## HIBERNATE - Relational Persistence for Idiomatic Java

## 1

# Hibernate Reference Documentation

### 3.5.6-Final

# King Gavin [FAMILY Given], Bauer Christian [FAMILY Given], Andersen Max [FAMILY Given], Bernard Emmanuel [FAMILY Given], # Ebersole Steve [FAMILY Given]

and thanks to Cobb James [FAMILY Given] (Graphic Design) # Weaver Cheyenne [FAMILY Given] (Graphic Design)

### ..... xi
**1. Tutorial** ..... 1
1.1. ###1 - #### Hibernate ######## ..... 1
1.1.1. Setup ..... 1
1.1.2. ###### ..... 3
1.1.3. ######### ..... 4
1.1.4. Hibernate ### ..... 7
1.1.5. Maven ###### ..... 9
1.1.6. ########### ..... 9
1.1.7. ############## ..... 10
1.2. ###2 - ######## ..... 13
1.2.1. Person ######### ..... 13
1.2.2. ### Set ##### ..... 14
1.2.3. ######## ..... 15
1.2.4. ######## ..... 17
1.2.5. ##### ..... 18
1.2.6. ######### ..... 19
1.3. ###3 - EventManager Web ######## ..... 20
1.3.1. #### Servlet ### ..... 20
1.3.2. ##### ..... 21
1.3.3. ######## ..... 23
1.4. ## ..... 24
**2. ######** ..... 25
2.1. ## ..... 25
2.2. ######### ..... 27
2.3. JMX #### ..... 28
2.4. JCA #### ..... 28
2.5. ############## ..... 28
**3. ##** ..... 31
3.1. ######### ..... 31
3.2. SessionFactory ##### ..... 32
3.3. JDBC ###### ..... 32
3.4. ############ ..... 34
3.4.1. SQL ###Dialect# ..... 39
3.4.2. ######## ..... 40
3.4.3. ######### ..... 40
3.4.4. ################ ..... 41
3.4.5. ########## ..... 41
3.4.6. Hibernate ## ..... 41
3.5. #### ..... 41
3.6. NamingStrategy ### ..... 42
3.7. XML ###### ..... 42
3.8. J2EE ################ ..... 43
3.8.1. ############ ..... 44

3.8.2. SessionFactory # JNDI #### ..... 45
3.8.3. JTA ######################### ..... 45
3.8.4. JMX ####### ..... 46
**4. #####** ..... 49
4.1. ### POJO ## ..... 49
4.1.1. ################## ..... 50
4.1.2. ##################### ..... 50
4.1.3. final ############### ..... 51
4.1.4. ################################### ..... 51
4.2. ##### ..... 51
4.3. equals() # hashCode()### ..... 52
4.4. ##### ..... 53
4.5. Tuplizer ..... 55
4.6. EntityNameResolvers ..... 56
**5. #### O/R #####** ..... 59
5.1. ####### ..... 59
5.1.1. Doctype ..... 60
5.1.2. Hibernate-mapping ..... 61
5.1.3. Class ..... 62
5.1.4. id ..... 64
5.1.5. Enhanced identifier generators ..... 68
5.1.6. Identifier generator optimization ..... 69
5.1.7. composite-id ..... 70
5.1.8. discriminator ..... 71
5.1.9. version######## ..... 72
5.1.10. timestamp######## ..... 73
5.1.11. property ..... 74
5.1.12. many-to-one ..... 75
5.1.13. one-to-one ..... 77
5.1.14. natural-id ..... 79
5.1.15. Component and dynamic-component ..... 80
5.1.16. ##### ..... 81
5.1.17. subclass ..... 82
5.1.18. joined-subclass ..... 82
5.1.19. union-subclass ..... 84
5.1.20. join ..... 85
5.1.21. Key ..... 86
5.1.22. column # formula ## ..... 86
5.1.23. Import ..... 87
5.1.24. Any ..... 88
5.2. Hibernate ## ..... 89
5.2.1. ######## ..... 89
5.2.2. ##### ..... 89
5.2.3. ##### ..... 90

5.3. ################## ..... 92
5.4. ########### SQL ### ..... 92
5.5. ########## ..... 92
5.5.1. XDoclet ######### ..... 93
5.5.2. JDK 5.0 ########## ..... 95
5.6. ####### ..... 95
5.7. Column read and write expressions ..... 96
5.8. ################ ..... 96
**6. ############** ..... 99
6.1. ########## ..... 99
6.2. ############ ..... 100
6.2.1. ########### ..... 101
6.2.2. ######### ..... 102
6.2.3. ################ ..... 102
6.2.4. ############### ..... 103
6.2.5. ##### ..... 105
6.3. ############### ..... 106
6.3.1. ############ ..... 106
6.3.2. ##### ..... 107
6.3.3. ##################### ..... 109
6.3.4. 3### ..... 110
6.3.5. Using an <idbag> ..... 110
6.4. ######## ..... 111
**7. ####### ..... 115**
7.1. ######### ..... 115
7.2. ##### ..... 115
7.2.1. Many-to-one ..... 115
7.2.2. One-to-one ..... 115
7.2.3. One-to-many ..... 116
7.3. ################ ..... 117
7.3.1. One-to-many ..... 117
7.3.2. Many-to-one ..... 118
7.3.3. One-to-one ..... 118
7.3.4. Many-to-many ..... 119
7.4. ##### ..... 120
7.4.1. ###/### ..... 120
7.4.2. One-to-one ..... 121
7.5. ################ ..... 122
7.5.1. ###/### ..... 122
7.5.2. ### ..... 123
7.5.3. Many-to-many ..... 123
7.6. ############ ..... 124
**8. ############** ..... 127
8.1. ######## ..... 127

8.2. ################ ..... 129
8.3. Map ################ ..... 130
8.4. ################ ..... 130
8.5. ######### ..... 132
**9. #######** ..... 133
9.1. 3#### ..... 133
9.1.1. ##############table-per-class-hierarchy# ..... 133
9.1.2. ############ #table-per-subclass# ..... 134
9.1.3. discriminator #### table-per-subclass ..... 134
9.1.4. table-per-subclass # table-per-class-hierarchy ### ..... 135
9.1.5. ##############table-per-concrete-class# ..... 136
9.1.6. ################ table-per-concrete-class ..... 136
9.1.7. ############################## ..... 137
9.2. ## ..... 138
**10. #########** ..... 141
10.1. Hibernate ############## ..... 141
10.2. ############### ..... 141
10.3. ########## ..... 142
10.4. ### ..... 143
10.4.1. ###### ..... 144
10.4.2. ############## ..... 147
10.4.3. ########## ..... 148
10.4.4. ##### SQL #### ..... 148
10.5. ########### ..... 149
10.6. detached ######### ..... 149
10.7. ######## ..... 150
10.8. ########### ..... 151
10.9. ################################ ..... 152
10.10. ########### ..... 152
10.11. ####### ..... 153
10.12. ######## ..... 154
**11. Read-only entities** ..... 157
11.1. Making persistent entities read-only ..... 157
11.1.1. Entities of immutable classes ..... 158
11.1.2. Loading persistent entities as read-only ..... 158
11.1.3. Loading read-only entities from an HQL query/criteria ..... 159
11.1.4. Making a persistent entity read-only ..... 160
11.2. Read-only affect on property type ..... 161
11.2.1. Simple properties ..... 162
11.2.2. Unidirectional associations ..... 163
11.2.3. Bidirectional associations ..... 164
**12. Transactions and Concurrency** ..... 167
12.1. session ##### transaction #### ..... 167
12.1.1. #####Unit of work# ..... 167

12.1.2. #### ..... 168
12.1.3. ############## ..... 169
12.1.4. ###### ..... 170
12.2. ################# ..... 170
12.2.1. ########## ..... 171
12.2.2. JTA ##### ..... 172
12.2.3. ######## ..... 173
12.2.4. ################ ..... 174
12.3. ######### ..... 174
12.3.1. ##################### ..... 175
12.3.2. ################## ..... 175
12.3.3. ######################## ..... 176
12.3.4. ################## ..... 177
12.4. ###### ..... 177
12.5. ########### ..... 178
**13. ############** ..... 179
13.1. ####### ..... 179
13.2. ######## ..... 181
13.3. Hibernate ########### ..... 182
**14. #####** ..... 183
14.1. ##### ..... 183
14.2. ##### ..... 184
14.3. StatelessSession ######## ..... 184
14.4. DML ####### ..... 185
**15. HQL: Hibernate #####** ..... 189
15.1. ########## ..... 189
15.2. from # ..... 189
15.3. ##### ..... 190
15.4. ####### ..... 191
15.5. ########### ..... 191
15.6. Select # ..... 192
15.7. #### ..... 193
15.8. ################ ..... 194
15.9. where # ..... 194
15.10. Expressions # ..... 196
15.11. order by # ..... 200
15.12. group by # ..... 200
15.13. ###### ..... 201
15.14. HQL ## ..... 202
15.15. ### UPDATE # DELETE ..... 204
15.16. Tips & Tricks ..... 204
15.17. ####### ..... 206
15.18. ########### ..... 206
**16. Criteria ###** ..... 209

16.1. Criteria ######### ..... 209
16.2. ########### ..... 209
16.3. ##### ..... 210
16.4. ## ..... 210
16.5. ######### ..... 212
16.6. ##### ..... 212
16.7. ########### ..... 213
16.8. ############### ..... 214
16.9. ########### ..... 215
**17. ##### SQL** ..... 217
17.1. Using a SQLQuery ..... 217
17.1.1. ######## ..... 217
17.1.2. ########## ..... 218
17.1.3. ############ ..... 218
17.1.4. ########### ..... 219
17.1.5. ################## ..... 220
17.1.6. ##### ..... 221
17.1.7. ##### ..... 221
17.2. #### SQL ### ..... 221
17.2.1. ################## return-property ### ..... 223
17.2.2. ######################### ..... 224
17.3. ################ SQL ..... 225
17.4. ########### SQL ..... 226
**18. ###########** ..... 229
18.1. Hibernate ##### ..... 229
**19. XML #####** ..... 233
19.1. XML ####### ..... 233
19.1.1. XML ################## ..... 233
19.1.2. XML ############ ..... 233
19.2. XML ########### ..... 234
19.3. XML ###### ..... 236
**20. ##########** ..... 239
20.1. ###### ..... 239
20.1.1. ####### ..... 239
20.1.2. ############## ..... 240
20.1.3. ######### ..... 241
20.1.4. ################ ..... 243
20.1.5. ########## ..... 244
20.1.6. ############## ..... 244
20.1.7. Fetch profiles ..... 245
20.1.8. ############### ..... 246
20.2. #2######## ..... 247
20.2.1. ########### ..... 247
20.2.2. read only ## ..... 248

20.2.3. read/write ## ..... 248
20.2.4. ###### read/write ## ..... 249
20.2.5. transactional ## ..... 249
20.2.6. Cache-provider/concurrency-strategy compatibility ..... 249
20.3. ######## ..... 249
20.4. ######## ..... 250
20.4.1. Enabling query caching ..... 251
20.4.2. Query cache regions ..... 252
20.5. ################## ..... 252
20.5.1. ## ..... 252
20.5.2. ################## list#map#idbag#set ..... 253
20.5.3. inverse ############### bag # list ..... 253
20.5.4. #### ..... 253
20.6. ############### ..... 254
20.6.1. SessionFactory ####### ..... 254
20.6.2. ##### ..... 255
**21. #########** ..... 257
21.1. ######### ..... 257
21.1.1. ############ ..... 257
21.1.2. ###### ..... 260
21.1.3. ##### ..... 261
21.1.4. Ant ##### ..... 261
21.1.5. ################ ..... 261
21.1.6. ################### Ant ### ..... 262
21.1.7. Schema validation ..... 262
21.1.8. ############## Ant ###### ..... 263
**22. ## #/##** ..... 265
22.1. ############# ..... 265
22.2. ###### ..... 265
22.3. ############## ..... 267
22.4. ###### unsaved-value ..... 268
22.5. ## ..... 269
**23. #: Weblog ########** ..... 271
23.1. ##### ..... 271
23.2. Hibernate ###### ..... 272
23.3. Hibernate #### ..... 274
**24. ## ##########** ..... 279
24.1. ###/### ..... 279
24.2. ##/## ..... 281
24.3. ##/##/## ..... 283
24.4. ############ ..... 285
24.4.1. ############## ..... 285
24.4.2. ###### ..... 285
24.4.3. ############### ..... 287

24.4.4. discrimination ###### ..... 288
24.4.5. ######## ..... 289
**25. #########** ..... 291
**26. Database Portability Considerations** ..... 295
26.1. Portability Basics ..... 295
26.2. Dialect ..... 295
26.3. Dialect resolution ..... 295
26.4. Identifier generation ..... 296
26.5. Database functions ..... 297
26.6. Type mappings ..... 297
References ..... 299

## ###

Working with object-oriented software and a relational database can be cumbersome and time consuming in today's enterprise environments. Hibernate is an Object/Relational Mapping tool for Java environments. The term Object/Relational Mapping (ORM) refers to the technique of mapping a data representation from an object model to a relational data model with a SQL-based schema.

Hibernate # Java ######################(## Java ######## SQL #############)####### ###################### SQL # JDBC ##########################################

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

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

1. Read *1#Tutorial* for a tutorial with step-by-step instructions. The source code for the tutorial is included in the distribution in the `doc/reference/tutorial/` directory.
2. Read *2########* to understand the environments where Hibernate can be used.
3. Hibernate ############# `eg/` ############### ############################ ### #### JDBC ##### `lib/` ######################################## `etc/hibernate.properties` ####### ############################## `ant eg` (Ant ###)## ##### Windows ###### `build eg` #######
4. Use this reference documentation as your primary source of information. Consider reading [*JPwH*] if you need more help with application design, or if you prefer a step-by-step tutorial. Also visit *http://caveatemptor.hibernate.org* and download the example application from [*JPwH*].
5. ########### (FAQ) # Hibernate ##############
6. Links to third party demos, examples, and tutorials are maintained on the Hibernate website.
7. Hibernate ######## Community Area ######################### (Tomcat# JBoss AS# Struts# EJB ##)#####################

######### Hibernate ################################ ### ################### JIRA (########) ######### Hibernate, ############ ##################### ############## ##### #######################

Hibernate ############## ######## ########### JBoss Inc ######### (http://www.hibernate.org/SupportTraining/ ###)# Hibernate ############################ JBoss Enterprise Middleware System (JEMS) #######################

# Tutorial

Intended for new users, this chapter provides an step-by-step introduction to Hibernate, starting with a simple application using an in-memory database. The tutorial is based on an earlier tutorial 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 have a limited knowledge of JAVA or SQL, it is advised that you start with a good introduction to that technology prior to attempting to learn Hibernate.

##

The distribution contains another example application under the `tutorial/eg` project source directory.

## 1.1. ###1 - #### Hibernate #########

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

##

Although you can use whatever database you feel comfortable using, we will use *HSQLDB* [http://hsqldb.org/] (an in-memory, Java database) to avoid describing installation/setup of any particular database servers.

### 1.1.1. Setup

The first thing we need to do is to set up the development environment. We will be using the "standard layout" advocated by alot of build tools such as *Maven* [http://maven.org]. Maven, in particular, has a good resource describing this *layout* [http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html]. As this tutorial is to be a web application, we will be creating and making use of `src/main/java`, `src/main/resources` and `src/main/webapp` directories.

We will be using Maven in this tutorial, taking advantage of its transitive dependency management capabilities as well as the ability of many IDEs to automatically set up a project for us based on the maven descriptor.

```
<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">

    <modelVersion
>4.0.0</modelVersion>

    <groupId
>org.hibernate.tutorials</groupId>
    <artifactId
>hibernate-tutorial</artifactId>
    <version
>1.0.0-SNAPSHOT</version>
    <name
>First Hibernate Tutorial</name>

    <build>
         <!-- we dont want the version to be part of the generated war file name -->
         <finalName
>${artifactId}</finalName>
    </build>

    <dependencies>
        <dependency>
            <groupId
>org.hibernate</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>

        <!-- Hibernate uses slf4j for logging, for our purposes here use the simple backend -->
        <dependency>
            <groupId
>org.slf4j</groupId>
            <artifactId
>slf4j-simple</artifactId>
        </dependency>

        <!-- Hibernate gives you a choice of bytecode providers between cglib and javassist -->
        <dependency>
            <groupId
>javassist</groupId>
            <artifactId
>javassist</artifactId>
        </dependency>
    </dependencies>

</project
```

```
>
```

#### 

It is not a requirement to use Maven. If you wish to use something else to build this tutorial (such as Ant), the layout will remain the same. The only change is that you will need to manually account for all the needed dependencies. If you use something like *Ivy* [http://ant.apache.org/ivy/] providing transitive dependency management you would still use the dependencies mentioned below. Otherwise, you'd need to grab *all* dependencies, both explicit and transitive, and add them to the project's classpath. If working from the Hibernate distribution bundle, this would mean `hibernate3.jar`, all artifacts in the `lib/required` directory and all files from either the `lib/bytecode/cglib` or `lib/bytecode/javassist` directory; additionally you will need both the servlet-api jar and one of the slf4j logging backends.

Save this file as `pom.xml` in the project root directory.

### 1.1.2. ######

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

```
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;
    }
}
```

This class uses standard JavaBean naming conventions for property getter and setter methods, as well as private visibility for the fields. Although this is the recommended design, it is not required. Hibernate can also access fields directly, the benefit of accessor methods is robustness for refactoring.

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

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

Save this file to the `src/main/java/org/hibernate/tutorial/domain` directory.

### 1.1.3. #########

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 package="org.hibernate.tutorial.domain">
[...]
</hibernate-mapping
>
```

Hibernate DTD ######################### DTD ####### IDE ## XML ################## ################## DTD ################################################################ ################### Hibernate ## web ## DTD ######################################

################# DTD ##### Hibernate ############ src/ ##########hibernate3.jar ##########

> ####
>
> ################# DTD ##############################

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

```
<hibernate-mapping package="org.hibernate.tutorial.domain">

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

    </class>

</hibernate-mapping
>
```

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

```
<hibernate-mapping package="org.hibernate.tutorial.domain">

    <class name="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. The name="id" mapping attribute declares the name of the JavaBean property and tells Hibernate to use the getId() and setId() methods to access the property. The column attribute tells Hibernate which column of the EVENTS table holds the primary key value.

The nested generator element specifies the identifier generation strategy (aka how are identifier values generated?). In this case we choose native, which offers a level of portability depending on the configured database dialect. Hibernate supports database generated, globally unique, as well as application assigned, identifiers. Identifier value generation is also one of Hibernate's many extension points and you can plugin in your own strategy.

