JPA Hints

1. javax.persistence.lock.timeout (Long – milliseconds)

This hint defines the timeout in milliseconds to acquire a pessitimistic lock.

2. javax.persistence.query.timeout (Long – milliseconds)

The *javax.persistence.query.timeout* hint defines how long a query is allowed to run before it gets canceled. Hibernate doesn't handle this timeout itself but provides it to the JDBC driver via the JDBC *Statement.setTimeout* method.

3. javax.persistence.cache.retrieveMode (CacheRetrieveMode – USE | BYPASS)

The *retrieveMode* hint supports the values *USE* and *BYPASS* and tells Hibernate if it shall *USE* the second-level cache to retrieve an entity or if it shall *BYPASS* it and get it directly from the database.

4. javax.persistence.cache.storeMode (CacheStoreMode – USE | BYPASS | REFRESH)

This hint defines how Hibernate shall write changed entities to the second-level cache. It can either *USE* the cache to add entities to the cache and updated existing ones, or *BYPASS* it for entities that are not already stored in the cache and only update the existing ones or *REFRESH* the entities located in the cache before they get retrieved from it.

5. javax.persistence.loadgraph (EntityGraph)

The *javax.persistence.loadgraph* hint allows you to provide an entity graph as a load graph to the query to define eager fetching specifically for this query.

You can read more about entity graphs in JPA 2.1 Entity Graph – Part 1: Named entity graphs and JPA 2.1 Entity Graph – Part 2: Define lazy/eager loading at runtime.

6. javax.persistence.fetchgraph (EntityGraph)

You can use this hint to provide an entity graph as a fetchgraph to a query.

You can read more about entity graphs in JPA 2.1 Entity Graph – Part 1: Named entity graphs and JPA 2.1 Entity Graph – Part 2: Define lazy/eager loading at runtime .

Hibernate Hints

7. org.hibernate.flushMode (FlushMode – AUTO | ALWAYS | COMMIT | MANUAL)

If you modify an entity, Hibernate keeps these changes in the first-level cache until it gets flushed. By default, this happens before each query but you can control it by providing a value of the *org.hibernate.FlushMode enum* as the *org.hibernate.flushMode* hint. You can choose between:

AUTO	Hibernate decides if the changes have to be written to the database,
ALWAYS	the Session gets flushed before every query,
COMMIT	Hibernate will not write any changes to the database until the transaction gets commited,
MANUAL	you have to flush the Session yourself.

8. org.hibernate.readOnly (boolean)

If you will not apply any changes to the selected entities, you can set the *org.hibernate.readOnly* hint to true. This allows Hibernate to deactivate dirty checking for these entities and can provide a performance benefit.

9. org.hibernate.fetchSize (Long – number of records)

Hibernate provides the value of this hint to the JDBC driver to define the number of rows the driver shall receive in one batch. This can improve the communication between the JDBC driver and the database, if it's supported by the driver.

10. org.hibernate.comment (String – custom comment)

If you set the *hibernate.use_sql_comments* property in your *persistence.xml* file to true, Hibernate generates a comment for each query and writes it to the log file. This can be useful, if you have to analyze huge or complex SQL logs.

You can use the *org.hibernate.comment* hint to provide your own comment for a query.

11. org.hibernate.cachable

If you want to use Hibernate's query cache, you have to activate it in the *persistence.xml* file and enable it for a specific query by setting the *org.hibernate.cachable* hint to true.