**✅ 1. Logging in Hibernate**

Hibernate uses **SLF4J** under the hood, typically configured via log4j.properties or logback.xml.

**🔹 Example (log4j.properties):**

log4j.rootLogger=INFO, stdout

log4j.logger.org.hibernate.SQL=DEBUG # Shows SQL queries

log4j.logger.org.hibernate.type=TRACE # Shows bind values

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

**✅ 2. Lazy Loading**

**Lazy Loading** delays loading of related entities until they are accessed.

**🔹 Example:**

@OneToMany(mappedBy="d\_code", fetch = FetchType.LAZY)

private Set<Emp> emp;

**🔸 Benefits:**

* Improves performance by avoiding unnecessary DB hits.

**🔸 Watch out:**

* Accessing lazy fields outside a session causes LazyInitializationException.

**✅ 3. Fetching Strategies**

Hibernate supports:

**a. Lazy Fetching (default for @OneToMany, @ManyToMany)**

@OneToMany(fetch = FetchType.LAZY)

**b. Eager Fetching (fetch immediately)**

@ManyToOne(fetch = FetchType.EAGER)

**c. Join Fetching (HQL):**

String hql = "from Emp e join fetch e.d\_code";

**✅ 4. Caching**

Hibernate supports **caching** at two levels to reduce database access:

**🔹 First Level Cache:**

* Enabled by default.
* Scoped to a Hibernate **Session**.
* Same session → no DB hit for same entity.

**🔹 Second Level Cache:**

* Must be configured explicitly.
* Scoped to **SessionFactory**.
* Used across sessions.
* Requires a caching provider (e.g., EhCache, Infinispan).

**✅ 5. Levels of Caching**

| **Level** | **Scope** | **Default** | **Configurable** | **Notes** |
| --- | --- | --- | --- | --- |
| **1st** | Session | ✅ Yes | ❌ No | One per session |
| **2nd** | SessionFactory | ❌ No | ✅ Yes | Shared across sessions |
| **Query Cache** | For HQL results | ❌ No | ✅ Yes | Works on top of 2nd-level |

**✅ Example: Enable Second-Level Cache with EhCache**

**a. In hibernate.cfg.xml:**

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

<property name="hibernate.cache.region.factory\_class">

org.hibernate.cache.ehcache.EhCacheRegionFactory

</property>

<property name="hibernate.cache.use\_query\_cache">true</property>

**b. In Entity:**

@Cache(usage = CacheConcurrencyStrategy.READ\_WRITE)

@Entity

@Cacheable

public class Emp {

...

}

**✅ Logging in Hibernate – With Example**

Hibernate uses [SLF4J (Simple Logging Facade for Java)](http://www.slf4j.org/) as a logging API, and commonly uses **Log4j**, **Logback**, or **java.util.logging** under the hood. You can configure it to print:

* SQL queries
* Parameter values (binds)
* Caching activity
* Transaction activity

**🔧 Step-by-Step Setup Using Log4j (Classic and Common for Hibernate)**

**1. ✅ Add Log4j dependency in pom.xml**

If you're using Maven:

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!-- SLF4J to Log4j Binding -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.30</version>

</dependency>

<!-- Log4j Core -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

**2. ✅ Add log4j.properties File in src/main/resources**

# Root logger

log4j.rootLogger=INFO, stdout

# Hibernate SQL and Binding Parameters

log4j.logger.org.hibernate.SQL=DEBUG

log4j.logger.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

# Console Appender

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{HH:mm:ss} %-5p [%c{1}] %m%n

**3. ✅ Example Output**

When you run Hibernate code (e.g., insert or fetch), you will see logs like:

21:45:30 DEBUG [SQL] select emp0\_.e\_code as e\_code1\_0\_, emp0\_.e\_name as e\_name2\_0\_ from TESTEMP emp0\_

21:45:30 TRACE [BasicBinder] binding parameter [1] as [VARCHAR] - [king]

**4. Optional: Enable Full Hibernate Debugging (Verbose)**

If needed:

log4j.logger.org.hibernate=DEBUG

log4j.logger.org.hibernate.engine=DEBUG

log4j.logger.org.hibernate.tool.hbm2ddl=DEBUG

**✅ Summary**

| **Logging Purpose** | **Property** |
| --- | --- |
| Print SQL | org.hibernate.SQL=DEBUG |
| Print bound values | org.hibernate.type.descriptor.sql.BasicBinder=TRACE |
| Print schema generation logs | org.hibernate.tool.hbm2ddl=DEBUG |

**✅ Hibernate First Level Cache – Demo**

Hibernate **first level cache** is the **Session-level cache**, enabled by default. It caches objects within the current Hibernate Session. Once you fetch an entity, subsequent reads of the same entity from that session **will not hit the database** again.

