name # ########Java # Date# ### Timestamp # ##JavaBeans#############

(3) access (optional - defaults to property): The strategy Hibernate should use for accessing the property value.

(4) unsaved-value ###### - ###### null ## ##################### ############################### ###Session######################### ############## # undefined ####### ###################

(5) source (##### - ###### vm): Hibernate############################# #################JVM######## ##################Hibernate#"###"######## ##################################### ###########JVM############## ############################# #### ######### ################# ########################### ######################(###Oracle 8)#

(6) generated (##### - ###### never): ## #5.6. ######### ##########

<timestamp> # <version type="timestamp"> ################### <timestamp source="db"> # <version type="dbtimestamp"> ###################

5.1.11. property

<property> ############JavaBean#################

```
<property
        name="propertyName"
  (1)
        column="column_name"
  (2)
        type="typename"
  (3)
        update="true|false"
  (4)
        insert="true|false"
  (4)
        formula="arbitrary SQL expression"
  (5)
        access="field|property|ClassName"
  (6)
```

```
        lazy="true|false"
  (7)
        unique="true|false"
  (8)
        not-null="true|false"
  (9)
        optimistic-lock="true|false"
  (10)
        generated="never|insert|always"
  (11)
        node="element-name|@attribute-name|element/@attribute|."
        index="index_name"
        unique_key="unique_key_id"
        length="L"
        precision="P"
        scale="S"
/>
```

(1) `name`################

(2) `column`###### - ############### ############################ ##### `<column>` ############

(3) `type`########Hibernate########

(4) `update, insert` ###### - ###### `true` ## ############SQL# `UPDATE` # `INSERT` ############### #### `false` ####### ############################ ##

(5) `formula`######## ## ################SQL## #################################

(6) `access` (optional - defaults to `property`): The strategy Hibernate should use for accessing the property value.

(7) `lazy` (optional - ###### `false`): ### (########################)#

(8) `unique` (#####):###############DDL############ ###`property-ref` ##################

(9) `not-null` (#####):####null######DDL############

(10) `optimistic-lock` (##### - ###### `true`): ##################################### ###

(11) `generated` (##### - ###### `never`): ################################### #5.6. ######### ##########

typename ############

1. Hibernate######### `integer, string, character, date, timestamp, float, binary, serializable, object, blob` ##
2. ##########Java#### ## `int, float, char, java.lang.String, java.util.Date, java.lang.Integer, java.sql.Clob` ##

3. #########Java#######
4. ############ com.illflow.type.MyCustomType ##

##########Hibernate####Hibernate########### ############################ Hibernate####2, 3, 4######### getter############################ ###################### ########type ############### #### Hibernate.DATE # Hibernate.TIMESTAMP ######## ######################

access #### ####Hibernate############################ #######Hibernate#######get/set########### access="field" ####### Hibernate###########get/set########################## ######## org.hibernate.property.PropertyAccessor # ##

################### ### ###SQL################### #############SQL#### SELECT ################

```
<property name="totalPrice"
    formula="( SELECT SUM (li.quantity*p.price) FROM LineItem li,
 Product p
                WHERE li.productId = p.productId
                AND li.customerId = customerId
                AND li.orderNumber = orderNumber )"/>
```

######(### customerId #########)################# ################################# ################ ##### <formula> #########################

5.1.12. many-to-one

################ many-to-one ############ #################### ##

```
<many-to-one
        name="propertyName"
  (1)
        column="column_name"
  (2)
        class="ClassName"
  (3)
        cascade="cascade_style"
  (4)
        fetch="join|select"
  (5)
        update="true|false"
  (6)
```

```
        insert="true|false"
  (6)
        property-ref="propertyNameFromAssociatedClass"
  (7)
        access="field|property|ClassName"
  (8)
        unique="true|false"
  (9)
        not-null="true|false"
  (10)
        optimistic-lock="true|false"
  (11)
        lazy="proxy|no-proxy|false"
  (12)
        not-found="ignore|exception"
  (13)
        entity-name="EntityName"
  (14)
        formula="arbitrary SQL expression"
  (15)
        node="element-name|@attribute-name|element/@attribute|."
        embed-xml="true|false"
        index="index_name"
        unique_key="unique_key_id"
        foreign-key="foreign_key_name"
/>
```

(1) name########

(2) column (optional): The name of the foreign key column. This may also be specified by nested <column> element(s).

(3) class (optional - defaults to the property type determined by reflection): The name of the associated class.

(4) cascade######## ##

(5) fetch (optional - defaults to select): Chooses between outer-join fetching or sequential select fetching.

(6) update, insert###### - ###### true ## ############SQL# UPDATE ### INSERT ################ #### false ####### ################################ ##

(7) property-ref######## ########################### ############################

(8) access (optional - defaults to property): The strategy Hibernate should use for accessing the property value.

(9) unique######## #####################DDL########### ###property-ref ################## ##########################

(10) not-null (#####):############ null######DDL##########

(11) `optimistic-lock` (##### - ###### `true`): ####################################### ##

(12) `lazy` (##### - ###### `proxy`): ############################ `lazy="no-proxy"` ########################## ###################### (#######################)# `lazy="false"` #########################

(13) `not-found` (##### - ###### `exception`): ################################ `ignore` #######null###########

(14) `entity-name` (optional): The entity name of the associated class.

(15) `formula` (#####): ##### ###############SQL#

`cascade` ### `none` ################## ############################ ########Hibernate############# `delete-orphan` # `all` ################### #### `cascade="persist,merge,evict"` # `cascade="all,delete-orphan"`## ########## `persist, merge, delete, save-update, evict, replicate, lock, refresh` ########## ###### #10.11. ########## ######### #######(many-to-one#one-to-one##)## #######orphan delete)#######################

`many-to-one`

```
<many-to-one name="product" class="Product" column="PRODUCT_ID"/>
```

`property-ref` ############################## ### #################### ### `Product` ##### ### # `unique` ###SchemaExport#######Hibernate#DDL##########

```
<property name="serialNumber" unique="true" type="string"
 column="SERIAL_NUMBER"/>
```

`OrderItem`

```
<many-to-one name="product" property-ref="serialNumber"
 column="PRODUCT_SERIAL_NUMBER"/>
```

###################

`<properties>`

###

```
<many-to-one name="owner" property-ref="identity.ssn"
 column="OWNER_SSN"/>
```

5.1.13. one-to-one

################one-to-one #########

```
<one-to-one
        name="propertyName"
  (1)
        class="ClassName"
  (2)
        cascade="cascade_style"
  (3)
        constrained="true|false"
  (4)
        fetch="join|select"
  (5)
        property-ref="propertyNameFromAssociatedClass"
  (6)
        access="field|property|ClassName"
  (7)
        formula="any SQL expression"
  (8)
        lazy="proxy|no-proxy|false"
  (9)
        entity-name="EntityName"
  (10)
        node="element-name|@attribute-name|element/@attribute|."
        embed-xml="true|false"
        foreign-key="foreign_key_name"
/>
```

(1) name########

(2) class (optional - defaults to the property type determined by reflection): The name of the associated class.

(3) cascade######## ##

(4) constrained######## ################################ ######################### ######## save() # delete() ################# ########################## ############################

(5) fetch (optional - defaults to select): Chooses between outer-join fetching or sequential select fetching.

(6) property-ref######## ############################### ###########################

(7) access (optional - defaults to property): The strategy Hibernate should use for accessing the property value.

(8) formula (#####): ### ########### ##############SQL######################## ### org.hibernate.test.onetooneformula ###########

(9) `lazy` (##### - ###### `proxy`): ############################ `lazy="no-proxy"` ########################## ###################### (#######################)# `lazy="false"` ######################## ## *`constrained="false"`* #### ###

(10) `entity-name` (optional): The entity name of the associated class.

#######2#######

- #####
- ##########

############################# ##2#################2##################### ####2################################# #########################

###################### `Employee` # `Person` ################

```
<one-to-one name="person" class="Person"/>
```

```
<one-to-one name="employee" class="Employee" constrained="true"/>
```

####PERSON#EMPLOYEE## #####`foreign` ######Hibernate#############

```
<class name="person" table="PERSON">
    <id name="id" column="PERSON_ID">
        <generator class="foreign">
            <param name="property">employee</param>
        </generator>
    </id>
    ...
    <one-to-one name="employee"
        class="Employee"
        constrained="true"/>
</class>
```

`Employee` ########`Person` # `employee` ################ ########## `Person` ######################### ######### `Person` ######## ## `Person` # `employee` ########## `Employee` #########################

##1########`Employee` ## `Person` ## ################################

```
<many-to-one name="person" class="Person" column="PERSON_ID"
 unique="true"/>
```

######### ###### `Person` ##############################

```
<one-to-one name="employee" class="Employee" property-ref="person"/>
```

5.1.14. natural-id

```
<natural-id mutable="true|false"/>
        <property ... />
        <many-to-one ... />
        ......
</natural-id>
```

###################### ################################## ############null######################### ############### <natural-id> ########################## Hibernate##########null############# ########################

####################### equals() # hashCode() ##############

#######################################

- mutable (#####, ###### false): #####################(##)###########

5.1.15. component, dynamic-component

<component> #### ####################################### ### #####################

```
<component
        name="propertyName"                 (1)
        class="className"                   (2)
        insert="true|false"                 (3)
        update="true|false"                 (4)
        access="field|property|ClassName"   (5)
        lazy="true|false"                   (6)
        optimistic-lock="true|false"        (7)
        unique="true|false"                 (8)
        node="element-name|."
>

        <property ...../>
        <many-to-one .... />
        ........
</component>
```

(1) name########

(2) class ###### - ################################ ##################

(3) insert: Do the mapped columns appear in SQL INSERTS?

(4) update: Do the mapped columns appear in SQL UPDATES?

(5) `access` (optional - defaults to `property`): The strategy Hibernate should use for accessing the property value.

(6) `lazy` (optional - ###### `false`): ###################### ########################## (#######################)

(7) `optimistic-lock` (##### - ###### `true`): ##################################### ###

(8) `unique` (##### - ###### `false`): ##

`<property>` ####

`<component>` #################### ######################### `<parent>` ###########

`<dynamic-component>` #### `Map` ############################## #######map###########8.5. ########### ##########

5.1.16. properties

`<properties>` ################### ################## ################ `property-ref` ############################ ####################################

```
<properties
        name="logicalName"                  (1)
        insert="true|false"                 (2)
        update="true|false"                 (3)
        optimistic-lock="true|false"        (4)
        unique="true|false"                 (5)
>

        <property ...../>
        <many-to-one .... />
        ........
</properties>
```

(1) `name` : ########### ########### ##### #

(2) `insert`: Do the mapped columns appear in SQL INSERTS?

(3) `update`: Do the mapped columns appear in SQL UPDATES?

(4) `optimistic-lock` (##### - ###### `true`): ####################################### ###

(5) `unique` (##### - ###### `false`): ##

############ `<properties>` ############

```
<class name="Person">
    <id name="personNumber"/>
    ...
    <properties name="name"
            unique="true" update="false">
        <property name="firstName"/>
        <property name="initial"/>
        <property name="lastName"/>
    </properties>
</class>
```

######## `Person` ################### ######################

```
<many-to-one name="person"
         class="Person" property-ref="name">
    <column name="firstName"/>
    <column name="initial"/>
    <column name="lastName"/>
</many-to-one>
```

##

5.1.17. subclass

##
table-per-class-hierarchy########## `<subclass>` #########

```
<subclass
        name="ClassName"                              (1)
        discriminator-value="discriminator_value"     (2)
        proxy="ProxyInterface"                        (3)
        lazy="true|false"                             (4)
        dynamic-update="true|false"
        dynamic-insert="true|false"
        entity-name="EntityName"
        node="element-name"
        extends="SuperclassName">

        <property .... />
        .....
</subclass>
```

(1) `name`###################

(2) `discriminator-value`###### - ############ ##################

(3) `proxy` (optional): Specifies a class or interface to use for lazy initializing proxies.

(4) `lazy` (optional, defaults to `true`): Setting `lazy="false"` disables the use of lazy fetching.

############################## <version> # <id> ####### ##################### ############### ##### discriminator-value ############## none##############Java##########

############### # 9. ####### ########

5.1.18. joined-subclass

##1###################################### (table-per-subclass mapping strategy)# ################################# <joined-subclass> #########

```
<joined-subclass
        name="ClassName"                        (1)
        table="tablename"                       (2)
        proxy="ProxyInterface"                  (3)
        lazy="true|false"                       (4)
        dynamic-update="true|false"
        dynamic-insert="true|false"
        schema="schema"
        catalog="catalog"
        extends="SuperclassName"
        persister="ClassName"
        subselect="SQL expression"
        entity-name="EntityName"
        node="element-name">

        <key .... >

        <property .... />
        .....
</joined-subclass>
```

(1) name###################

(2) table :#############

(3) proxy (optional): Specifies a class or interface to use for lazy initializing proxies.

(4) lazy (optional, defaults to true): Setting lazy="false" disables the use of lazy fetching.

########################## ########## <key> ###### #################################### #########################

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD//EN"

 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="eg">
```

```
        <class name="Cat" table="CATS">
                <id name="id" column="uid" type="long">
                        <generator class="hilo"/>
                </id>
                <property name="birthdate" type="date"/>
                <property name="color" not-null="true"/>
                <property name="sex" not-null="true"/>
                <property name="weight"/>
                <many-to-one name="mate"/>
                <set name="kittens">
                        <key column="MOTHER"/>
                        <one-to-many class="Cat"/>
                </set>
                <joined-subclass name="DomesticCat"
 table="DOMESTIC_CATS">
                    <key column="CAT"/>
                    <property name="name" type="string"/>
                </joined-subclass>
        </class>

        <class name="eg.Dog">
                <!-- mapping for Dog could go here -->
        </class>

</hibernate-mapping>
```

############### # 9. ####### ########

5.1.19. union-subclass

3####################################### (the table-per-concrete-class##)# ####################################### Hibernate######################### ############## ### `<class>` ##################### ###############(### ###############)######## `<union-subclass>` #################

```
<union-subclass
        name="ClassName"                    (1)
        table="tablename"                   (2)
        proxy="ProxyInterface"              (3)
        lazy="true|false"                   (4)
        dynamic-update="true|false"
        dynamic-insert="true|false"
        schema="schema"
        catalog="catalog"
        extends="SuperclassName"
        abstract="true|false"
        persister="ClassName"
        subselect="SQL expression"
        entity-name="EntityName"
        node="element-name">
```

```
        <property .... />
        .....
</union-subclass>
```

(1) `name`###################

(2) `table` :#############

(3) `proxy` (optional): Specifies a class or interface to use for lazy initializing proxies.

(4) `lazy` (optional, defaults to `true`): Setting `lazy="false"` disables the use of lazy fetching.

###############################

############### # 9. ####### ########

5.1.20. join

`<join>` ######### ##

```
<join
        table="tablename"                         (1)
        schema="owner"                            (2)
        catalog="catalog"                         (3)
        fetch="join|select"                       (4)
        inverse="true|false"                      (5)
        optional="true|false">                    (6)

        <key ... />

        <property ... />
        ...
</join>
```

(1) `table` :###########

(2) `schema` (optional): Override the schema name specified by the root `<hibernate-mapping>` element.

(3) `catalog` (optional): Override the catalog name specified by the root `<hibernate-mapping>` element.

(4) `fetch` (##### - ###### `join`): `join` ######## Hibernate########################## `<join>` ########################## `<join>` ################## `select` ########## Hibernate############ `<join>` #################### ## ######################## `<join>` ### ###########

(5) `inverse` (##### - ###### `false`): #########Hibernate##################### ############

(6) `optional` (##### - ###### `false`): #########Hibernate################null######## ################################

##################################### (##############################)#

```
<class name="Person"
    table="PERSON">

    <id name="id" column="PERSON_ID">...</id>

    <join table="ADDRESS">
        <key column="ADDRESS_ID"/>
        <property name="address"/>
        <property name="zip"/>
        <property name="country"/>
    </join>
    ...
```

################################ #################################### ###

5.1.21. key

`<key>` ########## ####################### ################################

```
<key
        column="columnname"                      (1)
        on-delete="noaction|cascade"             (2)
        property-ref="propertyName"              (3)
        not-null="true|false"                    (4)
        update="true|false"                      (5)
        unique="true|false"                      (6)
/>
```

(1) `column` (optional): The name of the foreign key column. This may also be specified by nested `<column>` element(s).

(2) `on-delete` (#####, ###### `noaction`): #######################################

(3) `property-ref` (#####): ##################################### (#################)#

(4) `not-null` (#####): ########null################ (###############################)#

(5) `update` (#####): ########################## (###############################)#

(6) `unique` (#####): ###########################
(###############################)#

######################## ####### `on-delete="cascade"`
############### #######Hibernate##`DELETE`
############## ########## `ON CASCADE DELETE` #########
#######################Hibernate#############
####################

`not-null` # `update` ######################
#########null######################## `<key not-null="true">`
############# ########### #

5.1.22. column # formula

`column` ####################### `<column>` ############ ### `<formula>` # `formula` ##########

```
<column
        name="column_name"
        length="N"
        precision="N"
        scale="N"
        not-null="true|false"
        unique="true|false"
        unique-key="multicolumn_unique_key_name"
        index="index_name"
        sql-type="sql_type_name"
        check="SQL expression"
        default="SQL expression"/>
```

```
<formula>SQL expression</formula>
```

##################### `column` # `formula` #################
#################

```
<many-to-one name="homeAddress" class="Address"
        insert="false" update="false">
    <column name="person_id" not-null="true" length="10"/>
    <formula>'MAILING'</formula>
</many-to-one>
```

5.1.23. import

##############2###########
Hibernate## ########
`auto-import="true"` ###### ############################
###

```
<import class="java.lang.Object" rename="Universe"/>
```

```
<import
        class="ClassName"              (1)
        rename="ShortName"             (2)
/>
```

(1) class#Java################

(2) rename###### - #################### #############

5.1.24. any

#################1######### <any> #########
#################################
########################1#######################
###
###
################################)###############

meta-type #### ##############################
#############id-type ####### ############################
meta-type##############################

```
<any name="being" id-type="long" meta-type="string">
    <meta-value value="TBL_ANIMAL" class="Animal"/>
    <meta-value value="TBL_HUMAN" class="Human"/>
    <meta-value value="TBL_ALIEN" class="Alien"/>
    <column name="table_name"/>
    <column name="id"/>
</any>
```

```
<any
        name="propertyName"                      (1)
        id-type="idtypename"                     (2)
        meta-type="metatypename"                 (3)
        cascade="cascade_style"                  (4)
        access="field|property|ClassName"        (5)
        optimistic-lock="true|false"             (6)
>
        <meta-value ... />
        <meta-value ... />
        .....
        <column .... />
        <column .... />
        .....
</any>
```

(1) name########

(2) id-type#######

(3) `meta-type`###### - ###### `string` ## ####################

(4) `cascade`###### - ###### `none` ## ###########

(5) `access` (optional - defaults to `property`): The strategy Hibernate should use for accessing the property value.

(6) `optimistic-lock` (##### - ###### `true`): ################################### ##

5.2. Hibernate##

5.2.1.

#############Java########################### #######2##################

###################################Java##### ############# #################### ##### ########### ############################### #####################ODMG############# ## ########################

##################### # ## #################### ###############(#############)# ################### #################################### ############### ##### # ## ######################### ##################### #################################

###################################### ############## ############################ ################### ####### ######################## `java.lang.String` ######################### #######JDK###########Java##(###)#################### ## ##################### ####################### ################################## #######################

###########################

Java######(######################)# SQL/############################# Hibernate################## ########### `<class>` # `<subclass>` ######### ####### `<property>`

`<component>` ###### `type` ######### #######Hibernate# ###### ###### Hibernate#(##JDK#######)##################

##############Hibernate#######null################

5.2.2.

The built-in *basic mapping types* may be roughly categorized into

`integer, long, short, float, double, character, byte, boolean, yes_no, true_false`
: Java############################## SQL############# `boolean`, `yes_no` # `true_false` ## ###Java# `boolean` ### `java.lang.Boolean` ###########

`string`
: `java.lang.String` ## `VARCHAR` ####Oracle# `VARCHAR2` ##########

`date, time, timestamp`
: `java.util.Date` ##########SQL## `DATE`, `TIME` , `TIMESTAMP` ############## #########

`calendar, calendar_date`
: `java.util.Calendar` ##SQL# ## `TIMESTAMP` , `DATE` (######################

`big_decimal, big_integer`
: `java.math.BigDecimal` # `java.math.BigInteger` ## `NUMERIC`####Oracle# `NUMBER` ##########

`locale, timezone, currency`
: `java.util.Locale` , `java.util.TimeZone` , `java.util.Currency` ## `VARCHAR` ####Oracle# `VARCHAR2` ########## `Locale` # `Currency` ######### ####ISO############## `TimeZone` ######### #### `ID` ###########

`class`
: `java.lang.Class` ## `VARCHAR` ####Oracle# `VARCHAR2` ########## `Class` #######################

`binary`
: ##########SQL#################

`text`
: ##Java#####SQL# `CLOB` ### `TEXT` ############

serializable

#########Java######SQL#################
#######################Java#### #####################
Hibernate### serializable ###########

clob, blob

JDBC### java.sql.Clob # java.sql.Blob ###########
blob#clob##############################
######################## #########################

imm_date, imm_time, imm_timestamp, imm_calendar, imm_calendar_date, imm_serializable, imm_binary

##############Java##############
Hibernate####Java############### #############################
imm_timestamp ################### Date.setTime()
############## ###########################
######################(#####)######################

#########################binary , blob , clob ###################
##########################

#########org.hibernate.Hibernate ###### Type ############
####Hibernate.STRING # string ##########

5.2.3.

########################### ####java.lang.BigInteger ########
VARCHAR ################### Hibernate######################
################################1########## ###################
########java.lang.String ## getName() / setName() Java###### FIRST_NAME , INITIAL, SURNAME ############

#############org.hibernate.UserType ###
org.hibernate.CompositeUserType ##### ############################
###################### org.hibernate.test.DoubleStringType
##########

```
<property name="twoStrings"
 type="org.hibernate.test.DoubleStringType">
    <column name="first_string"/>
    <column name="second_string"/>
</property>
```

<column> #### ###################################

CompositeUserType , EnhancedUserType , UserCollectionType , UserVersionType ####################################

########### UserType ############## #######UserType # org.hibernate.usertype.ParameterizedType ############## ################### ########### <type> ##########

```
<property name="priority">
    <type name="com.mycompany.usertypes.DefaultValueIntegerType">
        <param name="default">0</param>
    </type>
</property>
```

UserType ## ######### Properties ######### default #############################

UserType ################################### <typedef> ################### Typedefs################# ################

```
<typedef class="com.mycompany.usertypes.DefaultValueIntegerType"
 name="default_zero">
    <param name="default">0</param>
</typedef>
```

```
<property name="priority" type="default_zero"/>
```

########################### typedef###################################

Hibernate############################# ########### #### ############# #######################(###########)################### ################# ### MonetaryAmount ############################## CompositeUserType ######## ##############1######## ############################# #############################

5.3.

################################# ### ####### ############# (##########################)# ########################### ############################# Hibernate########################

```
<class name="Contract" table="Contracts"
        entity-name="CurrentContract">
    ...
    <set name="history" inverse="true"
            order-by="effectiveEndDate desc">
        <key column="currentContractId"/>
        <one-to-many entity-name="HistoricalContract"/>
    </set>
</class>
```

```
<class name="Contract" table="ContractHistory"
        entity-name="HistoricalContract">
    ...
    <many-to-one name="currentContract"
            column="currentContractId"
            entity-name="CurrentContract"/>
</class>
```

###class ##### entity-name ##### ######################

5.4. ########### SQL

Hibernate######SQL################### Hibernate#SQL# Dialect #################### ################SQL Server#######MySQL#############

```
<class name="LineItem" table="`Line Item`">
    <id name="id" column="`Item Id`"/><generator
 class="assigned"/></id>
    <property name="itemNumber" column="`Item #`"/>
    ...
</class>
```

5.5.

XML####### Hibernate##O/R##########################

5.5.1. XDoclet

###Hibernate####XDoclet# @hibernate.tags ##### ################################ #########XDoclet################################ ###XDoclet####### Cat #############

```
package eg;
import java.util.Set;
import java.util.Date;

/**
 * @hibernate.class
 *  table="CATS"
 */
public class Cat {
    private Long id; // identifier
    private Date birthdate;
    private Cat mother;
    private Set kittens
    private Color color;
```

```
    private char sex;
    private float weight;

    /*
     * @hibernate.id
     *  generator-class="native"
     *  column="CAT_ID"
     */
    public Long getId() {
        return id;
    }
    private void setId(Long id) {
        this.id=id;
    }

    /**
     * @hibernate.many-to-one
     *  column="PARENT_ID"
     */
    public Cat getMother() {
        return mother;
    }
    void setMother(Cat mother) {
        this.mother = mother;
    }

    /**
     * @hibernate.property
     *  column="BIRTH_DATE"
     */
    public Date getBirthdate() {
        return birthdate;
    }
    void setBirthdate(Date date) {
        birthdate = date;
    }
    /**
     * @hibernate.property
     *  column="WEIGHT"
     */
    public float getWeight() {
        return weight;
    }
    void setWeight(float weight) {
        this.weight = weight;
    }

    /**
     * @hibernate.property
     *  column="COLOR"
     *  not-null="true"
     */
    public Color getColor() {
        return color;
```

```
    }
    void setColor(Color color) {
        this.color = color;
    }
    /**
     * @hibernate.set
     *  inverse="true"
     *  order-by="BIRTH_DATE"
     * @hibernate.collection-key
     *  column="PARENT_ID"
     * @hibernate.collection-one-to-many
     */
    public Set getKittens() {
        return kittens;
    }
    void setKittens(Set kittens) {
        this.kittens = kittens;
    }
    // addKitten not needed by Hibernate
    public void addKitten(Cat kitten) {
        kittens.add(kitten);
    }

    /**
     * @hibernate.property
     *  column="SEX"
     *  not-null="true"
     *  update="false"
     */
    public char getSex() {
        return sex;
    }
    void setSex(char sex) {
        this.sex=sex;
    }
}
```

Hibernate##########XDoclet#Hibernate################

5.5.2. JDK 5.0

JDK5.0########################## ######XDoclet#################### ########XDoclet##################IDE############## ###IntelliJ IDEA##JDK5.0################################## EJB## (JSR-220)### JDK5.0################# Hibernate3##JSR-220 (###API)# `EntityManager` ##### ##################### ######## *Hibernate Annotations* ################ ###EJB3(JSR-220)#Hibernate3######################

###EJB##################POJO########

```
@Entity(access = AccessType.FIELD)
public class Customer implements Serializable {

    @Id;
    Long id;

    String firstName;
    String lastName;
    Date birthday;

    @Transient
    Integer age;

    @Embedded
    private Address homeAddress;

    @OneToMany(cascade=CascadeType.ALL)
    @JoinColumn(name="CUSTOMER_ID")
    Set<Order> orders;

    // Getter/setter and business methods
}
```

JDK5.0########(#JSR-220)##################################
#########Hibernate#########################

5.6.

#######################################
###Hibernate##
##################################
####################Hibernate#######
################################
Hibernate#INSERT#UPDATE#SQL########## ############### SELECT SQL ########

################################### #5.1.9. #version######## # #5.1.10. #timestamp######## # #5.1.11. #property# ##################

`never` (#####) - ############### ########################

`insert` - #########################
######################### ###########################
#5.1.9. #version######## # #5.1.10. #timestamp######## #
##

`always` - ###############################

5.7.

Hibernate########################## #####################################CREATE#DROP#### Hibernate##################################### ## #### `java.sql.Statement.execute()` ############# ###SQL#####ALTER#INSERT########### ##########################2##############

1#######CREATE#DROP################################

```
<hibernate-mapping>
    ...
    <database-object>
        <create>CREATE TRIGGER my_trigger ...</create>
        <drop>DROP TRIGGER my_trigger</drop>
    </database-object>
</hibernate-mapping>
```

2#######CREATE#DROP################################## ########## `org.hibernate.mapping.AuxiliaryDatabaseObject` ######## #############

```
<hibernate-mapping>
    ...
    <database-object>
        <definition class="MyTriggerDefinition"/>
    </database-object>
</hibernate-mapping>
```

##################################### ##############################

```
<hibernate-mapping>
    ...
    <database-object>
        <definition class="MyTriggerDefinition"/>
        <dialect-scope name="org.hibernate.dialect.Oracle9Dialect"/>
        <dialect-scope name="org.hibernate.dialect.OracleDialect"/>
    </database-object>
</hibernate-mapping>
```

#6#

6.1.

###################### ############################## ####

```
public class Product {
    private String serialNumber;
    private Set parts = new HashSet();

    public Set getParts() { return parts; }
    void setParts(Set parts) { this.parts = parts; }
    public String getSerialNumber() { return serialNumber; }
    void setSerialNumber(String sn) { serialNumber = sn; }
}
```

############## java.util.Set# java.util.Collection#java.util.List# java.util.Map#java.util.SortedSet# java.util.SortedMap ######## ##################### ######################### org.hibernate.usertype.UserCollectionType ####################

HashSet ################### ####################### ##################################### ############# #### persist() ####################### Hibernate# HashSet #Hibernate### Set ############## #######################

```
Cat cat = new DomesticCat();
Cat kitten = new DomesticCat();
....
Set kittens = new HashSet();
kittens.add(kitten);
cat.setKittens(kittens);
session.persist(cat);
kittens = cat.getKittens(); // Okay, kittens collection is a Set
(HashSet) cat.getKittens(); // Error!
```

Hibernate################################## HashMap # HashSet# TreeMap# TreeSet# ArrayList ##########

############################ ############################ ###################### ################################# ################################# #################################### ################################### null################ ###Hibernate##############################

####################
#######Java################################
#################################

6.2.

####

###########################
#################################
##
#######################

####<set> ### Set ##

```
<class name="Product">
    <id name="serialNumber" column="productSerialNumber"/>
    <set name="parts">
        <key column="productSerialNumber" not-null="true"/>
        <one-to-many class="Part"/>
    </set>
</class>
```

######### <set> ### <list># <map>#<bag># <array>#<primitive-array> ###### ######<map> ###########

```
<map
    name="propertyName"                                           (1)
    table="table_name"                                            (2)
    schema="schema_name"                                          (3)
    lazy="true|extra|false"                                       (4)
    inverse="true|false"                                          (5)

 cascade="all|none|save-update|delete|all-delete-orphan|delet(6)e-
orphan"
    sort="unsorted|natural|comparatorClass"                       (7)
    order-by="column_name asc|desc"                               (8)
    where="arbitrary sql where condition"                         (9)
    fetch="join|select|subselect"                                 (10)
    batch-size="N"                                                (11)
    access="field|property|ClassName"                             (12)
    optimistic-lock="true|false"                                  (13)
    mutable="true|false"                                          (14)
    node="element-name|."
    embed-xml="true|false"
>

    <key .... />
    <map-key .... />
```

```
    <element .... />
</map>
```

(1) name ##################

(2) table ###### - ########################## ################

(3) schema ################### ##########################

(4) lazy ###### - ###### true# ####################################
#####extra-lazy####################
#extra-lazy##############################
######################

(5) inverse ###### - ###### false# ##############################

(6) cascade ###### - ###### none# ########################

(7) sort ############################ natural #######
####Comparator##########

(8) order-by #######JDK1.4### Map#Set#bag###############
################ ######## asc#desc #######

(9) where ######################## ###SQL#WHERE ########
##

(10) fetch###### - ###### select# #####################sequential select fetch## #############sequential subselect fetch###############

(11) batch-size ###### - ###### 1# #######################################

(12) access ###### - ###### property# #################################

(13) optimistic-lock###### - ###### true# ####################
##
###################################

(14) mutable###### - ###### true# false ##########################
########################

6.2.1.

######################### ##################################
################## ########### ####### ############ <key> ##############

#########null############### ######################
############################null############# ####not-null="true" ##################

```
<key column="productSerialNumber" not-null="true"/>
```

######## ON DELETE CASCADE ###########

```
<key column="productSerialNumber" on-delete="cascade"/>
```

<key> ############################

6.2.2.

#########Hibernate############################### ########################### ################ ## ################################# ## ####2##

####### ########## ######## ###########<element> ### <composite-element> ########################### <one-to-many> ### <many-to-many> ############## ##

6.2.3.

set#bag#################### ############## ############ ####### ########### List ######### #### Map ############## Map ######### <map-key> ################# <map-key-many-to-many> ####################### #### <composite-map-key> ######################### ################# integer ### <list-index> ############### #######################################0##############

```
<list-index
        column="column_name"                (1)
        base="0|1|..."/>
```

(1) `column_name` (required): The name of the column holding the collection index values.

(1) `base` (optional, defaults to `0`): The value of the index column that corresponds to the first element of the list or array.