### ####

`native` is no longer consider the best strategy in terms of portability. for further discussion, see *#Identifier generation#*

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

```
<hibernate-mapping package="org.hibernate.tutorial.domain">

    <class name="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` ########### ########################

### ####

Hibernate makes this mapping type determination using reflection when the mapping files are processed. This can take time and resources, so if startup performance is important you should consider explicitly defining the type to use.

Save this mapping file as `src/main/resources/org/hibernate/tutorial/domain/Event.hbm.xml`.

### 1.1.4. Hibernate ###

At this point, you should have the persistent class and its mapping file in place. It is now time to configure Hibernate. First let's set up HSQLDB to run in "server mode"

##

We do this do that the data remains between runs.

We will utilize the Maven exec plugin to launch the HSQLDB server by running: `mvn exec:java -Dexec.mainClass="org.hsqldb.Server" -Dexec.args="-database.0 file:target/data/tutorial"` You will see it start up and bind to a TCP/IP socket; this is where our application will connect later. If you want to start with a fresh database during this tutorial, shutdown HSQLDB, delete all files in the `target/data` directory, and start HSQLDB again.

Hibernate will be connecting to the database on behalf of your application, so it needs to know how to obtain connections. For this tutorial we will be using a standalone connection pool (as opposed to a `javax.sql.DataSource`). Hibernate comes with support for two third-party open source JDBC connection pools: *c3p0* [https://sourceforge.net/projects/c3p0] and *proxool* [http://proxool.sourceforge.net/]. However, we will be using the Hibernate built-in connection pool for this tutorial.

##

The built-in Hibernate connection pool is in no way intended for production use. It lacks several features found on any decent connection pool.

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"
>update</property>

        <mapping resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>

    </session-factory>

</hibernate-configuration
>
```

##

## XML ####### DTD ################

############################ Hibernate # `SessionFactory` ########################
## ######################################## `<session-factory>` ##############

###4## `property` ### JDBC ##################### dialect ###### `property` #### Hibernate ######## SQL #########

####

In most cases, Hibernate is able to properly determine which dialect to use. See *#Dialect resolution#* for more information.

############## Hibernate ######################################## hbm2ddl.auto ## ###################### on ###############################config ########### off ### ## SchemaExport ### Ant ##################################################### #####

Save this file as `hibernate.cfg.xml` into the `src/main/resources` directory.

### 1.1.5. Maven ######

We will now build the tutorial with Maven. You will need to have Maven installed; it is available from the *Maven download page* [http://maven.apache.org/download.html]. Maven will read the `/pom.xml` file we created earlier and know how to perform some basic project tasks. First, lets run the `compile` goal to make sure we can compile everything so far:

```
[hibernateTutorial]$ mvn compile
[INFO] Scanning for projects...
[INFO] ----------------------------------------------------------------------
[INFO] Building First Hibernate Tutorial
[INFO]    task-segment: [compile]
[INFO] ----------------------------------------------------------------------
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Compiling 1 source file to /home/steve/projects/sandbox/hibernateTutorial/target/classes
[INFO] ----------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ----------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Tue Jun 09 12:25:25 CDT 2009
[INFO] Final Memory: 5M/547M
[INFO] ----------------------------------------------------------------------
```

### 1.1.6. ###########

## Event ################################################################## ######## Hibernate ######################################## SessionFactory ######## ############################################## org.hibernate.SessionFactory #### org.hibernate.Session ############### org.hibernate.Session ##############(Unit of Work)############ org.hibernate.SessionFactory ################################ #########

################# org.hibernate.SessionFactory ####### HibernateUtil ########### #############

```
package org.hibernate.tutorial.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
```

```
public class HibernateUtil {

    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            return 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;
    }

}
```

Save this code as `src/main/java/org/hibernate/tutorial/util/HibernateUtil.java`

This class not only produces the global `org.hibernate.SessionFactory` reference in its static initializer; it also hides the fact that it uses a static singleton. We might just as well have looked up the `org.hibernate.SessionFactory` reference from JNDI in an application server or any other location for that matter.

If you give the `org.hibernate.SessionFactory` a name in your configuration, Hibernate will try to bind it to JNDI under that name after it has been built. Another, better option is to use a JMX deployment and let the JMX-capable container instantiate and bind a `HibernateService` to JNDI. Such advanced options are discussed later.

You now need to configure a logging system. Hibernate uses commons logging and provides two choices: Log4j and JDK 1.4 logging. Most developers prefer Log4j: copy `log4j.properties` from the Hibernate distribution in the `etc/` directory to your `src` directory, next to `hibernate.cfg.xml`. If you prefer to have more verbose output than that provided in the example configuration, you can change the settings. By default, only the Hibernate startup message is shown on stdout.

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

### 1.1.7. ############

We are now ready to start doing some real work with Hibernate. Let's start by writing an `EventManager` class with a `main()` method:

```
package org.hibernate.tutorial;

import org.hibernate.Session;
```

```
import java.util.*;

import org.hibernate.tutorial.domain.Event;
import org.hibernate.tutorial.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();
    }

}
```

In `createAndStoreEvent()` we created a new `Event` object and handed it over to Hibernate. At that point, Hibernate takes care of the SQL and executes an `INSERT` on the database.

A org.hibernate.Session is designed to represent a single unit of work (a single atomic piece of work to be performed). For now we will keep things simple and assume a one-to-one granularity between a Hibernate org.hibernate.Session and a database transaction. To shield our code from the actual underlying transaction system we use the Hibernate `org.hibernate.Transaction` API. In this particular case we are using JDBC-based transactional semantics, but it could also run with JTA.

What does `sessionFactory.getCurrentSession()` do? First, you can call it as many times and anywhere you like once you get hold of your `org.hibernate.SessionFactory`. The `getCurrentSession()` method always returns the "current" unit of work. Remember that we switched the configuration option for this mechanism to "thread" in our `src/main/resources/hibernate.cfg.xml`? Due to that setting, the context of a current unit of work is bound to the current Java thread that executes the application.

#### ####

Hibernate offers three methods of current session tracking. The "thread" based method is not intended for production use; it is merely useful for prototyping and

tutorials such as this one. Current session tracking is discussed in more detail later on.

A org.hibernate.Session begins when the first call to `getCurrentSession()` is made for the current thread. It is then bound by Hibernate to the current thread. When the transaction ends, either through commit or rollback, Hibernate automatically unbinds the org.hibernate.Session from the thread and closes it for you. If you call `getCurrentSession()` again, you get a new org.hibernate.Session and can start a new unit of work.

#### (Unit of Work) ######## Hibernate # org.hibernate.Session #1######################### ########################################1## org.hibernate.Session ################### ########################### Hibernate # org.hibernate.Session ########## ### ######## ############ Hibernate org.hibernate.Session ########################################### ####### (###) ######################### ######### *Session* ############# (###) ####### ##########################

See *12#Transactions and Concurrency* for more information about transaction handling and demarcation. The previous example also skipped any error handling and rollback.

To run this, we will make use of the Maven exec plugin to call our class with the necessary classpath setup: `mvn exec:java -Dexec.mainClass="org.hibernate.tutorial.EventManager" -Dexec.args="store"`

##

You may need to perform `mvn compile` first.

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

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

This is the `INSERT` executed by Hibernate.

To list stored events an option is added to the main method:

```
        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;
    }
```

Here, we are using a Hibernate Query Language (HQL) query to load all existing `Event` objects from the database. Hibernate will generate the appropriate SQL, send it to the database and populate `Event` objects with the data. You can create more complex queries with HQL. See *15 #HQL: Hibernate #####* for more information.

Now we can call our new functionality, again using the Maven exec plugin: `mvn exec:java -Dexec.mainClass="org.hibernate.tutorial.EventManager" -Dexec.args="list"`

## 1.2. ###2 - ########

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

### 1.2.1. Person #########

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

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

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'

}
```

Save this to a file named `src/main/java/org/hibernate/tutorial/domain/Person.java`

Next, create the new mapping file as `src/main/resources/org/hibernate/tutorial/domain/Person.hbm.xml`

```
<hibernate-mapping package="org.hibernate.tutorial.domain">

    <class name="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#########################################################################
######################

## 1.2.2. ### Set #####

By adding a collection of events to the `Person` class, you can easily navigate to the events for a particular person, without executing an explicit query - by calling `Person#getEvents`. Multi-valued associations are represented in Hibernate by one of the Java Collection Framework contracts; here we choose a `java.util.Set` because the collection will not contain duplicate elements and the ordering is not relevant to our examples:

```
public class Person {

    private Set events = new HashSet();

    public Set getEvents() {
        return events;
    }

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

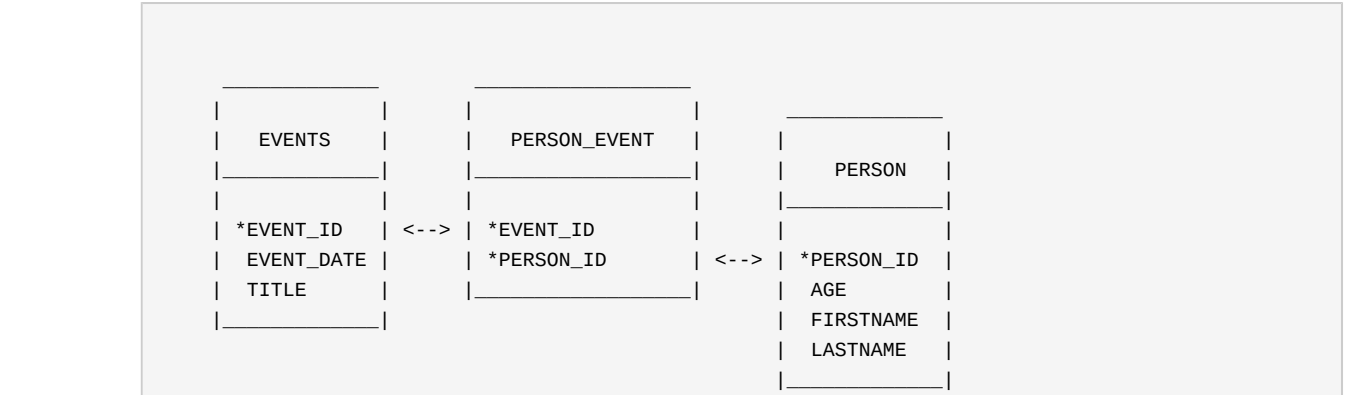
Before mapping this association, let's consider the other side. We could just keep this unidirectional or create another collection on the `Event`, if we wanted to be able to navigate it from both directions. This is not necessary, from a functional perspective. You can always execute an explicit query to retrieve the participants for a particular event. This is a design choice left to you,

but what is clear from this discussion is the multiplicity of the association: "many" valued on both sides is called a *many-to-many* association. Hence, we use Hibernate's many-to-many mapping:

```
<class name="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="Event"/>
    </set>

</class
>
```

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

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




## 1.2.3. #######

`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();
    }
```

After loading a `Person` and an `Event`, simply modify the collection using the normal collection methods. There is no explicit call to `update()` or `save()`; Hibernate automatically detects that the collection has been modified and needs to be updated. This is called *automatic dirty checking*. You can also try it by modifying the name or the date property of any of your objects. As long as they are in *persistent* state, that is, bound to a particular Hibernate `org.hibernate.Session`, Hibernate monitors any changes and executes SQL in a write-behind fashion. The process of synchronizing the memory state with the database, usually only at the end of a unit of work, is called *flushing*. In our code, the unit of work ends with a commit, or rollback, of the database transaction.

####### (Unit of Work) ############################################################ ## ###*detached*# ####### `org.hibernate.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 # java.lang.String ################## ## ############
############ ## ################## ID ############################## ##########
##########2###### firstname ####################### JDK ################## ####
Hibernate ################ JDK ############## ### Address # MonetaryAmount ######
#################

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

## 1.2.4. ########

Let's add a collection of email addresses to the `Person` entity. This will be represented as a `java.util.Set` of `java.lang.String` instances:

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

    public Set getEmailAddresses() {
        return emailAddresses;
    }

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

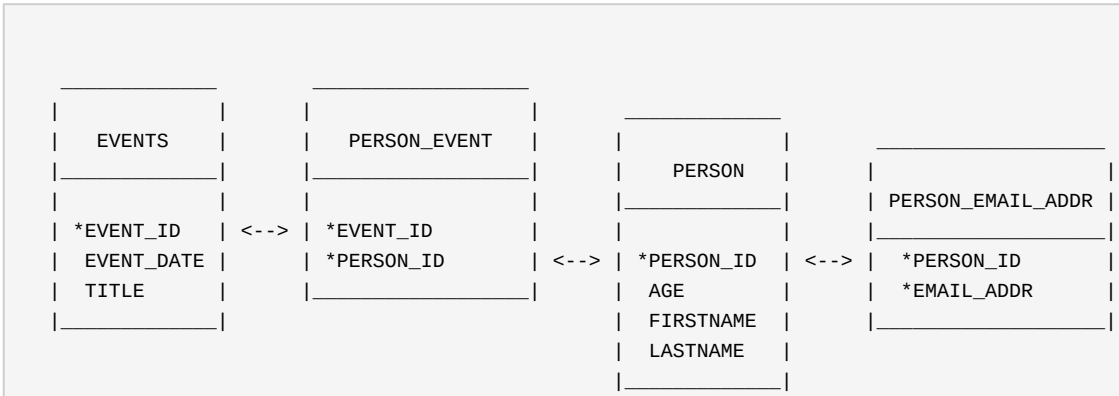
## Set #########

```
        <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);
        // adding to the emailAddress collection might trigger a lazy load of the collection
        aPerson.getEmailAddresses().add(emailAddress);

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

This time we did not use a *fetch* query to initialize the collection. Monitor the SQL log and try to optimize this with an eager fetch.

### 1.2.5. #####

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

## 1.2.6. #########

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

################################################################ 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` ################################## ######################################

## 1.3. ###3 - EventManager Web ########

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

### 1.3.1. #### Servlet ###

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

```
package org.hibernate.tutorial.web;

// Imports

public class EventManagerServlet extends HttpServlet {

    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();
            if ( ServletException.class.isInstance( ex ) ) {
                throw ( ServletException ) ex;
            }
            else {
                throw new ServletException( ex );
            }
```

```
        }
    }

}
```

Save this servlet as `src/main/java/org/hibernate/tutorial/web/EventManagerServlet.java`

### *session-per-request* ########## Servlet ############# `SessionFactory` # `getCurrentSession()` ########## Hibernate #### `Session` ############################ ############################################################################## ###

#################### Hibernate `Session` ### ######## ################## Hibernate `Session` ################## Java ############### `getCurrentSession()` ##########

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

############ HTML ################ (Unit of Work) ################################ ################################## `session-per-request` ########################## ########################################### *Open Session in View* ################ Hibernate # Web #### Wiki ################### JSP # HTML ###################### ################

## 1.3.2. #####

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

```
        // 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();
```

Java # HTML ########################################### ################## Hibernate ######################################## HTML ########################## ######### HTML ################################################## 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
>");
            Iterator it = result.iterator();
            while (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);
    }
```

##############################1## Session # Transaction ######################## ############## Hibernate ############################################################ ################ SessionFactory ################################################ ################################DAO############### Hibernate # Wiki ##########

### 1.3.3. ########

To deploy this application for testing we must create a Web ARchive (WAR). First we must define the WAR descriptor as `src/main/webapp/WEB-INF/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
>org.hibernate.tutorial.web.EventManagerServlet</servlet-class>
    </servlet>

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

########## `mvn package` ###### `hibernate-tutorial.war` ##### Tomcat # `webapp` ##########

##

If you do not have Tomcat installed, download it from *http://tomcat.apache.org/* and follow the installation instructions. Our application requires no changes to the standard Tomcat configuration.

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

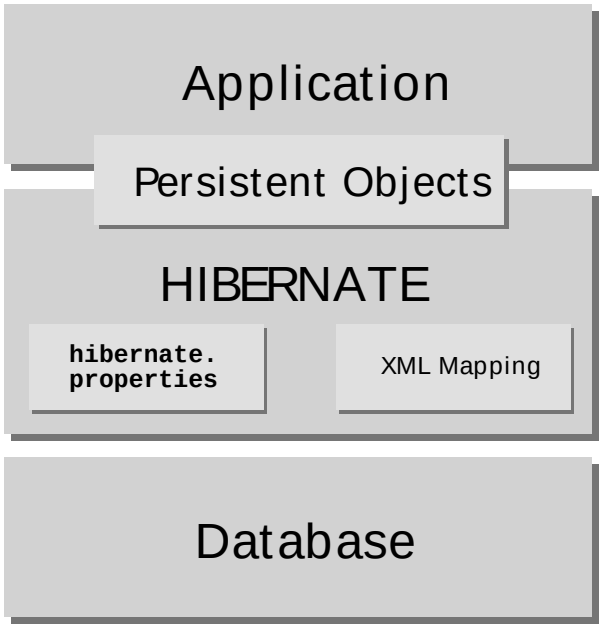
## 1.4. ##

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

# #######

## 2.1. ##

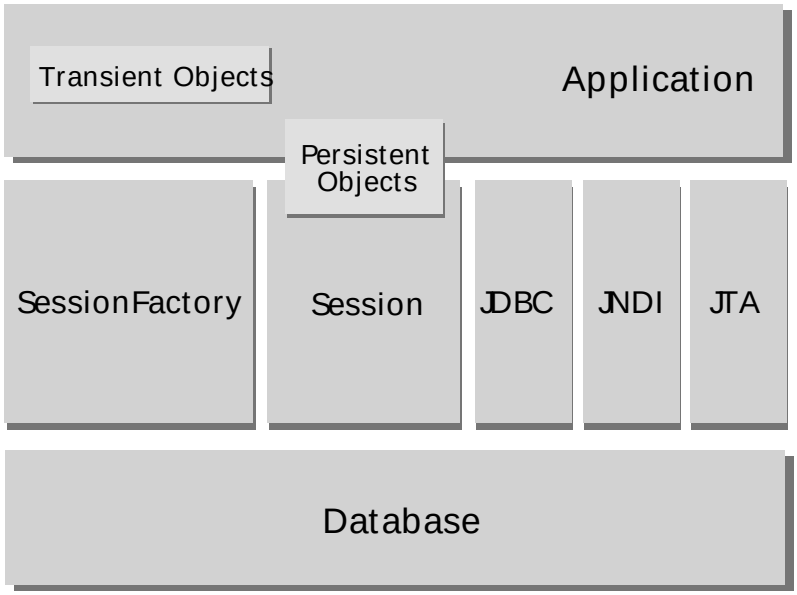
Hibernate ##########################



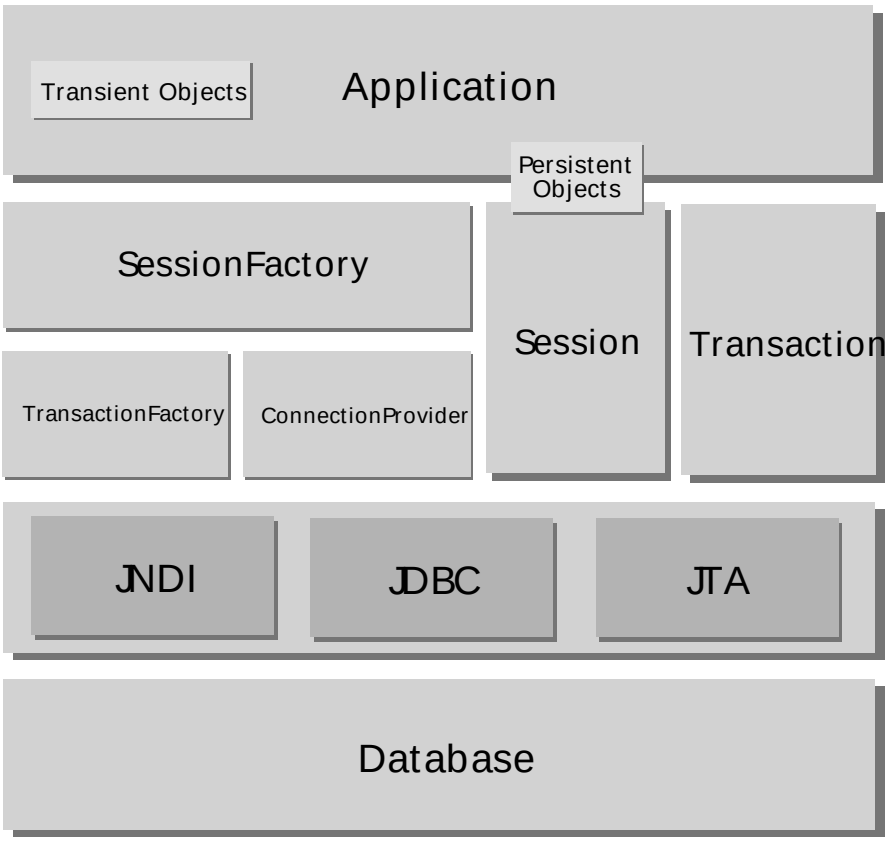

We do not have the scope in this document to provide a more detailed view of all the runtime architectures available; Hibernate is flexible and supports several different approaches. We will, however, show the two extremes: "minimal" architecture and "comprehensive" architecture.

