

Hibernate in Spring Boot

give me the deep and detailed explanation of Hibernate in spring boot so that i can able to understand it well and do not have any type of doubt in this.

so give me the deep and detailed explanation

Alright Piyush 👍

I'll explain **Hibernate in Spring Boot** from **absolute basics** → **internal working** → **real flow** → **why/when/how** everything is used, in **very simple language**, so that **no doubt remains**.

Think of this as a **complete mental model**, not just definitions.

◆ What is Hibernate (in Spring Boot)?

Hibernate is an **ORM (Object Relational Mapping)** framework that helps Java programs talk to databases using Java objects instead of SQL queries.

In **Spring Boot**, Hibernate is usually used **through JPA** (Java Persistence API).

- 👉 **Hibernate = Implementation**
- 👉 **JPA = Specification (rules/interface)**
- 👉 **Spring Boot = Auto-configures everything**

◆ Entity References (Important Technologies)

- Hibernate
- Spring Boot
- JPA
- MySQL

◆ Why Hibernate is Needed (Core Problem)

✗ Without Hibernate (JDBC way)

You must:

- Write SQL queries
- Manage connections
- Handle ResultSet
- Convert rows → Java objects
- Handle transactions manually

```
java
```

```
SELECT * FROM users WHERE id = 1;
```

Too much **boilerplate + error-prone + repetitive**.

✓ With Hibernate

You simply work with **Java objects**:

```
java
```

```
User user = entityManager.find(User.class, 1);
```

Hibernate:

- Generates SQL
 - Executes it
 - Converts DB row → Java object
 - Manages transactions
 - Handles caching
-

◆ Hibernate + Spring Boot = POWER COMBO

Spring Boot automatically configures Hibernate when it detects:

- `spring-boot-starter-data-jpa`
- Database driver

No XML

No manual configuration

No boilerplate

◆ Core Concepts (VERY IMPORTANT)

We'll go **one by one**, deeply.

1 Entity (Persistent Class)

What is an Entity?

An **Entity** is a **normal Java class** mapped to a **database table**.

```
java

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue
    private Long id;

    private String name;
    private String email;
}
```

What Hibernate Does

Java	Database
Class	Table
Object	Row
Field	Column

Hibernate **automatically maps** fields ↔ columns.

2 ORM (Object Relational Mapping) – Core Idea

Hibernate creates a **bridge** between:

- Java world (Objects)
- Database world (Tables)

pgsql

User Object <----> users table

You never manually convert data.

3 SessionFactory (Heavyweight Object)

What is SessionFactory?

- Created **once** at application startup
- Reads:
 - Entity mappings
 - DB configuration
- Used to create Session

👉 Spring Boot creates it internally

4 Session (VERY IMPORTANT)

What is Session?

- Represents **one database interaction**
- Manages:
 - CRUD operations
 - First-level cache
 - Object states

Think of Session as:

💡 Hibernate's **working memory** for DB operations

5 Object States in Hibernate (CRUCIAL)

Hibernate tracks **state of objects**.

◆ 1. Transient

java

```
User u = new User();  
u.setName("Piyush");
```

- Not associated with DB
 - No ID
 - Hibernate doesn't track it
-

◆ 2. Persistent

java

```
session.save(u);
```

- Now linked to DB
 - Hibernate tracks changes
 - Auto update happens
-

◆ 3. Detached

java

```
session.close();
```

- Object exists
 - But session is closed
 - Hibernate stops tracking
-

◆ 4. Removed

java

```
session.delete(u);
```

- Marked for deletion

6 Transactions (VERY IMPORTANT)

Why Transactions?

To ensure:

- Consistency
- Atomicity
- Rollback on failure

java

```
@Transactional
public void saveUser(User user) {
    userRepository.save(user);
}
```

Spring Boot:

- Begins transaction
- Commits if success
- Rollbacks if exception

7 Hibernate Cache (EXTREMELY IMPORTANT)

◆ Level 1 Cache (Session Cache)

- Enabled by default
- Per Session

java

```
session.get(User.class, 1);
session.get(User.class, 1); // NO DB HIT
```

◆ Level 2 Cache (Optional)

- Shared across sessions
- Uses tools like:
 - EhCache

- Redis

Improves performance drastically.

8 Hibernate Query Language (HQL)

Hibernate allows:

java

```
FROM User WHERE email = :email
```

Not SQL ❌

It uses **entity names**, not table names.