```
<map-key
        column="column_name"                (1)
        formula="any SQL expression"        (2)
        type="type_name"                    (3)
        node="@attribute-name"
        length="N"/>
```

(1) `column` (optional): The name of the column holding the collection index values.

(2) `formula` (optional): A SQL formula used to evaluate the key of the map.
(3) `type` (reguired): The type of the map keys.

```
<map-key-many-to-many
        column="column_name"                (1)
        formula="any SQL expression"        (2)(3)
        class="ClassName"
/>
```

(1) `column` (optional): The name of the foreign key column for the collection index values.
(2) `formula` (optional): A SQL formula used to evaluate the foreign key of the map key.
(3) `class` (required): The entity class used as the map key.

################################# `List` ######## Hibernate# <*bag*> ################## bag########################### ###################SQL##########order by###########

6.2.4.

################### ########## ###### ################# ############ ## #######################

#############`<element>`#########

```
<element
        column="column_name"                     (1)
        formula="any SQL expression"             (2)
        type="typename"                          (3)
        length="L"
        precision="P"
        scale="S"
        not-null="true|false"
        unique="true|false"
        node="element-name"
/>
```

(1) `column` (optional): The name of the column holding the collection element values.
(2) `formula` (optional): An SQL formula used to evaluate the element.
(3) `type` (required): The type of the collection element.

A *many-to-many association* is specified using the `<many-to-many>` element.

```
<many-to-many
        column="column_name"                               (1)
        formula="any SQL expression"                       (2)
```

```
        class="ClassName"                                 (3)
        fetch="select|join"                               (4)
        unique="true|false"                               (5)
        not-found="ignore|exception"                      (6)
        entity-name="EntityName"                          (7)
        property-ref="propertyNameFromAssociatedClass"    (8)
        node="element-name"
        embed-xml="true|false"
    />
```

(1) `column` (optional): The name of the element foreign key column.

(2) `formula` (optional): An SQL formula used to evaluate the element foreign key value.

(3) `class` (required): The name of the associated class.

(4) `fetch` (optional - defaults to `join`): enables outer-join or sequential select fetching for this association. This is a special case; for full eager fetching (in a single `SELECT`) of an entity and its many-to-many relationships to other entities, you would enable `join` fetching not only of the collection itself, but also with this attribute on the `<many-to-many>` nested element.

(5) `unique` (optional): Enable the DDL generation of a unique constraint for the foreign-key column. This makes the association multiplicity effectively one to many.

(6) `not-found` (optional - defaults to `exception`): Specifies how foreign keys that reference missing rows will be handled: `ignore` will treat a missing row as a null association.

(7) `entity-name` (optional): The entity name of the associated class, as an alternative to `class`.

(8) `property-ref`: (optional) The name of a property of the associated class that is joined to this foreign key. If not specified, the primary key of the associated class is used.

############## ###String#set#########

```
<set name="names" table="person_names">
    <key column="person_id"/>
    <element column="person_name" type="string"/>
</set>
```

######bag#bag# `order-by` ####################

```
<bag name="sizes"
        table="item_sizes"
        order-by="size asc">
    <key column="item_id"/>
    <element column="size" type="integer"/>
</bag>
```

######### - ##############

```
<array name="addresses"
        table="PersonAddress"
        cascade="persist">
    <key column="personId"/>
    <list-index column="sortOrder"/>
    <many-to-many column="addressId" class="Address"/>
</array>
```

#######map

```
<map name="holidays"
        table="holidays"
        schema="dbo"
        order-by="hol_name asc">
    <key column="id"/>
    <map-key column="hol_name" type="string"/>
    <element column="hol_date" type="date"/>
</map>
```

########list##############

```
<list name="carComponents"
        table="CarComponents">
    <key column="carId"/>
    <list-index column="sortOrder"/>
    <composite-element class="CarComponent">
        <property name="price"/>
        <property name="type"/>
        <property name="serialNumber" column="serialNum"/>
    </composite-element>
</list>
```

6.2.5.

#######2##################
############Java#########################

- ################## 2#############################
- ##############################
  #################2#############

`Product` ## `Part` ###### `Part` ##
`<one-to-many>` #########################

```
<one-to-many
        class="ClassName"                                   (1)
        not-found="ignore|exception"                        (2)
        entity-name="EntityName"                            (3)
        node="element-name"
        embed-xml="true|false"
```

```
    />
```

(1) `class` ####: #########

(2) `not-found` ###### - ###### `exception`#: ##################################### `ignore` ##############################

(3) `entity-name` #######: `class` #################### `class` #########################

`<one-to-many>` ############################## ### #### ################

########: ## `<one-to-many>` ########### `NOT NULL`######### `<key>` ###### `not-null="true"` ####### ############ `inverse="true"` ####### ######## ######## ##########################

########`Part` ############# `partName`# ### `Part` #######map######## formula###########################

```
<map name="parts"
        cascade="all">
    <key column="productId" not-null="true"/>
    <map-key formula="partName"/>
    <one-to-many class="Part"/>
</map>
```

6.3.

6.3.1.

Hibernate# `java.util.SortedMap` # `java.util.SortedSet` ###################### #####################################

```
<set name="aliases"
            table="person_aliases"
            sort="natural">
    <key column="person"/>
    <element column="name" type="string"/>
</set>

<map name="holidays" sort="my.custom.HolidayComparator">
    <key column="year_id"/>
    <map-key column="hol_name" type="string"/>
    <element column="hol_date" type="date"/>
</map>
```

`sort` ########## `unsorted` # `natural` ####`java.util.Comparator` ###############

################## java.util.TreeSet # java.util.TreeMap ##########

############################### set # bag#map # order-by ######## ######JDK1.4###################### #LinkedHashSet ### LinkedHashMap############## ############SQL############

```
<set name="aliases" table="person_aliases" order-by="lower(name)
 asc">
    <key column="person"/>
    <element column="name" type="string"/>
</set>

<map name="holidays" order-by="hol_date, hol_name">
    <key column="year_id"/>
    <map-key column="hol_name" type="string"/>
    <element column="hol_date type="date"/>
</map>
```

order-by #####SQL#######HQL#################

########### filter() ####### #######criteria#################

```
sortedUsers = s.createFilter( group.getUsers(), "order by this.name"
 ).list();
```

6.3.2.

A *bidirectional association* allows navigation from both "ends" of the association. Two kinds of bidirectional association are supported:

one-to-many
: ###set#bag############

many-to-many
: ###set#bag###

2############################# ### *inverse* ########### ###################### #####inverse##

################### ##

```
<class name="Category">
    <id name="id" column="CATEGORY_ID"/>
    ...
    <bag name="items" table="CATEGORY_ITEM">
        <key column="CATEGORY_ID"/>
        <many-to-many class="Item" column="ITEM_ID"/>
```

```
    </bag>
</class>

<class name="Item">
    <id name="id" column="ITEM_ID"/>
    ...

    <!-- inverse end -->
    <bag name="categories" table="CATEGORY_ITEM" inverse="true">
        <key column="ITEM_ID"/>
        <many-to-many class="Category" column="CATEGORY_ID"/>
    </bag>
</class>
```

###inverse############## ######
####Hibernate###
######A##B############B##A##############
Java#################Java######################
#################Java##############

```

category.getItems().add(item);          // The category now "knows"
 about the relationship
item.getCategories().add(category);     // The item now "knows"
 about the relationship

session.persist(item);                   // The relationship won't
 be saved!
session.persist(category);               // The relationship will be
 saved
```

###inverse####################################

################## ################################# ###
`inverse="true"` #######

```
<class name="Parent">
    <id name="id" column="parent_id"/>
    ....
    <set name="children" inverse="true">
        <key column="parent_id"/>
        <one-to-many class="Child"/>
    </set>
</class>

<class name="Child">
    <id name="id" column="child_id"/>
    ....
    <many-to-one name="parent"
        class="Parent"
        column="parent_id"
        not-null="true"/>
```

```
</class>
```

inverse="true" #######

6.3.3.

<list> # <map> ### ######################## ## ############## inverse="true"

```
<class name="Parent">
    <id name="id" column="parent_id"/>
    ....
    <map name="children" inverse="true">
        <key column="parent_id"/>
        <map-key column="name"
            type="string"/>
        <one-to-many class="Child"/>
    </map>
</class>

<class name="Child">
    <id name="id" column="child_id"/>
    ....
    <property name="name"
        not-null="true"/>
    <many-to-one name="parent"
        class="Parent"
        column="parent_id"
        not-null="true"/>
</class>
```

############################ ######################### ##################################### ############## inverse="true" ############ ######################

```
<class name="Parent">
    <id name="id" column="parent_id"/>
    ....
    <map name="children">
        <key column="parent_id"
            not-null="true"/>
        <map-key column="name"
            type="string"/>
        <one-to-many class="Child"/>
    </map>
</class>

<class name="Child">
    <id name="id" column="child_id"/>
    ....
    <many-to-one name="parent"
```

```
        class="Parent"
        column="parent_id"
        insert="false"
        update="false"
        not-null="true"/>
</class>
```

#####################

6.3.4. 3###

3##########3############# 1############### Map #############

```
<map name="contracts">
    <key column="employer_id" not-null="true"/>
    <map-key-many-to-many column="employee_id" class="Employee"/>
    <one-to-many class="Contract"/>
</map>
```

```
<map name="connections">
    <key column="incoming_node_id"/>
    <map-key-many-to-many column="outgoing_node_id" class="Node"/>
    <many-to-many column="connection_id" class="Connection"/>
</map>
```

2#################################### #########

###composite##############################

6.3.5. `<idbag>`###

########################(####)##################
##
##################### ####################################
################# ## ########
#####Hibernate############################ #################

bag############ List#### Collection## <idbag> #############

```
<idbag name="lovers" table="LOVERS">
    <collection-id column="ID" type="long">
        <generator class="sequence"/>
    </collection-id>
    <key column="PERSON1"/>
    <many-to-many column="PERSON2" class="Person" fetch="join"/>
</idbag>
```

####### <idbag> ###############
####id############## ##############################
#######Hibernate###########################

<idbag> ############# ### <bag> ################# Hibernate#################### list#map#set#####################

########native ###id##### <idbag> #####################

6.4.

#################################

```
package eg;
import java.util.Set;

public class Parent {
    private long id;
    private Set children;

    public long getId() { return id; }
    private void setId(long id) { this.id=id; }

    private Set getChildren() { return children; }
    private void setChildren(Set children) { this.children=children;
 }

    ....
    ....
}
```

Child ###################### #####child########parent##############################

```
<hibernate-mapping>

    <class name="Parent">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <set name="children">
            <key column="parent_id"/>
            <one-to-many class="Child"/>
        </set>
    </class>

    <class name="Child">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <property name="name"/>
    </class>

</hibernate-mapping>
```

#######################

```
create table parent ( id bigint not null primary key )
create table child ( id bigint not null primary key, name
 varchar(255), parent_id bigint )
alter table child add constraint childfk0 (parent_id) references
 parent
```

##parent# ## ##########################

```
<hibernate-mapping>

    <class name="Parent">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <set name="children" inverse="true">
            <key column="parent_id"/>
            <one-to-many class="Child"/>
        </set>
    </class>

    <class name="Child">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <property name="name"/>
        <many-to-one name="parent" class="Parent" column="parent_id"
 not-null="true"/>
    </class>

</hibernate-mapping>
```

`NOT NULL` ############

```
create table parent ( id bigint not null primary key )
create table child ( id bigint not null
                     primary key,
                     name varchar(255),
                     parent_id bigint not null )
alter table child add constraint childfk0 (parent_id) references
 parent
```

################################### `<key>` # `NOT NULL` ##########

```
<hibernate-mapping>

    <class name="Parent">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <set name="children">
            <key column="parent_id" not-null="true"/>
            <one-to-many class="Child"/>
```

```
        </set>
    </class>

    <class name="Child">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <property name="name"/>
    </class>

</hibernate-mapping>
```

######child####parent##################

```
<hibernate-mapping>

    <class name="Parent">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <set name="children" table="childset">
            <key column="parent_id"/>
            <many-to-many class="Child" column="child_id"/>
        </set>
    </class>

    <class name="Child">
        <id name="id">
            <generator class="sequence"/>
        </id>
        <property name="name"/>
    </class>

</hibernate-mapping>
```

##################

```
create table parent ( id bigint not null primary key )
create table child ( id bigint not null primary key, name
 varchar(255) )
create table childset ( parent_id bigint not null,
                        child_id bigint not null,
                        primary key ( parent_id, child_id ) )
alter table childset add constraint childsetfk0 (parent_id)
 references parent
alter table childset add constraint childsetfk1 (child_id)
 references child
```

parent/child#################################### # 21. ###/## ########

##################################

#7#

7.1.

############################## ############################ ## #####Person # Address ######

###################### ###################

######not null############ ###Hibernate########### not null#############################

7.2.

7.2.1.

######## #####################

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <many-to-one name="address"
        column="addressId"
        not-null="true"/>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
</class>
```

```
create table Person ( personId bigint not null primary key,
 addressId bigint not null )
create table Address ( addressId bigint not null primary key )
```

7.2.2.

############# ############ ######################

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
```

```
    </id>
    <many-to-one name="address"
        column="addressId"
        unique="true"
        not-null="true"/>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
</class>
```

```
create table Person ( personId bigint not null primary key,
 addressId bigint not null unique )
create table Address ( addressId bigint not null primary key )
```

############ #######ID############
###############################

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
</class>

<class name="Address">
    <id name="id" column="personId">
        <generator class="foreign">
            <param name="property">person</param>
        </generator>
    </id>
    <one-to-one name="person" constrained="true"/>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table Address ( personId bigint not null primary key )
```

7.2.3.

############# ############ #############

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <set name="addresses">
        <key column="personId"
```

```
            not-null="true"/>
        <one-to-many class="Address"/>
    </set>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table Address ( addressId bigint not null primary key,
 personId bigint not null )
```

###################################

7.3.

7.3.1.

#################### ########### unique="true" ####################
##################

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <set name="addresses" table="PersonAddress">
        <key column="personId"/>
        <many-to-many column="addressId"
            unique="true"
            class="Address"/>
    </set>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table PersonAddress ( personId not null, addressId bigint not
 null primary key )
create table Address ( addressId bigint not null primary key )
```

7.3.2.

############### # #######################

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <join table="PersonAddress"
        optional="true">
        <key column="personId" unique="true"/>
        <many-to-one name="address"
            column="addressId"
            not-null="true"/>
    </join>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table PersonAddress ( personId bigint not null primary key,
 addressId bigint not null )
create table Address ( addressId bigint not null primary key )
```

7.3.3.

############### ########## ###########

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <join table="PersonAddress"
        optional="true">
        <key column="personId"
            unique="true"/>
        <many-to-one name="address"
            column="addressId"
            not-null="true"
            unique="true"/>
    </join>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
```

```
    </id>
</class>
```

```

create table Person ( personId bigint not null primary key )
create table PersonAddress ( personId bigint not null primary key,
 addressId bigint not null unique )
create table Address ( addressId bigint not null primary key )

```

7.3.4.

############ ######

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <set name="addresses" table="PersonAddress">
        <key column="personId"/>
        <many-to-many column="addressId"
            class="Address"/>
    </set>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
</class>
```

```

create table Person ( personId bigint not null primary key )
create table PersonAddress ( personId bigint not null, addressId
 bigint not null, primary key (personId, addressId) )
create table Address ( addressId bigint not null primary key )

```

7.4.

7.4.1. ###/###

######## ############ ############

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <many-to-one name="address"
        column="addressId"
        not-null="true"/>
```

```
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
    <set name="people" inverse="true">
        <key column="addressId"/>
        <one-to-many class="Person"/>
    </set>
</class>
```

```

create table Person ( personId bigint not null primary key,
 addressId bigint not null )
create table Address ( addressId bigint not null primary key )

```

List ############################## ##### key #### not null ##### ################################# ############ #update="false" ## insert="false" ##############inverse######

```
<class name="Person">
   <id name="id"/>
   ...
   <many-to-one name="address"
      column="addressId"
      not-null="true"
      insert="false"
      update="false"/>
</class>

<class name="Address">
   <id name="id"/>
   ...
   <list name="people">
      <key column="addressId" not-null="true"/>
      <list-index column="peopleIdx"/>
      <one-to-many class="Person"/>
   </list>
</class>
```

########## NOT NULL ####### ############ <key> ### not-null="true" ########### ####### <column> ######### <key> ### not-null="true" #################

7.4.2.

############# ##########

```
<class name="Person">
```

```
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <many-to-one name="address"
        column="addressId"
        unique="true"
        not-null="true"/>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
   <one-to-one name="person"
        property-ref="address"/>
</class>
```

```
create table Person ( personId bigint not null primary key,
 addressId bigint not null unique )
create table Address ( addressId bigint not null primary key )
```

############ ####ID############

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <one-to-one name="address"/>
</class>

<class name="Address">
    <id name="id" column="personId">
        <generator class="foreign">
            <param name="property">person</param>
        </generator>
    </id>
    <one-to-one name="person"
        constrained="true"/>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table Address ( personId bigint not null primary key )
```

7.5.

7.5.1. ###/###

############### ### inverse="true" ####################
#################

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <set name="addresses"
        table="PersonAddress">
        <key column="personId"/>
        <many-to-many column="addressId"
            unique="true"
            class="Address"/>
    </set>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
    <join table="PersonAddress"
        inverse="true"
        optional="true">
        <key column="addressId"/>
        <many-to-one name="person"
            column="personId"
            not-null="true"/>
    </join>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table PersonAddress ( personId bigint not null, addressId
 bigint not null primary key )
create table Address ( addressId bigint not null primary key )
```

7.5.2.

############### ###############

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <join table="PersonAddress"
        optional="true">
```

```
        <key column="personId"
            unique="true"/>
        <many-to-one name="address"
            column="addressId"
            not-null="true"
            unique="true"/>
    </join>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
    <join table="PersonAddress"
        optional="true"
        inverse="true">
        <key column="addressId"
            unique="true"/>
        <many-to-one name="person"
            column="personId"
            not-null="true"
            unique="true"/>
    </join>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table PersonAddress ( personId bigint not null primary key,
 addressId bigint not null unique )
create table Address ( addressId bigint not null primary key )
```

7.5.3.

############ ######

```
<class name="Person">
    <id name="id" column="personId">
        <generator class="native"/>
    </id>
    <set name="addresses" table="PersonAddress">
        <key column="personId"/>
        <many-to-many column="addressId"
            class="Address"/>
    </set>
</class>

<class name="Address">
    <id name="id" column="addressId">
        <generator class="native"/>
    </id>
    <set name="people" inverse="true" table="PersonAddress">
        <key column="addressId"/>
```

```
        <many-to-many column="personId"
            class="Person"/>
    </set>
</class>
```

```
create table Person ( personId bigint not null primary key )
create table PersonAddress ( personId bigint not null, addressId
 bigint not null, primary key (personId, addressId) )
create table Address ( addressId bigint not null primary key )
```

7.6.

########## ### #### ############SQL########## #################### ####accountNumber, effectiveEndDate, effectiveStartDate ######account################### ################

```
<properties name="currentAccountKey">
    <property name="accountNumber" type="string" not-null="true"/>
    <property name="currentAccount" type="boolean">
        <formula>case when effectiveEndDate is null then 1 else 0
 end</formula>
    </property>
</properties>
<property name="effectiveEndDate" type="date"/>
<property name="effectiveStateDate" type="date" not-null="true"/>
```

####### #### ###### #effectiveEndDate #null################ ############

```
<many-to-one name="currentAccountInfo"
        property-ref="currentAccountKey"
        class="AccountInfo">
    <column name="accountNumber"/>
    <formula>'1'</formula>
</many-to-one>
```

##########Employee##### # Organization#### ##### Employment#### ###################### ########################## ####### ##### ######## ###### startDate #######################

```
<join>
    <key column="employeeId"/>
    <subselect>
        select employeeId, orgId
        from Employments
        group by orgId
        having startDate = max(startDate)
```

```
    </subselect>
    <many-to-one name="mostRecentEmployer"
            class="Organization"
            column="orgId"/>
</join>
```

############# ##############HQL#criteria################

#8#

####### #####Hibernate############ #################

8.1.

##################################
##
##
#######Person############

```
public class Person {
    private java.util.Date birthday;
    private Name name;
    private String key;
    public String getKey() {
        return key;
    }
    private void setKey(String key) {
        this.key=key;
    }
    public java.util.Date getBirthday() {
        return birthday;
    }
    public void setBirthday(java.util.Date birthday) {
        this.birthday = birthday;
    }
    public Name getName() {
        return name;
    }
    public void setName(Name name) {
        this.name = name;
    }
    ......
    ......
}
```

```
public class Name {
    char initial;
    String first;
    String last;
    public String getFirst() {
        return first;
    }
    void setFirst(String first) {
        this.first = first;
    }
    public String getLast() {
        return last;
    }
    void setLast(String last) {
```

```
        this.last = last;
    }
    public char getInitial() {
        return initial;
    }
    void setInitial(char initial) {
        this.initial = initial;
    }
}
```

###Name # Person ########### ################# Name ##########getter# setter## ##############

####################

```
<class name="eg.Person" table="person">
    <id name="Key" column="pid" type="string">
        <generator class="uuid"/>
    </id>
    <property name="birthday" type="date"/>
    <component name="Name" class="eg.Name"> <!-- class attribute
 optional -->
        <property name="initial"/>
        <property name="first"/>
        <property name="last"/>
    </component>
</class>
```

Person##### pid# birthday# initial# first# last #########

##################################### ##########Person#####################Person####### "######"####name##################### ########null########### ##### ### ##########################Hibernate################## null###############null######### #################

###############Hibernate##################many-to-one### ## ##### #Hibernate##

<component> ################################ ########### <parent> #############

```
<class name="eg.Person" table="person">
    <id name="Key" column="pid" type="string">
        <generator class="uuid"/>
    </id>
    <property name="birthday" type="date"/>
    <component name="Name" class="eg.Name" unique="true">
```

```
        <parent name="namedPerson"/> <!-- reference back to the
 Person -->
        <property name="initial"/>
        <property name="first"/>
        <property name="last"/>
    </component>
</class>
```

8.2.

Hibernate############################### Name ###### <element> ### <composite-element> ############# #######################

```
<set name="someNames" table="some_names" lazy="true">
    <key column="id"/>
    <composite-element class="eg.Name"> <!-- class attribute
 required -->
        <property name="initial"/>
        <property name="first"/>
        <property name="last"/>
    </composite-element>
</set>
```

##: ############ Set ######## equals() # hashCode() ################

<nested-composite-element> # ## ########one-to-many####################### ####################################### ######Java##

<set> ###############################null##### #########################Hibernate############### #####################################null############ #################################### ############not-null############### <list>#<map># <bag>#<idbag>

############################## <many-to-one> ##### ################## #################################### ################# #### Order ###Item ########## purchaseDate#price#quantity ############

```
<class name="eg.Order" .... >
    ....
    <set name="purchasedItems" table="purchase_items" lazy="true">
        <key column="order_id">
        <composite-element class="eg.Purchase">
```

```
            <property name="purchaseDate"/>
            <property name="price"/>
            <property name="quantity"/>
            <many-to-one name="item" class="eg.Item"/> <!-- class
 attribute is optional -->
        </composite-element>
    </set>
</class>
```

###########################purchase################
Purchase #### Order
#set######### ### Item ################

3########4###########

```
<class name="eg.Order" .... >
    ....
    <set name="purchasedItems" table="purchase_items" lazy="true">
        <key column="order_id">
        <composite-element class="eg.OrderLine">
            <many-to-one name="purchaseDetails class="eg.Purchase"/>
            <many-to-one name="item" class="eg.Item"/>
        </composite-element>
    </set>
</class>
```

############################# ###########################

8.3. Map###################

<composite-map-key> ### Map ##################
###################### hashCode() # equals() ###################

8.4.

####################################
##############################

- java.io.Serializable ###############
- ###########################equals() # hashCode() ################

##: Hibernate3#####2##################### ####################

############ IdentifierGenerator ###############
#################################

<id> ####### <composite-id> ### ####### <key-property>
########### #######OrderLine #### Order ######### ################

```
<class name="OrderLine">
```

```
    <composite-id name="id" class="OrderLineId">
        <key-property name="lineId"/>
        <key-property name="orderId"/>
        <key-property name="customerId"/>
    </composite-id>

    <property name="name"/>

    <many-to-one name="order" class="Order"
            insert="false" update="false">
        <column name="orderId"/>
        <column name="customerId"/>
    </many-to-one>
    ....