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

The "minimal" architecture has the application provide its own JDBC connections and manage its own transactions. This approach uses a minimal subset of Hibernate's APIs:




The "comprehensive" architecture abstracts the application away from the underlying JDBC/JTA APIs and allows Hibernate to manage the details.




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

SessionFactory (`org.hibernate.SessionFactory`)

A threadsafe, immutable cache of compiled mappings for a single database. A factory for `Session` and a client of `ConnectionProvider`, `SessionFactory` can hold an optional (second-level) cache of data that is reusable between transactions at a process, or cluster, level.

Session (`org.hibernate.Session`)

A single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps a JDBC connection and is a factory for `Transaction`. `Session` holds a mandatory first-level cache of persistent objects that are used when navigating the object graph or looking up objects by identifier.

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` # API #########

## 2.2. #########

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

transient
: The instance is not associated with any persistence context. It has no persistent identity or primary key value.

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

detached
: ########################################################################## ######################### ID ############################################ #### ID # Java # ID ###### Hibernate ########

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

Another feature available as a JMX service is runtime Hibernate statistics. See *#Hibernate ###* for more information.

## 2.4. JCA ####

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

## 2.5. #############

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

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

Starting with version 3.0.1, Hibernate added the `SessionFactory.getCurrentSession()` method. Initially, this assumed usage of `JTA` transactions, where the `JTA` transaction defined both the scope and context of a current session. Given the maturity of the numerous stand-alone `JTA TransactionManager` implementations, most, if not all, applications should be using `JTA` transaction management, whether or not they are deployed into a `J2EE` container. Based on that, the `JTA`-based contextual sessions are all you need to use.

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

See the Javadocs for the `org.hibernate.context.CurrentSessionContext` interface for a detailed discussion of its contract. It defines a single method, `currentSession()`, by which the implementation is responsible for tracking the current contextual session. Out-of-the-box, Hibernate comes with three implementations of this interface:

- `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. This is 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, you can utilize the JTA interfaces to demarcate transactions. If you execute in an EJB container that supports CMT, transaction boundaries are defined declaratively and you do not need any transaction or session demarcation operations in your code. Refer to *12#Transactions and Concurrency* 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"####################

## ##

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

## 3.1. ##########

org.hibernate.cfg.Configuration ######### Java ### SQL ######################### Configuration ######## SessionFactory ################## XML ###################### ######

### org.hibernate.cfg.Configuration ########### XML ########################### ########################################### 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);
```

Hibernate ##################################### /org/hibernate/auction/Item.hbm.xml # /org/hibernate/auction/Bid.hbm.xml ############################## ####

org.hibernate.cfg.Configuration ######################

```
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. java.util.Properties ####### Configuration.setProperties() ######
2. hibernate.properties ######################
3. System ###### java -Dproperty=value ############
4. <property> ### hibernate.cfg.xml ###########

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

`org.hibernate.cfg.Configuration` ###################### `SessionFactory` ######### ###############

## 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 ######

####### `org.hibernate.SessionFactory` ##### SessionFactory # JDBC ################ ######################### `org.hibernate.Session` ############

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

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

####### JDBC ############## Hibernate ################## Hibernate ############### `org.hibernate.cfg.Environment` ################## JDBC ####################

################## Hibernate ###################### `java.sql.DriverManager` #####:

### #3.1 Hibernate JDBC #####

| ###### | ## |
| --- | --- |
| hibernate.connection.driver_class | *JDBC driver class* |
| hibernate.connection.url | *JDBC URL* |
| hibernate.connection.username | *database user* |
| hibernate.connection.password | *database user password* |
| hibernate.connection.pool_size | *maximum number of pooled connections* |

Hibernate ################################################## ############## ###### ########################################################################## ######## `hibernate.connection.pool_size` ###################################### Hibernate ########################## C3P0 ######

C3P0 ######## JDBC ########### Hibernate # `lib` ################ `hibernate.c3p0.*` ############## Hibernate ## `C3P0ConnectionProvider` ######## Proxool ######### `hibernate.properties` ############ Hibernate # Web ######################

C3P0 ## `hibernate.properties` ###############

```
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
```

###################### Hibernate ######################################## `javax.sql.Datasource` # JNDI ###############################:

### #3.2 Hibernate ###########

| ###### | ## |
|---|---|
| hibernate.connection.datasource | *datasource JNDI name* |
| hibernate.jndi.url | *JNDI ###### URL* (#####) |
| hibernate.jndi.class | *JNDI #### `InitialContextFactory`* (#####) |
| hibernate.connection.username | ######### (#####) |
| hibernate.connection.password | ################ (#####) |

################### JNDI ######### `hibernate.properties` ############

```
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 ################################################

###################### "`hibernate.connnection`" ###################### charSet ######## hibernate.connection.charSet ######

JDBC ################################## `org.hibernate.connection.ConnectionProvider` ############################ hibernate.connection.provider_class ##########

## 3.4. ############

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

##

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

### #3.3 Hibernate #######

| ###### | ## |
|---|---|
| hibernate.dialect | Hibernate ##### `org.hibernate.dialect.Dialect` ######## ######################## SQL #######<br><br>`#full.classname.of.Dialect`<br><br>In most cases Hibernate will actually be able to choose the correct `org.hibernate.dialect.Dialect` implementation based on the `JDBC metadata` returned by the JDBC driver. |
| hibernate.show_sql | ######### SQL ######################## `org.hibernate.SQL` # `debug` ############ ##<br><br>`#true` \| `false` |
| hibernate.format_sql | ######### SQL ###########<br><br>`#true` \| `false` |
| hibernate.default_schema | ##### SQL ###############/###########<br><br>`#.SCHEMA_NAME` |
| hibernate.default_catalog | ##### SQL ################<br><br>`#CATALOG_NAME` |
| hibernate.session_factory_name | `org.hibernate.SessionFactory` ######### # JNDI ############<br><br>`#jndi/composite/name` |
| hibernate.max_fetch_depth | ################################### ######### `0` ###################### |

| ###### | ## |
| --- | --- |
| | ## ###### `0` ## `3` ##### |
| hibernate.default_batch_fetch_size | ############ Hibernate #############<br><br>## ###### `4` , `8` , `16` ### |
| hibernate.default_entity_mode | Sets a default mode for entity representation for all sessions opened from this `SessionFactory`<br><br>`dynamic-map`, `dom4j`, `pojo` |
| hibernate.order_updates | ############## SQL ##################################################<br>######<br><br>#`true` \| `false` |
| hibernate.generate_statistics | ###### Hibernate ######################<br>#######<br><br>#`true` \| `false` |
| hibernate.use_identifier_rollback | ####################################<br>##################<br><br>#`true` \| `false` |
| hibernate.use_sql_comments | ###### SQL ##########################<br>########## `false` ###<br><br>#`true` \| `false` |

## #3.4 Hibernate JDBC ############

| ###### | ## |
| --- | --- |
| hibernate.jdbc.fetch_size | ##0###### JDBC ############# ( `Statement.setFetchSize()` #####)# |
| hibernate.jdbc.batch_size | ##0###### Hibernate # JDBC2 ##########<br>##<br><br>## ###### `5` ## `30` ##### |
| hibernate.jdbc.batch_versioned_data | Set this property to `true` if your JDBC driver returns correct row counts from `executeBatch()`. It is usually safe to turn this option on. Hibernate will then use batched DML for automatically versioned data. Defaults to `false`.<br><br>#`true` \| `false` |

| ###### | ## |
| --- | --- |
| hibernate.jdbc.factory_class | #### `org.hibernate.jdbc.Batcher` ###### ##################################<br><br>#`classname.of.BatcherFactory` |
| hibernate.jdbc.use_scrollable_resultset | Hibernate ### JDBC2 ################## ############################## JDBC # ########################## Hibernate # ##################<br><br>#`true` \| `false` |
| hibernate.jdbc.use_streams_for_binary | JDBC #/## `binary` # `serializable` #####/## ############ (##############)#<br><br>#`true` \| `false` |
| hibernate.jdbc.use_get_generated_keys | ###################### JDBC3 `PreparedStatement.getGeneratedKeys()` # ############ JDBC3+ ##### JRE1.4+ ## ###### Hibernate ###################### false ################################## #############<br><br>#`true\|false` |
| hibernate.connection.provider_class | JDBC ####### Hibernate ######## `ConnectionProvider` ######<br><br>#`classname.of.ConnectionProvider` |
| hibernate.connection.isolation | JDBC ################################ # `java.sql.Connection` ################ ##################################### ##<br><br>#`1, 2, 4, 8` |
| hibernate.connection.autocommit | ######## JDBC ###################### ###<br><br>#`true` \| `false` |
| hibernate.connection.release_mode | Hibernate ### JDBC ################## ##################################### ###################### JTA ##### ######### JDBC ################## ######## `after_statement` ######### # JTA ########################## `after_transaction` ################## |

| ###### | ## |
| --- | --- |
| | ### auto ##### JTA # CMT ########### after_statement ####### JDBC ######### ### after_transaction ######### <br><br> #auto (#####) \| on_close \| after_transaction \| after_statement <br><br> This setting only affects Sessions returned from SessionFactory.openSession. For Sessions obtained through SessionFactory.getCurrentSession, the CurrentSessionContext implementation configured for use controls the connection release mode for those Sessions. See ##### ########## |
| hibernate.connection.*<propertyName>* | JDBC ##### *<propertyName>* # DriverManager.getConnection() ###### |
| hibernate.jndi.*<propertyName>* | ##### *<propertyName>* # JNDI InitialContextFactory ###### |

### #3.5 Hibernate ##########

| ###### | ## |
| --- | --- |
| hibernate.cache.provider_class | #### CacheProvider ######## <br><br> #classname.of.CacheProvider |
| hibernate.cache.use_minimal_puts | ################################## ################################## ################ Hibernate3 ########### ##################### <br><br> #true\|false |
| hibernate.cache.use_query_cache | ################################## <br><br> #true\|false |
| hibernate.cache.use_second_level_cache | ################################## ### <cache> ############ <br><br> #true\|false |
| hibernate.cache.query_cache_factory | #### QueryCache ##################### ###### StandardQueryCache ###### <br><br> **e.g.** classname.of.QueryCache |
| hibernate.cache.region_prefix | ################# |

| ###### | ## |
| --- | --- |
| | #prefix |
| hibernate.cache.use_structured_entries | ###################################<br><br>#true\|false |

## #3.6 Hibernate ##########

| ###### | ## |
| --- | --- |
| hibernate.transaction.factory_class | Hibernate Transaction API ######## TransactionFactory ######## ######## JDBCTransactionFactory ####<br><br>#classname.of.TransactionFactory > |
| jta.UserTransaction | ########## JTA UserTransaction ###### ##### JTATransactionFactory ##### JNDI ####<br><br>#jndi/composite/name |
| hibernate.transaction.manager_lookup_class | TransactionManagerLookup ######## JTA ####### JVM ################## hilo ################<br><br>#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 ######### .<br><br>#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 automatic session context management is preferred, see ######### .<br><br>#true \| false |

## #3.7 #########

| ###### | ## |
| --- | --- |
| hibernate.current_session_context_class | Supply a custom strategy for the scoping of the "current" Session. See ########## for more information about the built-in strategies. |

| ###### | ## |
| --- | --- |
| | #`jta` \| `thread` \| `managed` \| `custom.Class` |
| `hibernate.query.factory_class` | HQL ##############<br><br>#`org.hibernate.hql.ast.ASTQueryTranslatorFactory` or `org.hibernate.hql.classic.ClassicQueryTranslatorFactory` |
| `hibernate.query.substitutions` | HQL # SQL ############### ############ ############<br><br>#`hqlLiteral=SQL_LITERAL, hqlFunction=SQLFUNC` |
| `hibernate.hbm2ddl.auto` | `SessionFactory` ################# DDL ################## `create-drop` #### `SessionFactory` ######################## ########<br><br># `validate` \| `update` \| `create` \| `create-drop` |
| `hibernate.bytecode.use_reflection_optimizer` | Enables the use of bytecode manipulation instead of runtime reflection. This is a System-level property and cannot be set in `hibernate.cfg.xml`. Reflection can sometimes be useful when troubleshooting. Hibernate always requires either CGLIB or javassist even if you turn off the optimizer.<br><br>#`true` \| `false` |
| `hibernate.bytecode.provider` | Both javassist or cglib can be used as byte manipulation engines; the default is `javassist`.<br><br>**e.g.** `javassist` \| `cglib` |

## 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` |

| RDBMS | Dialect |
|---|---|
| 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 ############# | `org.hibernate.dialect.OracleDialect` |
| Oracle 9i | `org.hibernate.dialect.Oracle9iDialect` |
| Oracle 10g | `org.hibernate.dialect.Oracle10gDialect` |
| 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. ########

######### ANSI ## Oracle # Sybase ##################### *outer join fetching* ######## # SQL ########################################################################## #######################1## SQL # `SELECT` ####

`hibernate.max_fetch_depth` ######## `0` ############# ### ############# `1` ########## ############################## `fetch="join"` #############

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

### 3.4.3. #########

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

### 3.4.4. ################

The properties prefixed by `hibernate.cache` allow you to use a process or cluster scoped second-level cache system with Hibernate. See the ##2######### for more information.

### 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 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 # root ##################

```
<?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` # ###### ##################2################## 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` ############################### (CMT) ########### (BMT/ UserTransaction) ##########

- *JMX #######:* ## JMX ###################### JBoss AS# ###### 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 ########################

The easiest way to handle Sessions and transactions is Hibernate's automatic "current" Session management. For a discussion of contextual sessions see ###############. Using the "jta" session context, if there is no Hibernate Session associated with the current JTA

transaction, one will be started and associated with that JTA transaction the first time you call `sessionFactory.getCurrentSession()`. The `Session`s retrieved via `getCurrentSession()` in the `"jta"` context are set to automatically flush before the transaction completes, close after the transaction completes, and aggressively release JDBC connections after each statement. This allows the `Session`s to be managed by the life cycle of the JTA transaction to which it is associated, keeping user code clean of such management concerns. Your code can either use JTA programmatically through `UserTransaction`, or (recommended for portable code) use the Hibernate `Transaction` API to set transaction boundaries. If you run in an EJB container, declarative transaction demarcation with CMT is preferred.

### 3.8.4. JMX #######

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

JBoss AS #### 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 AS ###### ## JXM ##### EJB ######################

# #####

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

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

## 4.1. ### POJO ##

Most Java applications require a persistent class representing felines. For example:

```
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);
    }
}
```

The four main rules of persistent classes are explored in more detail in the following sections.

### 4.1.1. ##################

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

### 4.1.2. ######################

`Cat` has a property called `id`. This property maps to the primary key column of a database table. The property might have been called anything, and its type might have been any primitive type, any primitive "wrapper" type, `java.lang.String` or `java.util.Date`. If your legacy database table has composite keys, you can use a user-defined class with properties of these types (see the section on composite identifiers later in the chapter.)

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

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

- Transitive reattachment for detached objects (cascade update or cascade merge) - see ####
  #####
- `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` declares accessor methods for all its persistent fields. Many other ORM tools directly persist instance variables. It is better to provide an indirection between the relational schema and internal data structures of the class. By default, Hibernate persists JavaBeans style properties and recognizes method names of the form `getFoo`, `isFoo` and `setFoo`. If required, you can switch to direct field access for particular properties.

###### 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 ############################### Java ID ################### ############################### `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;
    }

}
```

A business key does not have to be as solid as a database primary key candidate (see ####### ########). Immutable or unique properties are usually good candidates for a business key.

## 4.4. #####

##

*The following features are currently considered experimental and may change in the near future.*

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

By default, Hibernate works in normal POJO mode. You can set a default entity representation mode for a particular `SessionFactory` using the `default_entity_mode` configuration option (see *#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();

// 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)##################

More information about the XML representation capabilities can be found in *19#XML #####*.

## 4.5. Tuplizer

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

Tuplizer ######################`org.hibernate.tuple.entity.EntityTuplizer` # `org.hibernate.tuple.component.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. EntityNameResolvers

The `org.hibernate.EntityNameResolver` interface is a contract for resolving the entity name of a given entity instance. The interface defines a single method `resolveEntityName` which is passed the entity instance and is expected to return the appropriate entity name (null is allowed and would indicate that the resolver does not know how to resolve the entity name of the given entity instance). Generally speaking, an `org.hibernate.EntityNameResolver` is going to be most useful in the case of dynamic models. One example might be using proxied interfaces as your domain model. The hibernate test suite has an example of this exact style of usage under the org.hibernate.test.dynamicentity.tuplizer2. Here is some of the code from that package for illustration.