9 JPA Repositories (Spring Boot Magic)

You usually don't use `Session` directly.

java

```
public interface UserRepository extends JpaRepository<User, Long> {  
    User findByEmail(String email);  
}
```

Spring Boot:

- Generates implementation
 - Uses Hibernate internally
 - Handles everything
-

◆ Hibernate Auto Configuration in Spring Boot

Spring Boot does automatically:

- ✓ DataSource
- ✓ EntityManager
- ✓ SessionFactory
- ✓ TransactionManager

- ✓ Dialect detection
- ✓ SQL generation

Just add:

```
properties
```

```
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true
```

◆ How Hibernate Works Internally (FLOW)

Step-by-step Flow

- 1 Controller receives request
- 2 Service calls repository
- 3 Repository uses Hibernate
- 4 Hibernate:
 - Opens session
 - Creates SQL
 - Executes query
 - Maps result to entity
- 5 Transaction commits
- 6 Response returned

◆ Hibernate vs JDBC (Quick Comparison)

Feature	JDBC	Hibernate
SQL	Manual	Automatic
Mapping	Manual	Automatic
Caching	No	Yes
Transactions	Manual	Automatic
Code	Large	Clean

Feature	JDBC	Hibernate
Performance	Low	Optimized

◆ Common Doubts (Answered)

? Is Hibernate slow?

✗ NO

When used correctly, Hibernate is **faster than JDBC** due to caching.

? Should I learn SQL?

✓ YES

Hibernate **generates SQL**, so understanding SQL is mandatory.

? Do we use Hibernate directly in Spring Boot?

Usually ✗

We use **Spring Data JPA**, which uses Hibernate internally.

◆ Real-World Example (Mental Model)

Think like this:

- **Entity** → Blueprint of table
 - **Hibernate** → Translator
 - **Session** → Working desk
 - **Transaction** → Safety lock
 - **Repository** → Interface to DB
 - **Spring Boot** → Manager who sets everything up
-

◆ Final Summary (One Shot)

- Hibernate is an ORM framework

- Spring Boot auto-configures Hibernate
 - Entities map Java → DB
 - Sessions manage object lifecycle
 - Transactions ensure data safety
 - Caching improves performance
 - Repositories simplify DB access
-

If you want, next we can go **EVEN DEEPER** into:

- 🔥 Hibernate Caching (L1, L2, Query Cache)
- 🔥 Lazy vs Eager Fetching (VERY IMPORTANT)
- 🔥 Hibernate Annotations (ALL with examples)
- 🔥 Hibernate Interview Questions (Real-world)
- 🔥 Internal SQL generation

Just tell me 👍

yes

Perfect 👍

Let's now go **DEEP** into **Hibernate Caching**, because this is one of the **most important + most confusing + most asked** topics in **Spring Boot + Hibernate** interviews.

I'll explain **from zero**, with **mental models, flow, examples, diagrams (in words)** so that **no doubt remains**.

🔥 Hibernate Caching – COMPLETE & DEEP EXPLANATION

◆ Entity References (Technologies Used)

- Hibernate
 - Spring Boot
 - Ehcache
 - Redis
 - MySQL
-

◆ First Understand: Why Caching is Needed?

✗ Without Cache

Every request → DB hit

DB is:

- Slow
- Expensive
- Limited connections

Example:

java

```
userRepository.findById(1);  
userRepository.findById(1);  
userRepository.findById(1);
```

➡ 3 database queries

✓ With Cache

Data is stored in **memory**.

- ➡ DB hit only **once**
 - ➡ Remaining calls served from cache
 - ➡ Faster response
 - ➡ Less DB load
-

◆ What is Cache in Hibernate?

Cache = Temporary in-memory storage of frequently used data

Hibernate provides **TWO LEVELS** of caching:

mathematica

Hibernate Cache Levels:

- 1 First Level Cache (L1) → Default
- 2 Second Level Cache (L2) → Optional

🔥 LEVEL 1 CACHE (FIRST LEVEL CACHE)

◆ What is L1 Cache?