</class>
```

#####OrderLine #####################
############################## OrderLine #####################

```
<many-to-one name="orderLine" class="OrderLine">
<!-- the "class" attribute is optional, as usual -->
    <column name="lineId"/>
    <column name="orderId"/>
    <column name="customerId"/>
</many-to-one>
```

#<column> ########## column ### ##########

OrderLine ## many-to-many ### ############

```
<set name="undeliveredOrderLines">
    <key column name="warehouseId"/>
    <many-to-many class="OrderLine">
        <column name="lineId"/>
        <column name="orderId"/>
        <column name="customerId"/>
    </many-to-many>
</set>
```

Order ### OrderLine ######## ###########

```
<set name="orderLines" inverse="true">
    <key>
        <column name="orderId"/>
        <column name="customerId"/>
    </key>
    <one-to-many class="OrderLine"/>
</set>
```

#<one-to-many> ####################

OrderLine ################## ##################

```
<class name="OrderLine">
    ....
    ....
    <list name="deliveryAttempts">
        <key>   <!-- a collection inherits the composite key type
 -->
            <column name="lineId"/>
            <column name="orderId"/>
            <column name="customerId"/>
        </key>
        <list-index column="attemptId" base="1"/>
        <composite-element class="DeliveryAttempt">
            ...
        </composite-element>
    </set>
</class>
```

8.5.

Map ###################

```
<dynamic-component name="userAttributes">
    <property name="foo" column="FOO" type="string"/>
    <property name="bar" column="BAR" type="integer"/>
    <many-to-one name="baz" class="Baz" column="BAZ_ID"/>
</dynamic-component>
```

<dynamic-component> ############## <component> ##bean#### ############DOM################################## ####Configuration #########Hibernate################### #################

#9#

9.1. 3####

Hibernate#3#########################

- #############table-per-class-hierarchy#
- table per subclass
- #############table-per-concrete-class#

###4#####Hibernate################################

- ###########

####Hibernate###### `<class>` #### `<subclass>` ######`<joined-subclass>` ###### `<union-subclass>` ###################### `<subclass>` ### `<join>` ############# ## `<class>` ##### table-per-hierarchy ### table-per-subclass

`subclass`, `union-subclass` # `joined-subclass` ################################## `hibernate-mapping` ########## ## #################################### `extends` ################ #################################### Hibernate3####extends#################################### ######################### ##)

```
<hibernate-mapping>
    <subclass name="DomesticCat" extends="Cat"
discriminator-value="D">
         <property name="name" type="string"/>
    </subclass>
</hibernate-mapping>
```

9.1.1. #############table-per-class-hierarchy#

############ `Payment` ######### `CreditCardPayment`#`CashPayment`# `ChequePayment` ####################### ###########

```
<class name="Payment" table="PAYMENT">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="native"/>
```

```
    </id>
    <discriminator column="PAYMENT_TYPE" type="string"/>
    <property name="amount" column="AMOUNT"/>
    ...
    <subclass name="CreditCardPayment" discriminator-value="CREDIT">
        <property name="creditCardType" column="CCTYPE"/>
        ...
    </subclass>
    <subclass name="CashPayment" discriminator-value="CASH">
        ...
    </subclass>
    <subclass name="ChequePayment" discriminator-value="CHEQUE">
        ...
    </subclass>
</class>
```

################## ######################### CCTYPE ##################### NOT NULL #########

9.1.2. ##############table-per-subclass#

table-per-subclass #################

```
<class name="Payment" table="PAYMENT">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="native"/>
    </id>
    <property name="amount" column="AMOUNT"/>
    ...
    <joined-subclass name="CreditCardPayment"
 table="CREDIT_PAYMENT">
        <key column="PAYMENT_ID"/>
        <property name="creditCardType" column="CCTYPE"/>
        ...
    </joined-subclass>
    <joined-subclass name="CashPayment" table="CASH_PAYMENT">
        <key column="PAYMENT_ID"/>
        ...
    </joined-subclass>
    <joined-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
        <key column="PAYMENT_ID"/>
        ...
    </joined-subclass>
</class>
```

#########4############3############ ############################# #####################

9.1.3. ####### table-per-subclass

Hibernate# table-per-subclass ####discriminator########################

Hibernate###O/R######table-per-subclass ############ ####################discriminator########### ## table-per-subclass ###discriminator########### `<subclass>` # `<join>` ######################

```
<class name="Payment" table="PAYMENT">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="native"/>
    </id>
    <discriminator column="PAYMENT_TYPE" type="string"/>
    <property name="amount" column="AMOUNT"/>
    ...
    <subclass name="CreditCardPayment" discriminator-value="CREDIT">
        <join table="CREDIT_PAYMENT">
            <key column="PAYMENT_ID"/>
            <property name="creditCardType" column="CCTYPE"/>
            ...
        </join>
    </subclass>
    <subclass name="CashPayment" discriminator-value="CASH">
        <join table="CASH_PAYMENT">
            <key column="PAYMENT_ID"/>
            ...
        </join>
    </subclass>
    <subclass name="ChequePayment" discriminator-value="CHEQUE">
        <join table="CHEQUE_PAYMENT" fetch="select">
            <key column="PAYMENT_ID"/>
            ...
        </join>
    </subclass>
</class>
```

`fetch="select"` #### ######################### ###### `ChequePayment`

9.1.4. table-per-subclass # table-per-class-hierarchy

##############table-per-hierarchy # table-per-subclass ### ##############

```
<class name="Payment" table="PAYMENT">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="native"/>
    </id>
    <discriminator column="PAYMENT_TYPE" type="string"/>
    <property name="amount" column="AMOUNT"/>
    ...
    <subclass name="CreditCardPayment" discriminator-value="CREDIT">
        <join table="CREDIT_PAYMENT">
```

```
            <property name="creditCardType" column="CCTYPE"/>
            ...
        </join>
    </subclass>
    <subclass name="CashPayment" discriminator-value="CASH">
        ...
    </subclass>
    <subclass name="ChequePayment" discriminator-value="CHEQUE">
        ...
    </subclass>
</class>
```

######################## Payment ##### ############# <many-to-one> ##############

```
<many-to-one name="payment" column="PAYMENT_ID" class="Payment"/>
```

9.1.5. #############table-per-concrete-class#

table-per-concrete-class ###################2###### 1### <union-subclass> ##########

```
<class name="Payment">
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class="sequence"/>
    </id>
    <property name="amount" column="AMOUNT"/>
    ...
    <union-subclass name="CreditCardPayment" table="CREDIT_PAYMENT">
        <property name="creditCardType" column="CCTYPE"/>
        ...
    </union-subclass>
    <union-subclass name="CashPayment" table="CASH_PAYMENT">
        ...
    </union-subclass>
    <union-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
        ...
    </union-subclass>
</class>
```

########3#####################################
############################

################## ################################## ## #Hibernate####################### <union-subclass> #### table-per-concrete-class ##################### ##################### union subclass ####################

##################abstract="true"##########
#######################################
#######################(########### PAYMENT)#

9.1.6. ################ table-per-concrete-class

############################

```
<class name="CreditCardPayment" table="CREDIT_PAYMENT">
    <id name="id" type="long" column="CREDIT_PAYMENT_ID">
        <generator class="native"/>
    </id>
    <property name="amount" column="CREDIT_AMOUNT"/>
    ...
</class>

<class name="CashPayment" table="CASH_PAYMENT">
    <id name="id" type="long" column="CASH_PAYMENT_ID">
        <generator class="native"/>
    </id>
    <property name="amount" column="CASH_AMOUNT"/>
    ...
</class>

<class name="ChequePayment" table="CHEQUE_PAYMENT">
    <id name="id" type="long" column="CHEQUE_PAYMENT_ID">
        <generator class="native"/>
    </id>
    <property name="amount" column="CHEQUE_AMOUNT"/>
    ...
</class>
```

Payment ####################################
####Payment ##
###############XML################## ### DOCTYPE ###### [<!ENTITY allproperties SYSTEM "allproperties.xml">] ########### &allproperties;#

############Hibernate##################SQL UNION ##########

###############Payment ############## ###<any> ##############

```
<any name="payment" meta-type="string" id-type="long">
    <meta-value value="CREDIT" class="CreditCardPayment"/>
    <meta-value value="CASH" class="CashPayment"/>
    <meta-value value="CHEQUE" class="ChequePayment"/>
    <column name="PAYMENT_CLASS"/>
    <column name="PAYMENT_ID"/>
</any>
```

9.1.7.

######################### ######### <class> #################
Payment #################
########### Payment ######################## #############

```
<class name="CreditCardPayment" table="CREDIT_PAYMENT">
    <id name="id" type="long" column="CREDIT_PAYMENT_ID">
        <generator class="native"/>
    </id>
    <discriminator column="CREDIT_CARD" type="string"/>
    <property name="amount" column="CREDIT_AMOUNT"/>
    ...
    <subclass name="MasterCardPayment" discriminator-value="MDC"/>
    <subclass name="VisaPayment" discriminator-value="VISA"/>
</class>

<class name="NonelectronicTransaction" table="NONELECTRONIC_TXN">
    <id name="id" type="long" column="TXN_ID">
        <generator class="native"/>
    </id>
    ...
    <joined-subclass name="CashPayment" table="CASH_PAYMENT">
        <key column="PAYMENT_ID"/>
        <property name="amount" column="CASH_AMOUNT"/>
        ...
    </joined-subclass>
    <joined-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
        <key column="PAYMENT_ID"/>
        <property name="amount" column="CHEQUE_AMOUNT"/>
        ...
    </joined-subclass>
</class>
```

##########Payment ############# ###Payment #####################
from Payment ####### Hibernate##### CreditCardPayment
##CreditCardPayment#######Payment ########## ####CashPayment
#ChequePayment ############# NonelectronicTransaction #############

9.2.

table-per-concrete-class #############################
#############<union-subclass> ########## #############

########Hibernate####table-per-concrete-class######
#######################

9.1.

####	######	######	######	######	Polymorphic **load()/ get()**	######	######	####
table per class-hierarchy	`<many-to-one>`	`<one-to-one>`	`<one-to-many>`	`<many-to-many>`	`s.get(Payment.class, id)`	`from Payment p`	`from Order o join o.payment p`	####
table per subclass	`<many-to-one>`	`<one-to-one>`	`<one-to-many>`	`<many-to-many>`	`s.get(Payment.class, id)`	`from Payment p`	`from Order o join o.payment p`	####
table per concrete-class (union-subclass)	`<many-to-one>`	`<one-to-one>`	`<one-to-many>` (for `inverse="true"` only)	`<many-to-many>`	`s.get(Payment.class, id)`	`from Payment p`	`from Order o join o.payment p`	####
table per concrete class (implicit polymorphism)	`<any>`	########	########	`<many-to-any>`	`s.createCriteria(Payment.class).add( Restrictions.idEq(id) ).uniqueResult()`	`from Payment p`	########	####

#10#

Hibernate ##########/########################
################################# ####### ####
####JDBC/SQL#########SQL #
Java###

#######Hibernate######################## ##
###########SQL########################
#########Hibernate################################## ##########

10.1. Hibernate#############

Hibernate################################

- *###(Transient)* - `new` ####### ############### Hibernate# `Session` ################### ###(transient)### ### ### ################### ##########(persistent)###########Hibernate# `Session` ######## #############SQL######Hibernate#########
- *###(Persistent)* - ################## ########################### ################################ ##### `Session` ################# Hibernate#######Unit of work########## ######################### ######################## ##########(transient)################# `UPDATE` ## `DELETE` #########
- *##(Detached)* - ######################### ########### `Session` ####################### ########################## ### ## ### `Session` ######### ## ################### ######## ###################*application transactions*# ######## ###########################

#############################Hibernate########## #########

10.2.

############################# Hibernate## ###*(transient)* ######## #################################### ####*(persistent)* ######

```
DomesticCat fritz = new DomesticCat();
fritz.setColor(Color.GINGER);
```

```
fritz.setSex('M');
fritz.setName("Fritz");
Long generatedId = (Long) sess.save(fritz);
```

Cat ###################### save() ######### #########
cat ################# Cat ################# assigned
#################### save() ########################### save()
######EJB3 ############# persist() ###########

- persist() makes a transient instance persistent. However, it doesn't guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time. persist() also guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. This is useful in long-running conversations with an extended Session/persistence context.
- save() does guarantee to return an identifier. If an INSERT has to be executed to get the identifier (e.g. "identity" generator, not "sequence"), this INSERT happens immediately, no matter if you are inside or outside of a transaction. This is problematic in a long-running conversation with an extended Session/persistence context.

############### save() ######### ##################

```
DomesticCat pk = new DomesticCat();
pk.setColor(Color.TABBY);
pk.setSex('F');
pk.setName("PK");
pk.setKittens( new HashSet() );
pk.addKitten(fritz);
sess.save( pk, new Long(1234) );
```

############################# ############# kittens #############
######### NOT NULL ########### ##################################
##################### #### NOT NULL ####################### save()
####### ################

##################### Hibernate# #######*(transitive persistence)* ###
################################ #### NOT NULL ###############
Hibernate##

10.3.

################################# Session # load() ################
load() ##Class ############# ###############################
################(persistent)#####

```
Cat fritz = (Cat) sess.load(Cat.class, generatedId);
```

```
// you need to wrap primitive identifiers
long id = 1234;
DomesticCat pk = (DomesticCat) sess.load( DomesticCat.class, new
 Long(id) );
```

##

```
Cat cat = new DomesticCat();
// load pk's state into cat
sess.load( cat, new Long(pkId) );
Set kittens = cat.getKittens();
```

DB############ `load()` ########## ###############
############################ `load()`
#################
#################
`batch-size` ###################### ###########################

######################## `get()` #############
#################################null######

```
Cat cat = (Cat) sess.get(Cat.class, id);
if (cat==null) {
    cat = new Cat();
    sess.save(cat, id);
}
return cat;
```

`LockMode` ##### `SELECT ... FOR UPDATE`###SQL#
######################## #######API#################

```
Cat cat = (Cat) sess.get(Cat.class, id, LockMode.UPGRADE);
```

################# `lock` # `all` # ################################ `FOR UPDATE` ### #### ############

`refresh()` #####################################
############### ########################
##

```
sess.save(cat);
sess.flush(); //force the SQL INSERT
sess.refresh(cat); //re-read the state (after the trigger executes)
```

###########################Hibernate##########
#########################SQL# `SELECT` ########### #### #######
###############19.1. ######## #########

10.4.

#######################################
Hibernate##########################
(HQL)############
########################Hibernate######Criteria#Example#####(QBC#QBE##
##########ResultSet############Hibernate###############
#############SQL##################

10.4.1.

HQL#######SQL##### `org.hibernate.Query` ##################
#########################ResultSet########
####################### ### `Query` ########### ##### `Session`
###########

```
List cats = session.createQuery(
    "from Cat as cat where cat.birthdate < ?")
    .setDate(0, date)
    .list();

List mothers = session.createQuery(
    "select mother from Cat as cat join cat.mother as mother where
 cat.name = ?")
    .setString(0, name)
    .list();

List kittens = session.createQuery(
    "from Cat as cat where cat.mother = ?")
    .setEntity(0, pk)
    .list();

Cat mother = (Cat) session.createQuery(
    "select cat.mother from Cat as cat where cat = ?")
    .setEntity(0, izi)
    .uniqueResult();]]

Query mothersWithKittens = (Cat) session.createQuery(
    "select mother from Cat as mother left join fetch
 mother.kittens");
Set uniqueMothers = new HashSet(mothersWithKittens.list());
```

######## `list()` ###################
####################################
#####################################
##########1###################### `uniqueResult()` #################
###################################### #######################
######################(############ ##### `Set` #################

10.4.1.1.

`iterate()` ####################
####################################
#####################
`iterate()` ## `list()` ###############################
################################select####
######################n##select######

```
// fetch ids
Iterator iter = sess.createQuery("from eg.Qux q order by
 q.likeliness").iterate();
while ( iter.hasNext() ) {
    Qux qux = (Qux) iter.next();  // fetch the object
    // something we couldnt express in the query
    if ( qux.calculateComplicatedAlgorithm() ) {
        // delete the current instance
        iter.remove();
        // dont need to process the rest
        break;
    }
}
```

10.4.1.2. #########tuple#######

Hibernate############################## ######################

```
Iterator kittensAndMothers = sess.createQuery(
            "select kitten, mother from Cat kitten join
 kitten.mother mother")
            .list()
            .iterator();

while ( kittensAndMothers.hasNext() ) {
    Object[] tuple = (Object[]) kittensAndMothers.next();
    Cat kitten = (Cat) tuple[0];
    Cat mother = (Cat) tuple[1];
    ....
}
```

10.4.1.3.

`select`
SQL############################
##################################

```
Iterator results = sess.createQuery(
        "select cat.color, min(cat.birthdate), count(cat) from Cat
 cat " +
```

```
        "group by cat.color")
        .list()
        .iterator();

while ( results.hasNext() ) {
    Object[] row = (Object[]) results.next();
    Color type = (Color) row[0];
    Date oldest = (Date) row[1];
    Integer count = (Integer) row[2];
    .....
}
```

10.4.1.4.

`Query` #############JDBC##### `?` ############################## *JDBC#####Hibernate######################* ######################## `:name` ######### ####################

- ##############################
- ####################
- ##########

```
//named parameter (preferred)
Query q = sess.createQuery("from DomesticCat cat where cat.name =
 :name");
q.setString("name", "Fritz");
Iterator cats = q.iterate();
```

```
//positional parameter
Query q = sess.createQuery("from DomesticCat cat where cat.name =
 ?");
q.setString(0, "Izi");
Iterator cats = q.iterate();
```

```
//named parameter list
List names = new ArrayList();
names.add("Izi");
names.add("Fritz");
Query q = sess.createQuery("from DomesticCat cat where cat.name in
 (:namesList)");
q.setParameterList("namesList", names);
List cats = q.list();
```

10.4.1.5.

ResultSet##################################### ####### `Query` ###################

```
Query q = sess.createQuery("from DomesticCat cat");
```

```
q.setFirstResult(20);
q.setMaxResults(10);
List cats = q.list();
```

#########DBMS#######SQL#########Hibernate########

10.4.1.6.

JDBC############# `ResultSet` ###########
`Query` ############# `ScrollableResults` #######
###

```
Query q = sess.createQuery("select cat.name, cat from DomesticCat
 cat " +
                            "order by cat.name");
ScrollableResults cats = q.scroll();
if ( cats.first() ) {

    // find the first name on each page of an alphabetical list of
 cats by name
    firstNamesOfPages = new ArrayList();
    do {
        String name = cats.getString(0);
        firstNamesOfPages.add(name);
    }
    while ( cats.scroll(PAGE_SIZE) );

    // Now get the first page of cats
    pageOfCats = new ArrayList();
    cats.beforeFirst();
    int i=0;
    while( ( PAGE_SIZE > i++ ) && cats.next() ) pageOfCats.add(
 cats.get(1) );

}
cats.close()
```

###
######################### `setMaxResult()` / `setFirstResult()` ########

10.4.1.7.

################################### ############################
`CDATA` ###### ####################

```
<query name="ByNameAndMaximumWeight"><![CDATA[
    from eg.DomesticCat as cat
        where cat.name = ?
        and cat.weight > ?
] ]></query>
```

####################################

```
Query q = sess.getNamedQuery("ByNameAndMaximumWeight");
q.setString(0, name);
q.setInt(1, minWeight);
List cats = q.list();
```

###
#############SQL################ ##########################
Hibernate#############

`<hibernate-mapping>` #################
################################# `<class>` ###
##
`eg.Cat.ByNameAndMaximumWeight`

10.4.2.

#########################
`this`

```
Collection blackKittens = session.createFilter(
    pk.getKittens(),
    "where this.color = ?")
    .setParameter( Color.BLACK,
 Hibernate.custom(ColorUserType.class) )
    .list()
);
```

###########Bag####################################
#####################"filter"################# ##################

`from`
###########################

```
Collection blackKittenMates = session.createFilter(
    pk.getKittens(),
    "select this.mate where this.color = eg.Color.BLACK.intValue")
    .list();
```

#################### ###################################

```
Collection tenKittens = session.createFilter(
    mother.getKittens(), "")
    .setFirstResult(0).setMaxResults(10)
    .list();
```

10.4.3.

HQL###################################API#### ###################### #############Hibernate##### `Criteria` ###API#########

```
Criteria crit = session.createCriteria(Cat.class);
crit.add( Restrictions.eq( "color", eg.Color.BLACK ) );
crit.setMaxResults(10);
List cats = crit.list();
```

`Criteria` # `Example` API##### # 15. *Criteria### ##########*

10.4.4. #####SQL####

`createSQLQuery()` #####SQL################# ####Hibernate##ResultSet ####################### `session.connection()` ################JDBC `Connection` #################### ###Hibernate API###############SQL#####################

```
List cats = session.createSQLQuery("SELECT {cat.*} FROM CAT {cat}
 WHERE ROWNUM<10")
    .addEntity("cat", Cat.class)
.list();
```

```
List cats = session.createSQLQuery(
    "SELECT {cat}.ID AS {cat.id}, {cat}.SEX AS {cat.sex}, " +
           "{cat}.MATE AS {cat.mate}, {cat}.SUBCLASS AS {cat.class},
 ... " +
    "FROM CAT {cat} WHERE ROWNUM<10")
    .addEntity("cat", Cat.class)
.list()
```

SQL#####Hibernate### Hibernate##########SQL############ # 16. *#####SQL ##########*

10.5.

############ ### `Session` ##### ### ################ `Session` # ##### #################################### ##################### `update()` ######################### ########### ################################# `load()` ## `Session` #########################

```
DomesticCat cat = (DomesticCat) sess.load( Cat.class, new Long(69)
 );
```

```
cat.setName("PK");
sess.flush();  // changes to cat are automatically detected and
 persisted
```

#################SQL# SELECT ################### SQL# UPDATE ################################ ############### #####Hibernate###############################detached######

Note that Hibernate does not offer its own API for direct execution of `UPDATE` or `DELETE` statements. Hibernate is a state management *service, you don't have to think in* statements *to use it. JDBC is a perfect API for executing SQL statements, you can get a JDBC `Connection` at any time by calling `session.connection()`. Furthermore, the notion of mass operations conflicts with object/relational mapping for online transaction processing-oriented applications. Future versions of Hibernate may however provide special mass operation functions. See # 13. ã##ã##ã##å#¦ç## for some possible batch operation tricks.*

10.6.

########################### #####################################UI##### ################################### #################################### "#####" ###################################

Hibernate## `Session.update()` # `Session.merge()` ##### ###################################

```
// in the first session
Cat cat = (Cat) firstSession.load(Cat.class, catId);
Cat potentialMate = new Cat();
firstSession.save(potentialMate);

// in a higher layer of the application
cat.setMate(potentialMate);

// later, in a new session
secondSession.update(cat);  // update cat
secondSession.update(mate); // update mate
```

###`catId` ### `Cat` #### `secondSession` #####################################

#################################### ####### `update()` ######## ####################################### `merge()` ######## ## ### `update()` ###############################

###################################### update() ####################### ## ### ####### ################## #10.11. ######### ##########

####################################
############################

```
//just reassociate:
sess.lock(fritz, LockMode.NONE);
//do a version check, then reassociate:
sess.lock(izi, LockMode.READ);
//do a version check, using SELECT ... FOR UPDATE, then reassociate:
sess.lock(pk, LockMode.UPGRADE);
```

lock() ####### LockMode ############### ####API############################### ########## lock() ###############

####################??? ########

10.7.

Hibernate#######2################################# #################################### ###########################/################## saveOrUpdate() ####################

```
// in the first session
Cat cat = (Cat) firstSession.load(Cat.class, catID);

// in a higher tier of the application
Cat mate = new Cat();
cat.setMate(mate);

// later, in a new session
secondSession.saveOrUpdate(cat);   // update existing state (cat has
 a non-null id)
secondSession.saveOrUpdate(mate);  // save the new instance (mate
 has a null id)
```

saveOrUpdate() ########## ####################### ### update() # saveOrUpdate() # merge() ############ #######################################

update() # saveOrUpdate() ########

- #################################
- #######UI########

- ####################
- #######################
- ##########2######### update() #####################

saveOrUpdate() ############

- ####################################
- ###################################### ########
- ##################### save() ####
- ## #########
  save() ####
- ######## <version> # <timestamp> #####
  ##############################
  ################################## ######### save() ####
- ################## update() ####

merge()

- ##
  ###################################
- ################################
  ##
- ################
- ############################## ##############

10.8.

Session.delete() #########################
###################################### ##### delete()
#############################

```
sess.delete(cat);
```

###
################################ NOT NULL ################
##

10.9.

############################### ##########################

```
//retrieve a cat from one database
Session session1 = factory1.openSession();
Transaction tx1 = session1.beginTransaction();
Cat cat = session1.get(Cat.class, catId);
tx1.commit();
```

```
session1.close();

//reconcile with a second database
Session session2 = factory2.openSession();
Transaction tx2 = session2.beginTransaction();
session2.replicate(cat, ReplicationMode.LATEST_VERSION);
tx2.commit();
session2.close();
```

########################### `replicate()` ##############
`ReplicationMode` #######

- `ReplicationMode.IGNORE` - #########################
################
- `ReplicationMode.OVERWRITE` - ############## ###########
- `ReplicationMode.EXCEPTION` - ######################### #########
- `ReplicationMode.LATEST_VERSION` - ###################
######################################

###################### ##############################
#######################ACID##################################

10.10.

JDBC###################################### SQL## `Session` ################## #### *flush* #####################

- #########
- `org.hibernate.Transaction.commit()` #######
- `Session.flush()` #######

SQL################

1. ################### `Session.save()` ######### ###################
2. ##############
3. ##############
4. ###########################
5. ##############
6. ###################`Session.delete()` #### #######################

(########## `native` ID ############### ######################

`flush()` ######### ##`Session` #JDBC############
########################### ## ### ####### ###Hibernate ##
`Query.list(..)` ###################### #########

`FlushMode`
####3################ ####################### #Hibernate#

Transaction API############# ############################# flush() ############################### ############################# (#11.3.2. ################### #########)#

```
sess = sf.openSession();
Transaction tx = sess.beginTransaction();
sess.setFlushMode(FlushMode.COMMIT); // allow queries to return
 stale state

Cat izi = (Cat) sess.load(Cat.class, id);
izi.setName(iznizi);

// might return stale data
sess.find("from Cat as cat left outer join cat.kittens kitten");

// change to izi is not flushed!
...
tx.commit(); // flush occurs
sess.close();
```

######################## #####DML################### ###############Hibernate############################ # 11. ############ #######

10.11.

#################################### ####################################### ##################### ###############

#################################### ###############################"#####"####### ################ ###################################### ################################## ########1####################### #####Hibernate############## ###

Hibernate# ###########

Hibernate#Session###### persist(), merge(), saveOrUpdate(), delete(), lock(), refresh(), evict(), replicate() ############# ######################### ################## create, merge, save-update, delete, lock, refresh, evict, replicate #############

###
###############

```
<one-to-one name="person" cascade="persist"/>
```

###########################

```
<one-to-one name="person" cascade="persist,delete,lock"/>
```

cascade="all" ######
cascade="none" ##########################

############# delete-orphan ########### #######
############################## delete() ##################

#####

- ### <many-to-one> # <many-to-many> ######## ####################
  <one-to-one> # <one-to-many> ######## ###################
- #################################### cascade="all,delete-orphan"
  ##### ########## ############## #####
- . ##############################
  ###
  ########################## cascade="persist,merge,save-update"
  ##############

cascade="all" ########################### ## #################
##########/##/##########/##/#################

##################################/############
####
cascade="delete-orphan" ########## <one-to-many> ### ######
#################################

- ## persist() ########## ###### persist() #######
- merge() ########## ###### merge() #######
- ## save() # update() # saveOrUpdate() ################ saveOrUpdate()
  #######
- ################################## saveOrUpdate() #######
- ################### delete() #######
- ########################## ########### #
  ####################################### ####
  cascade="delete-orphan" ########## #####################

################################## ###### ##### *flush*###
################ ##

################### #### `save-upate` # `delete-orphan` ## `Session` #flush####### ###########################

10.12.

Hibernate## ################################## ##############Hibernate###########"##" ############### ### ## ###########

Hibernate#`ClassMetadata` #`CollectionMetadata` ######## `Type` ################### ####################### `SessionFactory` ########

```
Cat fritz = ......;
ClassMetadata catMeta = sessionfactory.getClassMetadata(Cat.class);

Object[] propertyValues = catMeta.getPropertyValues(fritz);
String[] propertyNames = catMeta.getPropertyNames();
Type[] propertyTypes = catMeta.getPropertyTypes();

// get a Map of all properties which are not collections or
 associations
Map namedValues = new HashMap();
for ( int i=0; i<propertyNames.length; i++ ) {
    if ( !propertyTypes[i].isEntityType() &&
 !propertyTypes[i].isCollectionType() ) {
        namedValues.put( propertyNames[i], propertyValues[i] );
    }
}
```

#11#

Hibernate#################################
Hibernate########################JDBC#######JTA###########
JDBC#ANSI#################DBMS#################
########################

Hibernate###################### ################################
################# ####################### Session #####
##################################### ###########################

############################# SELECT FOR UPDATE
####### #####################API#######
#############API######################

#####################conversation####################
Configuration#SessionFactory#### Session
######Hibernate###################

11.1. session#####transaction####

SessionFactory ############# ##################
############################### ################## Configuration
###################

Session ###############################
#########################unit of work############# #############
Session ########## JDBC Connection##### DataSource#########
############################

############### ################################
######################### ##############################
###
###
###############################

##################### ###Hibernate Session ##
###################################### ###################### ##
Session ################ ######################################

11.1.1. #####Unit of work#

#####*session-per-operation* ###################
################################### Session ################
############################## #######################

###########planned sequence##### ##################
####SQL################## ###########################
########SQL###########################
Hibernate#####################
################################## #############################
##
####################### ##################################
###
###################### ######################

The most common pattern in a multi-user client/server application is *session-per-request*. In this model, a request from the client is sent to the server (where the Hibernate persistence layer runs), a new Hibernate `Session` is opened, and all database operations are executed in this unit of work. Once the work has been completed (and the response for the client has been prepared), the session is flushed and closed. You would also use a single database transaction to serve the clients request, starting and committing it when you open and close the `Session`. The relationship between the two is one-to-one and this model is a perfect fit for many applications.

The challenge lies in the implementation. Hibernate provides built-in management of the "current session" to simplify this pattern. All you have to do is start a transaction when a server request has to be processed, and end the transaction before the response is sent to the client. You can do this in any way you like, common solutions are `ServletFilter`, AOP interceptor with a pointcut on the service methods, or a proxy/interception container. An EJB container is a standardized way to implement cross-cutting aspects such as transaction demarcation on EJB session beans, declaratively with CMT. If you decide to use programmatic transaction demarcation, prefer the Hibernate `Transaction` API shown later in this chapter, for ease of use and code portability.

############################### ##
`sessionFactory.getCurrentSession()` ########
##################### #################### ##### #########
#################JTA################ ##2.5. ################
###########

################# ##### ######################################
####################### ###########################
################# ######################################
###################EJB#################
###################EJB############### ###################
Open Session in View
Hibernate#Web###################

11.1.2.

session-per-request####################################
##################################### ###########################
Web####################################
######################### #################

- ################### Session ## #######################
- 5#########Save########################## ####################### #######################

##################### ## ####################### ####### ##############################

########################Session ################################# ######################### ######################## ################### ################### ##

Clearly, we have to use several database transactions to implement the conversation. In this case, maintaining isolation of business processes becomes the partial responsibility of the application tier. A single conversation usually spans several database transactions. It will be atomic if only one of these database transactions (the last one) stores the updated data, all others simply read data (e.g. in a wizard-style dialog spanning several request/response cycles). This is easier to implement than it might sound, especially if you use Hibernate's features:

- *Automatic Versioning* - Hibernate can do automatic optimistic concurrency control for you, it can automatically detect if a concurrent modification occurred during user think time. Usually we only check at the end of the conversation.
- *###Detached#######* - ####### *session-per-request* ################ ################################ ##################### ############################### ### *session-per-request-with-detached-objects* ########## ###################################
- *Extended (or Long) Session* - The Hibernate `Session` may be disconnected from the underlying JDBC connection after the database transaction has been committed, and reconnected when a new client request occurs. This pattern is known as *session-per-conversation* and makes even

reattachment unnecessary. Automatic versioning is used to isolate concurrent modifications and the `Session` is usually not allowed to be flushed automatically, but explicitly.

session-per-request-with-detached-objects # *session-per-conversation* ##### ############# ##

11.1.3.

################ `Session` ## ################### ####### `Session` ####### ############################### ################################

#########