```
/**
 * A very trivial JDK Proxy InvocationHandler implementation where we proxy an interface as
 * the domain model and simply store persistent state in an internal Map.  This is an extremely
 * trivial example meant only for illustration.
 */
public final class DataProxyHandler implements InvocationHandler {
        private String entityName;
        private HashMap data = new HashMap();

        public DataProxyHandler(String entityName, Serializable id) {
                this.entityName = entityName;
                data.put( "Id", id );
        }

        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                String methodName = method.getName();
                if ( methodName.startsWith( "set" ) ) {
                        String propertyName = methodName.substring( 3 );
                        data.put( propertyName, args[0] );
                }
                else if ( methodName.startsWith( "get" ) ) {
                        String propertyName = methodName.substring( 3 );
                        return data.get( propertyName );
                }
                else if ( "toString".equals( methodName ) ) {
                        return entityName + "#" + data.get( "Id" );
                }
                else if ( "hashCode".equals( methodName ) ) {
                        return new Integer( this.hashCode() );
                }
                return null;
        }

        public String getEntityName() {
                return entityName;
        }

        public HashMap getData() {
                return data;
```

```
        }
}

/**
 *
 */
public class ProxyHelper {
    public static String extractEntityName(Object object) {
        // Our custom java.lang.reflect.Proxy instances actually bundle
        // their appropriate entity name, so we simply extract it from there
        // if this represents one of our proxies; otherwise, we return null
        if ( Proxy.isProxyClass( object.getClass() ) ) {
            InvocationHandler handler = Proxy.getInvocationHandler( object );
            if ( DataProxyHandler.class.isAssignableFrom( handler.getClass() ) ) {
                DataProxyHandler myHandler = ( DataProxyHandler ) handler;
                return myHandler.getEntityName();
            }
        }
        return null;
    }

    // various other utility methods ....

}

/**
 * The EntityNameResolver implementation.
 * IMPL NOTE : An EntityNameResolver really defines a strategy for how entity names should be
 * resolved.  Since this particular impl can handle resolution for all of our entities we want to
 * take advantage of the fact that SessionFactoryImpl keeps these in a Set so that we only ever
 * have one instance registered.  Why?  Well, when it comes time to resolve an entity name,
 * Hibernate must iterate over all the registered resolvers.  So keeping that number down
 * helps that process be as speedy as possible.  Hence the equals and hashCode impls
 */
public class MyEntityNameResolver implements EntityNameResolver {
    public static final MyEntityNameResolver INSTANCE = new MyEntityNameResolver();

    public String resolveEntityName(Object entity) {
        return ProxyHelper.extractEntityName( entity );
    }

    public boolean equals(Object obj) {
        return getClass().equals( obj.getClass() );
    }

    public int hashCode() {
        return getClass().hashCode();
    }
}

public class MyEntityTuplizer extends PojoEntityTuplizer {
       public MyEntityTuplizer(EntityMetamodel entityMetamodel, PersistentClass mappedEntity) {
                super( entityMetamodel, mappedEntity );
        }

        public EntityNameResolver[] getEntityNameResolvers() {
                return new EntityNameResolver[] { MyEntityNameResolver.INSTANCE };
        }
```

```
    public String determineConcreteSubclassEntityName(Object entityInstance, SessionFactoryImplementor factory) {
        String entityName = ProxyHelper.extractEntityName( entityInstance );
        if ( entityName == null ) {
            entityName = super.determineConcreteSubclassEntityName( entityInstance, factory );
        }
        return entityName;
    }

    ...
}
```

In order to register an `org.hibernate.EntityNameResolver` users must either:

1. Implement a custom *Tuplizer*, implementing the `getEntityNameResolvers` method.
2. Register it with the `org.hibernate.impl.SessionFactoryImpl` (which is the implementation class for `org.hibernate.SessionFactory`) using the `registerEntityNameResolver` method.

# #### 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 namespace` ####### `http://hibernate.sourceforge.net/` ######## ID ##### ################### Hibernate ##########################################
- `user namespace` ####### 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 version "3.5.6-Final">
```

```
<!ENTITY today "September 15, 2010">

    <!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>
```

Where `types.xml` is a resource in the `your.domain` package and contains a custom *typedef*.

## 5.1.2. Hibernate-mapping

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

```
<hibernate-mapping
        schema="schemaName"                                    ❶
        catalog="catalogName"                                  ❷
        default-cascade="cascade_style"                        ❸
        default-access="field|property|ClassName"              ❹
        default-lazy="true|false"                              ❺
        auto-import="true|false"                               ❻
        package="package.name"                                 ❼
 />
```

❶ schema`#####################`

❷ catalog `#####################`

❸ default-cascade `######` - `######` none`##` `################`

❹ default-access (`#####` - `######` property `##` Hibernate `#####################` PropertyAccessor `#################`

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

❻ auto-import `######` - `######` true`##########################################` `#########`

❼ 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"                                                    ❶
        table="tableName"                                                   ❷
        discriminator-value="discriminator_value"                           ❸
        mutable="true|false"                                                ❹
        schema="owner"                                                      ❺
        catalog="catalog"                                                   ❻
        proxy="ProxyInterface"                                              ❼
        dynamic-update="true|false"                                         ❽
        dynamic-insert="true|false"                                         ❾
        select-before-update="true|false"                                   ❿
        polymorphism="implicit|explicit"                                    ⓫
        where="arbitrary sql where condition"                               ⓬
        persister="PersisterClass"                                          ⓭
        batch-size="N"                                                      ⓮
        optimistic-lock="none|version|dirty|all"                            ⓯
        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"
/>
```

❶ name (#####)######################### Java ###################### POJO ###### ######################

❷ table (##### - ##################)###############

❸ discriminator-value (##### - ##########)# ################################### #### null # not null ###########

❹ mutable (############ true )# ######################################

❺ schema (#####): #### <hibernate-mapping> #######################

6. `catalog` (#####): #### `<hibernate-mapping>` ##########################
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. `polymorphism` (############## `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` (optional - defaults to the class name): Hibernate3 allows a class to be mapped multiple times, potentially to different tables. It also allows entity mappings that are represented by Maps or XML at the Java level. In these cases, you should provide an explicit arbitrary name for the entity. See ####### and *19#XML #####* for more information.
18. `check` ############################## *check* ####### SQL ##
19. `rowid` ######## Hibernate ##################### ROWID # ##################### Oracle ################## `rowid` ####### Hiberante # update ######### `rowid` ###### ############## ROWID ###################################
20. `subselect` (optional): maps an immutable and read-only entity to a database subselect. This is useful if you want to have a view instead of a base table. See below for more information.
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 (##### - ###### property ): Hibernate ##########################

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
>
```

############ org.hibernate.id.IdentifierGenerator ############################ ############################################## Hibernate ########################## ##########################

`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`

########################################### JDBC3 getGeneratedKeys ####### INSERT ################################## JDK 1.4 ###### Oracle 10g ########## ################### INSERT ######## Oracle ##############################

### 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"
>socialSecurityNumber</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 Optimization means that you do not have 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`. This is because `native` generally chooses between `identity` and `sequence` which have largely different semantics that can cause subtle issues in applications eyeing portability. `org.hibernate.id.enhanced.SequenceStyleGenerator`, however, achieves portability in a different manner. It chooses between 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 emulate 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 typically 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 typically 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, it is the name of the column on the table which is used to hold the value.
- `optimizer` (optional - defaults to `none`): See *#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, even though it actually functions much more like `org.hibernate.id.MultipleHiLoPerTableGenerator`, and secondly, as a re-implementation of `org.hibernate.id.MultipleHiLoPerTableGenerator` that utilizes the notion of pluggable optimizers. 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 that is used to hold the value.
- `segment_column_name` (optional - defaults to `sequence_name`): the name of the column on the table that is used to hold the "segment key". This is the value which 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 *#Identifier generator optimization#*

### 5.1.6. Identifier generator optimization

For identifier generators that 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 can 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 (*#Enhanced identifier generators#* support this operation.

- `none` (generally this is the default if no optimizer was specified): this will not perform any optimizations and hit the database for 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`: as with the case of `hilo`, this optimizer 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. Here, `increment_size` 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 (##########################): #################

We will describe a third, even more convenient approach, where the composite identifier is implemented as a component class in ###################. The attributes described below apply only to this alternative approach:

- name (##################): ############################(9##########)#
- access (##### - ###### property ): Hibernate ##########################
- 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"                                          ❶
        name="propertyName"                                              ❷
        type="typename"                                                  ❸
        access="field|property|ClassName"                                ❹
        unsaved-value="null|negative|undefined"                          ❺
        generated="never|always"                                         ❻
        insert="true|false"                                              ❼
        node="element-name|@attribute-name|element/@attribute|."
/>
```

1. `column` ###### - #############: ###################
2. `name` ################
3. `type` ###### - ###### `integer` ############
4. `access` (##### - ###### `property` ): Hibernate ###########################
5. `unsaved-value` (optional - defaults to `undefined`): a version property value that indicates that an instance is newly instantiated (unsaved), distinguishing it from detached instances that were saved or loaded in a previous session. `Undefined` specifies that the identifier property value should be used.
6. `generated` (optional - defaults to `never`): specifies that this version property value is generated by the database. See the discussion of *generated properties* for more information.
7. `insert` (##### - ###### `true` ): SQL# insert ######################################## ########## `0` ########### `false` #############

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

A version or timestamp property should never be null for a detached instance. Hibernate will detect any instance with a null version or timestamp as transient, irrespective of what other `unsaved-value` strategies are specified. *Declaring a nullable version or timestamp property is an easy way to avoid problems with transitive reattachment in Hibernate. It is especially useful for people using assigned identifiers or composite keys*.

## 5.1.10. timestamp#######

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

```
<timestamp
        column="timestamp_column"                                    ❶
        name="propertyName"                                          ❷
        access="field|property|ClassName"                            ❸
        unsaved-value="null|undefined"                               ❹
        source="vm|db"                                               ❺
        generated="never|always"                                     ❻
        node="element-name|@attribute-name|element/@attribute|."
/>
```

❶ `column`###### - ############## ####################

❷ `name` # ######## Java # `Date`#### `Timestamp` # ## JavaBeans ############

❸ `access` (##### - ###### `property` ): Hibernate ##########################

❹ `unsaved-value` ###### - ###### `null` ## ################### ################### ############## Session ######################################## # `undefined` ## ########################

❺ `source` (##### - ###### `vm` ): Hibernate ########################################## ##### JVM ########################### Hibernate # "###" ######################### ################################ JVM ######################################## ####### `Dialect` ############################################################## ##### (### Oracle 8 )#

❻ `generated` (optional - defaults to `never`): specifies that this timestamp property value is actually generated by the database. See the discussion of *generated properties* for more information.

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

## 5.1.11. property

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

```
<property
        name="propertyName"                                                  ❶
        column="column_name"                                                 ❷
        type="typename"                                                      ❸
        update="true|false"                                                  ❹
        insert="true|false"                                                  ❹
        formula="arbitrary SQL expression"                                   ❺
        access="field|property|ClassName"                                    ❻
        lazy="true|false"                                                    ❼
        unique="true|false"                                                  ❽
        not-null="true|false"                                                ❾
        optimistic-lock="true|false"                                         ❿
        generated="never|insert|always"                                      ⓫
        node="element-name|@attribute-name|element/@attribute|."
        index="index_name"
        unique_key="unique_key_id"
        length="L"
        precision="P"
        scale="S"
/>
```

❶ name# ##############

❷ column###### - ############## ############################## <column> ##### ######

❸ type######## Hibernate ########

❹ update, insert ###### - ###### true ## ############ SQL # UPDATE # INSERT #### ############## false ################################################ ######################

❺ formula######## ## ############### SQL ##################################

❻ access (##### - ###### property ): Hibernate ##########################

❼ lazy (##### - ###### false ): ############################################ (## ######################)#

❽ unique (#####):############## DDL ############## property-ref ############## ###

❾ not-null (optional): enables the DDL generation of a nullability constraint for the columns.

❿ optimistic-lock (##### - ###### true ): ######################################## ##########################################

⓫ generated (optional - defaults to never): specifies that this property value is actually generated by the database. See the discussion of *generated properties* for more information.

*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"                                    ❶
        column="column_name"                                   ❷
        class="ClassName"                                      ❸
        cascade="cascade_style"                                ❹
        fetch="join|select"                                    ❺
        update="true|false"                                    ❻
        insert="true|false"                                    ❻
```

```
        property-ref="propertyNameFromAssociatedClass"                    ❼
        access="field|property|ClassName"                                 ❽
        unique="true|false"                                               ❾
        not-null="true|false"                                             ❿
        optimistic-lock="true|false"                                      ⓫
        lazy="proxy|no-proxy|false"                                       ⓬
        not-found="ignore|exception"                                      ⓭
        entity-name="EntityName"                                          ⓮
        formula="arbitrary SQL expression"                                ⓯
        node="element-name|@attribute-name|element/@attribute|."
        embed-xml="true|false"
        index="index_name"
        unique_key="unique_key_id"
        foreign-key="foreign_key_name"
/>
```

❶ name########

❷ column (#####):################ <column> ###############

❸ class###### - ############################## #########

❹ cascade######## ########################################

❺ fetch###### - ###### select ## ##################sequential select fetch######## #####

❻ update, insert###### - ###### true ## ############ SQL # UPDATE ### INSERT ### ################ false ##################################################################################

❼ property-ref: (#####) ########################################################## ##

❽ access (##### - ###### property ): Hibernate ##########################

❾ unique######## ###################### DDL ############## property-ref ###### ###################################

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

⓫ optimistic-lock (##### - ###### true ): ########################################################################################

⓬ lazy (##### - ###### proxy ): ########################### lazy="no-proxy" ############################################## (###################### #)# lazy="false" #######################

⓭ not-found ###### - ###### exception#: ###########################: ignore #### ##########################

⓮ entity-name (#####):################

⓯ formula (#####): ##### ############## SQL #

Setting a value of the cascade attribute to any meaningful value other than none will propagate certain operations to the associated object. The meaningful values are divided into three

categories. First, basic operations, which include: `persist, merge, delete, save-update, evict, replicate, lock and refresh`; second, special values: `delete-orphan`; and third, `all` comma-separated combinations of operation names: `cascade="persist,merge,evict"` or `cascade="all,delete-orphan"`. See ######### for a full explanation. Note that single valued, many-to-one and one-to-one, associations do not support 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"                                                  ❶
        class="ClassName"                                                    ❷
        cascade="cascade_style"                                              ❸
        constrained="true|false"                                             ❹
        fetch="join|select"                                                  ❺
        property-ref="propertyNameFromAssociatedClass"                       ❻
```

```
        access="field|property|ClassName"                                    ❼
        formula="any SQL expression"                                         ❽
        lazy="proxy|no-proxy|false"                                          ❾
        entity-name="EntityName"                                             ❿
        node="element-name|@attribute-name|element/@attribute|."
        embed-xml="true|false"
        foreign-key="foreign_key_name"
/>
```

❶ name########

❷ class###### - ############################### #########

❸ cascade######## ########################################

❹ constrained######## ##################################################### ####### save() # delete() ######################################## ######## ##################

❺ fetch###### - ###### select ## ##################sequential select fetch######## #####

❻ property-ref######## ##################################################### ##

❼ access (##### - ###### property ): Hibernate ##########################

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

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

❿ entity-name (#####):################

#######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 and dynamic-component

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

```
<component
        name="propertyName"                                                  ❶
        class="className"                                                    ❷
        insert="true|false"                                                  ❸
        update="true|false"                                                  ❹
        access="field|property|ClassName"                                    ❺
        lazy="true|false"                                                    ❻
        optimistic-lock="true|false"                                         ❼
        unique="true|false"                                                  ❽
        node="element-name|."
>

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

❶ `name`########

❷ `class` ###### - ############################## ##################

❸ `insert`############# SQL # `INSERT` ####################

❹ `update`############# SQL # `UPDATE` ####################

❺ `access` (##### - ###### `property` ): Hibernate ##########################

❻ `lazy` (##### - ###### `false` ): ##################################################################################
(#######################)

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

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

## `<property>` ##################################

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

The `<dynamic-component>` element allows a `Map` to be mapped as a component, where the property names refer to keys of the map. See ########### for more information.

## 5.1.16. #####

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

```
<properties
        name="logicalName"                                          ❶
        insert="true|false"                                         ❷
        update="true|false"                                         ❸
        optimistic-lock="true|false"                                ❹
        unique="true|false"                                         ❺
>

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

❶ name : ###################### ##### #

❷ insert############# SQL # INSERT ####################

❸ update############# SQL # UPDATE ####################

❹ optimistic-lock (##### - ###### true ): ######################################## ############################################

❺ 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 (#####): ###################################
4. lazy (############ true ): lazy="false" ####################

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

For information about inheritance mappings see 9########.

## 5.1.18. joined-subclass

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

```
<joined-subclass
        name="ClassName"                                    1
```

```
        table="tablename"                                                    ❷
        proxy="ProxyInterface"                                               ❸
        lazy="true|false"                                                    ❹
        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
>
```

❶ name####################

❷ table :##############

❸ proxy (######): ###################################

❹ lazy (############ true ): lazy="false" ####################

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

For information about inheritance mappings see *9########*.

## 5.1.19. union-subclass

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

```
<union-subclass
        name="ClassName"                                                    ❶
        table="tablename"                                                   ❷
        proxy="ProxyInterface"                                              ❸
        lazy="true|false"                                                   ❹
        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
>
```

❶ `name`###################

❷ `table` :#############

❸ `proxy` (#####): ##################################

❹ `lazy` (########### `true` ): `lazy="false"` ####################

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

For information about inheritance mappings see *9########*.

## 5.1.20. join

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

```
<join
        table="tablename"                                                  ❶
        schema="owner"                                                     ❷
        catalog="catalog"                                                  ❸
        fetch="join|select"                                                ❹
        inverse="true|false"                                               ❺
        optional="true|false">                                             ❻

        <key ... />

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

❶ table :###########

❷ schema (#####): #### <hibernate-mapping> ########################

❸ catalog (#####): #### <hibernate-mapping> ########################

❹ fetch (##### - ###### join ): join ######## Hibernate ########################## <join> ########################## <join> ################## select ########## Hibernate ############ <join> ############################################ ##################################### <join> ##############

❺ inverse (##### - ###### false ): ######### Hibernate ########################## #######

❻ 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"                                                  ❶
        on-delete="noaction|cascade"                                         ❷
        property-ref="propertyName"                                          ❸
        not-null="true|false"                                                ❹
        update="true|false"                                                  ❺
        unique="true|false"                                                  ❻
/>
```

❶ column (#####):################ <column> ###############

❷ on-delete (#####, ###### noaction): ######################################

❸ property-ref (#####): ################################## (################ #)#

❹ not-null (#####): ######## null ############### (############################ #)#

❺ update (#####): ######################### (##############################)#

❻ 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"
        read="SQL expression"
        write="SQL expression"/>
```

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

Most of the attributes on `column` provide a means of tailoring the DDL during automatic schema generation. The `read` and `write` attributes allow you to specify custom SQL that Hibernate will use to access the column's value. For more on this, see the discussion of *column read and write expressions*.