**💡 Key Points**

* First-level cache is associated with the **Hibernate Session**.
* Automatically enabled – no config needed.
* Cleared when the session is closed or cleared (session.clear()).

**✅ Demo Example: First-Level Cache**

Assume we have the Emp entity mapped properly (like in your setup):

@Entity

@Table(name="TESTEMP")

public class Emp {

@Id

private int e\_code;

private String e\_name;

private double salary;

// getters, setters...

}

**🔍 Demo Code**

package com.ui;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

import com.bean.Emp;

public class FirstLevelCacheDemo {

public static void main(String[] args) {

SessionFactory factory = new Configuration().configure().buildSessionFactory();

Session session = factory.openSession();

System.out.println("Fetching employee with e\_code = 101 first time...");

Emp emp1 = session.get(Emp.class, 101); // Hits DB

System.out.println(emp1);

System.out.println("Fetching employee with e\_code = 101 second time...");

Emp emp2 = session.get(Emp.class, 101); // Cached

System.out.println(emp2);

System.out.println("Are emp1 and emp2 same object? " + (emp1 == emp2)); // true

session.close();

factory.close();

}

}

**🧠 What Happens**

* First session.get() triggers SQL query and caches Emp object.
* Second session.get() for same ID **returns cached object**, **no SQL is executed**.
* emp1 == emp2 will be true.

**🧪 Output (Sample)**

Fetching employee with e\_code = 101 first time...

Hibernate: select emp0\_.\* from TESTEMP emp0\_ where emp0\_.e\_code=?

Emp: 101 Raja 30000.0

Fetching employee with e\_code = 101 second time...

Emp: 101 Raja 30000.0

Are emp1 and emp2 same object? true

**❗ What if Session is Closed?**

Session session2 = factory.openSession();

Emp emp3 = session2.get(Emp.class, 101); // Hits DB again

Because session2 is a new session, first-level cache from session1 is no longer valid.

Here’s a full **demo** of both **Second-Level Cache** and **Query Cache** in Hibernate using **Ehcache** – the most common Hibernate caching provider.

**✅ 1. Second-Level Cache in Hibernate (Entity-level)**

**🔧 Step 1: Add Required Dependencies (pom.xml)**

<!-- Hibernate Ehcache support -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

<version>5.2.1.Final</version>

</dependency>

<!-- Ehcache -->

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache</artifactId>

<version>2.10.6</version>

</dependency>

**🗂️ Step 2: Update hibernate.cfg.xml**

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

<property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

<property name="net.sf.ehcache.configurationResourceName">/ehcache.xml</property>

**🧊 Step 3: Create ehcache.xml in src/main/resources**

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="ehcache.xsd"

updateCheck="false">

<defaultCache

maxEntriesLocalHeap="1000"

eternal="false"

timeToIdleSeconds="300"

timeToLiveSeconds="600"

overflowToDisk="false" />

<cache name="com.bean.Emp"

maxEntriesLocalHeap="1000"

eternal="false"

timeToIdleSeconds="300"

timeToLiveSeconds="600"

overflowToDisk="false" />

</ehcache>

**🧩 Step 4: Annotate Entity**

@Entity

@Cacheable

@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ\_ONLY)

@Table(name="TESTEMP")

public class Emp {

@Id

private int e\_code;

private String e\_name;

private double salary;

// constructors, getters, setters

}

**▶️ Step 5: Demo Code for Second-Level Cache**

SessionFactory factory = new Configuration().configure().buildSessionFactory();

// First session

Session session1 = factory.openSession();

Emp e1 = session1.get(Emp.class, 101); // Hits DB

System.out.println("First fetch: " + e1);

session1.close();

// Second session

Session session2 = factory.openSession();

Emp e2 = session2.get(Emp.class, 101); // Comes from 2nd level cache

System.out.println("Second fetch: " + e2);

session2.close();

📌 In this case, **only one SQL** is fired. The second session fetches from the second-level cache.