```
foo.getId().equals( bar.getId() )
```

JVM###

```
foo==bar
```

Then for objects attached to a *particular* `Session` (i.e. in the scope of a `Session`) the two notions are equivalent, and JVM identity for database identity is guaranteed by Hibernate. However, while the application might concurrently access the "same" (persistent identity) business object in two different sessions, the two instances will actually be "different" (JVM identity). Conflicts are resolved using (automatic versioning) at flush/commit time, using an optimistic approach.

##########Hibernate######################### ####################### ############################### ##################### `Session` ################### ####################synchronize########### `Session` ############################ == ###########

#####`Session` ### == ########################### ################## ################### `Set` # put######### ##################################### ################JVM############## ############ `equals()` # `hashCode()` ##################################### ###################################### ## ####################################### ######################## `Set` ##########################`Set` ############### ####################################### ######### `Set` #################### ###############Hibernate#Web############# ######Hibernate##########Java############### ###################

11.1.4.

session-per-user-session # *session-per-application* ####### ############################# ################################### ##############################

- `Session` ################ HTTP###########Bean#Swing######### ########### `Session` ############# ############# ########`HttpSession` ### Hibernate `Session` ######## HttpSession####################### ############################## ################## `Session` #######

- Hibernate################################### ### `Session` ###################### `Session` ########################## ################## ##################################### ########################## ## ############## #################### ########################

- `Session` ####persistent############### #########Hibernate##################### ######################## ######################### OutOfMemoryException######################## ########`Session` ############## `clear()` # `evict()` ######## ######################## ########################## ########### 13. ã##ã##ã##å#¦ç## ########### ############`Session` ################# ###########################

11.2.

Database (or system) transaction boundaries are always necessary. No communication with the database can occur outside of a database transaction (this seems to confuse many developers who are used to the auto-commit mode). Always use clear transaction boundaries, even for read-only operations. Depending on your isolation level and database capabilities this might not be required but there is no downside if you always demarcate transactions explicitly. Certainly, a single database transaction is going to perform better than many small transactions, even for reading data.

J2EE############# #################Web#Swing############ ##########Hibernate################# #############Hiberante######################### ##

####################################### ########################
##################################CMT#########
#########Bean########################## #################
###############################

However, it is often desirable to keep your persistence layer portable between non-managed resource-local environments, and systems that can rely on JTA but use BMT instead of CMT. In both cases you'd use programmatic transaction demarcation. Hibernate offers a wrapper API called `Transaction` that translates into the native transaction system of your deployment environment. This API is actually optional, but we strongly encourage its use unless you are in a CMT session bean.

###`Session` #####################

- ############
- ##############
- ###########
- ##########

################################## ############################
#####################################

11.2.1.

Hibernate##########################
###################DataSource##########
#################### Hibernate################################
###################################

```
// Non-managed environment idiom
Session sess = factory.openSession();
Transaction tx = null;
try {
    tx = sess.beginTransaction();

    // do some work
    ...

    tx.commit();
}
catch (RuntimeException e) {
    if (tx != null) tx.rollback();
    throw e; // or display error message
}
finally {
    sess.close();
}
```

Session # flush() ############ commit() ############################ ####### #10.10. ############## ######### close() ############################# close() ############# ######JDBC################# ###Java#########################JTA##############

#############Hibernate########### #current session############ #########################

```
// Non-managed environment idiom with getCurrentSession()
try {
    factory.getCurrentSession().beginTransaction();

    // do some work
    ...

    factory.getCurrentSession().getTransaction().commit();
}
catch (RuntimeException e) {
    factory.getCurrentSession().getTransaction().rollback();
    throw e; // or display error message
}
```

##############Hibernate############## RuntimeException ##### ## Hibernate################################## ######SessionFactory ###############

############## org.hibernate.transaction.JDBCTransactionFactory ########### #2######## hibernate.current_session_context_class # "thread" ###########

11.2.2. JTA#####

#######################EJB#####Bean############ Hibernate######################## #########JTA################# EJB##############JTA################ JTA#######Hibernate#############

Bean###########BMT#####Transaction API###### Hibernate#############BMT###################### ##

```
// BMT idiom
Session sess = factory.openSession();
Transaction tx = null;
```

```
try {
    tx = sess.beginTransaction();

    // do some work
    ...

    tx.commit();
}
catch (RuntimeException e) {
    if (tx != null) tx.rollback();
    throw e; // or display error message
}
finally {
    sess.close();
}
```

############## Session ######### ####################
getCurrentSession() ###### JTA# UserTransaction API##############

```
// BMT idiom with getCurrentSession()
try {
    UserTransaction tx = (UserTransaction)new InitialContext()
                            .lookup("java:comp/UserTransaction");

    tx.begin();

    // Do some work on Session bound to transaction
    factory.getCurrentSession().load(...);
    factory.getCurrentSession().persist(...);

    tx.commit();
}
catch (RuntimeException e) {
    tx.rollback();
    throw e; // or display error message
}
```

With CMT, transaction demarcation is done in session bean deployment descriptors, not programmatically, hence, the code is reduced to:

```
// CMT idiom
 Session sess = factory.getCurrentSession();

 // do some work
 ...
```

CMT/EJB########################## ##################################### RuntimeException ################# ######################### *####BMT####CMT######Hibernate Transaction API###################*

Note that you should choose `org.hibernate.transaction.JTATransactionFactory` if you use JTA directly (BMT), and `org.hibernate.transaction.CMTTransactionFactory` in a CMT session bean, when you configure Hibernate's transaction factory. Remember to also set `hibernate.transaction.manager_lookup_class`. Furthermore, make sure that your `hibernate.current_session_context_class` is either unset (backwards compatibility), or set to `"jta"`.

The `getCurrentSession()` operation has one downside in a JTA environment. There is one caveat to the use of `after_statement` connection release mode, which is then used by default. Due to a silly limitation of the JTA spec, it is not possible for Hibernate to automatically clean up any unclosed `ScrollableResults` or `Iterator` instances returned by `scroll()` or `iterate()`. You *must* release the underlying database cursor by calling `ScrollableResults.close()` or `Hibernate.close(Iterator)` explicitly from a `finally` block. (Of course, most applications can easily avoid using `scroll()` or `iterate()` at all from the JTA or CMT code.)

11.2.3.

`Session` ####`SQLException`########### ############################`Session.close()` ####`Session` ############### `Session` ############ ######### ###### ##### Hibernate############################### `finally` ####### `close()` ##### `Session` ############

`HibernateException` ##Hibernate################## #################Hibernate################# ############################## ############################## ############################ ### ############################ ###################### Hibernate##`HibernateException` ############ ################ ###############################

Hibernate wraps `SQLException`s thrown while interacting with the database in a `JDBCException`. In fact, Hibernate will attempt to convert the exception into a more meaningful subclass of `JDBCException`. The underlying `SQLException` is always available via `JDBCException.getCause()`. Hibernate converts the `SQLException` into an appropriate `JDBCException` subclass using the `SQLExceptionConverter` attached to the `SessionFactory`. By default, the `SQLExceptionConverter` is defined by the configured dialect; however, it is also possible to plug in a custom implementation (see the javadocs for the `SQLExceptionConverterFactory` class for details). The standard `JDBCException` subtypes are:

- `JDBCConnectionException` - #####JDBC############
- `SQLGrammarException` - ####SQL##################
- `ConstraintViolationException` - #####################
- `LockAcquisitionException` - #######################################
- `GenericJDBCException` - ########################

11.2.4.

One extremely important feature provided by a managed environment like EJB that is never provided for non-managed code is transaction timeout. Transaction timeouts ensure that no misbehaving transaction can indefinitely tie up resources while returning no response to the user. Outside a managed (JTA) environment, Hibernate cannot fully provide this functionality. However, Hibernate can at least control data access operations, ensuring that database level deadlocks and queries with huge result sets are limited by a defined timeout. In a managed environment, Hibernate can delegate transaction timeout to JTA. This functionality is abstracted by the Hibernate `Transaction` object.

```
Session sess = factory.openSession();
try {
    //set transaction timeout to 3 seconds
    sess.getTransaction().setTimeout(3);
    sess.getTransaction().begin();

    // do some work
    ...

    sess.getTransaction().commit()
}
catch (RuntimeException e) {
    sess.getTransaction().rollback();
    throw e; // or display error message
}
finally {
    sess.close();
}
```

CMT####### `setTimeout()` # ###################
###############################

11.3.

################################# ##########################
####################################
#######################################
Hibernate###############################

############## ##########################
##########

11.3.1.

Hibernate####################### ##################### Session
######### ########################## #######################
############################## ##############################
################################ ######EJB###############

```
// foo is an instance loaded by a previous Session
session = factory.openSession();
Transaction t = session.beginTransaction();

int oldVersion = foo.getVersion();
session.load( foo, foo.getKey() ); // load the current state
if ( oldVersion != foo.getVersion() ) throw new
 StaleObjectStateException();
foo.setProperty("bar");

t.commit();
session.close();
```

<version> #####version ###############
Hibernate########################## #### version
#####################

##
################################
############## ################ ##############
#####################################
####################

Clearly, manual version checking is only feasible in very trivial circumstances and not practical for most applications. Often not only single instances, but complete graphs of modified objects have to be checked. Hibernate offers automatic version checking with either an extended Session or detached instances as the design paradigm.

11.3.2.

Session #################### *session-per-conversation*
########### Hibernate################################
######################### ########################
################################### ########################

################# Session ######JDBC###############
################################

##
##
#########################

```
// foo is an instance loaded earlier by the old session
Transaction t = session.beginTransaction(); // Obtain a new JDBC
 connection, start transaction

foo.setProperty("bar");

session.flush();    // Only for last transaction in conversation
t.commit();         // Also return JDBC connection
session.close();    // Only for last transaction in conversation
```

foo ################# Session ##########
######################################
############################### ###########################
#######JDBC###########################
######################################
LockMode.READ
Session.lock() ############## #### ## ################### #####
Session # FlushMode.MANUAL ######## #########################
############################## ######################### flush()
############################### ###### close() ####

########################## Session ###########################
####HttpSession ################# Session ####################
####################### ##
########### Session ######## ##########################

#Hibernate############### Session # ##########################
#################### ###################################

Session ##################### ##########
Session ######### EJB###########Bean######### HttpSession
#########Web############ #########################

#################*session-per-conversation*###
################################### ########## CurrentSessionContext
################ Hibernate Wiki##############

11.3.3.

Session###########################
##
############### Session ####### ########################

####Session.update() #####Session.saveOrUpdate()# Session.merge() ########################

```
// foo is an instance loaded by a previous Session
foo.setProperty("bar");
session = factory.openSession();
Transaction t = session.beginTransaction();
session.saveOrUpdate(foo); // Use merge() if "foo" might have been
 loaded already
t.commit();
session.close();
```

Again, Hibernate will check instance versions during flush, throwing an exception if conflicting updates occurred.

############################# update() ######LockMode.READ ##### lock() ############ ##################################

11.3.4.

optimistic-lock ### false ############################### ########################

Legacy database schemas are often static and can't be modified. Or, other applications might also access the same database and don't know how to handle version numbers or even timestamps. In both cases, versioning can't rely on a particular column in a table. To force a version check without a version or timestamp property mapping, with a comparison of the state of all fields in a row, turn on optimistic-lock="all" in the <class> mapping. Note that this conceptually only works if Hibernate can compare the old and new state, i.e. if you use a single long Session and not session-per-request-with-detached-objects.

<class> ###### optimistic-lock="dirty" ###################Hibernate###################

In both cases, with dedicated version/timestamp columns or with full/dirty field comparison, Hibernate uses a single UPDATE statement (with an appropriate WHERE clause) per entity to execute the version check and update the information. If you use transitive persistence to cascade reattachment to associated entities, Hibernate might execute unnecessary updates. This is usually not a problem, but *on update* triggers in the database might be executed even when no changes have been made to detached instances. You can customize this behavior by setting select-before-update="true" in the <class> mapping, forcing Hibernate to SELECT the instance to ensure that changes did actually occur, before updating the row.

11.4.

It is not intended that users spend much time worrying about locking strategies. It's usually enough to specify an isolation level for the JDBC connections and then simply let the database do all the work. However, advanced users may sometimes wish to obtain exclusive pessimistic locks, or re-obtain locks at the start of a new transaction.

Hibernate######################### ########################

`LockMode` #####Hibernate####################### ######################

- `LockMode.WRITE` ## Hibernate##########################
- `LockMode.UPGRADE` ## ################### `SELECT ... FOR UPDATE` ################################
- `LockMode.UPGRADE_NOWAIT` ## Oracle # `SELECT ... FOR UPDATE NOWAIT` ##### ##########################
- `LockMode.READ` ## Repeatable Read####Serializable############################ ############################
- `LockMode.NONE` ################# `Transaction` ###### ############################## `update()` # `saveOrUpdate()` ############# ######################################

################################

- `LockMode` ##### `Session.load()` ######
- `Session.lock()` ######
- `Query.setLockMode()` ######

`UPGRADE` #### `UPGRADE_NOWAIT` ###### `Session.load()` ####### ## `SELECT ... FOR UPDATE` ##################### `load()` ################ #################################### Hibernate##############`lock()` ########

############ `READ` #####`UPGRADE`# `UPGRADE_NOWAIT` ######`Session.lock()` ###################### #`UPGRADE` #### `UPGRADE_NOWAIT` #### `SELECT ... FOR UPDATE` ########

################################ Hibernate################################ ###############################

11.5.

Hibernate######2.x##JDBC################# ########## Session ################### ################################## Hibernate 3.x#######JDBC##################### ######################## ############ ConnectionProvider ######### ######################## ##########org.hibernate.ConnectionReleaseMode # ################

- ON_CLOSE - is essentially the legacy behavior described above. The Hibernate session obtains a connection when it first needs to perform some JDBC access and holds unto that connection until the session is closed.
- AFTER_TRANSACTION - org.hibernate.Transaction ####### ##############
- AFTER_STATEMENT ############### - ##################################### #################################### ################## ############### org.hibernate.ScrollableResults #############

################ hibernate.connection.release_mode ######################## ##############

- auto ####### - ######## org.hibernate.transaction.TransactionFactory.getDefaultReleaseMode() ######################### ######## JTATransactionFactory##ConnectionReleaseMod JDBCTransactionFactory##ConnectionReleaseMode.AFTER_TRANSACTION###### ############################### ##################### ##############################
- on_close - ConnectionReleaseMode.ON_CLOSE###### ####################################
- after_transaction - ConnectionReleaseMode.AFTER_TRANSACTION###### #####JTA################## ConnectionReleaseMode.AFTER_TRANSACTION################## ######AFTER_STATEMENT#################################
- after_statement - ConnectionReleaseMode.AFTER_STATEMENT###### ######### ConnectionProvider ## #### (supportsAggressiveRelease()) ########### ############# ################ConnectionReleaseMode.AFTER_TRANSACTION ########## ################## ####ConnectionProvider.getConnection() ######## #####JDBC################### ####################################

#12#

#########Hibernate################################
###################### ##Hibernate######################

12.1.

Interceptor ############# #################################
################################
##################################### ################### ####
Interceptor # Auditable ########### createTimestamp ##### Auditable
########### lastUpdateTimestamp ############

Interceptor ######### #########EmptyInterceptor ###########

```
package org.hibernate.test;

import java.io.Serializable;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.EmptyInterceptor;
import org.hibernate.Transaction;
import org.hibernate.type.Type;

public class AuditInterceptor extends EmptyInterceptor {

    private int updates;
    private int creates;
    private int loads;

    public void onDelete(Object entity,
                         Serializable id,
                         Object[] state,
                         String[] propertyNames,
                         Type[] types) {
        // do nothing
    }

    public boolean onFlushDirty(Object entity,
                                Serializable id,
                                Object[] currentState,
                                Object[] previousState,
                                String[] propertyNames,
                                Type[] types) {

        if ( entity instanceof Auditable ) {
            updates++;
            for ( int i=0; i < propertyNames.length; i++ ) {
                if ( "lastUpdateTimestamp".equals( propertyNames[i]
 ) ) {
```

```
                    currentState[i] = new Date();
                    return true;
                }
            }
        }
        return false;
    }

    public boolean onLoad(Object entity,
                          Serializable id,
                          Object[] state,
                          String[] propertyNames,
                          Type[] types) {
        if ( entity instanceof Auditable ) {
            loads++;
        }
        return false;
    }

    public boolean onSave(Object entity,
                          Serializable id,
                          Object[] state,
                          String[] propertyNames,
                          Type[] types) {

        if ( entity instanceof Auditable ) {
            creates++;
            for ( int i=0; i<propertyNames.length; i++ ) {
                if ( "createTimestamp".equals( propertyNames[i] ) )
 {
                    state[i] = new Date();
                    return true;
                }
            }
        }
        return false;
    }

    public void afterTransactionCompletion(Transaction tx) {
        if ( tx.wasCommitted() ) {
            System.out.println("Creations: " + creates + ", Updates:
 " + updates, "Loads: " + loads);
        }
        updates=0;
        creates=0;
        loads=0;
    }

}
```

################# Session ######## SessionFactory ##########

Session ############### ####################### Interceptor ######SessionFactory.openSession() ######################

```
Session session = sf.openSession( new AuditInterceptor() );
```

SessionFactory ############## Configuration ################## ### SessionFactory ############## #################### SessionFactory ############################# ### SessionFactory ################################## ## ################################

```
new Configuration().setInterceptor( new AuditInterceptor() );
```

12.2.

############################## Hibernate3# #### ################### ###

Session ##################### 1################# ### LoadEvent#FlushEvent ######## #######################XML#######DTD# org.hibernate.event ################# ################1########### Hibernate# Session ############## ############################# ####################################### ##################################### #####LoadEvent ###### LoadEventListener ######################## ############ Session ############ load()

######################### ######################## ###################################

####Hibernate################################ ###################final############## ######### Configuration ######################### Hibernate#XML############## #######################################

```
public class MyLoadListener implements LoadEventListener {
    // this is the single method defined by the LoadEventListener
 interface
    public void onLoad(LoadEvent event, LoadEventListener.LoadType
 loadType)
            throws HibernateException {
        if ( !MySecurity.isAuthorized( event.getEntityClassName(),
 event.getEntityId() ) ) {
```

```
            throw MySecurityException("Unauthorized access");
        }
    }
}
```

####################Hibernate##########

```
<hibernate-configuration>
    <session-factory>
        ...
        <event type="load">
            <listener class="com.eg.MyLoadListener"/>
            <listener
 class="org.hibernate.event.def.DefaultLoadEventListener"/>
        </event>
    </session-factory>
</hibernate-configuration>
```

#########################

```
Configuration cfg = new Configuration();
LoadEventListener[] stack = { new MyLoadListener(), new
 DefaultLoadEventListener() };
cfg.EventListeners().setLoadEventListeners(stack);
```

################################### ### <listener/> ################# ################################## ############################ #########################

##################################### ## #########################on/off#################

12.3. Hibernate###########

####Hibernate####################################### ###Hiberenate3#JACC#######JAAS################## ######################################

#####################JAAS####################

```
<listener type="pre-delete"
 class="org.hibernate.secure.JACCPreDeleteEventListener"/>
<listener type="pre-update"
 class="org.hibernate.secure.JACCPreUpdateEventListener"/>
<listener type="pre-insert"
 class="org.hibernate.secure.JACCPreInsertEventListener"/>
<listener type="pre-load"
 class="org.hibernate.secure.JACCPreLoadEventListener"/>
```

############################# `<listener type="..." class="..."/>` # `<event type="..."><listener class="..."/></event>` ######################

`hibernate.cfg.xml`

```
<grant role="admin" entity-name="User"
 actions="insert,update,read"/>
<grant role="su" entity-name="User" actions="*"/>
```

###########JACC#################

#13# ã##ã##ã##å#¦ç##

Hibernateã##ä½¿ã#£ã#¦ã##ã#¼ã#¿ã##ã#¼ã#¹ã#«100,000è¡#ã##æ#¿å#¥ã##ã##æ##ç#´ã#ªã

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
}
tx.commit();
session.close();
```

ã##ã##ã#¯50,000ç#ªç#®ã#®è¡#ã#®ã##ã##ã##ã#§ `OutOfMemoryException` ã#§å¤±æ##ã##ã##ã#§ã##ã##ã##ã## Hibernateã##ã#»ã##ã#·ã#§ã#³ã#¬ã##ã#«ã#-ã#£ã##ã#·ã#¥ã#§ã## æ#°ã##ã##æ#¿å#¥ã##ã##ã##ã##ã#¹ã#¦ã#® `Customer` ã#¤ã#³ã#¹ã#¿ã#³ã#¹ã##ã#-ã#£ã##ã#·ã#¥ã##ã##ã##ã##ã#§ã##ã##

ã##ã#®ç« ã#§ã#¯ã##ã##ã#®å##é¡#ã##å##é#¿ã##ã##æ#¹æ³#ã##ç´¹ä»#ã##ã#¾ã##ã## ã##ã##ã##ã##ã##ã##å#¦ç##ã##ã##ã#ªã##ã##JDBCã##ã##ã##ä½¿ç#¨å#¯è#½ã#§ã# ã##ã##ã#§ã#ªã##ã##ã#°æ##é #ã#ªã##ã##ã#©ã#¼ã##ã#³ã#¹ã##å¾#ã##ã##ã#¾ã##ã##ã## JDBCã##ã##ã#µã#¤ã#ºã##æ##é #ã#ªæ#°å#¤ï¼#ä¾#ã##ã#°ã##10ã##ã##50ï¼#ã#«è¨-å®

```
hibernate.jdbc.batch_size 20
```

Note that Hibernate disables insert batching at the JDBC level transparently if you use an `identiy` identifier generator.

ã#¾ã##ä º#æ¬¡ã#-ã#£ã##ã#·ã#¥ã##å#¨ã##å#¹ã##ã#ªã##ã##ã#-ã#»ã#¹ã#§ã## ã##ã#®ã##ã##ã#ªä½#æ¥-ã##ã##ã##ã#¨æ##ã##ã##ã##ã##ã##ã#¾ã##ã##ï¼#

```
hibernate.cache.use_second_level_cache false
```

ã##ã##ã##ã##ã##ã#¯çµ¶å¯¾ã#«å¿#è¦#ã#¨ã##ã##ã##ã##ã#§ã#¯ã##ã##ã#¾ã##ã##ã## ã#ªã##ã#ªã##æ##ç¤ºç##ã#« `CacheMode` ã##è¨-å®#ã##ã#¦ã## äº#æ¬¡ã#-ã#£ã##ã#·ã#¥ã#¨ã#®ç#¸äº#ä½#ç#¨ã##ç#¡å#¹ã#«ã##ã##ã##ã#¨ã##ã#§ã##ã##ã##

13.1. ã##ã##ã##æ#¿å#¥

æ#°ã##ã##ã#ªã##ã#¸ã#§ã#¯ã##ã##æ°¸ç¶#å##ã##ã##ã#¨ã##ä¸#æ¬¡ã#-ã#£ã##ã#·ã#¥ã#© ã#»ã##ã#·ã#§ã#³ã## `flush()` ã##ã#¦ `clear()` ã##ã#ªã##ã##ã#°ã#ªã##ã#¾ã##ã##ã##

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
```

```
for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
    if ( i % 20 == 0 ) { //20, same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}

tx.commit();
session.close();
```

13.2. ã##ã##ã##æ#´æ#°

ã##ã#¼ã#¿ã##å¾©å##ã##ã##ã##æ#´æ#°ã##ã##ã##ã##ã#«ã#¯å##ã##ã#¢ã#¤ã##ã#£ã#
ã##ã##ã#«å# ã##ã#¦ã##ã##ã#¼ã#¿ã#®è¡#ã##å¤#ã##è¿#ã##ã#¯ã#¨ã#ªã#«å¯¾ã##ã#¦æ##å
ã#µã#¼ã##ã#¼ã#µã#¤ã##ã#®ã#«ã#¼ã#½ã#«ã#®å#©ç#¹ã##ç##ã##ã##ã##ã##ã#°
`scroll()` ã##ä½¿ã##å¿#è¦#ã##ã##ã##ã#¾ã##ã##

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

ScrollableResults customers = session.getNamedQuery("GetCustomers")
    .setCacheMode(CacheMode.IGNORE)
    .scroll(ScrollMode.FORWARD_ONLY);
int count=0;
while ( customers.next() ) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    if ( ++count % 20 == 0 ) {
        //flush a batch of updates and release memory:
        session.flush();
        session.clear();
    }
}

tx.commit();
session.close();
```

13.3. StatelessSessionã#¤ã#³ã#¿ã#¼ã##ã#§ã#¤ã#¹

ã#¾ã##å#¥ã#®æ#¹æ³#ã#¨ã##ã#¦ã##Hibernateã#¯ã#³ã##ã#³ã##æ##å##ã#®APIã##ç#¨æ##ã
ã##ã##ã#¯å##é#¢ã#ªã##ã#¸ã#§ã#¯ã##ã#®å½¢ã#§ã## ã##ã#¼ã#¿ã##ã#¼ã#¹ã#¨ã#®ã##ã#
`StatelessSession`
ã#¯é#¢é#£ã##ã##æ°¸ç¶#ã#³ã#³ã##ã#-ã#¹ã##ã##æ##ã##ã##ã## é«#ã#¬ã##ã#«ã#®ã#©ã#¤
ç#¹ã#«ã#¹ã##ã#¼ã##ã#¬ã#¹ã#»ã##ã#·ã#§ã#³ã#¯ã##ä¸#æ##ã#-ã#£ã##ã#·ã#¥ã##å®#è£#ã##
ã#¾ã##ã#©ã#®ã##ã##ã#ªä º#æ¬¡ã#-ã#£ã##ã#·ã#¥ã##ã#¯ã#¨ã#ªã#-ã#£ã##ã#·ã#¥ã#¨ã##ç#¸
ã##ã#©ã#³ã#¶ã#¯ã#·ã#§ã##ã#«ã#ªwrite-

behindã##è#ªå##ã##ã#¼ã##ã#£ã##ã#§ã##ã#¯ã##å®#è£#ã##ã#¾ã##ã##ã##
ã#¹ã##ã#¼ã##ã#¬ã#¹ã#»ã##ã#·ã#§ã#³ã##ä½¿ã#£ã#¦è¡#ã##ã##ã##æ##ä½#ã##ã##
é#¢é#£ã##ã##ã#¤ã#³ã#¹ã#¿ã#³ã#¹ã#¸ã#«ã#¹ã#±ã#¼ã##ã##ã##ã##ã##ã#¨ã#¯æ±ºã##ã#¦ã##
ã#³ã#¬ã#¯ã#·ã#§ã#³ã#¯ã##ã#¹ã##ã#¼ã##ã#¬ã#¹ã#»ã##ã#·ã#§ã#³ã##ã##ã#¯ç#¡è¦#ã##ã##ã
ã#¹ã##ã#¼ã##ã#¬ã#¹ã#»ã##ã#·ã#§ã#³ã##é##ã##ã#¦è¡#ã##ã##ã##æ##ä½#ã#¯ã##
Hibernateã#®ã#¤ã##ã#³ã##ã#¢ã##ã#«ã##ã#¤ã#³ã#¿ã#¼ã#»ã##ã#¿ã#®å½±é#¿ã##å##ã##ã
ä¸#æ##ã#-ã#£ã##ã#·ã#¥ã##æ##ã##ã#ªã##ã##ã## ã#¹ã##ã#¼ã##ã#¬ã#¹ã#»ã##ã#·ã#§ã
ã#¹ã##ã#¼ã##ã#¬ã#¹ã#»ã##ã#·ã#§ã#³ã#¯ä½#ã#¬ã##ã#«ã#®æ#½è±¡å##ã#§ã##ã##ã##JDE

```
StatelessSession session = sessionFactory.openStatelessSession();
Transaction tx = session.beginTransaction();

ScrollableResults customers = session.getNamedQuery("GetCustomers")
    .scroll(ScrollMode.FORWARD_ONLY);
while ( customers.next() ) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    session.update(customer);
}

tx.commit();
session.close();
```

ã##ã#®ã#³ã#¼ã##ä¾#ã#§ã#¯ã##ã#¯ã#¨ã#ªã##è¿#ã##

`Customer`

ã#¤ã#³ã#¹ã#¿ã#³ã#¹ã#¯å#³åº§ã#«ï¼#ã#»ã##ã#·ã#§ã#³ã##ã##ï¼#å##é#¢ã##ã##ã##ã##ã#¨ã
ã##ã##ã#¯ã##ã#©ã#®ã##ã##ã#ªæ°¸ç¶#ã#³ã#³ã##ã#-ã#¹ã##ã#¨ã##æ±ºã##ã#¦é#¢é#£ã##ã#¾

`StatelessSession`
ã#¤ã#³ã#¿ã#¼ã##ã#§ã#¤ã#¹ã#§å®#ç¾©ã##ã##ã#¦ã##ã## `insert()`, `update()`, `delete()` ã#¯ã##
ä½#ã#¬ã##ã#«ã#®ç#´æ#¥ç##ã#ªã##ã#¼ã#¿ã##ã#¼ã#¹æ##ä½#ã#¨è##ã##ã##ã##ã#¾ã##ã
çµ#æ##ã#¨ã##ã#¦ã##SQLã#® `INSERT`, `UPDATE`, `DELETE`
ã##ã##ã##ã##å#³åº§ã#«å®#è¡#ã##ã##ã#¾ã##ã##
ã##ã#®ã##ã##ã#«ã##ã##ã##ã#¯`Session`
ã#¤ã#³ã#¿ã#¼ã##ã#§ã#¤ã#¹ã#§å®#ç¾©ã##ã##ã#¦ã##ã##
`save()`, `saveOrUpdate()`, `delete()`
ã#¨ã#¯é##å¸¸ã#«ç#°ã#ªã##æ##å#³ã##æ##ã#¡ã#¾ã##ã##