The `column` and `formula` elements can even be combined within the same property or association mapping to express, for example, exotic join conditions.

```
<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"                                                    ❶
        rename="ShortName"                                                   ❷
/>
```

❶ `class`# Java ################

❷ 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"                                    ❶
        id-type="idtypename"                                   ❷
        meta-type="metatypename"                               ❸
        cascade="cascade_style"                                ❹
        access="field|property|ClassName"                      ❺
        optimistic-lock="true|false"                           ❻
>
        <meta-value ... />
        <meta-value ... />
        .....
        <column .... />
        <column .... />
        .....
</any
>
```

❶ name# #######

❷ id-type# ######

❸ meta-type###### - ###### string ## ####################

❹ cascade###### - ###### none ## ###########

❺ access (##### - ###### property ): Hibernate ###########################

❻ optimistic-lock (##### - ###### true ): ######################################## ###########################################

# 5.2. Hibernate ##

## 5.2.1. ########

In relation to the persistence service, Java language-level objects are classified into two groups:

###### ########################################################################## ###### Java ######################################## ##### ######################################################################## ODMG ################################################################################ ###

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

#################################################################################### ####################### ####### ########################### java.lang.String ################################### JDK ############ Java ## (## #) ################################################################################ ############################################################################ ###########

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

Java ###### (####################) # SQL /############################## Hibernate ############################## <class> # <subclass> ################ <property> # <component> ###### type ############### Hibernate # ###### ###### Hibernate # (## JDK #######) ##################################################################

############## Hibernate ####### null ################

## 5.2.2. #####

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

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

Even though Hibernate's rich range of built-in types and support for components means you will rarely need to use a custom type, it is considered good practice to use custom types for non-entity classes that occur frequently in your application. For example, a MonetaryAmount class is a

good candidate for a `CompositeUserType`, even though it could be mapped as a component. One reason for this is abstraction. With a custom type, your mapping documents would be protected against changes to the way monetary values are represented.

## 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) ################## Bean ################ #### JDK5.0 ################ Hibernate3 ## JSR-220 (### API) # `EntityManager` ####### ########################## *Hibernate Annotations* ################## EJB3 (JSR-220) # Hibernate3 ####################

### EJB ####### Bean ######## 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 ########

Properties marked as generated must additionally be non-insertable and non-updateable. Only *versions*, *timestamps*, and *simple properties*, can be marked as generated.

`never` (#####) - ####################################

`insert`: the given property value is generated on insert, but is not regenerated on subsequent updates. Properties like created-date fall into this category. Even though *version* and *timestamp* properties can be marked as generated, this option is not available.

`always` - ##############################

## 5.7. Column read and write expressions

Hibernate allows you to customize the SQL it uses to read and write the values of columns mapped to *simple properties*. For example, if your database provides a set of data encryption functions, you can invoke them for individual columns like this:

```
<property name="creditCardNumber">
        <column
          name="credit_card_num"
          read="decrypt(credit_card_num)"
          write="encrypt(?)"/>
</property
>
```

Hibernate applies the custom expressions automatically whenever the property is referenced in a query. This functionality is similar to a derived-property `formula` with two differences:

- The property is backed by one or more columns that are exported as part of automatic schema generation.
- The property is read-write, not read-only.

The `write` expression, if specified, must contain exactly one '?' placeholder for the value.

## 5.8. ################

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.Oracle9iDialect"/>
        <dialect-scope name="org.hibernate.dialect.Oracle10gDialect"/>
    </database-object>
</hibernate-mapping
>
```

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

## 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"                                                    ❶
    table="table_name"                                                     ❷
    schema="schema_name"                                                   ❸
    lazy="true|extra|false"                                                ❹
    inverse="true|false"                                                   ❺
    cascade="all|none|save-update|delete|all-delete-orphan|delete-orphan"  ❻
    sort="unsorted|natural|comparatorClass"                                ❼
    order-by="column_name asc|desc"                                        ❽
    where="arbitrary sql where condition"                                  ❾
    fetch="join|select|subselect"                                          ❿
    batch-size="N"                                                         ⓫
    access="field|property|ClassName"                                      ⓬
    optimistic-lock="true|false"                                           ⓭
    mutable="true|false"                                                   ⓮
    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"                                                    ❶
        base="0|1|..."/>
```

❶ `column_name` (required): the name of the column holding the collection index values.

❶ `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"                                                    ❶
        formula="any SQL expression"                                            ❷
        type="type_name"                                                        ❸
        node="@attribute-name"
        length="N"/>
```

❶ `column` (optional): the name of the column holding the collection index values.

❷ `formula` (optional): a SQL formula used to evaluate the key of the map.

❸ `type` (required): the type of the map keys.

```
<map-key-many-to-many
        column="column_name"                                                    ❶
        formula="any SQL expression"                                            ❷❸
        class="ClassName"
```

```
/>
```

1. `column` (optional): the name of the foreign key column for the collection index values.
2. `formula` (optional): a SQ formula used to evaluate the foreign key of the map key.
3. `class` (required): the entity class used as the map key.

If your table does not have an index column, and you still wish to use `List` as the property type, you can map the property as a Hibernate <*bag*>. A bag does not retain its order when it is retrieved from the database, but it can be optionally sorted or ordered.

### 6.2.4. ###############

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

############## `<element>` #########

```
<element
        column="column_name"                                    (1)
        formula="any SQ 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 SQ 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): enables 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.

Here are some examples.

A set of strings:

```
<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"                                    ❶
        not-found="ignore|exception"                         ❷
        entity-name="EntityName"                             ❸
        node="element-name"
        embed-xml="true|false"
```

```
    />
```

1. `class` ####: #########
2. `not-found` ###### - ###### `exception`#: ##################################: `ignore` ##############################
3. `entity-name` #######: `class` ##################### `class` ######################## ##

`<one-to-many>` ################################ #### ###############

##

If the foreign key column of a `<one-to-many>` association is declared `NOT NULL`, you must declare the `<key>` mapping `not-null="true"` or *use a bidirectional association* with the collection mapping marked `inverse="true"`. See the discussion of bidirectional associations later in this chapter for more information.

########`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
>
```

##

The value of the order-by attribute is an SQL ordering, not an HQL ordering.

############ filter() ############### criteria ##################

```
sortedUsers = s.createFilter( group.getUsers(), "order by this.name" ).list();
```

## 6.3.2. #####

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

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
>
```

Note that in this mapping, the collection-valued end of the association is responsible for updates to the foreign key.

### 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. `Using an <idbag>`

The majority of the many-to-many associations and collections of values shown previously all map to tables with composite keys, even though it has been have suggested that entities should have synthetic identifiers (surrogate keys). A pure association table does not seem to benefit much from a surrogate key, although a collection of composite values *might*. It is for this reason that Hibernate provides a feature that allows you to map many-to-many associations and collections of values to a table with a surrogate key.

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. #########

This section covers collection examples.

The following class has a collection of Child instances:

```
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; }

    ....
    ....
}
```

If each child has, at most, one parent, the most natural mapping is a one-to-many association:

```
<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
```

For more examples and a complete explanation of a parent/child relationship mapping, see *22 ### #/##* for more information.

Even more complex association mappings are covered in the next chapter.

# #######

## 7.1. #########

#################################################################################
#################### Person # Address ######

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

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

## 7.2. #####

### 7.2.1. Many-to-one

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

```
<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. One-to-one

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

```
<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. One-to-many

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

```
<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. One-to-many

################## ########## 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. Many-to-one

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

```
<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. One-to-one

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

```
<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. Many-to-many

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

```
<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
>
```

If the underlying foreign key column is `NOT NULL`, it is important that you define `not-null="true"` on the `<key>` element of the collection mapping. Do not only declare `not-null="true"` on a possible nested `<column>` element, but on the `<key>` element.

## 7.4.2. One-to-one

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

```
<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. Many-to-many

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

```
<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 #################

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

####### ##### 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
>
```

> ####
>
> The `column` element is an alternative to the `column` attribute everywhere. Using the `column` element just gives more declaration options, which are mostly useful when utilizing `hbm2ddl`

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

# #######

## 9.1. 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. discriminator #### 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#####################################################################

The limitation of this approach is that if a property is mapped on the superclass, the column name must be the same on all subclass tables. The identity generator strategy is not allowed in union subclass inheritance. The primary key seed has to be shared across all unioned subclasses of a hierarchy.

################### 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> ######################

The following table shows the limitations of table per concrete-class mappings, and of implicit polymorphism, in Hibernate.

### #9.1 ##########

| #### | Polymorp many-to-one | ###### ###### | ###### ###### | ###### ###### | #### ##### load()/get() | ###### ###### ### | ####### ####### | Outer join fetching |
|---|---|---|---|---|---|---|---|---|
| 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 | *supported* |

| #### | Polymorp many-to-one | ###### ###### | ###### ###### | ###### ###### | #### ##### load()/ get() | ###### ###### ### | ####### ####### | Outer join fetching |
|---|---|---|---|---|---|---|---|---|
| table per subclass | <many-to-one> | <one-to-one> | <one-to-many> | <many-to-many> | s.get(Payment.class, id) | from Payment p | select o from Order o join o.payment p | *supported* |
| 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 | select o from Order o join o.payment p | *supported* |
| table per concrete class (implicit polymorphism) | <any> | *not supported* | *not supported* | <many-to-any> | s.createCriteria(Payment.class).add( Restrictions. | from Payment p | *not supported* | *not supported* |

# #########

Hibernate ##########/################################################################ ##### #### ########## JDBC/SQL ######### SQL `statements` ######### Java ######### ####################################

####### Hibernate ######################### ## ################ SQL ############# ################### Hibernate ##########################################

## 10.1. Hibernate #############

Hibernate ############################:

- *Transient* - `new` ###################### Hibernate # `Session` ################# ## transient ############################################ Transient ########## ################################################################ (persistent) ########### Hibernate # `Session` ################### SQL ###### Hibernate #########
- *### (Persistent)* - ################################################################ ################### `Session` ################ Hibernate #######Unit of Work####### ############################################################ transient ########## ###### `UPDATE` ## `DELETE` #########
- *Detached* - detached ############################ `Session` ############## ########################################detached ########################## detached ############################################ `Session` ############# ############################################################ ############# #####*application transactions*# ##################################

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

## 10.2. ##############

Newly instantiated instances of a persistent class are considered *transient* by Hibernate. We can make a transient instance *persistent* by associating it with a session:

```
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()` ############ `cat` ###################### `save()` ###### EJB3 ############# `persist()` ###########

- `persist()` makes a transient instance persistent. However, it does not 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.

Alternatively, you can assign the identifier using an overloaded version of `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)
```

How much does Hibernate load from the database and how many SQL `SELECT`s will it use? This depends on the *fetching strategy*. This is explained in ########.

## 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();
```

The `Criteria` and the associated `Example` API are discussed in more detail in *16#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 queries can contain named and positional parameters, just like Hibernate queries. More information about native SQL queries in Hibernate can be found in *17###### 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 ##### ###########

> ####
>
> Hibernate does not offer its own API for direct execution of UPDATE or DELETE statements. Hibernate is a *state management* service, you do not 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 can, however, provide special mass operation functions. See *14### ###* for some possible batch operation tricks.

## 10.6. detached #########

############################################################### UI ############ ################################################################################ ############################

Hibernate ## Session.update() # Session.merge() ######### detached ################ ############

```
// 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()` ########### detached ################################ ###### `update()` ##############################

The application should individually `update()` detached instances that are reachable from the given detached instance *only* if it wants their state to be updated. This can be automated using *transitive persistence*. See ######### for more information.

`lock()` ######################################## detached ######################

```
//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()` ###############

Other models for long units of work are discussed in ###########.

## 10.7. ########

Hibernate ########2################################################ transient ################################ detached #########/################### `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() ########## transient ################

```
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(..) ##############################

It is possible to change the default behavior so that flush occurs less frequently. The FlushMode class defines three different modes: only flush at commit time when the Hibernate Transaction API is used, flush automatically using the explained routine, or never flush unless flush() is called explicitly. The last mode is useful for long running units of work, where a Session is kept open and disconnected for a long time (see ####################).

```
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();
```

During flush, an exception might occur (e.g. if a DML operation violates a constraint). Since handling exceptions involves some understanding of Hibernate's transactional behavior, we discuss it in *12#Transactions and Concurrency*.

## 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() #######
- transient ### detached ##################### 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] );
    }
}
```

# Read-only entities

####

Hibernate's treatment of *read-only* entities may differ from what you may have encountered elsewhere. Incorrect usage may cause unexpected results.

When an entity is read-only:

- Hibernate does not dirty-check the entity's simple properties or single-ended associations;
- Hibernate will not update simple properties or updatable single-ended associations;
- Hibernate will not update the version of the read-only entity if only simple properties or single-ended updatable associations are changed;

In some ways, Hibernate treats read-only entities the same as entities that are not read-only:

- Hibernate cascades operations to associations as defined in the entity mapping.
- Hibernate updates the version if the entity has a collection with changes that dirties the entity;
- A read-only entity can be deleted.

Even if an entity is not read-only, its collection association can be affected if it contains a read-only entity.

For details about the affect of read-only entities on different property and association types, see *#Read-only affect on property type#*.

For details about how to make entities read-only, see *#Making persistent entities read-only#*

Hibernate does some optimizing for read-only entities:

- It saves execution time by not dirty-checking simple properties or single-ended associations.
- It saves memory by deleting database snapshots.

## 11.1. Making persistent entities read-only

Only persistent entities can be made read-only. Transient and detached entities must be put in persistent state before they can be made read-only.

Hibernate provides the following ways to make persistent entities read-only:

- you can map an entity class as *immutable*; when an entity of an immutable class is made persistent, Hibernate automatically makes it read-only. see *#Entities of immutable classes#* for details
- you can change a default so that entities loaded into the session by Hibernate are automatically made read-only; see *#Loading persistent entities as read-only#* for details
- you can make an HQL query or criteria read-only so that entities loaded when the query or criteria executes, scrolls, or iterates, are automatically made read-only; see *#Loading read-only entities from an HQL query/criteria#* for details
- you can make a persistent entity that is already in the in the session read-only; see *#Making a persistent entity read-only#* for details

### 11.1.1. Entities of immutable classes

When an entity instance of an immutable class is made persistent, Hibernate automatically makes it read-only.

An entity of an immutable class can created and deleted the same as an entity of a mutable class.

Hibernate treats a persistent entity of an immutable class the same way as a read-only persistent entity of a mutable class. The only exception is that Hibernate will not allow an entity of an immutable class to be changed so it is not read-only.

### 11.1.2. Loading persistent entities as read-only

##

Entities of immutable classes are automatically loaded as read-only.

To change the default behavior so Hibernate loads entity instances of mutable classes into the session and automatically makes them read-only, call:

```
Session.setDefaultReadOnly( true );
```

To change the default back so entities loaded by Hibernate are not made read-only, call:

```
Session.setDefaultReadOnly( false );
```

You can determine the current setting by calling:

```
Session.isDefaultReadOnly();
```

If Session.isDefaultReadOnly() returns true, entities loaded by the following are automatically made read-only:

- Session.load()
- Session.get()
- Session.merge()
- executing, scrolling, or iterating HQL queries and criteria; to override this setting for a particular HQL query or criteria see *#Loading read-only entities from an HQL query/criteria#*

Changing this default has no effect on:

- persistent entities already in the session when the default was changed
- persistent entities that are refreshed via Session.refresh(); a refreshed persistent entity will only be read-only if it was read-only before refreshing
- persistent entities added by the application via Session.persist(), Session.save(), and Session.update() Session.saveOrUpdate()

### 11.1.3. Loading read-only entities from an HQL query/criteria

##

Entities of immutable classes are automatically loaded as read-only.

If Session.isDefaultReadOnly() returns false (the default) when an HQL query or criteria executes, then entities and proxies of mutable classes loaded by the query will not be read-only.

You can override this behavior so that entities and proxies loaded by an HQL query or criteria are automatically made read-only.

For an HQL query, call:

```
Query.setReadOnly( true );
```

`Query.setReadOnly( true )` must be called before `Query.list()`, `Query.uniqueResult()`, `Query.scroll()`, or `Query.iterate()`

For an HQL criteria, call:

```
Criteria.setReadOnly( true );
```

`Criteria.setReadOnly( true )` must be called before `Criteria.list()`, `Criteria.uniqueResult()`, or `Criteria.scroll()`

Entities and proxies that exist in the session before being returned by an HQL query or criteria are not affected.

Uninitialized persistent collections returned by the query are not affected. Later, when the collection is initialized, entities loaded into the session will be read-only if Session.isDefaultReadOnly() returns true.

Using `Query.setReadOnly( true )` or `Criteria.setReadOnly( true )` works well when a single HQL query or criteria loads all the entities and intializes all the proxies and collections that the application needs to be read-only.

When it is not possible to load and initialize all necessary entities in a single query or criteria, you can temporarily change the session default to load entities as read-only before the query is executed. Then you can explicitly initialize proxies and collections before restoring the session default.

```
Session session = factory.openSession();
Transaction tx = session.beginTransaction();

setDefaultReadOnly( true );
Contract contract =
   ( Contract ) session.createQuery(
           "from Contract where customerName = 'Sherman'" )
           .uniqueResult();
Hibernate.initialize( contract.getPlan() );
Hibernate.initialize( contract.getVariations() );
Hibernate.initialize( contract.getNotes() );
setDefaultReadOnly( false );
...
tx.commit();
session.close();
```

If Session.isDefaultReadOnly() returns true, then you can use Query.setReadOnly( false ) and Criteria.setReadOnly( false ) to override this session setting and load entities that are not read-only.

### 11.1.4. Making a persistent entity read-only

##

Persistent entities of immutable classes are automatically made read-only.

To make a persistent entity or proxy read-only, call:

```
Session.setReadOnly(entityOrProxy, true)
```

To change a read-only entity or proxy of a mutable class so it is no longer read-only, call:

```
Session.setReadOnly(entityOrProxy, false)
```

####

When a read-only entity or proxy is changed so it is no longer read-only, Hibernate assumes that the current state of the read-only entity is consistent with its database representation. If this is not true, then any non-flushed changes made before or while the entity was read-only, will be ignored.

To throw away non-flushed changes and make the persistent entity consistent with its database representation, call:

```
session.refresh( entity );
```

To flush changes made before or while the entity was read-only and make the database representation consistent with the current state of the persistent entity:

```
// evict the read-only entity so it is detached
session.evict( entity );

// make the detached entity (with the non-flushed changes) persistent
session.update( entity );

// now entity is no longer read-only and its changes can be flushed
s.flush();
```

## 11.2. Read-only affect on property type

The following table summarizes how different property types are affected by making an entity read-only.

