**<https://docs.jboss.org/hibernate/orm/3.5/reference/en/html/performance.html>**

**Cache Hibernate**

We know, file reading with [BufferedInputStream](http://way2java.com/io/bufferedinputstream-and-bufferedoutputstream/) gives more performance (around 1500 times more) than [FileInputStream](http://way2java.com/io/semantics-of-file-copying/). It is due to **buffering** effect. Same case with database accesses also. More the database hits, less the performance. To increase the database access performance, Hibernate comes with **caching mechanism** to give **high performance**.

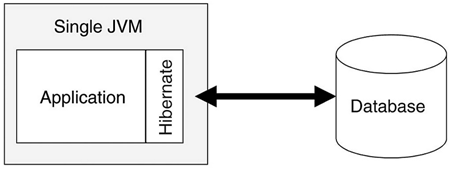
Especially, in Web applications, performance is a bottleneck due to database traffic. **Caching**is the most preferred technique to solve this problem. **Caching is nothing but some buffer where a record is stored when first time retrieved from the database.** When second time needed the same record, Hibernate does not access the database and instead reads from the **cache**. This type of adjustment decreases the database hits. This is what Cache Hibernate is. Accessing cache is much faster than accessing the database.

###### Cache Hibernate implementations

Hibernate comes with two types of cache mechanism – **First-level cache** and **Second-level cache**.(SessionFactory)

**1. First-level cache with Session Object**

First-level cache is associated with **Session** object. It is the default cache Hibernate uses (where programmer need not write any extra code).

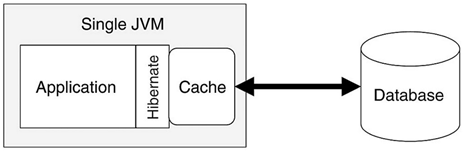
[](http://way2java.com/wp-content/uploads/2013/12/image41.png)

How it works? Hibernate does not process each database hit separately and instead preserves the query in the buffer memory (associated with Session). Infact, it writes one**addBatch()**statement and stores with the**Session** object. When the transaction is committed or session is flushed, Hibernate executes all the queries, stored earlier, with **executeBatch()** statement. This obviously increases the performance.

Session cache is useful on a per-transaction basis. What does it mean? Let us see clearly. As the first level cache is associated with the **Session**object, its scope is limited to one session only. If we fetch a record (better call as persistent object in Hibernate), say with [get()](http://way2java.com/hibernate/hibernate-get-load-delete-and-update-methods/) method, from the database, it is stored with the Session cache. If the same record is fetched again second time, database hit is not made as it gets from the **Session** cache. This you can see in the Console screen in the coming programs. If you fetch the same record with another **Session**object, database hit is made.

**2. Second-level cache with SessionFactory Object**

Second-level cache always associates with the **SessionFactor**y object. While running the transactions, whatever records (or persistent objects) the session object fetches from the database are preserved in **SessionFactory cache (buffer)**. These records are available not only to the current Session object but all Session objects created from SessionFactory object. Any Session object requires the same record again, the database is not hit and instead reads from the **SessionFactory cache**. This we can prove later in the client program.

[](http://way2java.com/wp-content/uploads/2013/12/image42.png)

The entire applications can access the **SessionFactory** cache. It is like **ServletContext** in case of servlets. If you would like the object data should be available to all the threads in the client program, the best choice is second-level cache.

Hibernate comes with four open-source cache implementations to support second-level caching.

1**. EHCache (Easy Hibernate Cache)  
2. OSCache (Open Symphony Cache)  
3. Swarm Cache  
4. JBoss Tree Cache.**

Each style differs in their performance, memory usage and support for different properties supported by servers like clustering etc.