13.4. DMLã#¹ã#¿ã#¤ã#«ã#®æ##ä½#

ã##ã#§ã#«è-°è«#ã##ã##ã##ã#«ã##è#ªå##ç##ã##ã#¤é##é##ç##ã#ªã#ªã##ã#¸ã#§ã#¯ã##
ã#ªã#¬ã#¼ã#·ã#§ã##ã#«ã##ã##ã#³ã#°ã#¯ã## ã#ªã##ã#¸ã#§ã#¯ã##ã#®ç#¶æ##ã#®ç®¡ç
ã##ã##ã#¯ã#¡ã#¢ã#ªå##ã#®ã#ªã##ã#¸ã#§ã#¯ã##ã#®ç#¶æ##ã##å©ç#¨ã#§ã##ã##ã#¨ã##ã#
ã##ã#®ã##ã##ï¼#SQLã#® ã##ã#¼ã#¿æ##ä½#è¨#èª# (DML)
æ##ï¼# `INSERT`, `UPDATE`, `DELETE`
ã##ä½¿ã#£ã#¦ï¼#ã##ã#¼ã#¿ã##ã#¼ã#¹å##ã#®ã##ã#¼ã#¿ã##ç#´æ#¥æ##ä½#ã##ã#¦ã##ã

ã#¡ã#¢ã#ªå##ã#®ç#¶æ##ã#«ã#¯å½±é#¿ã##ä¸#ã##ã#¾ã##ã##ã## ã##ã##ã##Hibernateã#¯ã
ã##ã##ã#¯Hibernateã#¯ã#¨ã#ªè¨#èª#ï¼# 14. *HQL: The Hibernate Query Language*ï¼# ã##é##ã##ã#¦å®#è¡#ã##ã##ã#¾ã##ã##

UPDATE ã#¨ DELETE æ##ã#®ç##ä¼¼æ§#æ##ã#¯ï¼#(UPDATE | DELETE) FROM? ã#¨ã#³ã##ã#£ã##ã#£å## (WHERE æ#¡ä»¶ç¯#)? ã#§ã##ã## æ³¨æ##ã##ã#¹ã##ç#¹ã##ã##ã##ã#¤ã##ã##ã##ã#¾ã##ï¼#

- fromç¯#ã#«ã##ã##ã#¦ã##FROMã#-ã#¼ã#¯ã#¼ã##ã#¯ã#ªã##ã#·ã#§ã#³ã#§ã##ã##
- fromç¯#ã#§ã#¯å##ä¸#ã#®ã#¨ã#³ã##ã#£ã##ã#£å##ã# ã##ã##å#¯è#½ã#§ã## ä»»æ##ã#§å#¥å##ã##ä»#ã##ã##ã##ã#¨ã##ã#§ã##ã#¾ã##ã## ã#¨ã#³ã##ã#£ã##ã#£å##ã ã##ã#®å#¥å##ã##ä½¿ã#£ã#¦ä¿®é£¾ã##ã#ªã##ã##ã#°ã#ªã##ã#¾ã##ã##ã## ã##ã##ã#¨ã#³ã##ã#£ã##ã#£å##ã#«å#¥å##ã##ä¸#ã##ã##ã##ã#ªã##ã##ã#°ã## ã#©ã#®ã##ã##ã#ªã##ã#-ã##ã##ã#£å##ç#§ã##ä¿®é£¾ã##ã#¦ã#¯ã#ªã##ã#¾ã##ã##ã##
- ï¼#æ##é»#ç##ã#§ã##ã##æ##ç¤ºç##ã#§ã##ã##ï¼##14.4. ######### ã##ã##ã#«ã#¯HQLã#¯ã#¨ã#ªå##ã#§æ##å®#ã##ã##ã##ã#¨ã#¯ã#§ã##ã#¾ã##ã##ã## ã#µã##ã#¯ã#¨ã#ªã#¯whereç¯#ã#§ä½¿ã##ã##ã#¨ã##ã#§ã##ã#¾ã## ã#µã##ã#¯ã#¨ã#ªã##
- whereç¯#ã#¯ã#ªã##ã#·ã#§ã#³ã#§ã##ã##

ä¾#ã#¨ã##ã#¦ã##HQLã#® UPDATE ã##å®#è¡#ã##ã##ã#«ã#¯ã## Query.executeUpdate() ã#¡ã#½ã##ã##ã##ä½¿ã#£ã#¦ã##ã# ã##ã##ã## ï¼#ã##ã#®ã#¡ã#½ã##ã##ã#¯ã##ã#ªã##ã#¿ã#®JDBC PreparedStatement.executeUpdate() ã##ã##å##ä»#ã##ã##ã##ã#¾ã##ã##ï¼#ï¼# d

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

String hqlUpdate = "update Customer c set c.name = :newName where
 c.name = :oldName";
// or String hqlUpdate = "update Customer set name = :newName where
 name = :oldName";
int updatedEntities = s.createQuery( hqlUpdate )
        .setString( "newName", newName )
        .setString( "oldName", oldName )
        .executeUpdate();
tx.commit();
session.close();
```

HQLã#® UPDATE æ##ã#¯ã##ã##ã##ã#©ã#«ã##ã#§ã#¯ã##ä½#ç#¨ã##ã##ã#¨ã#³ã##ã#£ã##ã#£ã#® #5.1.9. #version######## ã## #5.1.10. #timestamp######## ã##ã#-ã##ã##ã#£ã#®å#¤ã#«ã#¯å½±é#¿ã##ã#¾ã##ã##ã## ã##ã##ã#¯EJB3ã#®ä»#æ§#ã#«ã##å##ã##ç¶#ã##ã##ã#¦ã##ã#¾ã##ã## ã##ã##ã## versioned update ã##ä½¿ã#£ã#¦ã## version ã## timestamp

ã##ã#-ã##ã##ã#£ã#®å#¤ã##å¼·å#¶ç##ã#«ã#ªã#»ã##ã##ã##ã##ã##ã#¨ã##ã#§ã##ã#¾ã
ã##ã##ã#¯UPDATE ã#-ã#¼ã#¯ã#¼ã##ã#®å¾#ã#«VERSIONED
ã#-ã#¼ã#¯ã#¼ã##ã##è¿½å# ã##ã##ã##ã#¨ã#§è¡#ã##ã#¾ã##ã##

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
String hqlVersionedUpdate = "update versioned Customer set name =
 :newName where name = :oldName";
int updatedEntities = s.createQuery( hqlUpdate )
        .setString( "newName", newName )
        .setString( "oldName", oldName )
        .executeUpdate();
tx.commit();
session.close();
```

ã#«ã#¹ã#¿ã# ã##ã#¼ã#¸ã#§ã#³å##ï¼#`org.hibernate.usertype.UserVersionType`ï¼#
ã#¯`update versioned`
æ##ã#¨ä¸#ç·#ã#«ä½¿ã##ã#ªã##ã##ã#¨ã#«æ³¨æ##ã##ã#¦ã##ã# ã##ã##ã##

HQLã#® `DELETE` ã##å®#è¡#ã##ã##ã#«ã#¯ã## å##ã##
`Query.executeUpdate()` ã#¡ã#½ã##ã##ã##ä½¿ã#£ã#¦ã##ã# ã##ã##ï¼#

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

String hqlDelete = "delete Customer c where c.name = :oldName";
// or String hqlDelete = "delete Customer where name = :oldName";
int deletedEntities = s.createQuery( hqlDelete )
        .setString( "oldName", oldName )
        .executeUpdate();
tx.commit();
session.close();
```

`Query.executeUpdate()` ã#¡ã#½ã##ã##ã##è¿#ã##
`int`
ã#®å#¤ã#¯ã##ã##ã#®æ##ä½#ã##å½±é#¿ã##å##ã#¼ã##ã##ã#¨ã#³ã##ã#£ã##ã#£ã#®æ#°ã
ã##ã##ã##å½±é#¿ã##ã##ã##ã#¼ã#¿ã##ã#¼ã#¹å##ã#®è¡#æ#°ã#¨ã##ç#¸ä#ºã#«é#¢ä¿#ã##
HQLã##ã#«ã#¯æ##ä½#ã#¯ã##çµ#æ##ã#¨ã##ã#¦ã##å®#é##ã#®SQLæ##ã##è¤#æ#°å®#è¡#
ä¾ã##ã#°joined-subclassã#§ã##ã## è¿#ã##ã##æ#°ã#¯ã##ã#®æ##ã#«ã##ã#£ã#¦å½
joined-
subclassã#®ä¾ã#«æ#»ã##ã#¨ã##ã#µã##ã#¯ã#©ã#¹ã#®ä¸#ã#¤ã#«å¯¾ã##ã##å##é#¤ã#¯ã
ã##ã#®ã#µã##ã#¯ã#©ã#¹ã##ã##ã##ã#³ã#°ã##ã##ã##ã#¼ã##ã#«ã# ã##ã#§ã#¯ã#ªã
ã##ã#«ã#¼ã##ã##ã##ã#¼ã##ã#«ã#¨ç¶#æ#¿é##å±¤ã##ã##ã##ã#«ä¸#ã#£ã##joined-
subclassã#®ã##ã#¼ã##ã#«ã#®å##é#¤ã#«ã#ªã##ã#¾ã##ã##

`INSERT` æ##ã#®ç##ä¼¼æ§#æ##ã#¯ï¼#`INSERT INTO`
`ã#¨ã#³ã##ã#£ã##ã#£å## ã##ã#-ã##ã##ã#£ã#ªã#¹ã## selectæ##` ã#§ã##ã##
æ³¨æ##ã##ã#¹ã##ç#¹ã##ã##ã##ã#¤ã##ã##ã##ã#¾ã##ï¼#

- INSERT INTO ... SELECT ... ã#®å½¢å¼#ã# ã##ã##ã#µã##ã#¼ã##ã##ã##ã#¦ã##ã#¾ã##ã## INSERT INTO ... VALUES ... ã#®å½¢å¼#ã#¯ã#µã##ã#¼ã##ã##ã##ã#¦ã##ã#¾ã##ã##ã##

  ã##ã#-ã##ã##ã#£ã#ªã#¹ã##ã#¯ã##SQLã#® INSERT æ##ã#«ã##ã##ã## ã#«ã#©ã# ã#®ä»#æ§# ã#«é¡#ä¼¼ã##ã#¦ã##ã#¾ã##ã## ç¶#æ#¿ã#®ã##ã##ã##ã#³ã#°ã#«å#‹ ã#¯ã#©ã#¹ã#¬ã##ã#«ã#§ç#´æ#¥å®#ç¾©ã##ã##ã##ã#-ã##ã##ã#£ã# ã##ã##ã##ã# ã#¹ã#¼ã##ã#¼ã#¯ã#©ã#¹ã#®ã##ã#-ã##ã##ã#£ã#¯èª#ã##ã##ã##ã##ã##ã#µã##ã#¯ã#©ã è¨#ã##æ##ã##ã##ã#¨ INSERT æ##ã#¯ã##æ#¬è³ªç##ã#«ã##ã#ªã#¢ã#¼ã##ã#£ã##ã#¯ã#§ã#¯ã##ã##ã#¾ã##ã##ã##
- selectæ##ã#®è¿#ã##å#¤ã#®å##ã##insertæ##ã##æ##å¾#ã##ã##å##ã#¨ã##ã##ã##ã# ã##ã#®selectæ##ã#¯å¦¥å½ã#ªHQL selectã#¯ã#¨ã#ªã#¨ã#ªã##ã##ã#¾ã##ã## ç#¾å#¨ã##ã#®ã##ã#§ã##ã#¯ã##ã##ã#¼ã#¿ã## ã##ã#®ã##ã#¨ã#¯ã##*equal*ã#¨ã#¯é##ã##ã## Hibernateã#® `Type` é##ã#® *equivalent* ã#«é#¢ã##ã## å##é¡#ã##å¼#ã##èµ·ã##ã##ã##ã#¨ã#«æ³¨æ##ã##ã#¦ã##ã# ã##ã##ã## ã##ã##ã#¯`org.hibernate.type.DataType` ã#¨ã##ã#¦å®#ç¾©ã##ã##ã##ã#-ã##ã##ã#£ã#¨ã## `org.hibernate.type.TimestampType` ã#¨ã##ã#¦å®#ç¾©ã##ã##ã##ã#-ã##ã##ã#£ã#®é##ã ã##ã#¼ã#¿ã##ã#¼ã#¹ã##ã##ã##ã##å#ºå#¥ã#§ã##ã#ªã##ã#¦ã##ã##å¤#æ##ã##ã#
- idã##ã#-ã##ã##ã#£ã#«å¯¾ã##ã#¦ã##insertæ##ã#«ã#¯äº#ã#¤ã#®é#¸æ##è#¢ã##ã##ã# ã##ã#-ã##ã##ã#£ã#ªã#¹ã##ã#§æ##ç¤ºç##ã#«idã##ã#-ã##ã##ã#£ã##æ##å®#ã##ã##ã## ï¼#ã##ã#®å ´å##ã##å¯¾å¿#ã##ã##selectå¼#ã##ã##å#¤ã##å##ã##ã##ã#³ã##ï¼#ã## ã##ã#-ã##ã##ã#£ã#ªã#¹ã##ã##ã##é#¤å¤#ã##ã##ã## ï¼#ã##ã#®å ´å##ã##ç##æ##ã##ã# å¾#è##ã#®é#¸æ##è#¢ã#¯ã##ã##ã#¼ã#¿ã##ã#¼ã#¹å##ã##æ##ä½#ã##ã##idã#¸ã#§ã##ã ã##ã#®é#¸æ##è#¢ã##æ#¡ã##å ´å##ã##ã#¤ã#³ã#¡ã#¢ã#ªã##å##ã#®ã#¸ã#§ã##ã#¬ã#¼ ã##ã#®è-°è«#ã#§ã#¯ã##ã#¤ã#³ã##ã#¼ã#¿ã##ã#¼ã#¹å##ã#¸ã#§ã##ã#¬ã#¼ã#¿ã#¯ `org.hibernate.id.SequenceGenerator` ï¼#ã#¨ã##ã#®ã#µã##ã#¯ã#©ã#¹ï¼#ã#¨ã##`org.hibern` ã#®å®#è£#ã#§ã##ã##ã#¨è##ã##ã#¦ã##ã#¾ã##ã## ã##ã##ã#§æ##ã##æ³¨æ##ã#¹ã##ã ã#§ã##ã## å#¤ã##å##å¾#ã##ã##é#¸æ##å#¯è#½ã#ªæ#¹æ³#ã##ã#ªã##ã##ã##ã##ã##ã#®ã#¸ã#§ã##ã
- `version` ã## `timestamp` ã#¨ã##ã#¦ã##ã##ã##ã#³ã#°ã##ã##ã##ã##ã#-ã##ã##ã#£ã#«å¯¾ã##ã#¦ã## insertæ##ã#«ã#¯äº#ã#¤ã#®é#¸æ##è#¢ã##ã##ã#¾ã##ã## ã##ã#-ã##ã##ã#£ã#ªã#¹ã## ï¼#ã##ã#®å ´å##ã##å¯¾å¿#ã##ã##selectå¼#ã##ã##å#¤ã##å##ã##ã##ã#¾ã##ï¼#ã## ã##ã#-ã##ã##ã#£ã#ªã#¹ã##ã##ã##é#¤å¤#ã##ã##ã## ï¼#ã##ã#®å ´å##ã##`org.hibernate.type.VersionType` ã#§å®#ç¾©ã##ã##ã## ã#·ã#¼ã##å#¤ ã##ä½¿ã##ã##ã#¾ã##ï¼#ã#®ã##ã##ã##ã##ã#§ã##ã##

HQLã#® `INSERT` æ##ã#®å®#è¡#ä¾#ã#§ã##ï¼#

```
Session session = sessionFactory.openSession();
```

```
Transaction tx = session.beginTransaction();

String hqlInsert = "insert into DelinquentAccount (id, name) select
 c.id, c.name from Customer c where ...";
int createdEntities = s.createQuery( hqlInsert )
        .executeUpdate();
tx.commit();
session.close();
```

#14# HQL: The Hibernate Query Language

Hibernate#SQL########(#######)##################
###SQL#################HQL################
############################

14.1.

####Java################################ ### SeLeCT # sELEct ####### SELECT ####### net.sf.hibernate.eg.FOO # net.sf.hibernate.eg.Foo ####### foo.barSet # foo.BARSET #######

#############HQL############
##
####Java#########################

14.2. from#

#######Hibernate###########

```
from eg.Cat
```

eg.Cat
######auto-import
################## ############################

```
from Cat
```

################## Cat ###########################

```
from Cat as cat
```

####### Cat ####### cat #############
############################### as
#################################

```
from Cat cat
```

####################################

```
from Formula, Parameter
```

```
from Formula as form, Parameter as param
```

#######Java############### ################################(### domesticCat)#

14.3.

############################## #####################

```
from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
```

```
from Cat as cat left join cat.mate.kittens as kittens
```

```
from Formula form full join form.parameter param
```

###############ANSI SQL######

- `inner join`
- `left outer join`
- `right outer join`
- `full join` (############)

`inner join`#`left outer join`#`right outer join`################

```
from Cat as cat
    join cat.mate as mate
    left join cat.kittens as kitten
```

HQL# `with` #############################

```
from Cat as cat
    left join cat.kittens as kitten
        with kitten.bodyWeight > 10.0
```

#####################################1##select######### ########################
#19.1. ######## ###############

```
from Cat as cat
    inner join fetch cat.mate
    left join fetch cat.kittens
```

##################### `where` #(###########)
###
##################################
####################################

```
from Cat as cat
    inner join fetch cat.mate
    left join fetch cat.kittens child
    left join fetch child.kittens
```

fetch ### iterate() ############################## #### scroll() ######### ######################### fetch # setMaxResults() # setFirstResult() ################## ##eager### ##### fetch ######## with ### ############### ## #######################bag############################## ###################### ### ############ # ############ ####################

fetch all properties ###### Hibernate######################################

```
from Document fetch all properties order by name
```

```
from Document doc fetch all properties where lower(doc.name) like
 '%cats%'
```

14.4.

HQL###################### # ####

####################### #### #### from######join############# ##############

######join######################### ############### #########HQL####### ### #######SQL##################

```
from Cat as cat where cat.mate.name like '%s%'
```

14.5. Refering to identifier property

There are, generally speaking, 2 ways to refer to an entity's identifier property:

- The special property (lowercase) `id` may be used to reference the identifier property of an entity *provided that entity does not define a non-identifier property named id*.
- If the entity defines a named identifier property, you may use that property name.

References to composite identifier properties follow the same naming rules. If the entity has a non-identifier property named id, the composite identifier property can only be referenced by its defined named; otherwise, the special `id` property can be used to rerference the identifier property.

Note: this has changed significantly starting in version 3.2.2. In previous versions, `id` *always* referred to the identifier property no matter what its actual name. A ramification of that decision was that non-identifier properties named `id` could never be referenced in Hibernate queries.

14.6. Select#

`select` ##:

```
select mate
from Cat as cat
    inner join cat.mate as mate
```

######### `Cat` # `mate` ####### #######################:

```
select cat.mate from Cat cat
```

###:

```
select cat.name from DomesticCat cat
where cat.name like 'fri%'
```

```
select cust.name.firstName from Customer as cust
```

###############(###)####### `Object[]` #############

```
select mother, offspr, mate.name
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

`List`

```
select new list(mother, offspr, mate.name)
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

###########Java############

```
select new Family(mother, mate, offspr)
from DomesticCat as mother
    join mother.mate as mate
    left join mother.kittens as offspr
```

`Family`

select## `as` ###################

```
select max(bodyWeight) as max, min(bodyWeight) as min, count(*) as n
from Cat cat
```

`select new map` #################

```
select new map( max(bodyWeight) as max, min(bodyWeight) as min,
 count(*) as n )
from Cat cat
```

##########select#### `Map` ######

14.7.

HQL########################

```
select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat)
from Cat cat
```

#####################

- `avg(...), sum(...), min(...), max(...)`
- `count(*)`
- `count(...), count(distinct ...), count(all...)`

select#################SQL#############

```
select cat.weight + sum(kitten.weight)
from Cat cat
    join cat.kittens kitten
group by cat.id, cat.weight
```

```
select firstName||' '||initial||' '||upper(lastName) from Person
```

SQL######## `distinct` # `all` ################

```
select distinct cat.name from Cat cat

select count(distinct cat.name), count(cat) from Cat cat
```

14.8.

#########

```
from Cat as cat
```

`Cat` #############`DomesticCat` ################Hibernate#### ### Java############# `from` #############

##
########################:

```
from java.lang.Object o
```

Named ##############################:

```
from Named n, Named m where n.name = m.name
```

################SQL SELECT #################### ##### order by ############################### (#######Query.scroll() #################################)#

14.9. where#

where ############################# ###########################

```
from Cat where name='Fritz'
```

########################

```
from Cat as cat where cat.name='Fritz'
```

###'Fritz'### Cat ##############

```
select foo
from Foo foo, Bar bar
where foo.startDate = bar.date
```

##HQL##Foo # startDate ########## date ########## bar ############# #### Foo ############# ###############cat.mate.name### where ################

```
from Cat cat where cat.mate.name is not null
```

######################SQL########### #################

```
from Foo foo
where foo.bar.baz.customer.address.city is not null
```

###################4###############SQL###########

= ##

```
from Cat cat, Cat rival where cat.mate = rival.mate
```

```
select cat, mate
from Cat cat, Cat mate
```

```
where cat.mate = mate
```

id (###)##############
#################################(#####################)

```
from Cat as cat where cat.id = 123

from Cat as cat where cat.mate.id = 69
```

2##############################

########################## Person # country # medicareNumber
#####################

```
from bank.Person person
where person.id.country = 'AU'
    and person.id.medicareNumber = 123456
```

```
from bank.Account account
where account.owner.id.country = 'AU'
    and account.owner.id.medicareNumber = 123456
```

##########2########################

class
#########################discriminator##########
where########Java########discriminator#########

```
from Cat cat where cat.class = DomesticCat
```

##################(#######################)###############
##
store.owner # address

"any"############# id # class ####
#######################(AuditLog.item # <any> #################)#

```
from AuditLog log, Payment payment
where log.item.class = 'Payment' and log.item.id = payment.id
```

log.item.class # payment.class #
###

14.10. Expressions

SQL# where ###################HQL#########:

- ######+, -, *, /
- 2#######=, >=, <=, <>, !=, like

- ######`and, or, not`
- ############`( )`
- `in`, `not in`, `between`, `is null`, `is not null`, `is empty`, `is not empty`, `member of` and `not member of`
- "####"#case `case ... when ... then ... else ... end`# "###"#case `case when ... then ... else ... end`
- ######## `...||...` ### `concat(...,...)`
- `current_date()`, `current_time()`, `current_timestamp()`
- `second(...)`, `minute(...)`, `hour(...)`, `day(...)`, `month(...)`, `year(...)`,
- EJB-QL 3.0##############: `substring(), trim(), lower(), upper(), length(), locate(), abs(), sqrt(), bit_length()`
- `coalesce()` # `nullif()`
- ########String######## `str()`#
- 2######Hibernate####### `cast(... as ...)` # `extract(... from ...)`# ##############ANSI `cast()` # `extract()` ###############
- ############################HQL# `index()` ###
- ##############HQL### `size(), minelement(), maxelement(), minindex(), maxindex()` # `some, all, exists, any, in` ################# `elements()` # `indices` ###########
- `sign()`, `trunc()`, `rtrim()`, `sin()` #################SQL######
- JDBC############ `?`
- #########: `:name`, `:start_date`, `:x1`
- SQL#####`'foo'`, `69`, `6.66E+2`, `'1970-01-01 10:00:01.0'`
- Java# `public static final` ###`eg.Color.TABBY`

`in` # `between` ##############:

```
from DomesticCat cat where cat.name between 'A' and 'B'
```

```
from DomesticCat cat where cat.name in ( 'Foo', 'Bar', 'Baz' )
```

###################

```
from DomesticCat cat where cat.name not between 'A' and 'B'
```

```
from DomesticCat cat where cat.name not in ( 'Foo', 'Bar', 'Baz' )
```

`is null` # `is not null` #null#################

Hibernate#######HQL query substitutions#######boolean################

```
<property name="hibernate.query.substitutions">true 1, false
 0</property>
```

##########HQL#SQL######## `true` , `false` ###### `1` , `0` ##########:

```
from Cat cat where cat.alive = true
```

######## size######### size() ########################:

```
from Cat cat where cat.kittens.size > 0
```

```
from Cat cat where size(cat.kittens) > 0
```

##################minindex # maxindex ############################## ####minelement # maxelement ##### #############################

```
from Calendar cal where maxelement(cal.holidays) > current_date
```

```
from Order order where maxindex(order.items) > 100
```

```
from Order order where minelement(order.items) > 10000
```

######################elements # indices ############################### SQL## any, some, all, exists, in ##########

```
select mother from Cat as mother, Cat as kit
where kit in elements(foo.kittens)
```

```
select p from NameList list, Person p
where p.name = some elements(list.names)
```

```
from Cat cat where exists elements(cat.kittens)
```

```
from Player p where 3 > all elements(p.scores)
```

```
from Show show where 'fizard' in indices(show.acts)
```

size, elements, indices, minindex, maxindex, minelement, maxelement #Hibernate3#where#######################

################arrays, lists, maps###### ##############where#######

```
from Order order where order.items[0].id = 1234
```

```
select person from Person person, Calendar calendar
where calendar.holidays['national day'] = person.birthDay
    and person.nationality.calendar = calendar
```

```
select item from Item item, Order order
```

```
where order.items[ order.deliveredItemIndices[0] ] = item and
 order.id = 11
```

```
select item from Item item, Order order
where order.items[ maxindex(order.items) ] = item and order.id = 11
```

`[]` #################

```
select item from Item item, Order order
where order.items[ size(order.items) - 1 ] = item
```

#######################HQL###### `index()` ###########

```
select item, index(item) from Order order
    join order.items item
where index(item) < 5
```

#########################SQL#########

```
from DomesticCat cat where upper(cat.name) like 'FRI%'
```

#######################SQL#########################:

```
select cust
from Product prod,
    Store store
    inner join store.customers cust
where prod.name = 'widget'
    and store.location.name in ( 'Melbourne', 'Sydney' )
    and prod = all elements(cust.currentOrder.lineItems)
```

###: #############

```
SELECT cust.name, cust.address, cust.phone, cust.id,
 cust.current_order
FROM customers cust,
    stores store,
    locations loc,
    store_customers sc,
    product prod
WHERE prod.name = 'widget'
    AND store.loc_id = loc.id
    AND loc.name IN ( 'Melbourne', 'Sydney' )
    AND sc.store_id = store.id
    AND sc.cust_id = cust.id
    AND prod.id = ALL(
        SELECT item.prod_id
        FROM line_items item, orders o
        WHERE item.order_id = o.id
            AND cust.current_order = o.id
    )
```

14.11. order by#

######list##

```
from DomesticCat cat
order by cat.name asc, cat.weight desc, cat.birthdate
```

`asc` # `desc`

14.12. group by#

###:

```
select cat.color, sum(cat.weight), count(cat)
from Cat cat
group by cat.color
```

```
select foo.id, avg(name), max(name)
from Foo foo join foo.names name
group by foo.id
```

`having` #######

```
select cat.color, sum(cat.weight), count(cat)
from Cat cat
group by cat.color
having cat.color in (eg.Color.TABBY, eg.Color.BLACK)
```

########################## `having` # `order by` ##SQL############# ####MySQL#########

```
select cat
from Cat cat
    join cat.kittens kitten
group by cat.id, cat.name, cat.other, cat.properties
having avg(kitten.weight) > 100
order by count(kitten) asc, sum(kitten.weight) desc
```

Note that neither the `group by` clause nor the `order by` clause may contain arithmetic expressions. Also note that Hibernate currently does not expand a grouped entity, so you can't write `group by cat` if all properties of `cat` are non-aggregated. You have to list all non-aggregated properties explicitly.

14.13.

#########################Hibernate################### ########################SQL##################### ########(#########################)#########

```
from Cat as fatcat
where fatcat.weight > (
    select avg(cat.weight) from DomesticCat cat
)
```

```
from DomesticCat as cat
where cat.name = some (
    select name.nickName from Name as name
)
```

```
from Cat as cat
where not exists (
    from Cat as mate where mate.mate = cat
)
```

```
from DomesticCat as cat
where cat.name not in (
    select name.nickName from Name as name
)
```

```
select cat.id, (select max(kit.weight) from cat.kitten kit)
from Cat as cat
```

HQL########select###where####################

Note that subqueries can also utilize `row value constructor` syntax. See #14.18. #Row value constructor syntax# for more details.

14.14. HQL##

Hibernate################################Hibernate##################
#####################################
#####################################!

######################################ID#
##########################
#################################SQL#### `ORDER`, `ORDER_LINE`, `PRODUCT`, `CATALOG` ### `PRICE` #######4######## (#####)#############

```
select order.id, sum(price.amount), count(item)
from Order as order
    join order.lineItems as item
    join item.product as product,
    Catalog as catalog
    join catalog.prices as price
where order.paid = false
    and order.customer = :customer
    and price.product = product
    and catalog.effectiveDate < sysdate
```

```
    and catalog.effectiveDate >= all (
        select cat.effectiveDate
        from Catalog as cat
        where cat.effectiveDate < sysdate
    )
group by order
having sum(price.amount) > :minAmount
order by sum(price.amount) desc
```

################! ##:

```
select order.id, sum(price.amount), count(item)
from Order as order
    join order.lineItems as item
    join item.product as product,
    Catalog as catalog
    join catalog.prices as price
where order.paid = false
    and order.customer = :customer
    and price.product = product
    and catalog = :currentCatalog
group by order
having sum(price.amount) > :minAmount
order by sum(price.amount) desc
```

AWAITING_APPROVAL ############ ######2######## PAYMENT, PAYMENT_STATUS ### PAYMENT_STATUS_CHANGE #####################SQL###########

```
select count(payment), status.name
from Payment as payment
    join payment.currentStatus as status
    join payment.statusChanges as statusChange
where payment.status.name <> PaymentStatus.AWAITING_APPROVAL
    or (
        statusChange.timeStamp = (
            select max(change.timeStamp)
            from PaymentStatusChange change
            where change.payment = payment
        )
        and statusChange.user <> :currentUser
    )
group by status.name, status.sortOrder
order by status.sortOrder
```

####set#####list### statusChanges ####### ##################################

```
select count(payment), status.name
from Payment as payment
    join payment.currentStatus as status
where payment.status.name <> PaymentStatus.AWAITING_APPROVAL
```

