**Caching in Hibernate**

Hibernate is a Java-based **Object-Relational Mapping (ORM)** framework that simplifies the process of mapping object-oriented domain models to relational databases.

**What does that mean?**

Well, it means that Hibernate allows developers to create **Java classes that represent database tables, and it automatically maps those classes to their corresponding tables in the database**.

In addition to other features like **automatic schema generation**, **lazy loading of data**, and **support for transactions**, Hibernate also includes a feature called **caching**.

**Without caching, Hibernate has to fetch data from the database every time it is requested. This process can be time-consuming and can cause performance issues, especially for frequently accessed data. Additionally, as the size of the data grows, the time required to fetch and process the data increases exponentially.**

This is where caching comes in. **Caching is a technique that allows frequently accessed data to be stored in memory so that it can be quickly retrieved without having to go back to the database every time**. By caching data, Hibernate can reduce the number of database queries it needs to execute, improving performance and reducing the load on the database server.

**Levels of Caching in Hibernate**

Hibernate offers caching at three levels:

**1. First-Level Cache (also known as Session Cache)**

First-level caching is an in-built caching mechanism provided by Hibernate that stores the data in the memory of the Session object. Whenever an entity is fetched from the database, it is stored in the cache. If the same entity is requested again within the same session, Hibernate retrieves it from the cache instead of hitting the database. This can significantly improve the performance of the application by reducing the number of database queries.

The first-level cache has session-level scope, which means that the data stored in the cache is available only within the session. When the session is closed, the cache is cleared automatically. The cache is also not shared between different sessions, which means that each session has its own separate cache.

Hibernate manages the first-level cache automatically, and there is no need to configure it manually. The cache is enabled by default and can’t be turned off. When an entity is fetched from the database, Hibernate checks if it is already present in the cache. If it is, the cached entity is returned, otherwise, Hibernate fetches the entity from the database and stores it in the cache.

**2. Second-Level Cache**

Second-level caching is an optional caching mechanism provided by Hibernate that stores data across sessions. It is more powerful than first-level caching, which is limited to a single session.

To enable Second-level Caching in Hibernate, you need to configure a cache provider in your Hibernate configuration file. Hibernate supports multiple cache providers such as Ehcache, Infinispan, and Hazelcast. Check Ehcache is supported in hibernate 6 or not.

**3. Query Cache**

Query caching is a caching mechanism provided by Hibernate to cache the results of queries. When a query is executed, Hibernate checks if the query results are already cached. If the results are found in the cache, they are returned without hitting the database, otherwise, the query is executed and the results are cached for future use.

To enable query caching in Hibernate, you need to set the hibernate.cache.use\_query\_cache property to true.

spring.jpa.properties.hibernate.cache.use\_query\_cache=true  
spring.jpa.properties.hibernate.cache.region.factory\_class=org.hibernate.cache.ehcache.EhCacheRegionFactory

@Repository  
public interface AuthorRepository extends CrudRepository<Author, Integer> {  
 @QueryHints({ @QueryHint(name = "org.hibernate.cacheable", value ="true") })  
 Author findByName(String name);  
}

**Configuring Caching in Hibernate**

Hibernate provides various configuration properties to enable and configure caching. Here are some of the commonly used properties:

1. hibernate.cache.use\_second\_level\_cache - Set this to true to enable the second-level cache.
2. hibernate.cache.region.factory\_class - This property specifies the caching provider to be used for the second-level and query cache. For example, to use EHCache, set it to org.hibernate.cache.ehcache.EhCacheRegionFactory.
3. hibernate.cache.use\_query\_cache - Set this to true to enable the query cache.
4. hibernate.cache.provider\_configuration\_file\_resource\_path - This property specifies the path to the configuration file for the caching provider. For example, for EHCache, it can be set to ehcache.xml.