### #11.1 Affect of read-only entity on property types

| Property/Association Type | Changes flushed to DB? |
|---|---|
| Simple | no* |

| Property/Association Type | Changes flushed to DB? |
|---|---|
| (*#Simple properties#*) | |
| Unidirectional one-to-one<br><br>Unidirectional many-to-one<br><br>(*#Unidirectional one-to-one and many-to-one#*) | no*<br><br>no* |
| Unidirectional one-to-many<br><br>Unidirectional many-to-many<br><br>(*#Unidirectional one-to-many and many-to-many#*) | yes<br><br>yes |
| Bidirectional one-to-one<br><br>(*#Bidirectional one-to-one#*) | only if the owning entity is not read-only* |
| Bidirectional one-to-many/many-to-one<br><br>inverse collection<br><br>non-inverse collection<br><br>(*#Bidirectional one-to-many/many-to-one#*) | only added/removed entities that are not read-only*<br><br>yes |
| Bidirectional many-to-many<br><br>(*#Bidirectional many-to-many#*) | yes |

* Behavior is different when the entity having the property/association is read-only, compared to when it is not read-only.

## 11.2.1. Simple properties

When a persistent object is read-only, Hibernate does not dirty-check simple properties.

Hibernate will not synchronize simple property state changes to the database. If you have automatic versioning, Hibernate will not increment the version if any simple properties change.

```
Session session = factory.openSession();
Transaction tx = session.beginTransaction();

// get a contract and make it read-only
Contract contract = ( Contract ) session.get( Contract.class, contractId );
session.setReadOnly( contract, true );

// contract.getCustomerName() is "Sherman"
contract.setCustomerName( "Yogi" );
tx.commit();
```

```
tx = session.beginTransaction();

contract = ( Contract ) session.get( Contract.class, contractId );
// contract.getCustomerName() is still "Sherman"
...
tx.commit();
session.close();
```

## 11.2.2. Unidirectional associations

### 11.2.2.1. Unidirectional one-to-one and many-to-one

Hibernate treats unidirectional one-to-one and many-to-one associations in the same way when the owning entity is read-only.

We use the term *unidirectional single-ended association* when referring to functionality that is common to unidirectional one-to-one and many-to-one associations.

Hibernate does not dirty-check unidirectional single-ended associations when the owning entity is read-only.

If you change a read-only entity's reference to a unidirectional single-ended association to null, or to refer to a different entity, that change will not be flushed to the database.

##

If an entity is of an immutable class, then its references to unidirectional single-ended associations must be assigned when that entity is first created. Because the entity is automatically made read-only, these references can not be updated.

If automatic versioning is used, Hibernate will not increment the version due to local changes to unidirectional single-ended associations.

In the following examples, Contract has a unidirectional many-to-one association with Plan. Contract cascades save and update operations to the association.

The following shows that changing a read-only entity's many-to-one association reference to null has no effect on the entity's database representation.

```
// get a contract with an existing plan;
// make the contract read-only and set its plan to null
tx = session.beginTransaction();
Contract contract = ( Contract ) session.get( Contract.class, contractId );
session.setReadOnly( contract, true );
contract.setPlan( null );
tx.commit();

// get the same contract
```

```
tx = session.beginTransaction();
contract = ( Contract ) session.get( Contract.class, contractId );

// contract.getPlan() still refers to the original plan;

tx.commit();
session.close();
```

The following shows that, even though an update to a read-only entity's many-to-one association has no affect on the entity's database representation, flush still cascades the save-update operation to the locally changed association.

```
// get a contract with an existing plan;
// make the contract read-only and change to a new plan
tx = session.beginTransaction();
Contract contract = ( Contract ) session.get( Contract.class, contractId );
session.setReadOnly( contract, true );
Plan newPlan = new Plan( "new plan"
contract.setPlan( newPlan);
tx.commit();

// get the same contract
tx = session.beginTransaction();
contract = ( Contract ) session.get( Contract.class, contractId );
newPlan = ( Contract ) session.get( Plan.class, newPlan.getId() );

// contract.getPlan() still refers to the original plan;
// newPlan is non-null because it was persisted when
// the previous transaction was committed;

tx.commit();
session.close();
```

#### 11.2.2.2. Unidirectional one-to-many and many-to-many

Hibernate treats unidirectional one-to-many and many-to-many associations owned by a read-only entity the same as when owned by an entity that is not read-only.

Hibernate dirty-checks unidirectional one-to-many and many-to-many associations;

The collection can contain entities that are read-only, as well as entities that are not read-only.

Entities can be added and removed from the collection; changes are flushed to the database.

If automatic versioning is used, Hibernate will update the version due to changes in the collection if they dirty the owning entity.

### 11.2.3. Bidirectional associations

#### 11.2.3.1. Bidirectional one-to-one

If a read-only entity owns a bidirectional one-to-one association:

- Hibernate does not dirty-check the association.
- updates that change the association reference to null or to refer to a different entity will not be flushed to the database.
- If automatic versioning is used, Hibernate will not increment the version due to local changes to the association.

##

If an entity is of an immutable class, and it owns a bidirectional one-to-one association, then its reference must be assigned when that entity is first created. Because the entity is automatically made read-only, these references cannot be updated.

When the owner is not read-only, Hibernate treats an association with a read-only entity the same as when the association is with an entity that is not read-only.

### 11.2.3.2. Bidirectional one-to-many/many-to-one

A read-only entity has no impact on a bidirectional one-to-many/many-to-one association if:

- the read-only entity is on the one-to-many side using an inverse collection;
- the read-only entity is on the one-to-many side using a non-inverse collection;
- the one-to-many side uses a non-inverse collection that contains the read-only entity

When the one-to-many side uses an inverse collection:

- a read-only entity can only be added to the collection when it is created;
- a read-only entity can only be removed from the collection by an orphan delete or by explicitly deleting the entity.

### 11.2.3.3. Bidirectional many-to-many

Hibernate treats bidirectional many-to-many associations owned by a read-only entity the same as when owned by an entity that is not read-only.

Hibernate dirty-checks bidirectional many-to-many associations.

The collection on either side of the association can contain entities that are read-only, as well as entities that are not read-only.

Entities are added and removed from both sides of the collection; changes are flushed to the database.

If automatic versioning is used, Hibernate will update the version due to changes in both sides of the collection if they dirty the entity owning the respective collections.

# Transactions and Concurrency

Hibernate ################################ Hibernate ######################## JDBC ####### JTA ########### JDBC # ANSI #################DBMS############### ########################

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

############################ `SELECT FOR UPDATE` ############################ API ################### API ####################

####################conversation###################
`Configuration#SessionFactory`#### `Session` ###### Hibernate ##################

## 12.1. session ##### transaction ####

`SessionFactory` ################################################################ ############## `Configuration` ##################

`Session` ##################################################unit of work###### ################## `Session` ######### JDBC `Connection`##### `DataSource`########### ########################

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

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

### 12.1.1. #####Unit of work#

First, let's define a unit of work. A unit of work is a design pattern described by Martin Fowler as # [maintaining] a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems. #[PoEAA] In other words, its a series of operations we wish to carry out against the database together. Basically, it is a transaction, though fulfilling a unit of work will often span multiple physical database transactions (see #### ##). So really we are talking about a more abstract notion of a transaction. The term "business transaction" is also sometimes used in lieu of unit of work.

##### *session-per-operation* ################################################## # `Session` ################################################################ #############planned sequence############################ SQL ########## ############################################ SQL ########################

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

################################################ *session-per-request* ################################ Hibernate ############################## Hibernate `Session` ################################################################ session ######################################################## `Session` ##################################################################

################## Hibernate ######################## "current session" ############################################################################################ ################### `ServletFilter` ################### AOP ######### proxy/interception ####### EJB ##### EJB ##### Bean ######################################## #### CMT ################################################################################ #### Hibernate `Transaction` API #########

Your application code can access a "current session" to process the request by calling `sessionFactory.getCurrentSession()`. You will always get a `Session` scoped to the current database transaction. This has to be configured for either resource-local or JTA environments, see ###############.

################## ##### ################################################################################################################################ ############################## EJB ######################################## EJB # ################################### *Open Session in View* ################### Hibernate # Web ###################

## 12.1.2. ####

session-per-request ################################################################################################################ Web ######################################################################################:

- ################## `Session` ################################################################################
- 5######## "Save" ################################################################################

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

######################## `Session` ################################################################################################################################

##############################################################################
##############################################################################
##############################################################################
################# Hibernate ##################:

- ######### - Hibernate ##########################################################
#####################
- *###Detached####### - ####### session-per-request ################################ ################################################################ session-per-request-with-detached-objects ##########################################*
- ################ - Hibernate # `Session` ################################## # JDBC ################################################### *session-per-conversation* ################################################################ ############ `Session` ###########################

*session-per-request-with-detached-objects # session-per-conversation* ####################
#####################################

### 12.1.3. #############

################ `Session` ############################ `Session` ################
##################################################

#########

```
foo.getId().equals( bar.getId() )
```

JVM ###

```
foo==bar
```

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

########## Hibernate ##########################################################
######################################### `Session` ############################
########## synchronize ########### `Session` ############################ == ######
#####

##### `Session` ### == ##############################################
##################### `Set` # put ######################################
########################## JVM ########################### `equals()` #
`hashCode()` ##############################################################
##############################################################################
############################### `Set` ########################### `Set` ############

################################################ `Set` ######################## ############ Hibernate # Web ################### Hibernate ########## Java ####### ############################

### 12.1.4. ######

*session-per-user-session # session-per-application ##################################### ##############################################################*

- `Session` ################ HTTP ########### Bean # Swing ##################### `Session` ################################### `HttpSession` ### Hibernate `Session` ## ###### HttpSession ############################################################ ############ `Session` #######
- Hibernate ###################################### `Session` ######## ############ # `Session` ############################################################################ ############################################################
- The `Session` caches every object that is in a persistent state (watched and checked for dirty state by Hibernate). If you keep it open for a long time or simply load too much data, it will grow endlessly until you get an OutOfMemoryException. One solution is to call `clear()` and `evict()` to manage the `Session` cache, but you should consider a Stored Procedure if you need mass data operations. Some solutions are shown in *14######*. Keeping a `Session` open for the duration of a user session also means a higher probability of stale data.

## 12.2. ################

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

J2EE ############# ################# Web # Swing ###################### Hibernate ############################### Hiberante ########################################### ################################################################################ ######################################### #CMT# ################# Bean ######## ############################################################

############################ JTA ######### #CMT #### BMT# ###################### ############################################ `Transaction` ####### API # Hibernate # ######## API ########### CMT ###### Bean ########################

### `Session` ###################:

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

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

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

### 12.2.1. ##########

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();
}
```

You do not have to `flush()` the `Session` explicitly: the call to `commit()` automatically triggers the synchronization depending on the *FlushMode* for the session. A call to `close()` marks the end of a session. The main implication of `close()` is that the JDBC connection will be relinquished by the session. This Java code is portable and runs in both non-managed and JTA environments.

############# 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" ###########

## 12.2.2. 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
}
```

CMT ################### Bean ################################################ #####:

```
// CMT idiom
 Session sess = factory.getCurrentSession();

 // do some work
 ...
```

CMT/EJB ################################### Bean ################### `RuntimeException` ######################################## #### *BMT* #### *CMT* ## ## *Hibernate* `Transaction` *API* ################################################ ######

Hibernate ###################### JTA ##### #BMT## ### `org.hibernate.transaction.JTATransactionFactory` ## CMT ##### Bean #### `org.hibernate.transaction.CMTTransactionFactory` ################## #### `hibernate.transaction.manager_lookup_class` ###################### `hibernate.current_session_context_class` ################ `"jta"` ###########

`getCurrentSession()` ######### JTA ######################### `after_statement` ## ############################## JTA ############# `scroll()` ### `iterate()` ####### ###### `ScrollableResults` ### `Iterator` ####### Hibernate ######################## `finally` ######## `ScrollableResults.close()` ### `Hibernate.close(Iterator)` ####### ###################### ########### ################### JTA # CMT #### `scroll()` # `iterate()` ###############

## 12.2.3. ########

`Session` ### #`SQLException`#### ################################## `Session.close()` #### `Session` ############### `Session` ###################### ###### ##### Hibernate ############################## `finally` ####### `close()` ##### `Session` ############

`HibernateException` ## Hibernate ################################## # Hibernate ### ############ ################################################################ ######################################### ############ ######################### ############# ################### # Hibernate ##`HibernateException` ############# ##############################################

Hibernate ################### `SQLException` # `JDBCException` ################### ## `JDBCException` ################### `SQLException` ## `JDBCException.getCause()` ############# Hibernate ## `SessionFactory` ######## `SQLExceptionConverter` #### `SQLException` #### `JDBCException` ################### `SQLExceptionConverter` ##### ### SQL ################################## ##### `SQLExceptionConverterFactory` # ### Javadoc ############### `JDBCException` ###############

- `JDBCConnectionException` - ##### JDBC ############
- `SQLGrammarException` - #### SQL ##################
- `ConstraintViolationException` - ####################
- `LockAcquisitionException` - ######################################
- `GenericJDBCException` - #######################

### 12.2.4. ###############

EJB ############################################################################ ################################################################################ ## #JTA# ##### Hibernate ######################## Hibernate ######################## ################################################################################ Hibernate ################ JTA ############# Hibernate # `Transaction` ############# #######

```
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 Bean #### `setTimeout()` ################################################

## 12.3. #########

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

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

## 12.3.1. ####################

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

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

#################################################################################
############################################ Hibernate ############### Session ##
#############################

## 12.3.2. ##################

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

## 12.3.3. ########################

### 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();
```

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

############################## update() ###### LockMode.READ ##### lock() ######## #### ################################

### 12.3.4. ################

###### optimistic-lock ### false ################################################ ##########################################

################################################################################ ################################################################################ ################################################ <class> ###### optimistic-lock="all" ########################################## Hibernate ################ ################################### session-per-request-with-detached-objects ####### ### Session ########

############################################ <class> ###### optimistic-lock="dirty" ################## Hibernate ###################

############################################################ Hibernate ######## ##1## UPDATE ## #### WHERE ##### ################################################ ################################################################################ ####### *on update* ################## <class> ###### select-before-update="true" # ################################################################ SELECT ####

## 12.4. ######

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

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

## 12.5. ############

Hibernate ######2.x## JDBC ############################ Session ################ ###################################### Hibernate 3.x ######## JDBC ############## ############################################ ConnectionProvider ############## ############################ org.hibernate.ConnectionReleaseMode ############ ####

- ON_CLOSE - ###################### Hibernate ######### JDBC ################### ######################################
- AFTER_TRANSACTION - org.hibernate.Transaction ###################
- AFTER_STATEMENT ############## - ############################################ ################################################################ org.hibernate.ScrollableResults ############

################ hibernate.connection.release_mode ########################## ##########:

- auto ####### - ######## org.hibernate.transaction.TransactionFactory.getDefaultReleaseMode() ######## ########################## JTATransactionFactory ## ConnectionReleaseMode.AFTER_STATEMENT #### 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 ###### ##################################################

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

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

## 13.1. #######

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() );
```

## 13.2. ########

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

## 13.3. 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 ##################

# #####

Hibernate ###########100,000#######################

```
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();
```

###50,000######### `OutOfMemoryException` ########## Hibernate ###################### ######## `Customer` ###################

######################################## JDBC ################################### ############## JDBC #################10##50###########

```
hibernate.jdbc.batch_size 20
```

`identiy` ############Hibernate # JDBC ################################

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

```
hibernate.cache.use_second_level_cache false
```

################################### `CacheMode` ################################### ##

## 14.1. #####

############################################ `flush()` ## `clear()` ###########

```
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();
```

## 14.2. #####

#################################################################################
###### scroll() ###########

```
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();
```

## 14.3. StatelessSession ########

########## Hibernate ######## API ################################################ ########### StatelessSession ################################################ ########################################################################### write-behind ################################################################################ ########################################################### Hibernate ############# ################################################################################ ######## JDBC ############

```
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();
```

################ `Customer` ########## ######### ################################## ##############

`StatelessSession` ################ `insert(), update()` # `delete()` ################ ##################### SQL # `INSERT, UPDATE` ### `DELETE` ######################## `Session` ################ `save(), saveOrUpdate()` # `delete()` ################

## 14.4. DML #######

As already discussed, automatic and transparent object/relational mapping is concerned with the management of the object state. The object state is available in memory. This means that manipulating data directly in the database (using the SQL `Data Manipulation Language` (DML) the statements: `INSERT`, `UPDATE`, `DELETE`) will not affect in-memory state. However, Hibernate provides methods for bulk SQL-style DML statement execution that is performed through the Hibernate Query Language (*HQL*).

`UPDATE` # `DELETE` ######## `( UPDATE | DELETE ) FROM? ######## (WHERE ###)?` ###### ##############

Some points to note:

- from ###### FROM ###############
- from ##########################################################################################################################
- No *joins*, either implicit or explicit, can be specified in a bulk HQL query. Sub-queries can be used in the where-clause, where the subqueries themselves may contain joins.
- where ##########

##### HQL # `UPDATE` ######## `Query.executeUpdate()` ########################## JDBC `PreparedStatement.executeUpdate()` #############

```
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();
```

In keeping with the EJB3 specification, HQL `UPDATE` statements, by default, do not effect the *version* or the *timestamp* property values for the affected entities. However, you can force

Hibernate to reset the `version` or `timestamp` property values through the use of a `versioned update`. This is achieved by adding the `VERSIONED` keyword after the `UPDATE` keyword.