- Default cache
- Enabled automatically
- Associated with **Session**
- Cannot be disabled ❌

👉 One session = one cache

◆ How L1 Cache Works (Step-by-Step)

java

```
Session session = sessionFactory.openSession();

User u1 = session.get(User.class, 1); // DB HIT
User u2 = session.get(User.class, 1); // NO DB HIT
```

Internal Flow

- 1 Hibernate checks Session cache
- 2 If entity exists → return it
- 3 Else → hit DB + store in cache

◆ Key Points of L1 Cache

Feature	Value
Scope	Session
Enabled	Always
Shared	❌ No

Feature	Value
Speed	Very fast
Lifetime	Till session closes

◆ When L1 Cache is Cleared?

- `session.clear()`
- `session.close()`
- Transaction ends

🔥 LEVEL 2 CACHE (SECOND LEVEL CACHE)

This is where real performance boost comes.

◆ What is L2 Cache?

- Shared across **multiple sessions**
- Stores **entity data**
- Needs external cache provider
- Optional (manual configuration)

pgsql

```

Session 1  ↵
Session 2  |→ Second Level Cache → Database
Session 3  ↵

```

◆ Why L2 Cache is Needed?

L1 Cache ❌ is limited to a single session.

In real apps:

- Each HTTP request = new session

- So L1 cache is useless across requests

➡ L2 cache solves this problem

◆ Popular L2 Cache Providers

Provider	Use Case
Ehcache	Simple, local cache
Redis	Distributed cache
Hazelcast	Cluster cache

◆ How L2 Cache Works (Flow)

- 1 Request comes
 - 2 Hibernate checks L2 cache
 - 3 If found → return entity
 - 4 If not found:
 - Fetch from DB
 - Store in L2 cache
 - Return result
-

◆ Enabling L2 Cache in Spring Boot

Step 1 Add Dependency (Ehcache Example)

xml

```
<dependency>
  <groupId>org.ehcache</groupId>
  <artifactId>ehcache</artifactId>
</dependency>
```

Step 2 Enable Cache in Properties

properties

```
spring.jpa.properties.hibernate.cache.use_second_level_cache=true
spring.jpa.properties.hibernate.cache.region.factory_class=jcache
spring.jpa.properties.javax.cache.provider=org.ehcache.jsr107.EhcacheCachingProvi
```

Step 3 Mark Entity as Cacheable

java

```
@Entity
@Cacheable
@org.hibernate.annotations.Cache(
    usage = CacheConcurrencyStrategy.READ_WRITE
)
public class User {
}
```

◆ Cache Concurrency Strategies (VERY IMPORTANT)

Strategy	Use Case
READ_ONLY	Data never changes
READ_WRITE	Data changes
NONSTRICT_READ_WRITE	Less consistency
TRANSACTIONAL	Strict ACID

👉 Most used: READ_WRITE

🔥 QUERY CACHE (ADVANCED)

◆ What is Query Cache?

- Caches **query results**
 - Not entity data
 - Works with L2 cache
-

Example

java

```
@QueryHints({
    @QueryHint(name = "org.hibernate.cacheable", value = "true")
})
List<User> findByRole(String role);
```

◆ Important Rule

! Query Cache depends on L2 Cache

If L2 cache is OFF → Query cache useless.

🔥 Cache vs Database (Comparison)

Aspect	Cache	Database
Storage	Memory	Disk
Speed	Very fast	Slow
Cost	Low	High
Persistence	Temporary	Permanent

🔥 Common Interview Doubts (CLEARED)

? Is Hibernate cache same as Spring cache?

✗ No

Hibernate cache works at **ORM level**

Spring cache works at **method level**

? Does save() update cache?

✓ YES

Hibernate keeps cache consistent.

? When NOT to use cache?

- Frequently changing data
 - Huge tables
 - One-time reads
-

🔥 Real-Life Analogy (Best Understanding)

- Database → Library
 - Cache → Table near you
 - L1 Cache → Your notebook
 - L2 Cache → Shared whiteboard
-

◆ Final Summary (Must Remember)

- ✓ Hibernate cache improves performance
 - ✓ L1 cache is default and session-based
 - ✓ L2 cache is shared and configurable
 - ✓ Query cache stores query results
 - ✓ Cache strategy matters
 - ✓ Not all data should be cached
-

🔥 NEXT (Tell me what you want)

We can now go **even deeper** into:

- 1 Lazy vs Eager Fetching (VERY IMPORTANT)
- 2 Hibernate Annotations (ALL with examples)
- 3 N+1 Query Problem (REAL WORLD ISSUE)
- 4 Hibernate Internals (SQL generation)
- 5 Hibernate Interview Questions

👉 Just tell me **which one** 🚀