```
    or payment.statusChanges[ maxIndex(payment.statusChanges) ].user
 <> :currentUser
group by status.name, status.sortOrder
order by status.sortOrder
```

#####MS SQL Server # `isNull()` ########### ######3#######1###### # ### `ACCOUNT`, `PAYMENT`, `PAYMENT_STATUS`, `ACCOUNT_TYPE`, `ORGANIZATION` ### `ORG_USER` #################SQL########

```
select account, payment
from Account as account
    left outer join account.payments as payment
where :currentUser in elements(account.holder.users)
    and PaymentStatus.UNPAID = isNull(payment.currentStatus.name,
 PaymentStatus.UNPAID)
order by account.type.sortOrder, account.accountNumber,
 payment.dueDate
```

#################(#######)#######################

```
select account, payment
from Account as account
    join account.holder.users as user
    left outer join account.payments as payment
where :currentUser = user
    and PaymentStatus.UNPAID = isNull(payment.currentStatus.name,
 PaymentStatus.UNPAID)
order by account.type.sortOrder, account.accountNumber,
 payment.dueDate
```

14.15. ###UPDATE#DELETE

HQL### `update` # `delete`#`insert ... select ...` ########HQL################# #13.4. #DMLã#¹ã#¿ã#¤ã#«ã#®æ##ä½## #########

14.16. Tips & Tricks

################################:

```
( (Integer) session.createQuery("select count(*) from
 ....").iterate().next() ).intValue()
```

##:

```
select usr.id, usr.name
from User as usr
    left join usr.messages as msg
```

```
group by usr.id, usr.name
order by count(msg)
```

####################################where####################:

```
from User usr where size(usr.messages) >= 1
```

###:

```
select usr.id, usr.name
from User usr.name
    join usr.messages msg
group by usr.id, usr.name
having count(msg) >= 1
```

###################message#### ### `User`
###########################

```
select usr.id, usr.name
from User as usr
    left join usr.messages as msg
group by usr.id, usr.name
having count(msg) = 0
```

JavaBean#######################################

```
Query q = s.createQuery("from foo Foo as foo where foo.name=:name
 and foo.size=:size");
q.setProperties(fooBean); // fooBean has getName() and getSize()
List foos = q.list();
```

############## `Query` ##################################:

```
Query q = s.createFilter( collection, "" ); // the trivial filter
q.setMaxResults(PAGE_SIZE);
q.setFirstResult(PAGE_SIZE * pageNumber);
List page = q.list();
```

##:

```
Collection orderedCollection = s.filter( collection, "order by
 this.amount" );
Collection counts = s.filter( collection, "select this.type,
 count(this) group by this.type" );
```

#####################################:

```
( (Integer) session.createQuery("select count(*) from
 ....").iterate().next() ).intValue();
```

14.17. translator-credits

Components might be used in just about every way that simple value types can be used in HQL queries. They can appear in the `select` clause:

```
select p.name from Person p
```

```
select p.name.first from Person p
```

where the Person's name property is a component. Components can also be used in the `where` clause:

```
from Person p where p.name = :name
```

```
from Person p where p.name.first = :firstName
```

Components can also be used in the `order by` clause:

```
from Person p order by p.name
```

```
from Person p order by p.name.first
```

Another common use of components is in row value constructors.

14.18. Row value constructor syntax

HQL supports the use of ANSI SQL `row value constructor` syntax (sometimes called `tuple` syntax), even though the underlying database may not support that notion. Here we are generally referring to multi-valued comparisons, typically associated with components. Consider an entity Person which defines a name component:

```
from Person p where p.name.first='John' and
 p.name.last='Jingleheimer-Schmidt'
```

That's valid syntax, although a little verbose. It be nice to make this a bit more concise and use `row value constructor` syntax:

```
from Person p where p.name=('John', 'Jingleheimer-Schmidt')
```

It can also be useful to specify this in the `select` clause:

```
select p.name from Person p
```

Another time using `row value constructor` syntax can be beneficial is when using subqueries needing to compare against multiple values:

```
from Cat as cat
where not ( cat.name, cat.color ) in (
    select cat.name, cat.color from DomesticCat cat
)
```

One thing to consider when deciding if you want to use this syntax is that the query will be dependent upon the ordering of the component sub-properties in the metadata.

#15# Criteria###

Hibernate############criteria###API##########

15.1. `Criteria`

`org.hibernate.Criteria` ################################## `Session` # `Criteria` ################

```
Criteria crit = sess.createCriteria(Cat.class);
crit.setMaxResults(50);
List cats = crit.list();
```

15.2.

`org.hibernate.criterion.Criterion` ################## ################################# `org.hibernate.criterion.Restrictions` ############ `Criterion` ############################

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "Fritz%") )
    .add( Restrictions.between("weight", minWeight, maxWeight) )
    .list();
```

Restriction#####################

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "Fritz%") )
    .add( Restrictions.or(
        Restrictions.eq( "age", new Integer(0) ),
        Restrictions.isNull("age")
    ) )
    .list();
```

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.in( "name", new String[] { "Fritz", "Izi",
 "Pk" } ) )
    .add( Restrictions.disjunction()
        .add( Restrictions.isNull("age") )
        .add( Restrictions.eq("age", new Integer(0) ) )
        .add( Restrictions.eq("age", new Integer(1) ) )
        .add( Restrictions.eq("age", new Integer(2) ) )
    ) )
    .list();
```

####Criterion##`Restrictions` ####### #####################SQL#############

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.sqlRestriction("lower({alias}.name) like
 lower(?)", "Fritz%", Hibernate.STRING) )
    .list();
```

{alias} ############ ###################################

criterion########## Property ################## Property.forName() ####### Property ###############

```

Property age = Property.forName("age");
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.disjunction()
        .add( age.isNull() )
        .add( age.eq( new Integer(0) ) )
        .add( age.eq( new Integer(1) ) )
        .add( age.eq( new Integer(2) ) )
    ) )
    .add( Property.forName("name").in( new String[] { "Fritz",
 "Izi", "Pk" } ) )
    .list();
```

15.3.

org.hibernate.criterion.Order #####################

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "F%")
    .addOrder( Order.asc("name") )
    .addOrder( Order.desc("age") )
    .setMaxResults(50)
    .list();
```

```
List cats = sess.createCriteria(Cat.class)
    .add( Property.forName("name").like("F%") )
    .addOrder( Property.forName("name").asc() )
    .addOrder( Property.forName("age").desc() )
    .setMaxResults(50)
    .list();
```

15.4.

createCriteria() ################### #########################

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "F%") )
    .createCriteria("kittens")
        .add( Restrictions.like("name", "F%") )
```

```
    .list();
```

2### `createCriteria()` ##`kittens` ################## `Criteria` ######################

#####################

```
List cats = sess.createCriteria(Cat.class)
    .createAlias("kittens", "kt")
    .createAlias("mate", "mt")
    .add( Restrictions.eqProperty("kt.name", "mt.name") )
    .list();
```

#`createAlias()` #### `Criteria` ################

##2############## `Cat` ################kittens######## criteria############## #### ############# ##criteria#####kitten########## `ResultTransformer` ##############

```
List cats = sess.createCriteria(Cat.class)
    .createCriteria("kittens", "kt")
        .add( Restrictions.eq("name", "F%") )
    .setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP)
    .list();
Iterator iter = cats.iterator();
while ( iter.hasNext() ) {
    Map map = (Map) iter.next();
    Cat cat = (Cat) map.get(Criteria.ROOT_ALIAS);
    Cat kitten = (Cat) map.get("kt");
}
```

15.5.

`setFetchMode()` #### ######################

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "Fritz%") )
    .setFetchMode("mate", FetchMode.EAGER)
    .setFetchMode("kittens", FetchMode.EAGER)
    .list();
```

############## `mate` # `kittens` ############ ######## #19.1. ######## ##########

15.6.

`org.hibernate.criterion.Example` ##### #################################

```
Cat cat = new Cat();
cat.setSex('F');
cat.setColor(Color.BLACK);
List results = session.createCriteria(Cat.class)
    .add( Example.create(cat) )
    .list();
```

######################### #######null################

`Example` ######

```
Example example = Example.create(cat)
    .excludeZeroes()           //exclude zero valued properties
    .excludeProperty("color")  //exclude the property named "color"
    .ignoreCase()              //perform case insensitive string
 comparisons
    .enableLike();             //use like for string comparisons
List results = session.createCriteria(Cat.class)
    .add(example)
    .list();
```

#########criteria#########Example###########

```
List results = session.createCriteria(Cat.class)
    .add( Example.create(cat) )
    .createCriteria("mate")
        .add( Example.create( cat.getMate() ) )
    .list();
```

15.7.

`org.hibernate.criterion.Projections` #### `Projection` ################ `setProjection()` ######### #############

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.rowCount() )
    .add( Restrictions.eq("color", Color.BLACK) )
    .list();
```

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.projectionList()
        .add( Projections.rowCount() )
        .add( Projections.avg("weight") )
        .add( Projections.max("weight") )
        .add( Projections.groupProperty("color") )
    )
    .list();
```

########criteria#####group by############### ####Projection## ####### ######## SQL# `group by` ########

#########################restriction#ordering##########
#####################

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.alias(
 Projections.groupProperty("color"), "colr" ) )
    .addOrder( Order.asc("colr") )
    .list();
```

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.groupProperty("color").as("colr") )
    .addOrder( Order.asc("colr") )
    .list();
```

`alias()` # `as()` ###### Projection############ `Projection` #######
########## ##

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.projectionList()
        .add( Projections.rowCount(), "catCountByColor" )
        .add( Projections.avg("weight"), "avgWeight" )
        .add( Projections.max("weight"), "maxWeight" )
        .add( Projections.groupProperty("color"), "color" )
    )
    .addOrder( Order.desc("catCountByColor") )
    .addOrder( Order.desc("avgWeight") )
    .list();
```

```
List results = session.createCriteria(Domestic.class, "cat")
    .createAlias("kittens", "kit")
    .setProjection( Projections.projectionList()
        .add( Projections.property("cat.name"), "catName" )
        .add( Projections.property("kit.name"), "kitName" )
    )
    .addOrder( Order.asc("catName") )
    .addOrder( Order.asc("kitName") )
    .list();
```

`Property.forName()`

```
List results = session.createCriteria(Cat.class)
    .setProjection( Property.forName("name") )
    .add( Property.forName("color").eq(Color.BLACK) )
    .list();
```

```
List results = session.createCriteria(Cat.class)
    .setProjection( Projections.projectionList()
        .add( Projections.rowCount().as("catCountByColor") )
        .add( Property.forName("weight").avg().as("avgWeight") )
        .add( Property.forName("weight").max().as("maxWeight") )
        .add( Property.forName("color").group().as("color" )
```

```
    )
    .addOrder( Order.desc("catCountByColor") )
    .addOrder( Order.desc("avgWeight") )
    .list();
```

15.8.

DetachedCriteria ####### ###################### ###### Session ############

```
DetachedCriteria query = DetachedCriteria.forClass(Cat.class)
    .add( Property.forName("sex").eq('F') );

Session session = ....;
Transaction txn = session.beginTransaction();
List results =
 query.getExecutableCriteria(session).setMaxResults(100).list();
txn.commit();
session.close();
```

DetachedCriteria ##################### ########Criterion######## Subqueries #### Property ########

```
DetachedCriteria avgWeight = DetachedCriteria.forClass(Cat.class)
    .setProjection( Property.forName("weight").avg() );
session.createCriteria(Cat.class)
    .add( Property.forName("weight").gt(avgWeight) )
    .list();
```

```
DetachedCriteria weights = DetachedCriteria.forClass(Cat.class)
    .setProjection( Property.forName("weight") );
session.createCriteria(Cat.class)
    .add( Subqueries.geAll("weight", weights) )
    .list();
```

#####################

```
DetachedCriteria avgWeightForSex =
 DetachedCriteria.forClass(Cat.class, "cat2")
    .setProjection( Property.forName("weight").avg() )
    .add( Property.forName("cat2.sex").eqProperty("cat.sex") );
session.createCriteria(Cat.class, "cat")
    .add( Property.forName("weight").gt(avgWeightForSex) )
    .list();
```

15.9.

criteria################### ###################### ########################### ################################

################# ##################
################################## ##############criteria API#############

####`<natural-id>` ##### ######################## ###################

```
<class name="User">
    <cache usage="read-write"/>
    <id name="id">
        <generator class="increment"/>
    </id>
    <natural-id>
        <property name="name"/>
        <property name="org"/>
    </natural-id>
    <property name="password"/>
</class>
```

#########################

###Hibernate################

####`Restrictions.naturalId()` #### ##########################

```
session.createCriteria(User.class)
    .add( Restrictions.naturalId()
        .set("name", "gavin")
        .set("org", "hb")
    ).setCacheable(true)
    .uniqueResult();
```

#16# #####SQL

############SQL###################### #######Oracle# `CONNECT` ########## ########################### SQL/JDBC##################Hibernate##############

Hibernate3########################### SQL########################

16.1. `SQLQuery`

######SQL####### `SQLQuery` ################## `SQLQuery` ######### `Session.createSQLQuery()` ############ ##API#######################

16.1.1.

######SQL#######################

```
sess.createSQLQuery("SELECT * FROM CATS").list();
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").list();
```

#########CATS################## Object###Object[]########### ######################## Hibernate#ResultSetMetadata#######

`ResultSetMetadata` ###################### #####################`addScalar()` ######

```
sess.createSQLQuery("SELECT * FROM CATS")
 .addScalar("ID", Hibernate.LONG)
 .addScalar("NAME", Hibernate.STRING)
 .addScalar("BIRTHDATE", Hibernate.DATE)
```

########################

- SQL######
- #########

This will still return Object arrays, but now it will not use `ResultSetMetadata` but will instead explicitly get the ID, NAME and BIRTHDATE column as respectively a Long, String and a Short from the underlying resultset. This also means that only these three columns will be returned, even though the query is using `*` and could return more than the three listed columns.

####################

```
sess.createSQLQuery("SELECT * FROM CATS")
 .addScalar("ID", Hibernate.LONG)
```

```
.addScalar("NAME")
.addScalar("BIRTHDATE")
```

#################### NAME # BIRTHDATE ######### ResultSetMetaData ##########ID ################

ResultSetMetaData ###### java.sql.Types # Hibernate ### ###########Dialect ####### ### Dialect # registerHibernateType ###### ###########

16.1.2.

############################# ############################ #####addEntity() #########SQL##### ##########################

```
sess.createSQLQuery("SELECT * FROM CATS").addEntity(Cat.class);
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM
 CATS").addEntity(Cat.class);
```

##########################

- SQL######
- #############SQL#######

Cat#ID, NAME, BIRTHDATE######################## ###############Cat###################

################# ### ############### ### ##################column not found(###########)########### * ############################## ########Dog # ### ###### #################

```
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE, DOG_ID FROM
 CATS").addEntity(Cat.class);
```

cat.getDog()

16.1.3.

########################## Dog ############ ### addJoin() ############ #####################

```
sess.createSQLQuery("SELECT c.ID, NAME, BIRTHDATE, DOG_ID, D_ID,
 D_NAME FROM CATS c, DOGS d WHERE c.DOG_ID = d.D_ID")
 .addEntity("cat", Cat.class)
 .addJoin("cat.dog");
```

########### Cat ## ################ ######### dog ###########
######################## ####cat###################
###################### #####Cat ##### Dog ###################

```
sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE, D_ID, D_NAME,
 CAT_ID FROM CATS c, DOGS d WHERE c.ID = d.CAT_ID")
 .addEntity("cat", Cat.class)
 .addJoin("cat.dogs");
```

####### Hibernate##########SQL##########
##########################
###########

16.1.4.

#################### ##################################
######################## ###########SQL#################

#################### ##############column alias injection#######

```
sess.createSQLQuery("SELECT c.*, m.*  FROM CATS c, CATS m WHERE
 c.MOTHER_ID = c.ID")
 .addEntity("cat", Cat.class)
 .addEntity("mother", Cat.class)
```

The intention for this query is to return two Cat instances per row, a cat and its mother. This will fail since there is a conflict of names since they are mapped to the same column names and on some databases the returned column aliases will most likely be on the form "c.ID", "c.NAME", etc. which are not equal to the columns specified in the mappings ("ID" and "NAME").

#######################

```
sess.createSQLQuery("SELECT {cat.*}, {mother.*}  FROM CATS c, CATS m
 WHERE c.MOTHER_ID = c.ID")
 .addEntity("cat", Cat.class)
 .addEntity("mother", Cat.class)
```

########################

- SQL###### #Hibernate##########################
- #################

The {cat.*} and {mother.*} notation used above is a shorthand for "all properties". Alternatively, you may list the columns explicitly, but even in this case we let Hibernate inject the SQL column aliases for each property. The placeholder for a column alias is just the property name qualified by the table alias. In the following example, we retrieve Cats and their mothers from a different table (cat_log) to the one declared in the mapping metadata. Notice that we may even use the property aliases in the where clause if we like.

```
String sql = "SELECT ID as {c.id}, NAME as {c.name}, " +
         "BIRTHDATE as {c.birthDate}, MOTHER_ID as {c.mother},
 {mother.*} " +
         "FROM CAT_LOG c, CAT_LOG m WHERE {c.mother} = c.ID";

List loggedCats = sess.createSQLQuery(sql)
        .addEntity("cat", Cat.class)
        .addEntity("mother", Cat.class).list()
```

16.1.4.1.

############################### ###############################
############################
###################Hibernate##############

############################### ###############
#######################################

 # 16.1. #########

##	##	#
########	{[aliasname].[propertyname]}	A_NAME as {item.name}
#######	{[aliasname].[componentname].[propertyname]}	CURRENCY as {item.amount.currency}, VALUE as {item.amount.value}
################	{[aliasname].class}	DISC as {item.class}
#############	{[aliasname].*}	{item.*}
#########	{[aliasname].key}	ORGID as {coll.key}
#######ID	{[aliasname].id}	EMPID as {coll.id}
#########	{[aliasname].element}	XID as {coll.element}
###############	{[aliasname].element.[propertyname]}	NAME as {coll.element.name}
################	{[aliasname].element.*}	{coll.element.*}
#############	{[aliasname].*}	{coll.*}

16.1.5.

#####SQL#### ResultTransformer ########
#################################

```
sess.createSQLQuery("SELECT NAME, BIRTHDATE FROM CATS")

 .setResultTransformer(Transformers.aliasToBean(CatDTO.class))
```

########################

- SQL######
- #########

#################NAME # BIRTHDATE ### ######################## CatDTO ##########

16.1.6.

##################################SQL##### ###

16.1.7.

#####SQL############ ##########:name####################

```
Query query = sess.createSQLQuery("SELECT * FROM CATS WHERE NAME
 like ?").addEntity(Cat.class);
List pusList = query.setString(0, "Pus%").list();

query = sess.createSQLQuery("SELECT * FROM CATS WHERE NAME like
 :name").addEntity(Cat.class);
List pusList = query.setString("name", "Pus%").list();
```

16.2. ####SQL###

####SQL######################### ####HQL##################### #####addEntity() ######## ## ###

```
<sql-query name="persons">
    <return alias="person" class="eg.Person"/>
    SELECT person.NAME AS {person.name},
           person.AGE AS {person.age},
           person.SEX AS {person.sex}
    FROM PERSON person
    WHERE person.NAME LIKE :namePattern
</sql-query>
```

```
List people = sess.getNamedQuery("persons")
    .setString("namePattern", namePattern)
    .setMaxResults(50)
    .list();
```

###################################### <return-join> # <load-collection> ########

```
<sql-query name="personsWith">
    <return alias="person" class="eg.Person"/>
```

```
    <return-join alias="address" property="person.mailingAddress"/>
    SELECT person.NAME AS {person.name},
           person.AGE AS {person.age},
           person.SEX AS {person.sex},
           address.STREET AS {address.street},
           address.CITY AS {address.city},
           address.STATE AS {address.state},
           address.ZIP AS {address.zip}
    FROM PERSON person
    JOIN ADDRESS address
        ON person.ID = address.PERSON_ID AND address.TYPE='MAILING'
    WHERE person.NAME LIKE :namePattern
</sql-query>
```

####SQL################### `<return-scalar>` ####### #####Hibernate################

```
<sql-query name="mySqlQuery">
    <return-scalar column="name" type="string"/>
    <return-scalar column="age" type="long"/>
    SELECT p.NAME AS name,
           p.AGE AS age,
    FROM PERSON p WHERE p.NAME LIKE 'Hiber%'
</sql-query>
```

You can externalize the resultset mapping informations in a `<resultset>` element to either reuse them across several named queries or through the `setResultSetMapping()` API.

```
<resultset name="personAddress">
    <return alias="person" class="eg.Person"/>
    <return-join alias="address" property="person.mailingAddress"/>
</resultset>

<sql-query name="personsWith" resultset-ref="personAddress">
    SELECT person.NAME AS {person.name},
           person.AGE AS {person.age},
           person.SEX AS {person.sex},
           address.STREET AS {address.street},
           address.CITY AS {address.city},
           address.STATE AS {address.state},
           address.ZIP AS {address.zip}
    FROM PERSON person
    JOIN ADDRESS address
        ON person.ID = address.PERSON_ID AND address.TYPE='MAILING'
    WHERE person.NAME LIKE :namePattern
</sql-query>
```

#####hbm##################### ##Java#############

```
List cats = sess.createSQLQuery(
```

```
        "select {cat.*}, {kitten.*} from cats cat, cats kitten where
 kitten.mother = cat.id"
    )
    .setResultSetMapping("catAndKitten")
    .list();
```

16.2.1. ################### return-property

########## {} ########## <return-property> ####
####################

```
<sql-query name="mySqlQuery">
    <return alias="person" class="eg.Person">
        <return-property name="name" column="myName"/>
        <return-property name="age" column="myAge"/>
        <return-property name="sex" column="mySex"/>
    </return>
    SELECT person.NAME AS myName,
           person.AGE AS myAge,
           person.SEX AS mySex,
    FROM PERSON person WHERE person.NAME LIKE :name
</sql-query>
```

<return-property> ########### ############################## {}
############

```
<sql-query name="organizationCurrentEmployments">
    <return alias="emp" class="Employment">
        <return-property name="salary">
            <return-column name="VALUE"/>
            <return-column name="CURRENCY"/>
        </return-property>
        <return-property name="endDate" column="myEndDate"/>
    </return>
        SELECT EMPLOYEE AS {emp.employee}, EMPLOYER AS
 {emp.employer},
        STARTDATE AS {emp.startDate}, ENDDATE AS {emp.endDate},
        REGIONCODE as {emp.regionCode}, EID AS {emp.id}, VALUE,
 CURRENCY
        FROM EMPLOYMENT
        WHERE EMPLOYER = :id AND ENDDATE IS NULL
        ORDER BY STARTDATE ASC
</sql-query>
```

############ {} ######### <return-property> ##################
#########################

discriminator ########## discriminator
###########<return-discriminator> #############

16.2.2.

Hibernate######3############################## ########## ##################### ###################Hibernate######## 1################################## Oracle 9#######################################

```
CREATE OR REPLACE FUNCTION selectAllEmployments
    RETURN SYS_REFCURSOR
AS
    st_cursor SYS_REFCURSOR;
BEGIN
    OPEN st_cursor FOR
 SELECT EMPLOYEE, EMPLOYER,
 STARTDATE, ENDDATE,
 REGIONCODE, EID, VALUE, CURRENCY
 FROM EMPLOYMENT;
      RETURN  st_cursor;
 END;
```

Hibernate############## #######################

```
<sql-query name="selectAllEmployees_SP" callable="true">
    <return alias="emp" class="Employment">
        <return-property name="employee" column="EMPLOYEE"/>
        <return-property name="employer" column="EMPLOYER"/>
        <return-property name="startDate" column="STARTDATE"/>
        <return-property name="endDate" column="ENDDATE"/>
        <return-property name="regionCode" column="REGIONCODE"/>
        <return-property name="id" column="EID"/>
        <return-property name="salary">
            <return-column name="VALUE"/>
            <return-column name="CURRENCY"/>
        </return-property>
    </return>
    { ? = call selectAllEmployments() }
</sql-query>
```

####################################### `<return-join>` # `<load-collection>` #############

16.2.2.1.

Hibernate###################### ############################## ###################Hibernate############ ############################### `session.connection()` ################# #################### ############################# #####################

setFirstResult()/setMaxResults() ##### ###############################

################SQL92######## { ? = call functionName(<parameters>) } # { ? = call procedureName(<parameters>) } ### ##########################

Oracle################

- ######################## ########################## OUT ########### Oracle 9#10###SYS_REFCURSOR ############### Oracle## REF CURSOR ############### Oracle#############

Sybase#MS SQL######################

- ############################ ############################### Hibernate#1############################ ###############
- ######### SET NOCOUNT ON ######### ################# ####################

16.3. #################SQL

Hibernate3##################SQL######### ## #insertsql#deletesql#updatesql################### ########## <sql-insert># <sql-delete># <sql-update> ###################

```
<class name="Person">
    <id name="id">
        <generator class="increment"/>
    </id>
    <property name="name" not-null="true"/>
    <sql-insert>INSERT INTO PERSON (NAME, ID) VALUES ( UPPER(?), ?
 )</sql-insert>
    <sql-update>UPDATE PERSON SET NAME=UPPER(?) WHERE
 ID=?</sql-update>
    <sql-delete>DELETE FROM PERSON WHERE ID=?</sql-delete>
</class>
```

SQL################################### #########SQL############################

callable ########## ###################

```
<class name="Person">
    <id name="id">
        <generator class="increment"/>
    </id>
    <property name="name" not-null="true"/>
```

```
    <sql-insert callable="true">{call createPerson (?,
 ?)}</sql-insert>
    <sql-delete callable="true">{? = call deletePerson
 (?)}</sql-delete>
    <sql-update callable="true">{? = call updatePerson (?,
 ?)}</sql-update>
</class>
```

########################## #####Hibernate##################

`org.hiberante.persister.entity` ###########
##############################
###################################### ########SQL########
###################Hibernate########SQL##########
####SQL###############################

##############/##/##################
############################# ####Hibernate#SQL################
Hibernate##CUD#################### SQL####################

```
CREATE OR REPLACE FUNCTION updatePerson (uid IN NUMBER, uname IN
 VARCHAR2)
    RETURN NUMBER IS
BEGIN

    update PERSON
    set
        NAME = uname,
    where
        ID = uid;

    return SQL%ROWCOUNT;

END updatePerson;
```

16.4. ###########SQL

#################SQL#####HQL############

```
<sql-query name="person">
    <return alias="pers" class="Person" lock-mode="upgrade"/>
    SELECT NAME AS {pers.name}, ID AS {pers.id}
    FROM PERSON
    WHERE ID=?
    FOR UPDATE
</sql-query>
```

############################## ##############################

```
<class name="Person">
```

```
    <id name="id">
        <generator class="increment"/>
    </id>
    <property name="name" not-null="true"/>
    <loader query-ref="person"/>
</class>
```

#######################

###################################

```
<set name="employments" inverse="true">
    <key/>
    <one-to-many class="Employment"/>
    <loader query-ref="employments"/>
</set>
```

```
<sql-query name="employments">
    <load-collection alias="emp" role="Person.employments"/>
    SELECT {emp.*}
    FROM EMPLOYMENT emp
    WHERE EMPLOYER = :id
    ORDER BY STARTDATE ASC, EMPLOYEE ASC
</sql-query>
```

########################### ##################

```
<sql-query name="person">
    <return alias="pers" class="Person"/>
    <return-join alias="emp" property="pers.employments"/>
    SELECT NAME AS {pers.*}, {emp.*}
    FROM PERSON pers
    LEFT OUTER JOIN EMPLOYMENT emp
        ON pers.ID = emp.PERSON_ID
    WHERE ID=?
</sql-query>
```

#17#

Hibernate3##
Hibernate filter ################################
###Hibernate#######################

17.1. Hibernate#####

Hibernate3####################
######################################
########################## ###############################
#########################where##############
##############################
######################################
##

###############
`<hibernate-mapping/>` #### `<filter-def/>` ##########

```
<filter-def name="myFilter">
    <filter-param name="myFilterParam" type="string"/>
</filter-def>
```

##########################

```
<class name="myClass" ...>
    ...
    <filter name="myFilter" condition=":myFilterParam =
 MY_FILTERED_COLUMN"/>
</class>
```

###########################

```
<set ...>
    <filter name="myFilter" condition=":myFilterParam =
 MY_FILTERED_COLUMN"/>
</set>
```

########(##########)###############

The methods on `Session` are: `enableFilter(String filterName)`, `getEnabledFilter(String filterName)`, and `disableFilter(String filterName)`. By default, filters are *not* enabled for a given session; they must be explcitly enabled through use of the `Session.enableFilter()` method, which returns an instance of the `Filter` interface. Using the simple filter defined above, this would look like:

```
session.enableFilter("myFilter").setParameter("myFilterParam",
 "some-value");
```

org.hibernate.Filter###############
Hibernate####################################

##################################

```
<filter-def name="effectiveDate">
    <filter-param name="asOfDate" type="date"/>
</filter-def>

<class name="Employee" ...>
...
    <many-to-one name="department" column="dept_id"
 class="Department"/>
    <property name="effectiveStartDate" type="date"
 column="eff_start_dt"/>
    <property name="effectiveEndDate" type="date"
 column="eff_end_dt"/>
...
    <!--
        Note that this assumes non-terminal records have an
 eff_end_dt set to
        a max db date for simplicity-sake
    -->
    <filter name="effectiveDate"
            condition=":asOfDate BETWEEN eff_start_dt and
 eff_end_dt"/>
</class>

<class name="Department" ...>
...
    <set name="employees" lazy="true">
        <key column="dept_id"/>
        <one-to-many class="Employee"/>
        <filter name="effectiveDate"
                condition=":asOfDate BETWEEN eff_start_dt and
 eff_end_dt"/>
    </set>
</class>
```

############################
####################################

```
Session session = ...;
session.enableFilter("effectiveDate").setParameter("asOfDate", new
 Date());
List results = session.createQuery("from Employee as e where
 e.salary > :targetSalary")
         .setLong("targetSalary", new Long(1000000))
         .list();
```

###HQL############################
############################100####################

(HQL################################ #################
########################## #####################################

translator-credits

```
<filter-def name="myFilter" condition="abc > xyz">...</filter-def>
<filter-def name="myOtherFilter">abc=xyz</filter-def>
```

This default condition will then be used whenever the filter is attached to something without specifying a condition. Note that this means you can give a specific condition as part of the attachment of the filter which overrides the default condition in that particular case.

#18# XML#####

XML######Hibernate3.0#########################

18.1. XML#######

Hibernate######POJO######################
####XML##############

Hibernate#XML###########API###dom4j###########
########dom4j#####################
################################
##XML#################dom4j################
Hibernate##################################
####`persist(), saveOrUpdate(), merge(), delete(), replicate()`
####(#################)#

#########/####### JMS#################SOAP#XSLT###########
#################

#####################XML###########
############################### ##################
XML#######################

18.1.1. XML###################

###POJO#XML################

```
<class name="Account"
        table="ACCOUNTS"
        node="account">

    <id name="accountId"
            column="ACCOUNT_ID"
            node="@id"/>

    <many-to-one name="customer"
            column="CUSTOMER_ID"
            node="customer/@id"
            embed-xml="false"/>

    <property name="balance"
            column="BALANCE"
            node="balance"/>

    ...