```
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();
```

##########org.hibernate.usertype.UserVersionType## update versioned ########## ###########

HQL # DELETE ########## Query.executeUpdate() ##############

```
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() ####### int ############################################ ######################### HQL ################ SQL ##################### joined-subclass ######################################### joined-subclass ################ ########################################################## joined-subclass ######## ######

INSERT ######## INSERT INTO ############### select # ###################

- INSERT INTO ... SELECT ... ################# INSERT INTO ... VALUES ... ######### #######

  ########## SQL # INSERT ##### ###### ############################################ ################################################################################ INSERT ########################
- select ######## insert ################### select ##### HQL select ############## ################################################ *equal* ##### Hibernate # Type ## *equivalent* ########################### org.hibernate.type.DateType ########### #### org.hibernate.type.TimestampType ######################################## ########################################

- id ########## insert ############################## id ########### ########## select ######################################## (##################)# ########## ########## id ################################################################ ############################## org.hibernate.id.SequenceGenerator ########## ## org.hibernate.id.PostInsertIdentifierGenerator ########################## ## org.hibernate.id.TableHiLoGenerator ######################################## ####
- version # timestamp ####################### insert ################## ################################## select ################################## org.hibernate.type.VersionType ###### #### ###############

HQL # INSERT ########

```
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();
```

# HQL: Hibernate #####

Hibernate # SQL ######## (#######) ###################### SQL ################## HQL ##########################################

## 15.1. ###########

#### Java ################################## `SeLeCT` # `sELEct` ####### `SELECT` ###### `org.hibernate.eg.FOO` # `org.hibernate.eg.Foo` ####### `foo.barSet` # `foo.BARSET` ######

############## HQL ########################################################## Java ##########################

## 15.2. from #

######## Hibernate ##########:

```
from eg.Cat
```

This returns all instances of the class `eg.Cat`. You do not usually need to qualify the class name, since `auto-import` is the default. For example:

```
from Cat
```

In order to refer to the `Cat` in other parts of the query, you will need to assign an *alias*. For example:

```
from Cat as cat
```

######## `Cat` ######## `cat` ########################################### `as` ######## ########################

```
from Cat cat
```

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

```
from Formula, Parameter
```

```
from Formula as form, Parameter as param
```

######## Java ############################################## (### `domesticCat`)#

## 15.3. #####

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

```
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
```

A "fetch" join allows associations or collections of values to be initialized along with their parent objects using a single select. This is particularly useful in the case of a collection. It effectively overrides the outer join and lazy declarations of the mapping file for associations and collections. See ######## for more information.

```
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%'
```

## 15.4. #######

HQL ################### ### # ####

######################### #### #### from ###### join ###########################

### ###### join ################################### ### ######### HQL ####### ### ####### SQL #################

```
from Cat as cat where cat.mate.name like '%s%'
```

## 15.5. ###########

#####################2###########:

- ######## (###) `id` ## *id* ################################## ###################### #########
- ##############################################

######################################## id ######################################## ############################## `id` ##############################

> ####
>
> ##: ######### 3.2.2 ###################### `id` ############### ## ###### ################## `id` ################### Hibernate ################

## 15.6. 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
```

#### Family ############################

```
select new Family(mother, mate, offspr)
from DomesticCat as mother
    join mother.mate as mate
    left join mother.kittens as offspr
```

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 ######

## 15.7. ####

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
```

## 15.8. ################

#########

```
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
```

###2#######2#### SQL `SELECT` ######################### `order by` ################## ############ (######## `Query.scroll()` ################################)#

## 15.9. where #

`where` ######################################################

```
from Cat where name='Fritz'
```

######################:

```
from Cat as cat where cat.name='Fritz'
```

### 'Fritz' ### Cat ##############

The following query:

```
select foo
from Foo foo, Bar bar
where foo.startDate = bar.date
```

## HQL ## Foo # startDate ######### date ######### bar ################# Foo ###### ################# 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
```

The special property (lowercase) id can be used to reference the unique identifier of an object. See ############## for more information.

```
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########################

See ############## for more information regarding referencing identifier properties)

### class ##################################### discriminator ########## where ######## Java ######## discriminator #########

```
from Cat cat where cat.class = DomesticCat
```

You can also use components or composite user types, or properties of said component types. See ######### for more information.

"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 ################################################

## 15.10. Expressions #

Expressions used in the where clause include the following:

- ######+, -, *, /
- 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()`, `mod()`
- `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
    )
```

## 15.11. order by #

###### list ######################################

```
from DomesticCat cat
order by cat.name asc, cat.weight desc, cat.birthdate
```

###### asc # desc ####################

## 15.12. 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
```

group by ## order by ############################### Hibernate ################## ########################## cat ################## group by cat ################## ###########################

## 15.13. ######

######################### 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 ############# for more information.

## 15.14. 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
```

## 15.15. ### UPDATE # DELETE

HQL now supports `update`, `delete` and `insert ... select ...` statements. See *#DML ### #####* for more information.

## 15.16. 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
    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();
```

## 15.17. #######

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

```
select p.name from Person p
```

```
select p.name.first from Person p
```

############################## `where` #########:

```
from Person p where p.name = :name
```

```
from Person p where p.name.first = :firstName
```

######## `order by` #########:

```
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*.

## 15.18. ############

############## ANSI SQL `row value constructor` ## (`tuple` ###############) ######### ###### HQL ############################################################################ #### Person ##############:

```
from Person p where p.name.first='John' and p.name.last='Jingleheimer-Schmidt'
```

############################## `row value constructor` ###############:

```
from Person p where p.name=('John', 'Jingleheimer-Schmidt')
```

### `select` ###############

```
select p.name from Person p
```

## row value constructor ##########:

```
from Cat as cat
where not ( cat.name, cat.color ) in (
    select cat.name, cat.color from DomesticCat cat
)
```

##########

# Criteria ###

Hibernate ############ criteria ### API ##########

## 16.1. `Criteria` ##########

`org.hibernate.Criteria` ################################ `Session` # `Criteria` ##### ##########

```
Criteria crit = sess.createCriteria(Cat.class);
crit.setMaxResults(50);
List cats = crit.list();
```

## 16.2. ############

`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();
```

## 16.3. #####

`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();
```

## 16.4. ##

By navigating associations using `createCriteria()` you can specify constraints upon related entities:

```
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");
}
```

Additionally you may manipulate the result set using a left outer join:

```
List cats = session.createCriteria( Cat.class )
    .createAlias("mate", "mt", Criteria.LEFT_JOIN, Restrictions.like("mt.name",
 "good%") )
       .addOrder(Order.asc("mt.age"))
       .list();
```

This will return all of the `Cats` with a mate whose name starts with "good" ordered by their mate's age, and all cats who do not have a mate. This is useful when there is a need to order or limit

in the database prior to returning complex/large result sets, and removes many instances where multiple queries would have to be performed and the results unioned by java in memory.

Without this feature, first all of the cats without a mate would need to be loaded in one query.

A second query would need to retreive the cats with mates who's name started with "good" sorted by the mates age.

Thirdly, in memory; the lists would need to be joined manually.

## 16.5. #########

`setFetchMode()` ##########################

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "Fritz%") )
    .setFetchMode("mate", FetchMode.EAGER)
    .setFetchMode("kittens", FetchMode.EAGER)
    .list();
```

This query will fetch both `mate` and `kittens` by outer join. See ######## for more information.

## 16.6. #####

`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();
```

## 16.7. ##########

`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` ########

An alias can be assigned to a projection so that the projected value can be referred to in restrictions or orderings. Here are two different ways to do this:

```
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();
```

## 16.8. ###############

`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();
```

## 16.9. ###########

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
>
```

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

#### Restrictions.naturalId() ##############################

```
session.createCriteria(User.class)
    .add( Restrictions.naturalId()
        .set("name", "gavin")
        .set("org", "hb")
    ).setCacheable(true)
    .uniqueResult();
```

# ##### SQL

############ SQL ############################## Oracle # `CONNECT` ################ ################## SQL/JDBC ################## Hibernate ##############

Hibernate3 ########################### SQL ########################

## 17.1. Using a `SQLQuery`

###### SQL ####### `SQLQuery` ################## `SQLQuery` ######### `Session.createSQLQuery()` ############## API ######################

### 17.1.1. ########

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

##### Object ######### `ResultSetMetdata` ################################ ID#NAME#BIRTHDATE ######## Long#String#Short ################################ ############## * ##############################

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

```
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` ############ #####

## 17.1.2. ##########

############################################################ `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() ##########

## 17.1.3. ############

########################### `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 ############################################ ###########################################

### 17.1.4. ###########

############################################################################ ################# 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)
```

################## Cat ############################################################ ######################################## "c.ID"#"c.NAME" ######################## #"ID" # "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 ##########################
- #################

######### {cat.*} # {mother.*} ############################################################ ##### Hibernate ############ SQL ############################################################ ############################### cat_log ## ################ Cat ################## ## where ####################

```
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()
```

### 17.1.4.1. ###############

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

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

#### #17.1 #########

| ## | ## | # |
|---|---|---|
| ######## | {[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.*} |
| All properties of the collection | {[aliasname].*} | {coll.*} |

### 17.1.5. ##################

##### SQL #### ResultTransformer ########################################

```
sess.createSQLQuery("SELECT NAME, BIRTHDATE FROM CATS")
        .setResultTransformer(Transformers.aliasToBean(CatDTO.class))
```

#######################:

- SQL ######
- #########

################## NAME # BIRTHDATE ########################## `CatDTO` ########
##

### 17.1.6. #####

################################## SQL ######################################
########

### 17.1.7. #####

##### 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();
```

## 17.2. #### 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();
```

The `<return-join>` element is use to join associations and the `<load-collection>` element is used to define queries which initialize collections,

```
<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
>
```

################ <resultset> ################################ 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();
```

### 17.2.1. ################## 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> ####
#########

## 17.2.2. ######################

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> ###############`

### 17.2.2.1. ######################

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` ##############################################

## 17.3. ################ SQL

Hibernate3 can use custom SQL for create, update, and delete operations. The SQL can be overridden at the statement level or inidividual column level. This section describes statement overrides. For columns, see *#Column read and write expressions#*.

The class and collection persisters in Hibernate already contain a set of configuration time generated strings (insertsql, deletesql, updatesql etc.). The mapping tags `<sql-insert>`, `<sql-delete>`, and `<sql-update>` override these strings:

```
<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;
```

## 17.4. ########### SQL

You can also declare your own SQL (or HQL) queries for entity loading. As with inserts, updates, and deletes, this can be done at the individual column level as described in *#Column read and write expressions#* or at the statement level. Here is an example of a statement level override:

```
<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
>
```

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

Hibernate3 ######################################## *Hibernate filter* ################ ################### Hibernate #######################

## 18.1. 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
>
```

`######## (##########) ###############`

`Session ####### enableFilter(String filterName)# getEnabledFilter(String filterName)# disableFilter(String filterName) ################################ ##### # Filter ############# Session.enabledFilter() ############################ ################################`

```
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 ##########################################################################
#########################################

##############################################################################
############## <filter-def/> ####### CDATA ################################

```
<filter-def name="myFilter" condition="abc
> xyz"
>...</filter-def>
<filter-def name="myOtherFilter"
>abc=xyz</filter-def
>
```

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

# XML #####

*XML Mapping is an experimental feature in Hibernate 3.0 and is currently under active development.*

## 19.1. XML #######

Hibernate ###### POJO ########################### XML ################### XML # ### POJO ########################################

Hibernate # XML ########### API ### dom4j ################### dom4j ############### ######################################### XML ################## dom4j ########## ###### Hibernate ################################ #### `persist(), saveOrUpdate(), merge(), delete(), replicate()` #### (################)#

#########/####### JMS ################## SOAP # XSLT ########################## ###

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

### 19.1.1. 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
>
```

### 19.1.2. 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 ####################

## 19.2. 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
>
```

## 19.3. 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()` ########################

# ##########

## 20.1. ######

###### ############################### 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` ################################################ ####

### 20.1.1. #######

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

## 20.1.2. ##############

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

### 20.1.3. #########

############### 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
>
```

Then proxies for instances of `Cat` and `DomesticCat` can be returned by `load()` or `iterate()`.

```
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();
```

**Note**

`list()` does not usually return proxies.

####################### `Cat` ####################### `CatImpl` ########

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

- `equals()` ####### `equals()` ##############
- `hashCode()` ####### `hashCode()` ##############
- #### getter ####

Hibernate # `equals()` # `hashCode()` #####################

###### `lazy="proxy"` ###### `lazy="no-proxy"` ##################################################################

## 20.1.4. ###############

`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();
```

## 20.1.5. ##########

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 ###

######################################## `Person` # `Cat` ################ 10 ## Person
# `Sesssion` ################ Person ######## `getCats()` ##########10## `SELECT` #####
#### `Person` ####### `cats` ######################## Hibernate ####################

```
<class name="Person">
    <set name="cats" batch-size="3">
        ...
    </set>
</class
>
```

`batch-size` # 3 #### Hibernate # 4 ## `SELECT` # 3 ## 3 ## 3 ## 1 ######################
`Session` ##############################

##############################################################################
##### *set* # ####### ##############

## 20.1.6. #############

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

### 20.1.7. Fetch profiles

Another way to affect the fetching strategy for loading associated objects is through something called a fetch profile, which is a named configuration associated with the `org.hibernate.SessionFactory` but enabled, by name, on the `org.hibernate.Session`. Once enabled on a `org.hibernate.Session`, the fetch profile wull be in affect for that `org.hibernate.Session` until it is explicitly disabled.

So what does that mean? Well lets explain that by way of an example. Say we have the following mappings:

```
<hibernate-mapping>
    <class name="Customer">
        ...
        <set name="orders" inverse="true">
            <key column="cust_id"/>
            <one-to-many class="Order"/>
        </set>
    </class>
    <class name="Order">
        ...
    </class>
</hibernate-mapping
>
```

Now normally when you get a reference to a particular customer, that customer's set of orders will be lazy meaning we will not yet have loaded those orders from the database. Normally this is a good thing. Now lets say that you have a certain use case where it is more efficient to load the customer and their orders together. One way certainly is to use "dynamic fetching" strategies via an HQL or criteria queries. But another option is to use a fetch profile to achieve that. Just add the following to your mapping:

```
<hibernate-mapping>
    ...
    <fetch-profile name="customer-with-orders">
        <fetch entity="Customer" association="orders" style="join"/>
    </fetch-profile>
</hibernate-mapping
>
```

or even:

```
<hibernate-mapping>
    <class name="Customer">
        ...
        <fetch-profile name="customer-with-orders">
            <fetch association="orders" style="join"/>
        </fetch-profile>
```

```
    </class>
    ...
</hibernate-mapping
>
```

Now the following code will actually load both the customer *and their orders*:

```
    Session session = ...;
    session.enableFetchProfile( "customer-with-orders" );  // name matches from mapping
    Customer customer = (Customer) session.get( Customer.class, customerId );
```

Currently only join style fetch profiles are supported, but they plan is to support additional styles. See *HHH-3414* [http://opensource.atlassian.com/projects/hibernate/browse/HHH-3414] for details.

## 20.1.8. ###############

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
>
```

########################################################## HQL # Criteria ##########
############################################

HQL # `fetch all properties` ########################################

## 20.2. #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#########
#####

### #20.1 ##########

| ##### | ######## | ### | ####### | ########## |
|---|---|---|---|---|
| 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.jbc.JBossCacheRegionFactory` | ######ip ########## ######## | yes (replication or invalidation) | yes########### |

### 20.2.1. ############

################## `<cache>` ############

```
<cache
    usage="transactional|read-write|nonstrict-read-write|read-only"   ❶
    region="RegionName"                                                ❷
    include="all|non-lazy"                                             ❸
/>
```

❶ usage (##) ############### transactional# read-write# nonstrict-read-write ## # read-only

❷ region (################################) 2####################

❸ include (###### all #########) non-lazy ## ###### lazy ############## lazy="true" ###########################################

################ hibernate.cfg.xml # <class-cache> # <collection-cache> ######## #######

usage ### ########### #######

## 20.2.2. read only ##

############################################ read-only ########################## ####################################

```
<class name="eg.Immutable" mutable="false">
    <cache usage="read-only"/>
    ....
</class
>
```

## 20.2.3. 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
```

```
>
```

### 20.2.4. ###### read/write ##

########################################################################## ########################## nonstrict-read-write ########################## JTA ########## hibernate.transaction.manager_lookup_class ########################## Session.close() # Session.disconnect() ###################################

### 20.2.5. transactional ##

transactional ######## JBoss TreeCache ######################################## ######### JTA ########### hibernate.transaction.manager_lookup_class ########## ####

### 20.2.6. Cache-provider/concurrency-strategy compatibility

####

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

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

**#20.2 ##################**

| ##### | 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 | | |

## 20.3. ########

####### 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
```

## 20.4. ########

Query result sets can also be cached. This is only useful for queries that are run frequently with the same parameters.

### 20.4.1. Enabling query caching

Caching of query results introduces some overhead in terms of your applications normal transactional processing. For example, if you cache results of a query against Person Hibernate will need to keep track of when those results should be invalidated because changes have been committed against Person. That, coupled with the fact that most applications simply gain no benefit from caching query results, leads Hibernate to disable caching of query results by default. To use query caching, you will first need to enable the query cache:

```
hibernate.cache.use_query_cache true
```

This setting creates two new cache regions:

- `org.hibernate.cache.StandardQueryCache`, holding the cached query results
- `org.hibernate.cache.UpdateTimestampsCache`, holding timestamps of the most recent updates to queryable tables. These are used to validate the results as they are served from the query cache.

#### ####

If you configure your underlying cache implementation to use expiry or timeouts is very important that the cache timeout of the underlying cache region for the UpdateTimestampsCache be set to a higher value than the timeouts of any of the query caches. In fact, we recommend that the the UpdateTimestampsCache region not be configured for expiry at all. Note, in particular, that an LRU cache expiry policy is never appropriate.

As mentioned above, most queries do not benefit from caching or their results. So by default, individual queries are not cached even after enabling query caching. To enable results caching for a particular query, call `org.hibernate.Query.setCacheable(true)`. This call allows the query to look for existing cache results or add its results to the cache when it is executed.

#### ##

The query cache does not cache the state of the actual entities in the cache; it caches only identifier values and results of value type. For this reaso, the query cache should always be used in conjunction with the second-level cache for those entities expected to be cached as part of a query result cache (just as with collection caching).

### 20.4.2. Query cache regions

############################ `Query.setCacheRegion()` ########################### ##########

```
List blogs = sess.createQuery("from Blog blog where blog.blogger = :blogger")
        .setEntity("blogger", blogger)
        .setMaxResults(15)
        .setCacheable(true)
        .setCacheRegion("frontpages")
        .list();
```

If you want to force the query cache to refresh one of its regions (disregard any cached results it finds there) you can use `org.hibernate.Query.setCacheMode(CacheMode.REFRESH)`. In conjunction with the region you have defined for the given query, Hibernate will selectively force the results cached in that particular region to be refreshed. This is particularly useful in cases where underlying data may have been updated via a separate process and is a far more efficient alternative to bulk eviction of the region via `org.hibernate.SessionFactory.evictQueries()`.

## 20.5. ##################

In the previous sections we have covered collections and their applications. In this section we explore some more issues in relation to collections at runtime.

### 20.5.1. ##

Hibernate #3#######################:

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

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

- ##############
- set
- bag

################### ############ # <key> # <index> ##############################
############################## Hibernate ########################################

set # <key> ################################################################################################################################ SchemaExport #### <set> ###################### not-null="true" ###############

<idbag> ##################################################

bag ########## bag ################################################ Hibernate ##################### Hibernate ################################### DELETE ######## ###################################

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

## 20.5.2. ################# list#map#idbag#set

############################ set ####################################

############################################ set ################### Set ######## ## Hibernate ################### UPDATE ##### Set ############### INSERT # DELETE ### ###################################

#############################list#map#idbag ################inverse ############## ####### set ############### Hibernate ############ set ############################ "set" #######################

############# Hibernate ############################## inverse="true" ########### ###################################################

## 20.5.3. inverse ############## bag # list

bag ########### bag #### list ### set ############################## inverse="true" ########################### 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();
```

## 20.5.4. ####

############################################ Hibernate ########################## list.clear() ###################################### Hibernate # DELETE ######### ###########

###20######################################## Hibernate #### `INSERT` ##### `DELETE` ### #### ######## bag ######################

####18#########2#########3###############################

- 18###########3######
- ############ `DELETE` # SQL #########5##################

Hibernate ######2################################ Hibernate ###################### ########################################