**✅ 2. Query Cache (For Reusable HQL)**

**🔧 Update hibernate.cfg.xml**

<property name="hibernate.cache.use\_query\_cache">true</property>

**▶️ Demo Code for Query Cache**

SessionFactory factory = new Configuration().configure().buildSessionFactory();

Session session1 = factory.openSession();

Query<Emp> query1 = session1.createQuery("from Emp where salary > :sal", Emp.class);

query1.setParameter("sal", 25000.0);

query1.setCacheable(true);

List<Emp> list1 = query1.list(); // Hits DB

list1.forEach(System.out::println);

session1.close();

Session session2 = factory.openSession();

Query<Emp> query2 = session2.createQuery("from Emp where salary > :sal", Emp.class);

query2.setParameter("sal", 25000.0);

query2.setCacheable(true);

List<Emp> list2 = query2.list(); // Comes from cache

list2.forEach(System.out::println);

session2.close();

📌 **Identical HQL + same parameters + setCacheable(true)** = Hibernate will cache query result.

**🔁 Summary: Caching Levels**

| **Cache Type** | **Scope** | **Enabled By Default** | **Config Required** | **Example Usage** |
| --- | --- | --- | --- | --- |
| First-Level Cache | Session | ✅ Yes | ❌ No | session.get() |
| Second-Level | SessionFactory | ❌ No | ✅ Yes (Ehcache) | @Cacheable Entity |
| Query Cache | SessionFactory | ❌ No | ✅ Yes | query.setCacheable(true) |

The ehcache.xml file is the configuration file for **Ehcache**, which is a widely used caching library in Hibernate to support **Second-Level Caching** and **Query Caching**.

**✅ Basic Structure of ehcache.xml**

Here is a minimal and typical example:

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="ehcache.xsd"

updateCheck="false">

<!-- Default cache settings -->

<defaultCache

maxEntriesLocalHeap="1000"

eternal="false"

timeToIdleSeconds="300"

timeToLiveSeconds="600"

overflowToDisk="false" />

<!-- Specific cache region for Emp entity -->

<cache name="com.bean.Emp"

maxEntriesLocalHeap="1000"

eternal="false"

timeToIdleSeconds="300"

timeToLiveSeconds="600"

overflowToDisk="false" />

</ehcache>

**🔍 Explanation of Attributes**

**<ehcache ...>**

Root tag; attributes:

* updateCheck="false": Disables checking for newer Ehcache versions online.

**🔧 <defaultCache ... />**

This sets the **default behavior** for all caches **unless overridden**.

| **Attribute** | **Description** |
| --- | --- |
| maxEntriesLocalHeap | Max number of elements to store in memory (heap). |
| eternal | If true, elements never expire (ignores TTL & TTI). |
| timeToIdleSeconds | Time (seconds) an item can remain idle before being evicted. |
| timeToLiveSeconds | Max time (seconds) an item can stay in cache after creation. |
| overflowToDisk | Whether to store evicted elements to disk. |

**🔧 <cache name="...">**

Specifies **custom cache settings** for a given region or entity.

In the example:

<cache name="com.bean.Emp"

maxEntriesLocalHeap="1000"

eternal="false"

timeToIdleSeconds="300"

timeToLiveSeconds="600"

overflowToDisk="false" />

* name="com.bean.Emp": This **must match** the fully-qualified class name of the entity you want to cache.
* Other attributes override the defaultCache settings just for this region.

**💡 How Hibernate Maps Entities to Cache Regions**

If you annotate like this:

@Entity

@Cacheable

@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ\_ONLY)

public class Emp { ... }

Hibernate maps the entity Emp to a region named com.bean.Emp, and will look for a matching <cache name="com.bean.Emp"/> in ehcache.xml.

**⚠️ Tips and Best Practices**

* For frequently read but rarely updated data (like master tables), use:  
  usage = CacheConcurrencyStrategy.READ\_ONLY
* If entities are updated often, prefer:  
  READ\_WRITE or NONSTRICT\_READ\_WRITE (with transactional cache like JTA)
* Avoid disk-based caching (overflowToDisk="false") unless your dataset is large.
* Don’t forget to enable caching in hibernate.cfg.xml:

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

<property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>