</class>
```

18.1.2. XML############

###POJO################

```
<class entity-name="Account"
        table="ACCOUNTS"
        node="account">

    <id name="id"
            column="ACCOUNT_ID"
            node="@id"
            type="string"/>

    <many-to-one name="customerId"
            column="CUSTOMER_ID"
            node="customer/@id"
            embed-xml="false"
            entity-name="Customer"/>

    <property name="balance"
            column="BALANCE"
            node="balance"
            type="big_decimal"/>

    ...

</class>
```

###########dom4j##### ######/########java# `Map#### ############# ##########HQL#####################

18.2. XML###########

###Hibernate######### `node` ########## #####XML## `node` ###############1#############

- `"element-name"` - ####XML###########
- `"@attribute-name"` - ####XML###########
- `"."` - ############
- `"element-name/@attribute-name"` - ##########################

################### #### `embed-xml` ######## ###### `embed-xml="true"` ######## ###########(##########)#XML##### ################XML############# ####`embed-xml="false"` ######## #########################XML#### #######################

############# `embed-xml="true"` ################### XML#############

```
<class name="Customer"
        table="CUSTOMER"
        node="customer">

    <id name="id"
            column="CUST_ID"
            node="@id"/>

    <map name="accounts"
            node="."
            embed-xml="true">
        <key column="CUSTOMER_ID"
                not-null="true"/>
        <map-key column="SHORT_DESC"
                node="@short-desc"
                type="string"/>
        <one-to-many entity-name="Account"
                embed-xml="false"
                node="account"/>
    </map>

    <component name="name"
            node="name">
        <property name="firstName"
                node="first-name"/>
        <property name="initial"
                node="initial"/>
        <property name="lastName"
                node="last-name"/>
    </component>

    ...

</class>
```

#########account######### account#id####################
###HQL######

```
from Customer c left join fetch c.accounts where c.lastName like
 :lastName
```

##################

```
<customer id="123456789">
    <account short-desc="Savings">987632567</account>
    <account short-desc="Credit Card">985612323</account>
    <name>
        <first-name>Gavin</first-name>
        <initial>A</initial>
        <last-name>King</last-name>
    </name>
    ...
```

```
</customer>
```

<one-to-many> ###### embed-xml="true" ######## ################

```
<customer id="123456789">
    <account id="987632567" short-desc="Savings">
        <customer id="123456789"/>
        <balance>100.29</balance>
    </account>
    <account id="985612323" short-desc="Credit Card">
        <customer id="123456789"/>
        <balance>-2370.34</balance>
    </account>
    <name>
        <first-name>Gavin</first-name>
        <initial>A</initial>
        <last-name>King</last-name>
    </name>
    ...
</customer>
```

18.3. XML######

XML####################################
####dom4j####################

```
Document doc = ....;

Session session = factory.openSession();
Session dom4jSession = session.getSession(EntityMode.DOM4J);
Transaction tx = session.beginTransaction();

List results = dom4jSession
    .createQuery("from Customer c left join fetch c.accounts where
 c.lastName like :lastName")
    .list();
for ( int i=0; i<results.size(); i++ ) {
    //add the customer data to the XML document
    Element customer = (Element) results.get(i);
    doc.add(customer);
}

tx.commit();
session.close();
```

```
Session session = factory.openSession();
Session dom4jSession = session.getSession(EntityMode.DOM4J);
Transaction tx = session.beginTransaction();

Element cust = (Element) dom4jSession.get("Customer", customerId);
for ( int i=0; i<results.size(); i++ ) {
    Element customer = (Element) results.get(i);
```

```
    //change the customer name in the XML and database
    Element name = customer.element("name");
    name.element("first-name").setText(firstName);
    name.element("initial").setText(initial);
    name.element("last-name").setText(lastName);
}

tx.commit();
session.close();
```

XML#############/################ Hibernate# `replicate()` ################ ########

#19#

19.1.

###########################
####Hibernate##################################O/R######
###############HQL# `Criteria` ################

Hibernate3####################

- ###### - Hibernate# `OUTER JOIN` ##### ###################### `SELECT` #######
- ######## - 2### `SELECT` ########################## `lazy="false"` ########################2###select#### ########################
- ########### - 2### `SELECT` ## ###### `lazy="false"` ########################2###select#### ########################
- ####### - ###################### - Hibernate##########################1## `SELECT` ################################ ########

Hibernate################

- ###### - ###################### #######################
- ############# - ################# #############################(########################)
- *"#####"*########### - ########## ##################################### Hibernate################################ ###########################
- ######## - ###########getter### ##################################
- *"######"*#### - ################# ### ####################################### ## #######################################
- ######## - ########################## ## ##########

###############: ## ########### #### ##### ###########SQL###### ############### fetch #################### lazy ############################## ####################

19.1.1.

########Hibernate3######################### ### ###########################

#:hibernate.default_batch_fetch_size ##########Hibernate# ############################ ###############################

#################################### Hibernate#session################################# ###########

```
s = sessions.openSession();
Transaction tx = s.beginTransaction();

User u = (User) s.createQuery("from User u where u.name=:userName")
    .setString("userName", userName).uniqueResult();
Map permissions = u.getPermissions();

tx.commit();
s.close();

Integer accessLevel = (Integer) permissions.get("accounts");  //
 Error!
```

Session ###########permissions####### ###################################### *Hibernate*######################### ###### ############################## ##########################

lazy="false" ################# ############################# ###################################### ####################################### ###########Hibernate################

################################### #################################### ###################################### Hibernate3################################# ##########

19.1.2.

################N+1####################
#############################

```
<set name="permissions"
            fetch="join">
    <key column="userId"/>
    <one-to-many class="Permission"/>
</set
```

```
<many-to-one name="mother" class="Cat" fetch="join"/>
```

############ #### ###############

- `get()` # `load()` #####
- ########################
- `Criteria` ###
- ###### #######HQL###

###
##############HQL################SELECT#########
################

################################ #######################HQL# `left join fetch` # ##################################
###Hibernate## `Criteria` ####API### `setFetchMode(FetchMode.JOIN)` ###########

`get()` # `load()` ##### ######################## `Criteria`

```
User user = (User) session.createCriteria(User.class)
                .setFetchMode("permissions", FetchMode.JOIN)
                .add( Restrictions.idEq(userId) )
                .uniqueResult();
```

#########ORM########"fetch plan"###############

N+1########################2################

19.1.3.

###############Hibernate####################
###################################

##################################Hibernate#
######CGLIB######################
################################

########Hibernate3########################### ######## many-to-one # one-to-one ### #################

########## proxy ###########################
###########################Hibernate##################
##
###

##

```
<class name="Cat" proxy="Cat">
    ......
    <subclass name="DomesticCat">
        .....
    </subclass>
</class>
```

Cat ######## DomesticCat ########################### DomesticCat

```
Cat cat = (Cat) session.load(Cat.class, id);  // instantiate a proxy
 (does not hit the db)
if ( cat.isDomesticCat() ) {                  // hit the db to
 initialize the proxy
    DomesticCat dc = (DomesticCat) cat;       // Error!
    ....
}
```

######### == ###############

```
Cat cat = (Cat) session.load(Cat.class, id);            //
 instantiate a Cat proxy
DomesticCat dc =
        (DomesticCat) session.load(DomesticCat.class, id);  //
 acquire new DomesticCat proxy!
System.out.println(cat==dc);                            // false
```

###
######################################

```
cat.setWeight(11.0);  // hit the db to initialize the proxy
System.out.println( dc.getWeight() );  // 11.0
```

final #### final ########### CGLIB###########

########################(##########################)
##
############################

#######Java#######################################
##
######################################

```
<class name="CatImpl" proxy="Cat">
    ......
    <subclass name="DomesticCatImpl" proxy="DomesticCat">
        .....
    </subclass>
</class>
```

`CatImpl` # `Cat` ################## `DomesticCatImpl` # `DomesticCat` ####### #### `load()` # `iterate()` ## `Cat` # `DomesticCat` ############# ###### `list()` ###########################

```
Cat cat = (Cat) session.load(CatImpl.class, catid);
Iterator iter = session.createQuery("from CatImpl as cat where
 cat.name='fritz'").iterate();
Cat fritz = (Cat) iter.next();
```

####################### `Cat` ########## ############## `CatImpl` #########

######### ###### #########

- `equals()` ####### `equals()` # ##############
- `hashCode()` ####### `hashCode()` # ##############
- ####getter####

Hibernate# `equals()` # `hashCode()` ########## ############

`lazy="proxy"` ###### `lazy="no-proxy"` #
##
##########################

19.1.4.

`LazyInitializationException` ## `Session` ######## #################################Hibernate############ ###

`Session` ############################# ################## `cat.getSex()` # `cat.getKittens().size()` ############################# #######################################

static##### `Hibernate.initialize()` # `Hibernate.isInitialized()` ### `Hibernate.initialize(cat)` ## `Session` ############ `cat` ################ `Hibernate.initialize( cat.getKittens() )` #kittens############# ########

###################################### `Session` ################################## ############Hibernate######################## ### ################# `Session` ############ ######################2##############

- Web############### ############################### `Session` ################################# *Open Session in View* ## ###################################### `Session` #################### ############ Hibernate#Wiki######"Open Session in View"################

- ################################## Web######################"##"########### #############################/Web##### ####################################### ###########################Web########### ######## `Hibernate.initialize()` ####### ################################### Hibernate#### `FETCH` ## `Criteria` # `FetchMode.JOIN` #################### ### *Session Facade* ######### *Command* ####################

- ####################################### `merge()` # `lock()` ####### `Session` ############################### ################################Hibernate# ########### ########### #

#########################

###

```
( (Integer) s.createFilter( collection, "select count(*)"
 ).list().get(0) ).intValue()
```

`createFilter()` ################################### #######################

```
s.createFilter( lazyCollection,
 "").setFirstResult(0).setMaxResults(10).list();
```

19.1.5.

Hibernate### #####Hibernate## ### ###################

Session #######25## Cat ########### ##### Cat # owner ### Person ########## Person #### lazy="true" ################## #######Cat######## getOwner() #######Hibernate# #######25## SELECT #####owner########### ####### Person ####### batch-size

```
<class name="Person" batch-size="10">...</class>
```

Hibernate###########################10, 10, 5###

You may also enable batch fetching of collections. For example, if each Person has a lazy collection of Cats, and 10 persons are currently loaded in the Session, iterating through all persons will generate 10 SELECTs, one for every call to getCats(). If you enable batch fetching for the cats collection in the mapping of Person, Hibernate can pre-fetch collections:

```
<class name="Person">
    <set name="cats" batch-size="3">
        ...
    </set>
</class>
```

batch-size #3####Hibernate#4## SELECT #3##3##3##1######################## Session ##############################

############################## ################## ############################## #####*set* # ####### ##############

19.1.6.

######################################Hibernate# ### ################################

19.1.7.

Hibernate3################################## ######### ################### ### ##################

##
#############################

###################################### `lazy` #############

```
<class name="Document">
       <id name="id">
        <generator class="native"/>
    </id>
    <property name="name" not-null="true" length="50"/>
    <property name="summary" not-null="true" length="200"
 lazy="true"/>
    <property name="text" not-null="true" length="2000"
 lazy="true"/>
</class>
```

#######################################
####################Hibernate################### ############

###############Ant############

```
<target name="instrument" depends="compile">
    <taskdef name="instrument"
 classname="org.hibernate.tool.instrument.InstrumentTask">
        <classpath path="${jar.path}"/>
        <classpath path="${classes.dir}"/>
        <classpath refid="lib.class.path"/>
    </taskdef>

    <instrument verbose="true">
        <fileset
 dir="${testclasses.dir}/org/hibernate/auction/model">
            <include name="*.class"/>
        </fileset>
    </instrument>
</target>
```

A different (better?) way to avoid unnecessary column reads, at least for read-only transactions is to use the projection features of HQL or Criteria queries. This avoids the need for buildtime bytecode processing and is certainly a preferred solution.

HQL# `fetch all properties` #############
############################

19.2. #2########

Hibernate# `Session` ############################
class-by-class#collection-by-collection############JVM###
`SessionFactory`

##
###########################
#####################################

Hibernate##############hibernate.cache.provider_class ######
org.hibernate.cache.CacheProvider #########################
Hibernate##
##
#####3.2#####EhCache##################################
#####3.2###############

19.1.

#####	########	###	#######	#############
Hashtable##############	org.hibernate.cache.HashtableCacheProvider	###		yes
EHCache	org.hibernate.cache.EhCacheProvider	##########		yes
OSCache	org.hibernate.cache.OSCacheProvider	##########		yes
SwarmCache	org.hibernate.cache.SwarmCacheProvider	####ip###	yes###########	
JBoss Cache 1.x	org.hibernate.cache.TreeCacheProvider	####ip###	yes########	yes##########
JBoss Cache 2	org.hibernate.cache.jbc2.JBossCacheRegionFactory	####ip###	yes (replication or invalidation)	yes##########

19.2.1.

################# <cache> ###########

```
<cache
    usage="transactional|read-write|nonstrict-read-write|read-only"
  (1)
    region="RegionName"
  (2)
    include="all|non-lazy"
  (3)
/>
```

(1) `usage` (required) specifies the caching strategy: `transactional`, `read-write`, `nonstrict-read-write` or `read-only`

(2) `region` (optional, defaults to the class or collection role name) specifies the name of the second level cache region

(3) `include` (optional, defaults to `all`) `non-lazy` specifies that properties of the entity mapped with `lazy="true"` may not be cached when attribute-level lazy fetching is enabled

Alternatively (preferably?), you may specify `<class-cache>` and `<collection-cache>` elements in `hibernate.cfg.xml`.

`usage` ### ########### #######

19.2.2. read only##

`read-only` ##############################

```
<class name="eg.Immutable" mutable="false">
    <cache usage="read-only"/>
    ....
</class>
```

19.2.3. read/write##

############################# `read-write` ############ ## ###########################JTA############JTA `TransactionManager` ############### `hibernate.transaction.manager_lookup_class` ############################## `Session.close()` # `Session.disconnect()` ########################### ############### ### ####################### ################### ############ #

```
<class name="eg.Cat" .... >
    <cache usage="read-write"/>
    ....
    <set name="kittens" ... >
        <cache usage="read-write"/>
        ....
    </set>
</class>
```

19.2.4. ######read/write##

`nonstrict-read-write` ################### #########JTA############ `hibernate.transaction.manager_lookup_class` ######################## `Session.close()` # `Session.disconnect()` ###########################

19.2.5. transactional##

`transactional` ########JBoss TreeCache################ ####################### ###########JTA########### `hibernate.transaction.manager_lookup_class` ##############

19.2.6. Cache-provider/concurrency-strategy compatibility

####

None of the cache providers support all of the cache concurrency strategies.

The following table shows which providers are compatible with which concurrency strategies.

 # 19.2. #################

#####	read-only	######read-write	read-write	transactional
Hashtable#########	yes###########	yes	yes	
EHCache	yes	yes	yes	
OSCache	yes	yes	yes	
SwarmCache	yes	yes		
JBoss Cache 1.x	yes			yes
JBoss Cache 2	yes			yes

19.3.

####### `save()` # `update()` # `saveOrUpdate()` ######### `load()` # `get()` # `list()` # `iterate()` # `scroll()` ###################### ######### `Session` ################

`flush()` ################################# ### ########### `evict()` #################

```
ScrollableResult cats = sess.createQuery("from Cat as
 cat").scroll(); //a huge result set
while ( cats.next() ) {
    Cat cat = (Cat) cats.get(0);
```

```
    doSomethingWithACat(cat);
    sess.evict(cat);
}
```

Session ## contains() ############

####################################Session.clear() ############

############# SessionFactory ## ### ###########################

```
sessionFactory.evict(Cat.class, catId); //evict a particular Cat
sessionFactory.evict(Cat.class);  //evict all Cats
sessionFactory.evictCollection("Cat.kittens", catId); //evict a
 particular collection of kittens
sessionFactory.evictCollection("Cat.kittens"); //evict all kitten
 collections
```

CacheMode ################################## ######

- CacheMode.NORMAL - #############################
- CacheMode.GET - ########################## ##############################
- CacheMode.PUT - ############################# ###############
- CacheMode.REFRESH - ############################# ############# hibernate.cache.use_minimal_puts ###

############################## Statistics API# #########

```
Map cacheEntries = sessionFactory.getStatistics()
        .getSecondLevelCacheStatistics(regionName)
        .getEntries();
```

#################################### ###################Hibernate#######

```
hibernate.generate_statistics true
hibernate.cache.use_structured_entries true
```

19.4.

###

```
hibernate.cache.use_query_cache true
```

###
org.hibernate.cache.StandardQueryCache
##1##########################
org.hibernate.cache.UpdateTimestampsCache ########
######################################
####################################
###################################

##
############### Query.setCacheable(true) ###########
########

############################# Query.setCacheRegion()
#####################################

```
List blogs = sess.createQuery("from Blog blog where blog.blogger =
 :blogger")
    .setEntity("blogger", blogger)
    .setMaxResults(15)
    .setCacheable(true)
    .setCacheRegion("frontpages")
    .list();
```

##
Query.setCacheMode(CacheMode.REFRESH) #############
##############################Hibernate############
###############
SessionFactory.evictQueries() ########################

19.5.

################################
#############################2#3#######

19.5.1.

Hibernate#3#########################

- ########
- #####
- #####

###
##

##########Hibernate##########################
##################################

- ###############
- set
- bag

###################map#list##### <key>
<index>
######################################Hibernate#
##

Sets have a primary key consisting of <key> and element columns. This may be less efficient for some types of collection element, particularly composite elements or large text or binary fields; the database may not be able to index a complex primary key as efficiently. On the other hand, for one to many or many to many associations, particularly in the case of synthetic identifiers, it is likely to be just as efficient. (Side-note: if you want SchemaExport to actually create the primary key of a <set> for you, you must declare all columns as not-null="true".)

<idbag> ###################### ################################

bag##########bag################################
###################Hibernate######################
Hibernate################################ #### DELETE
######################### ###################

1###############################
##
#Hibernate####################################

19.5.2. ################## list#map#idbag#set

##############################set##########
############################

##
set################### Set ###
#######Hibernate################### UPDATE ##### Set
##############INSERT # DELETE ############################
#########

##############################list#map#idbag#####
###########inverse####################set#

##############Hibernate#############set#############
################set#########################

##############Hibernate############################
inverse="true" #####1#################
###

19.5.3. inverse##############bag#list

bag############bag####list###set#########################
inverse="true" #############1##############
bag####################bag#list############ ### Collection.add()
Collection.addAll() #bag# List ####true################ # Set
########## ###########################

```
Parent p = (Parent) sess.load(Parent.class, id);
Child c = new Child();
c.setParent(p);
p.getChildren().add(c);  //no need to fetch the collection!
sess.flush();
```

19.5.4.

###
Hibernate########################### list.clear()
###Hibernate# DELETE
#####################

###20## Hibernate#### INSERT
DELETE ####### ########bag########################

####18##########2##########3################# ################

- 18############3######
- ############# DELETE #SQL##########5######## ###########

Hibernate######2##############################
####Hibernate##################################
#############################

##
##########################2#################### #################

########### inverse="true" ####################

19.6.

######################################
Hibernate###################### Hibernate###### `SessionFactory` ##########

19.6.1. SessionFactory#######

`SessionFactory` ###############2########## ####### `sessionFactory.getStatistics()` ###### ### `Statistics` ###############

`StatisticsService` MBean##########Hibernate#JMX#### ##################1##MBean##### `SessionFactory` ###########SessionFactory######MBean############## #########################

```
// MBean service registration for a specific SessionFactory
Hashtable tb = new Hashtable();
tb.put("type", "statistics");
tb.put("sessionFactory", "myFinancialApp");
ObjectName on = new ObjectName("hibernate", tb); // MBean object
 name

StatisticsService stats = new StatisticsService(); // MBean
 implementation
stats.setSessionFactory(sessionFactory); // Bind the stats to a
 SessionFactory
server.registerMBean(stats, on); // Register the Mbean on the server
```

```
// MBean service registration for all SessionFactory's
Hashtable tb = new Hashtable();
tb.put("type", "statistics");
tb.put("sessionFactory", "all");
ObjectName on = new ObjectName("hibernate", tb); // MBean object
 name

StatisticsService stats = new StatisticsService(); // MBean
 implementation
server.registerMBean(stats, on); // Register the MBean on the server
```

TODO: ###################MBean############2######## ####session factory######JNDI############## `hibernateStatsBean.setSessionFactoryJNDIName("my/JNDI/Name")` #########

`SessionFactory` #############################

- ###### `hibernate.generate_statistics` # `false` ####
- ##### `sf.getStatistics().setStatisticsEnabled(true)` ### `hibernateStatsBean.setStatisticsEnabled(true)` #######

Statistics can be reset programmatically using the `clear()` method. A summary can be sent to a logger (info level) using the `logSummary()` method.

19.6.2.

########################## `Statistics` #########API########3##########

- ########## `Session` ############# ##############JDBC####################
- ############################### #########################
- ##

For example, you can check the cache hit, miss, and put ratio of entities, collections and queries, and the average time a query needs. Beware that the number of milliseconds is subject to approximation in Java. Hibernate is tied to the JVM precision, on some platforms this might even only be accurate to 10 seconds.

###getter### ### ############HQL#SQL############################### `Statistics` # `EntityStatistics` # `CollectionStatistics` # `SecondLevelCacheStatistics` # `QueryStatistics` API#javadoc# ########################

```
Statistics stats = HibernateUtil.sessionFactory.getStatistics();

double queryCacheHitCount  = stats.getQueryCacheHitCount();
double queryCacheMissCount = stats.getQueryCacheMissCount();
double queryCacheHitRatio =
  queryCacheHitCount / (queryCacheHitCount + queryCacheMissCount);

log.info("Query Hit ratio:" + queryCacheHitRatio);

EntityStatistics entityStats =
  stats.getEntityStatistics( Cat.class.getName() );
long changes =
        entityStats.getInsertCount()
        + entityStats.getUpdateCount()
        + entityStats.getDeleteCount();
log.info(Cat.class.getName() + " changed " + changes + "times"  );
```

`getQueries()` # `getEntityNames()`# `getCollectionRoleNames()` # `getSecondLevelCacheRegionNames()` #####

#20# Toolset Guide

Hibernate######################
Eclipse#####################Ant##############

Hibernate Tools ##########################Ant########EclipseIDE############

- #########*:* Hibernate#XML#################
  #################################/#####################
  ###XML############
- *Console:* #####################
  ##
  #######HQL###########################
- ####### Hibernate#Eclipse#####################
  #########Hibernate#######(cfg.xml)###########
  ##############POJO#########Hibernate#############
  #########################
  ###
- *Ant Tasks:*

######## *Hibernate Tools* ########################

####Hibernate########## *SchemaExport* # ## `hbm2ddl` #####(Hibernate###############)#

20.1.

DDL#Hibernate##################################
##(########)######
#######################################

DDL#######################`hibernate.dialect` ######SQL# ## ### ########## #

###

20.1.1.

###Hibernate################# `length` ################
#####################(###NUMERIC/DECIMAL###############)#

```
<property name="zip" length="5"/>
```

```
<property name="balance" precision="12" scale="2"/>
```

not-null ############ NOT NULL ######### unique ############ UNIQUE ######################

```
<many-to-one name="bar" column="barId" not-null="true"/>
```

```
<element column="serialNumber" type="long" not-null="true"
 unique="true"/>
```

unique-key ################################ ###unique-key ################ #### # ################################

```
<many-to-one name="org" column="orgId" unique-key="OrgEmployeeId"/>
<property name="employeeId" unique-key="OrgEmployee"/>
```

index #####################################
##

```
<property name="lastName" index="CustName"/>
<property name="firstName" index="CustName"/>
```

foreign-key ######################################

```
<many-to-one name="bar" column="barId" foreign-key="FKFooBar"/>
```

############# <column> ###################################

```
<property name="name" type="my.customtypes.Name"/>
    <column name="last" not-null="true" index="bar_idx"
 length="30"/>
    <column name="first" not-null="true" index="bar_idx"
 length="20"/>
    <column name="initial"/>
</property>
```

default ################### (############################
########################)#

```
<property name="credits" type="integer" insert="false">
    <column name="credits" default="10"/>
</property>
```

```
<version name="version" type="integer" insert="false">
    <column name="version" default="0"/>
</property>
```

sql-type ##########Hibernate###SQL##########################

```
<property name="balance" type="float">
    <column name="balance" sql-type="decimal(13,3)"/>
</property>
```

check ######################

```
<property name="foo" type="integer">
    <column name="foo" check="foo > 10"/>
</property>
```

```
<class name="Foo" table="foos" check="bar < 100.0">
    ...
    <property name="bar" type="float"/>
</class>
```

20.1.

##	#	##
length	##	######
precision	##	####DECIMAL#####precision#
scale	##	####DECIMAL#######scale#
not-null	true|false	####null###############
unique	true|false	######################
index	#######	(######)################
unique-key	#######	######################
foreign-key	#####	specifies the name of the foreign key constraint generated for an association, for a <one-to-one>, <many-to-one>, <key>, or <many-to-many> mapping element. Note that inverse="true" sides will not be considered by SchemaExport.
sql-type	SQL#####	overrides the default column type (attribute of <column> element only)
default	SQL#	#################
check	SQL#	#########SQL#############

<comment> ##############################

```
<class name="Customer" table="CurCust">
    <comment>Current customers only</comment>
    ...
```

```
</class>
```

```
<property name="balance">
    <column name="bal">
        <comment>Balance in USD</comment>
    </column>
</property>
```

##########DDL# `comment on table` # `comment on column` ########

20.1.2.

`SchemaExport` #########DDL###########DDL############

`java -cp` *hibernate_classpaths* `org.hibernate.tool.hbm2ddl.SchemaExport` *options mapping_files*

20.2. `SchemaExport` Command Line Options `SchemaExport`

#####	##
`--quiet`	don't output the script to stdout
`--drop`	###############
`--create`	################
`--text`	##################
`--output=my_schema.ddl`	DDL#################
`--naming=eg.MyNamingStrategy`	select a `NamingStrategy`
`--config=hibernate.cfg.xml`	XML######Hibernate############
`--properties=hibernate.properties`	read database properties from a file
`--format`	###########SQL###################
`--delimiter=x`	###################

######### `SchemaExport` #############

```
Configuration cfg = ....;
new SchemaExport(cfg).create(false, true);
```

20.1.3.

#################################

- `-D`<*property*> ##################

- `hibernate.properties` ######
- `--properties` ###################

#################

20.3. SchemaExport###########

######	##
`hibernate.connection.driver_class`	jdbc#########
`hibernate.connection.url`	jdbc#url
`hibernate.connection.username`	##########
`hibernate.connection.password`	########
`hibernate.dialect`	########

20.1.4. Ant#####

Ant########### `SchemaExport` #############:

```
<target name="schemaexport">
    <taskdef name="schemaexport"
        classname="org.hibernate.tool.hbm2ddl.SchemaExportTask"
        classpathref="class.path"/>

    <schemaexport
        properties="hibernate.properties"
        quiet="no"
        text="no"
        drop="no"
        delimiter=";"
        output="schema-export.sql">
        <fileset dir="src">
            <include name="**/*.hbm.xml"/>
        </fileset>
    </schemaexport>
</target>
```

20.1.5.

`SchemaUpdate` ###########################
`SchemaUpdate` #JDBC######API#########
#########JDBC############################

`java -cp` *hibernate_classpaths* `org.hibernate.tool.hbm2ddl.SchemaUpdate` *options mapping_files*

20.4. `SchemaUpdate`

#####	##
`--quiet`	don't output the script to stdout
`--text`	########################
`--naming=eg.MyNamingStrategy`	select a `NamingStrategy`
`--properties=hibernate.properties`	read database properties from a file
`--config=hibernate.cfg.xml`	specify a `.cfg.xml` file

######### `SchemaUpdate` ##############

```
Configuration cfg = ....;
new SchemaUpdate(cfg).execute(false);
```

20.1.6. ####################Ant###

Ant####### `SchemaUpdate` ##############

```
<target name="schemaupdate">
    <taskdef name="schemaupdate"
        classname="org.hibernate.tool.hbm2ddl.SchemaUpdateTask"
        classpathref="class.path"/>

    <schemaupdate
        properties="hibernate.properties"
        quiet="no">
        <fileset dir="src">
            <include name="**/*.hbm.xml"/>
        </fileset>
    </schemaupdate>
</target>
```

20.1.7. Schema validation

`SchemaValidator` ####################################"####"#########
`SchemaValidator` #JDBC######API############################JDBC##############################

`java -cp` *hibernate_classpaths* `org.hibernate.tool.hbm2ddl.SchemaValidator` *options mapping_files*

20.5. SchemaValidator

#####	##
--naming=eg.MyNamingStrategy	select a NamingStrategy
--properties=hibernate.properties	read database properties from a file
--config=hibernate.cfg.xml	specify a .cfg.xml file

SchemaValidator ######################:

```
Configuration cfg = ....;
new SchemaValidator(cfg).validate();
```

20.1.8. ##############Ant######

Ant####### SchemaValidator #######:

```
<target name="schemavalidate">
    <taskdef name="schemavalidator"
        classname="org.hibernate.tool.hbm2ddl.SchemaValidatorTask"
        classpathref="class.path"/>

    <schemavalidator
        properties="hibernate.properties">
        <fileset dir="src">
            <include name="**/*.hbm.xml"/>
        </fileset>
    </schemavalidator>
</target>
```

#21# ###/##

######Hibernate####################################
#######################################
############################# ## ## ## <one-to-many>
#############################
########### ## # <composite-element> ##############
####Hibernate##
########################## #################################
################### #################################

21.1.

Hibernate################################
###################################### ##################

- #####################################
  ##################################
- ################################
  ################)###########################
  ########################
  ###
- #################################)##########
  ####################################
  ##
  ###

##
###
###########/##

21.2.

`Parent` ## `Child` ##### `<one-to-many>` ############

```
<set name="children">
    <key column="parent_id"/>
    <one-to-many class="Child"/>
</set>
```

#############

```
Parent p = .....;
Child c = new Child();
```

```
p.getChildren().add(c);
session.save(c);
session.flush();
```

Hibernate####SQL#######:

- c##############INSERT
- p##c##########UPDATE

###############parent_id ####### NOT NULL ######### ############# not-null="true" #########null################:

```
<set name="children">
    <key column="parent_id" not-null="true"/>
    <one-to-many class="Child"/>
</set>
```

#################

#############p ## c ##### ##### parent_id)# Child #################### #### INSERT ################### #############Child################

```
<many-to-one name="parent" column="parent_id" not-null="true"/>
```

(## Child #### parent ###################)

Child ############################ ############################## inverse

```
<set name="children" inverse="true">
    <key column="parent_id"/>
    <one-to-many class="Child"/>
</set>
```

############### Child ##############

```
Parent p = (Parent) session.load(Parent.class, pid);
Child c = new Child();
c.setParent(p);
p.getChildren().add(c);
session.save(c);
session.flush();
```

######SQL# INSERT #####################

###########Parent # addChild() ###########

```
public void addChild(Child c) {
```

```
    c.setParent(this);
    children.add(c);
}
```

Child ####################

```
Parent p = (Parent) session.load(Parent.class, pid);
Child c = new Child();
p.addChild(c);
session.save(c);
session.flush();
```

21.3.

save()

```
<set name="children" inverse="true" cascade="all">
    <key column="parent_id"/>
    <one-to-many class="Child"/>
</set>
```

##########################

```
Parent p = (Parent) session.load(Parent.class, pid);
Child c = new Child();
p.addChild(c);
session.flush();
```

Parent ### ####### p

```
Parent p = (Parent) session.load(Parent.class, pid);
session.delete(p);
session.flush();
```

##########

```
Parent p = (Parent) session.load(Parent.class, pid);
Child c = (Child) p.getChildren().iterator().next();
p.getChildren().remove(c);
c.setParent(null);
session.flush();
```

######## c ########p ########## ############ NOT NULL ################# Child # delete() ##############

```
Parent p = (Parent) session.load(Parent.class, pid);
Child c = (Child) p.getChildren().iterator().next();
p.getChildren().remove(c);
session.delete(c);
```

```
session.flush();
```

########### Child ###################### ################ Child #################### ####### cascade="all-delete-orphan" ##############

```
<set name="children" inverse="true" cascade="all-delete-orphan">
    <key column="parent_id"/>
    <one-to-many class="Child"/>
</set>
```

################# inverse="true" ####### ## ### ########################### setParent() ##############

21.4. ###### unsaved-value

Parent #### Session #######UI################ update() ############################### Parent #################################### Hibernate## Parent # Child #### java.lang.Long ###################### Hibernate################## ??? ##)#Hibernate3###### #### unsaved-value #################

####### parent # child #####newChild #######

```
//parent and child were both loaded in a previous session
parent.addChild(child);
Child newChild = new Child();
parent.addChild(newChild);
session.update(parent);
session.flush();
```

###Hibernate## ###Session################################# #####Hibernate##

21.5.

############################Hiberna

############################### <composite-element> ############### ################################## ###################################

1####################################
####################################

#22# #: Weblog########

22.1.

################################### ########################## set#########bag###########

```
package eg;

import java.util.List;

public class Blog {
    private Long _id;
    private String _name;
    private List _items;

    public Long getId() {
        return _id;
    }
    public List getItems() {
        return _items;
    }
    public String getName() {
        return _name;
    }
    public void setId(Long long1) {
        _id = long1;
    }
    public void setItems(List list) {
        _items = list;
    }
    public void setName(String string) {
        _name = string;
    }
}
```

```
package eg;

import java.text.DateFormat;
import java.util.Calendar;

public class BlogItem {
    private Long _id;
    private Calendar _datetime;
    private String _text;
    private String _title;
    private Blog _blog;

    public Blog getBlog() {
        return _blog;
    }
```

```
    public Calendar getDatetime() {
        return _datetime;
    }
    public Long getId() {
        return _id;
    }
    public String getText() {
        return _text;
    }
    public String getTitle() {
        return _title;
    }
    public void setBlog(Blog blog) {
        _blog = blog;
    }
    public void setDatetime(Calendar calendar) {
        _datetime = calendar;
    }
    public void setId(Long long1) {
        _id = long1;
    }
    public void setText(String string) {
        _text = string;
    }
    public void setTitle(String string) {
        _title = string;
    }
}
```

22.2. Hibernate######

XML#####################

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="eg">

    <class
        name="Blog"
        table="BLOGS">

        <id
            name="id"
            column="BLOG_ID">

            <generator class="native"/>

        </id>

        <property
```

```
            name="name"
            column="NAME"
            not-null="true"
            unique="true"/>

        <bag
            name="items"
            inverse="true"
            order-by="DATE_TIME"
            cascade="all">

            <key column="BLOG_ID"/>
            <one-to-many class="BlogItem"/>

        </bag>

    </class>

</hibernate-mapping>
```

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="eg">

    <class
        name="BlogItem"
        table="BLOG_ITEMS"
        dynamic-update="true">

        <id
            name="id"
            column="BLOG_ITEM_ID">

            <generator class="native"/>

        </id>

        <property
            name="title"
            column="TITLE"
            not-null="true"/>

        <property
            name="text"
            column="TEXT"
            not-null="true"/>

        <property
            name="datetime"
            column="DATE_TIME"
            not-null="true"/>
```

```
        <many-to-one
            name="blog"
            column="BLOG_ID"
            not-null="true"/>

    </class>

</hibernate-mapping>
```

22.3. Hibernate####

######## Hibernate##############################

```
package eg;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Iterator;
import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.tool.hbm2ddl.SchemaExport;

public class BlogMain {

    private SessionFactory _sessions;

    public void configure() throws HibernateException {
        _sessions = new Configuration()
            .addClass(Blog.class)
            .addClass(BlogItem.class)
            .buildSessionFactory();
    }

    public void exportTables() throws HibernateException {
        Configuration cfg = new Configuration()
            .addClass(Blog.class)
            .addClass(BlogItem.class);
        new SchemaExport(cfg).create(true, true);
    }

    public Blog createBlog(String name) throws HibernateException {

        Blog blog = new Blog();
        blog.setName(name);
        blog.setItems( new ArrayList() );
```

```
        Session session = _sessions.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            session.persist(blog);
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
        return blog;
    }

    public BlogItem createBlogItem(Blog blog, String title, String
text)
                        throws HibernateException {

        BlogItem item = new BlogItem();
        item.setTitle(title);
        item.setText(text);
        item.setBlog(blog);
        item.setDatetime( Calendar.getInstance() );
        blog.getItems().add(item);

        Session session = _sessions.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            session.update(blog);
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
        return item;
    }

    public BlogItem createBlogItem(Long blogid, String title, String
text)
                        throws HibernateException {

        BlogItem item = new BlogItem();
        item.setTitle(title);
        item.setText(text);
        item.setDatetime( Calendar.getInstance() );
```

```
        Session session = _sessions.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            Blog blog = (Blog) session.load(Blog.class, blogid);
            item.setBlog(blog);
            blog.getItems().add(item);
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
        return item;
    }

    public void updateBlogItem(BlogItem item, String text)
                    throws HibernateException {

        item.setText(text);

        Session session = _sessions.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            session.update(item);
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
    }

    public void updateBlogItem(Long itemid, String text)
                    throws HibernateException {

        Session session = _sessions.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            BlogItem item = (BlogItem) session.load(BlogItem.class,
itemid);
            item.setText(text);
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
```

```
            throw he;
        }
        finally {
            session.close();
        }
    }

    public List listAllBlogNamesAndItemCounts(int max)
                    throws HibernateException {

        Session session = _sessions.openSession();
        Transaction tx = null;
        List result = null;
        try {
            tx = session.beginTransaction();
            Query q = session.createQuery(
                "select blog.id, blog.name, count(blogItem) " +
                "from Blog as blog " +
                "left outer join blog.items as blogItem " +
                "group by blog.name, blog.id " +
                "order by max(blogItem.datetime)"
            );
            q.setMaxResults(max);
            result = q.list();
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
        return result;
    }

    public Blog getBlogAndAllItems(Long blogid)
                    throws HibernateException {

        Session session = _sessions.openSession();
        Transaction tx = null;
        Blog blog = null;
        try {
            tx = session.beginTransaction();
            Query q = session.createQuery(
                "from Blog as blog " +
                "left outer join fetch blog.items " +
                "where blog.id = :blogid"
            );
            q.setParameter("blogid", blogid);
            blog  = (Blog) q.uniqueResult();
            tx.commit();
        }
        catch (HibernateException he) {
```

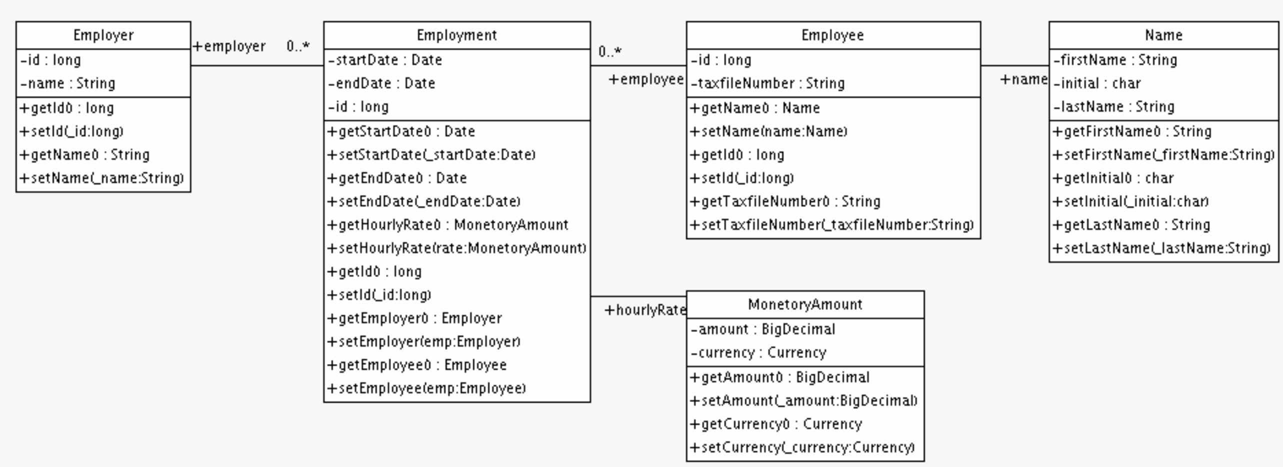
```
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
        return blog;
    }

    public List listBlogsAndRecentItems() throws HibernateException
 {

        Session session = _sessions.openSession();
        Transaction tx = null;
        List result = null;
        try {
            tx = session.beginTransaction();
            Query q = session.createQuery(
                "from Blog as blog " +
                "inner join blog.items as blogItem " +
                "where blogItem.datetime > :minDate"
            );

            Calendar cal = Calendar.getInstance();
            cal.roll(Calendar.MONTH, false);
            q.setCalendar("minDate", cal);

            result = q.list();
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
        return result;
    }
}
```

#23#

################################

23.1. ###/###

Employer # Employee ############### ####################
Employment # ###### #######2##################################
######################################



##################

```
<hibernate-mapping>

    <class name="Employer" table="employers">
        <id name="id">
            <generator class="sequence">
                <param name="sequence">employer_id_seq</param>
            </generator>
        </id>
        <property name="name"/>
    </class>

    <class name="Employment" table="employment_periods">

        <id name="id">
            <generator class="sequence">
                <param name="sequence">employment_id_seq</param>
            </generator>
        </id>
        <property name="startDate" column="start_date"/>
        <property name="endDate" column="end_date"/>

        <component name="hourlyRate" class="MonetaryAmount">
            <property name="amount">
                <column name="hourly_rate" sql-type="NUMERIC(12,
 2)"/>
```

```
            </property>
            <property name="currency" length="12"/>
        </component>

        <many-to-one name="employer" column="employer_id"
 not-null="true"/>
        <many-to-one name="employee" column="employee_id"
 not-null="true"/>

    </class>

    <class name="Employee" table="employees">
        <id name="id">
            <generator class="sequence">
                <param name="sequence">employee_id_seq</param>
            </generator>
        </id>
        <property name="taxfileNumber"/>
        <component name="name" class="Name">
            <property name="firstName"/>
            <property name="initial"/>
            <property name="lastName"/>
        </component>
    </class>

</hibernate-mapping>
```

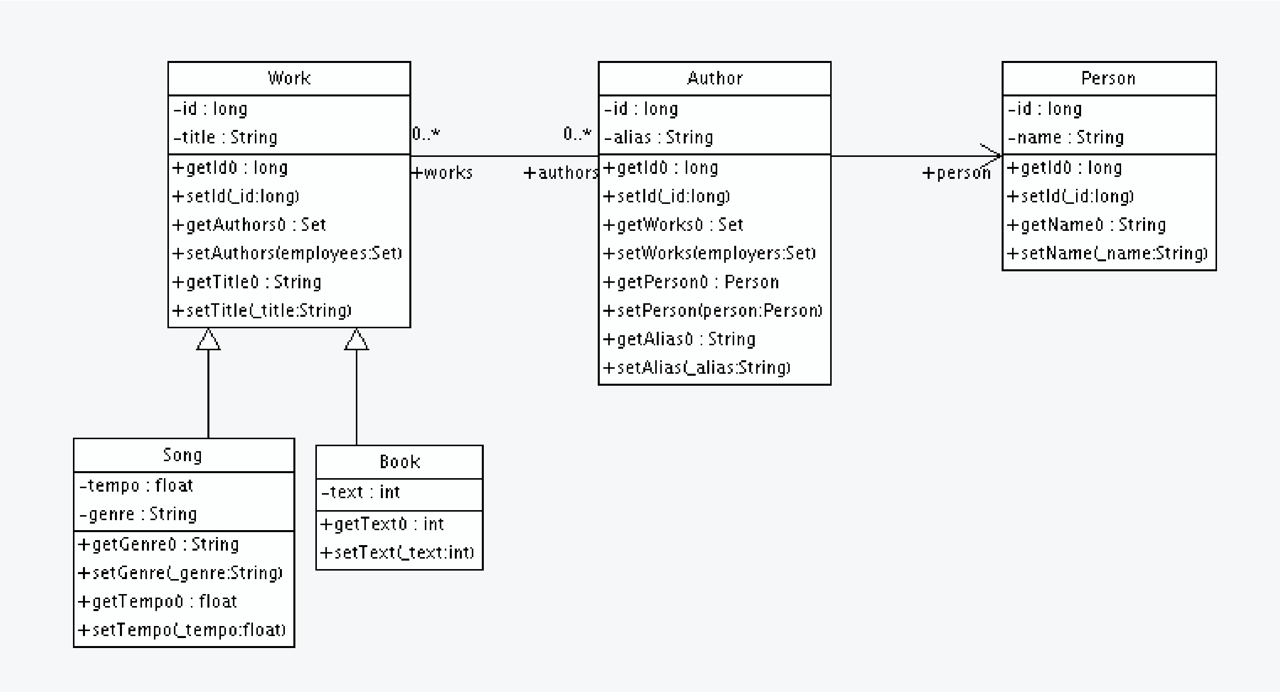
`SchemaExport` #################

```
create table employers (
    id BIGINT not null,
    name VARCHAR(255),
    primary key (id)
)

create table employment_periods (
    id BIGINT not null,
    hourly_rate NUMERIC(12, 2),
    currency VARCHAR(12),
    employee_id BIGINT not null,
    employer_id BIGINT not null,
    end_date TIMESTAMP,
    start_date TIMESTAMP,
    primary key (id)
)

create table employees (
    id BIGINT not null,
    firstName VARCHAR(255),
    initial CHAR(1),
    lastName VARCHAR(255),
    taxfileNumber VARCHAR(255),
    primary key (id)
)
```

```
alter table employment_periods
    add constraint employment_periodsFK0 foreign key (employer_id)
 references employers
alter table employment_periods
    add constraint employment_periodsFK1 foreign key (employee_id)
 references employees
create sequence employee_id_seq
create sequence employment_id_seq
create sequence employer_id_seq
```

23.2. ##/##

`Work` , `Author` ### `Person` ######################## `Work` # `Author` ################## `Author` # `Person` #################### ### `Author` # `Person` #################



####################################

```
<hibernate-mapping>

    <class name="Work" table="works" discriminator-value="W">

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

        <property name="title"/>
        <set name="authors" table="author_work">
            <key column name="work_id"/>
```

```
            <many-to-many class="Author" column name="author_id"/>
        </set>

        <subclass name="Book" discriminator-value="B">
            <property name="text"/>
        </subclass>

        <subclass name="Song" discriminator-value="S">
            <property name="tempo"/>
            <property name="genre"/>
        </subclass>

    </class>

    <class name="Author" table="authors">

        <id name="id" column="id">
            <!-- The Author must have the same identifier as the
 Person -->
            <generator class="assigned"/>
        </id>

        <property name="alias"/>
        <one-to-one name="person" constrained="true"/>

        <set name="works" table="author_work" inverse="true">
            <key column="author_id"/>
            <many-to-many class="Work" column="work_id"/>
        </set>

    </class>

    <class name="Person" table="persons">
        <id name="id" column="id">
            <generator class="native"/>
        </id>
        <property name="name"/>
    </class>

</hibernate-mapping>
```

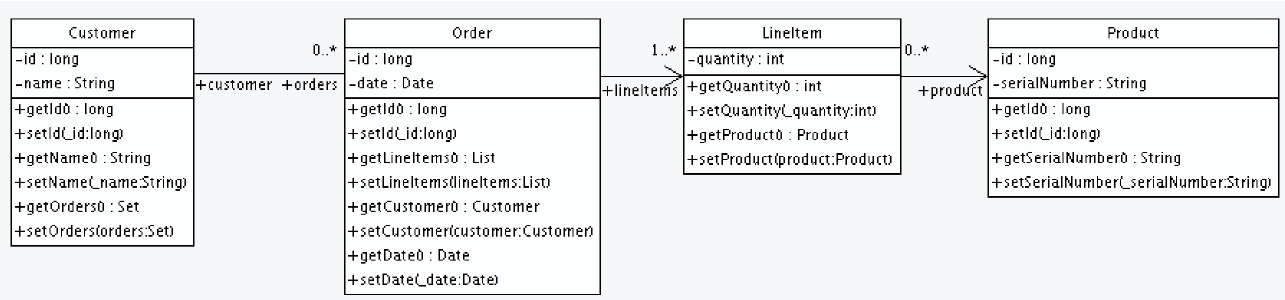
#########4############ `works` , `authors` , `persons` ######################## `author_work` ##################### ### `SchemaExport` ################

```
create table works (
    id BIGINT not null generated by default as identity,
    tempo FLOAT,
    genre VARCHAR(255),
    text INTEGER,
    title VARCHAR(255),
    type CHAR(1) not null,
    primary key (id)
```

```
)

create table author_work (
    author_id BIGINT not null,
    work_id BIGINT not null,
    primary key (work_id, author_id)
)

create table authors (
    id BIGINT not null generated by default as identity,
    alias VARCHAR(255),
    primary key (id)
)

create table persons (
    id BIGINT not null generated by default as identity,
    name VARCHAR(255),
    primary key (id)
)

alter table authors
    add constraint authorsFK0 foreign key (id) references persons
alter table author_work
    add constraint author_workFK0 foreign key (author_id) references
 authors
alter table author_work
    add constraint author_workFK1 foreign key (work_id) references
 works
```

23.3. ##/##/##

`Customer` , `Order` , `LineItemProduct` #################### `Customer` # `Order` ########### `Order` / `LineItem` / `Product` ################### `LineItem` ##`Order` # `Product` ################################ Hibernate###################



############

```
<hibernate-mapping>

    <class name="Customer" table="customers">
        <id name="id">
```

```
            <generator class="native"/>
        </id>
        <property name="name"/>
        <set name="orders" inverse="true">
            <key column="customer_id"/>
            <one-to-many class="Order"/>
        </set>
    </class>

    <class name="Order" table="orders">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="date"/>
        <many-to-one name="customer" column="customer_id"/>
        <list name="lineItems" table="line_items">
            <key column="order_id"/>
            <list-index column="line_number"/>
            <composite-element class="LineItem">
                <property name="quantity"/>
                <many-to-one name="product" column="product_id"/>
            </composite-element>
        </list>
    </class>

    <class name="Product" table="products">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="serialNumber"/>
    </class>

</hibernate-mapping>
```

customers , orders , line_items , products ################################ line_items ############################

```
create table customers (
    id BIGINT not null generated by default as identity,
    name VARCHAR(255),
    primary key (id)
)

create table orders (
    id BIGINT not null generated by default as identity,
    customer_id BIGINT,
    date TIMESTAMP,
    primary key (id)
)

create table line_items (
    line_number INTEGER not null,
```

```
    order_id BIGINT not null,
    product_id BIGINT,
    quantity INTEGER,
    primary key (order_id, line_number)
)

create table products (
    id BIGINT not null generated by default as identity,
    serialNumber VARCHAR(255),
    primary key (id)
)

alter table orders
    add constraint ordersFK0 foreign key (customer_id) references
 customers
alter table line_items
    add constraint line_itemsFK0 foreign key (product_id) references
 products
alter table line_items
    add constraint line_itemsFK1 foreign key (order_id) references
 orders
```

23.4.

##########Hibernate################# ############################ Hibernate############ `test` #############

TODO: #########

23.4.1.

```
<class name="Person">
    <id name="name"/>
    <one-to-one name="address"
            cascade="all">
        <formula>name</formula>
        <formula>'HOME'</formula>
    </one-to-one>
    <one-to-one name="mailingAddress"
            cascade="all">
        <formula>name</formula>
        <formula>'MAILING'</formula>
    </one-to-one>
</class>

<class name="Address" batch-size="2"
        check="addressType in ('MAILING', 'HOME', 'BUSINESS')">
    <composite-id>
        <key-many-to-one name="person"
                column="personName"/>
        <key-property name="type"
                column="addressType"/>
```

```
    </composite-id>
    <property name="street" type="text"/>
    <property name="state"/>
    <property name="zip"/>
</class>
```

23.4.2.

```
<class name="Customer">

    <id name="customerId"
        length="10">
        <generator class="assigned"/>
    </id>

    <property name="name" not-null="true" length="100"/>
    <property name="address" not-null="true" length="200"/>

    <list name="orders"
            inverse="true"
            cascade="save-update">
        <key column="customerId"/>
        <index column="orderNumber"/>
        <one-to-many class="Order"/>
    </list>

</class>

<class name="Order" table="CustomerOrder" lazy="true">
    <synchronize table="LineItem"/>
    <synchronize table="Product"/>

    <composite-id name="id"
            class="Order$Id">
        <key-property name="customerId" length="10"/>
        <key-property name="orderNumber"/>
    </composite-id>

    <property name="orderDate"
            type="calendar_date"
            not-null="true"/>

    <property name="total">
        <formula>
            ( select sum(li.quantity*p.price)
            from LineItem li, Product p
            where li.productId = p.productId
                and li.customerId = customerId
                and li.orderNumber = orderNumber )
        </formula>
    </property>

    <many-to-one name="customer"
```

```
            column="customerId"
            insert="false"
            update="false"
            not-null="true"/>

    <bag name="lineItems"
            fetch="join"
            inverse="true"
            cascade="save-update">
        <key>
            <column name="customerId"/>
            <column name="orderNumber"/>
        </key>
        <one-to-many class="LineItem"/>
    </bag>

</class>

<class name="LineItem">

    <composite-id name="id"
            class="LineItem$Id">
        <key-property name="customerId" length="10"/>
        <key-property name="orderNumber"/>
        <key-property name="productId" length="10"/>
    </composite-id>

    <property name="quantity"/>

    <many-to-one name="order"
            insert="false"
            update="false"
            not-null="true">
        <column name="customerId"/>
        <column name="orderNumber"/>
    </many-to-one>

    <many-to-one name="product"
            insert="false"
            update="false"
            not-null="true"
            column="productId"/>

</class>

<class name="Product">
    <synchronize table="LineItem"/>

    <id name="productId"
        length="10">
        <generator class="assigned"/>
    </id>

    <property name="description"
```

```
        not-null="true"
        length="200"/>
    <property name="price" length="3"/>
    <property name="numberAvailable"/>

    <property name="numberOrdered">
        <formula>
            ( select sum(li.quantity)
            from LineItem li
            where li.productId = productId )
        </formula>
    </property>

</class>
```

23.4.3.

```
<class name="User" table="`User`">
    <composite-id>
        <key-property name="name"/>
        <key-property name="org"/>
    </composite-id>
    <set name="groups" table="UserGroup">
        <key>
            <column name="userName"/>
            <column name="org"/>
        </key>
        <many-to-many class="Group">
            <column name="groupName"/>
            <formula>org</formula>
        </many-to-many>
    </set>
</class>

<class name="Group" table="`Group`">
    <composite-id>
        <key-property name="name"/>
        <key-property name="org"/>
    </composite-id>
    <property name="description"/>
    <set name="users" table="UserGroup" inverse="true">
        <key>
            <column name="groupName"/>
            <column name="org"/>
        </key>
        <many-to-many class="User">
            <column name="userName"/>
            <formula>org</formula>
        </many-to-many>
    </set>
</class>
```

23.4.4. discrimination######

```
<class name="Person"
    discriminator-value="P">

    <id name="id"
        column="person_id"
        unsaved-value="0">
        <generator class="native"/>
    </id>


    <discriminator
        type="character">
        <formula>
            case
                when title is not null then 'E'
                when salesperson is not null then 'C'
                else 'P'
            end
        </formula>
    </discriminator>

    <property name="name"
        not-null="true"
        length="80"/>

    <property name="sex"
        not-null="true"
        update="false"/>

    <component name="address">
        <property name="address"/>
        <property name="zip"/>
        <property name="country"/>
    </component>

    <subclass name="Employee"
        discriminator-value="E">
            <property name="title"
                length="20"/>
            <property name="salary"/>
            <many-to-one name="manager"/>
    </subclass>

    <subclass name="Customer"
        discriminator-value="C">
            <property name="comments"/>
            <many-to-one name="salesperson"/>
    </subclass>

</class>
```

23.4.5.

```
<class name="Person">

    <id name="id">
        <generator class="hilo"/>
    </id>

    <property name="name" length="100"/>

    <one-to-one name="address"
        property-ref="person"
        cascade="all"
        fetch="join"/>

    <set name="accounts"
        inverse="true">
        <key column="userId"
            property-ref="userId"/>
        <one-to-many class="Account"/>
    </set>

    <property name="userId" length="8"/>

</class>

<class name="Address">

    <id name="id">
        <generator class="hilo"/>
    </id>

    <property name="address" length="300"/>
    <property name="zip" length="5"/>
    <property name="country" length="25"/>
    <many-to-one name="person" unique="true" not-null="true"/>

</class>

<class name="Account">
    <id name="accountId" length="32">
        <generator class="uuid"/>
    </id>

    <many-to-one name="user"
        column="userId"
        property-ref="userId"/>

    <property name="type" not-null="true"/>

</class>
```

#24#

############ <component> ############

street####, suburb ####, state###, postcode ##############

Address############### #####################################

#########################

Hibernate#################### ##################

############################## ##################

#############

######################### <natural-id> ############

######################## equals() # hashCode() #########

##################################

############################## com.eg.Foo #####

com/eg/Foo.hbm.xml ################# #########################

######################

####################################

#####################

####ANSI#####SQL##########################

##

##############

JDBC####################"?"##########

######################################

############################

JDBC###################

Hibernate###########JDBC#######################

##################### ##############################

org.hibernate.connection.ConnectionProvider #################

################

##############Java####################

###

######## org.hibernate.UserType ############

#####Hibernate################################

#############JDBC#############

######################### #######JDBC################

################### ###### ############

##JDBC##

JDBC############Hibernate# Session

######## JDBC####################
#####################################

`Session` ################

Session###############################
#####################################
##
###################

3##############################

/ ###################### ####### /
JSP####################################
##############Session######### ## `Session.merge()` #
`Session.saveOrUpdate()` #############################

2################################

################# ##################################
########## ################ ##################
############1#######unit of work#######
###################################/#################
##
########2#######################
###############################
###################################
###############JDBC############ ##################
################################# 1##Session#############
##########################

#######################

############################### ##########
`Transaction` ########## `Session` ############
######Hibernate##################################
##
`Session.load()` ############## ###### `Session.get()` #############

######################

################# ###############################
######################
`lazy="false"`
######################## ###################### #### `left join fetch` #########

########################### ###################*(open session in view)* ########### ########*(assembly phase)* ########

Hibernate# *Data Transfer Objects* (DTO)####################
####EJB#########DTO###########

1################################## 2#######################
########################DTO############################
Hibernate##1##############
###
##
##
###Hibernate############ ###################################

Hibernate##########################
##########Hibernate##################### *DAO*
 # Thread Local Session ############### `UserType`
#Hibernate######## ########JDBC####################
###################################
#####5#############################

########################
################################
#################################
############2##1################# ############1#####1####
####################################

################
######################## ##############
###################################