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

########## `inverse="true"` ###################

## 20.6. ###############

######################################## Hibernate ######################## Hibernate ###### `SessionFactory` ##########

### 20.6.1. 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
```

SessionFactory ############################

- ###### hibernate.generate_statistics # false ####

- ##### sf.getStatistics().setStatisticsEnabled(true) ### hibernateStatsBean.setStatisticsEnabled(true) #######

### clear() ############################## logSummary() ######## logger ########## #info #######

## 20.6.2. #####

######################### Statistics ######### API ########3#########:

- ########## Session ############################ JDBC ####################

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

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

########################################################################## Java ##### ################ Hibernate # JVM ######################10###################

### getter ##########################################################################

# #########

Hibernate ###################### Eclipse ###################### Ant ##############

*Hibernate Tools* ########################### Ant ######## Eclipse IDE ###########:

- *#########:* Hibernate # XML ##################################################/ ######################### XML ############
- *Console:* ################################################################## ################### HQL ############################
- ####### Hibernate # Eclipse ############################### Hibernate ####### (cfg.xml) ######################### POJO ######### Hibernate ################### ##############################################################
-

######## *Hibernate Tools* ########################

#### Hibernate ########## *SchemaExport* ### `hbm2ddl` #####(Hibernate ################ #)#

## 21.1. #########

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

DDL ######################### `hibernate.dialect` ###### SQL # ## ### ########## #

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

### 21.1.1. ###########

### Hibernate ################## `length`# `precision`# `scale` ########################## ########################

```
<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
>
```

The following table summarizes these optional attributes.

### #21.1 ##

| ## | # | ## |
|---|---|---|
| length | ## | ###### |
| precision | ## | #### DECIMAL #####precision# |
| scale | ## | #### DECIMAL #######scale# |
| not-null | true\|false | #### null ############## |
| unique | true\|false | ###################### |
| index | index_name | (######)############### |
| unique-key | unique_key_name | ###################### |
| foreign-key | foreign_key_name | <one-to-one># <many-to-one># <key># ## <many-to-many> ###################### ###################### inverse="true" ## SchemaExport ###################### |
| sql-type | SQL column type | ###################### ( <column> ########) |
| 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` ########

## 21.1.2. ######

`SchemaExport` ######### DDL ########### DDL #############

The following table displays the `SchemaExport` command line options

`java -cp` *hibernate_classpaths* `org.hibernate.tool.hbm2ddl.SchemaExport` *options mapping_files*

### #21.2 `SchemaExport` ##############

| ##### | ## |
|---|---|
| `--quiet` | ################## |
| `--drop` | ############### |
| `--create` | ############### |
| `--text` | ################## |
| `--output=my_schema.ddl` | DDL ################## |
| `--naming=eg.MyNamingStrategy` | `NamingStrategy` ### |
| `--config=hibernate.cfg.xml` | XML ###### Hibernate ############ |
| `--properties=hibernate.properties` | ########################## |
| `--format` | ########### SQL ################### |
| `--delimiter=;` | ################## |

######### `SchemaExport` ##############

```
Configuration cfg = ....;
```

```
new SchemaExport(cfg).create(false, true);
```

### 21.1.3. #####

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

- `-D<property>` ################
- `hibernate.properties` ######
- `--properties` ###################

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

#### #21.3 SchemaExport ###########

| ###### | ## |
| --- | --- |
| `hibernate.connection.driver_class` | jdbc ######### |
| `hibernate.connection.url` | jdbc # url |
| `hibernate.connection.username` | ########### |
| `hibernate.connection.password` | ######### |
| `hibernate.dialect` | ######## |

### 21.1.4. 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
>
```

### 21.1.5. ################

`SchemaUpdate` ############################ `SchemaUpdate` # JDBC ###### API ####### ########### JDBC ##############################

`java -cp` *hibernate_classpaths* `org.hibernate.tool.hbm2ddl.SchemaUpdate` *options mapping_files*

### #21.4 `SchemaUpdate` ##############

| ##### | ## |
|---|---|
| `--quiet` | ################# |
| `--text` | ####################### |
| `--naming=eg.MyNamingStrategy` | `NamingStrategy` ### |
| `--properties=hibernate.properties` | ######################### |
| `--config=hibernate.cfg.xml` | `.cfg.xml` ####### |

######### `SchemaUpdate` ##############

```
Configuration cfg = ....;
new SchemaUpdate(cfg).execute(false);
```

## 21.1.6. ##################### 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
>
```

## 21.1.7. Schema validation

`SchemaValidator` ##################################################
`SchemaValidator` # JDBC ###### API ############################## JDBC ###########
##############################

`java -cp` *hibernate_classpaths* `org.hibernate.tool.hbm2ddl.SchemaValidator` *options mapping_files*

The following table displays the `SchemaValidator` command line options:

### #21.5 `SchemaValidator` ##############

| ##### | ## |
|---|---|
| `--naming=eg.MyNamingStrategy` | `NamingStrategy` ### |
| `--properties=hibernate.properties` | ########################## |
| `--config=hibernate.cfg.xml` | `.cfg.xml` ####### |

`SchemaValidator` ######################:

```
Configuration cfg = ....;
new SchemaValidator(cfg).validate();
```

## 21.1.8. ############## 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
>
```

## ## #/##

####### Hibernate ################################################################ ######################################## # ## ## ## <one-to-many> ##### # # ## # ######################################## ## # <composite-element> ################ # Hibernate ################################################################ ################################## ################### ################### ########################## ########

## 22.1. ############

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

- ##################################################################
- ##############################################) ############################ ##################################################################
- ##############################) ############################################ ################################################################################ ################################

################################################################################ ##########################################/######################################## ###

## 22.2. ######

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();
```

## 22.3. ##############

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

## 22.4. ###### unsaved-value

Suppose we loaded up a `Parent` in one `Session`, made some changes in a UI action and wanted to persist these changes in a new session by calling `update()`. The `Parent` will contain a collection of children and, since the cascading update is enabled, Hibernate needs to know which children are newly instantiated and which represent existing rows in the database. We will also assume that both `Parent` and `Child` have generated identifier properties of type `Long`. Hibernate will use the identifier and version/timestamp property value to determine which of the children are new. (See ##########.) *In Hibernate3, it is no longer necessary to specify an `unsaved-value` explicitly.*

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

## 22.5. ##

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

############################## <composite-element> ############################
################################################### 1#########################
###########################################

# #: Weblog ########

## 23.1. #####

############################################################ 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;
    }
}
```

## 23.2. 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
>
```

## 23.3. 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);
```

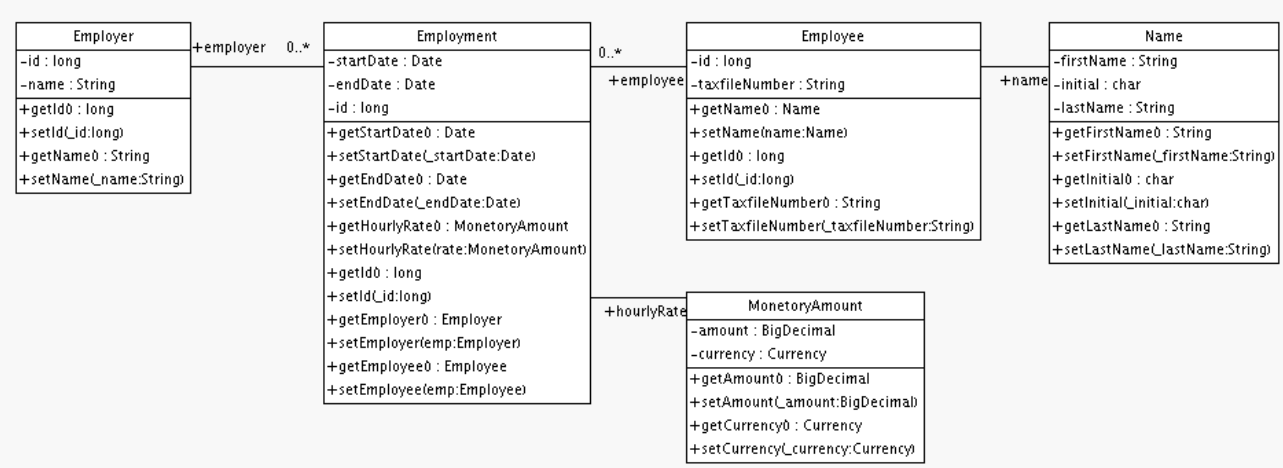
```
            result = q.list();
            tx.commit();
        }
        catch (HibernateException he) {
            if (tx!=null) tx.rollback();
            throw he;
        }
        finally {
            session.close();
        }
        return result;
    }
}
```

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

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

## 24.1. ###/###

`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
>
```

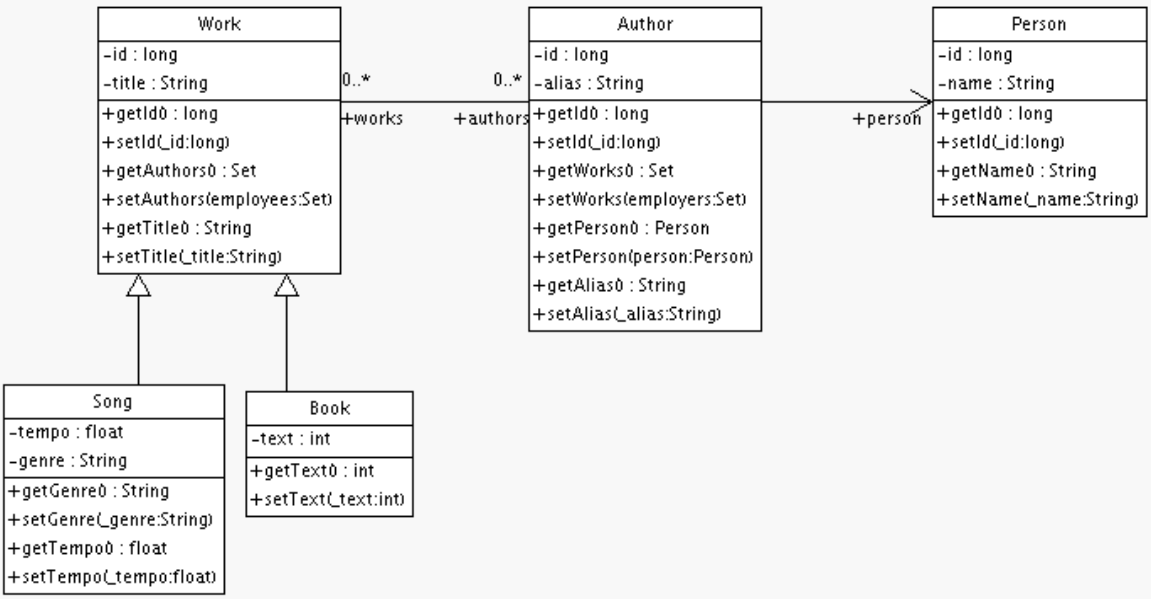
SchemaExport ##################

```
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
```

## 24.2. ##/##

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
>
```

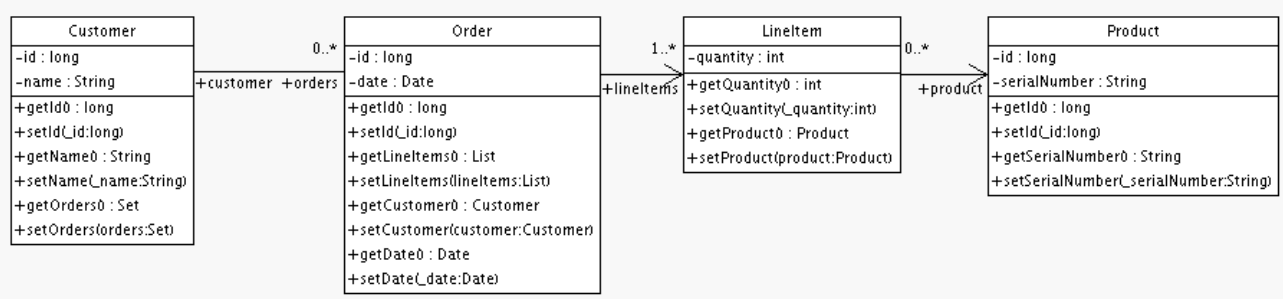
#########4############ works # authors , persons ######################## author_work ######################## SchemaExport ################

```
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
```

## 24.3. ##/##/##

### Customer # Order # LineItem # Product ##################### 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
```

## 24.4. ##########

########## Hibernate ########################################## Hibernate ##### ####### `test` ##########

### 24.4.1. ##############

```
<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
>
```

### 24.4.2. ######

```
<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
>
```

### 24.4.3. ###############

```
<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>
```

### 24.4.4. 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
>
```

### 24.4.5. #######

```
<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
>
```

# #########

```
############ <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 #############
    In performance-critical areas of the system, some kinds of operations might benefit from
    direct JDBC. Do not assume, however, that JDBC is necessarily faster. Please wait until you
    know something is a bottleneck. If you need to use direct JDBC, you can open a Hibernate
    Session, wrap your JDBC operation as a org.hibernate.jdbc.Work object and using that
    JDBC connection. This way you can still use the same transaction strategy and underlying
    connection provider.
```

`Session` ###############

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

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

###### / ##### Bean ###################### / JSP ######## Bean #### ################################### 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 #2######### 1########## Bean ####################2####################################################### DTO ############################ Hibernate ##1##################################################################################################################################### ###### Hibernate ##############################################

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

##########Hibernate ###################### *DAO # Thread Local Session* ########## ##### `UserType` # Hibernate ################ JDBC ##############################################################5##############################

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

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

```
################
    ##########################################################################
```

# Database Portability Considerations

## 26.1. Portability Basics

One of the selling points of Hibernate (and really Object/Relational Mapping as a whole) is the notion of database portability. This could mean an internal IT user migrating from one database vendor to another, or it could mean a framework or deployable application consuming Hibernate to simultaneously target multiple database products by their users. Regardless of the exact scenario, the basic idea is that you want Hibernate to help you run against any number of databases without changes to your code, and ideally without any changes to the mapping metadata.

## 26.2. Dialect

The first line of portability for Hibernate is the dialect, which is a specialization of the `org.hibernate.dialect.Dialect` contract. A dialect encapsulates all the differences in how Hibernate must communicate with a particular database to accomplish some task like getting a sequence value or structuring a SELECT query. Hibernate bundles a wide range of dialects for many of the most popular databases. If you find that your particular database is not among them, it is not terribly difficult to write your own.

## 26.3. Dialect resolution

Originally, Hibernate would always require that users specify which dialect to use. In the case of users looking to simultaneously target multiple databases with their build that was problematic. Generally this required their users to configure the Hibernate dialect or defining their own method of setting that value.

Starting with version 3.2, Hibernate introduced the notion of automatically detecting the dialect to use based on the `java.sql.DatabaseMetaData` obtained from a `java.sql.Connection` to that database. This was much better, expect that this resolution was limited to databases Hibernate know about ahead of time and was in no way configurable or overrideable.

Starting with version 3.3, Hibernate has a fare more powerful way to automatically determine which dialect to should be used by relying on a series of delegates which implement the `org.hibernate.dialect.resolver.DialectResolver` which defines only a single method:

```
public Dialect resolveDialect(DatabaseMetaData metaData) throws JDBCConnectionException
```

. The basic contract here is that if the resolver 'understands' the given database metadata then it returns the corresponding Dialect; if not it returns null and the process continues to the next resolver. The signature also identifies `org.hibernate.exception.JDBCConnectionException` as possibly being thrown. A JDBCConnectionException here is interpreted to imply a "non transient" (aka non-recoverable) connection problem and is used to indicate an immediate stop to resolution attempts. All other exceptions result in a warning and continuing on to the next resolver.

The cool part about these resolvers is that users can also register their own custom resolvers which will be processed ahead of the built-in Hibernate ones. This might be useful in a number of different situations: it allows easy integration for auto-detection of dialects beyond those shipped with HIbernate itself; it allows you to specify to use a custom dialect when a particular database is recognized; etc. To register one or more resolvers, simply specify them (seperated by commas, tabs or spaces) using the 'hibernate.dialect_resolvers' configuration setting (see the `DIALECT_RESOLVERS` constant on `org.hibernate.cfg.Environment`).

## 26.4. Identifier generation

When considering portability between databases, another important decision is selecting the identifier generation stratagy you want to use. Originally Hibernate provided the *native* generator for this purpose, which was intended to select between a *sequence*, *identity*, or *table* strategy depending on the capability of the underlying database. However, an insidious implication of this approach comes about when targtetting some databases which support *identity* generation and some which do not. *identity* generation relies on the SQL definition of an IDENTITY (or auto-increment) column to manage the identifier value; it is what is known as a post-insert generation strategy becauase the insert must actually happen before we can know the identifier value. Because Hibernate relies on this identifier value to uniquely reference entities within a persistence context it must then issue the insert immediately when the users requests the entitiy be associated with the session (like via save() e.g.) regardless of current transactional semantics.

##

Hibernate was changed slightly once the implication of this was better understood so that the insert is delayed in cases where that is feasible.

The underlying issue is that the actual semanctics of the application itself changes in these cases.

Starting with version 3.2.3, Hibernate comes with a set of *enhanced* [http://in.relation.to/2082.lace] identifier generators targetting portability in a much different way.

##

There are specifically 2 bundled *enhanced*generators:

- `org.hibernate.id.enhanced.SequenceStyleGenerator`
- `org.hibernate.id.enhanced.TableGenerator`

The idea behind these generators is to port the actual semantics of the identifer value generation to the different databases. For example, the `org.hibernate.id.enhanced.SequenceStyleGenerator` mimics the behavior of a sequence on databases which do not support sequences by using a table.

## 26.5. Database functions

##

This is an area in Hibernate in need of improvement. In terms of portability concerns, this function handling currently works pretty well from HQL; however, it is quite lacking in all other aspects.

SQL functions can be referenced in many ways by users. However, not all databases support the same set of functions. Hibernate, provides a means of mapping a *logical* function name to a delegate which knows how to render that particular function, perhaps even using a totally different physical function call.

####

Technically this function registration is handled through the `org.hibernate.dialect.function.SQLFunctionRegistry` class which is intended to allow users to provide custom function definitions without having to provide a custom dialect. This specific behavior is not fully completed as of yet.

It is sort of implemented such that users can programatically register functions with the `org.hibernate.cfg.Configuration` and those functions will be recognized for HQL.

## 26.6. Type mappings

This section scheduled for completion at a later date...

# References

[PoEAA] *Patterns of Enterprise Application Architecture*. 0-321-12742-0. # Fowler Martin [FAMILY Given]. #### © 2003 Pearson Education, Inc.. Addison-Wesley Publishing Company.

[JPwH] *Java Persistence with Hibernate*. Second Edition of Hibernate in Action. 1-932394-88-5. *http://www.manning.com/bauer2* . # Bauer Christian [FAMILY Given] # King Gavin [FAMILY Given]. #### © 2007 Manning Publications Co.. Manning Publications Co..