

Day 25- Facilitation Guide

Index

- I. Hibernate caching
- II. Second level cache
- III. Q&A Discussion
- IV. Interview Q&A

For (1.5 hrs) ILT

During the previous session about the HQL queries,Named queries and inheritance in hibernate, we covered a variety of important concepts:

HQL Queries (Hibernate Query Language):

HQL is an object-oriented query language in Hibernate, used to query and manipulate data from a relational database.It operates on Hibernate entities and their properties, making it more abstract and powerful than SQL.HQL queries are automatically translated to SQL queries by Hibernate at runtime.

Named Queries:

Named queries are pre-defined HQL or SQL queries associated with an entity in Hibernate.They are stored in mapping files or annotations and provide a way to separate query logic from application code.Named queries improve code readability and maintainability and can be reused across the application.

Inheritance in Hibernate:

Hibernate supports different inheritance strategies for mapping class hierarchies to database tables.

Three common strategies are: Table Per Hierarchy (Single Table), Table Per Subclass (Joined Subclass), and Table Per Concrete Class (Table Per Class).The default inheritance strategy is "Table Per Hierarchy," where all subclasses share a single table with a discriminator column.

During this session, we will explore the Hibernate caching mechanism, specifically focusing on the second level cache.

I. Hibernate caching

Caching in Hibernate refers to the technique of storing frequently accessed data in memory to improve the performance of an application that uses Hibernate as an Object-Relational Mapping (ORM) framework. Hibernate provides two levels of caching:

1. **First-Level Cache:** Hibernate uses a session-level cache, also known as a first-level cache, to store the data that is currently being used by a specific session. When an entity is loaded or updated for the first time in a session, it is stored in the session-level cache. Any subsequent request to fetch the same entity within the same session will be served from the cache, avoiding a database round-trip. The session-level cache is enabled by default and cannot be disabled.
2. **Second-Level Cache:** Hibernate also supports a second-level cache, which is a shared cache across multiple sessions. This cache stores data that is frequently used across different sessions, reducing the number of database queries and improving the overall performance of the application. The second-level cache can be configured with various caching providers, such as Ehcache, Infinispan, and Hazelcast.

Using caching in Hibernate can significantly improve the performance of an application by reducing the number of database round-trips and improving response times. However, it is important to use caching carefully, as it can also cause issues such as stale data, memory leaks, and increased complexity in managing the cache.

Advantages of caching

- **Improved Performance:** Hibernate caching can significantly improve the performance of an application by reducing the number of database round-trips required to retrieve data. By storing frequently accessed data in memory, Hibernate can serve subsequent requests for the same data from the cache, reducing the time it takes to retrieve the data from the database.
- **Reduced Database Load:** Caching in Hibernate can reduce the load on the database by minimizing the number of queries made to the database. This

can help to improve the scalability of the application and reduce the risk of database overload.

- **Increased Concurrency:** Hibernate caching can increase concurrency by reducing the time that multiple users or threads spend waiting for the database. By storing frequently accessed data in memory, Hibernate can serve multiple requests for the same data from the cache simultaneously.
- **Customizable Cache Configuration:** Hibernate provides a customizable caching framework that allows developers to configure the cache to meet the specific needs of their application. This includes the ability to configure the cache provider, cache regions, cache strategies, and cache eviction policies.

First-Level Cache:

The first-level cache is also known as the session cache. It is a cache associated with the Hibernate Session object. Here's how it works:

- When you load an entity using `session.get()` or `session.load()`, the entity is cached in the first-level cache for the duration of the session.
- If you retrieve the same entity within the same session, Hibernate will return it from the cache, rather than hitting the database again. This reduces the number of database queries.
- The first-level cache is limited to the scope of a single Hibernate session and is cleared when the session is closed.

Let's explore with practical demonstration:

Here we are retrieving the students record based on student Id:

Entity Class:

```
package com.sms;
```

```
import java.time.LocalDate;  
import java.util.List;
```

```
import javax.persistence.Column;  
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
```

@Entity

```
public class Student {
    @Id
    @Column(name = "StudentId", length = 10)
    private String studentId;

    @Column(name = "FirstName", length = 50)
    private String firstName;

    @Column(name = "LastName", length = 25)
    private String lastName;

    @Column(name = "DateOfBirth")
    private LocalDate dateOfBirth;

    @Column(name = "Gender", length = 25)
    private String gender;

    @Column(name = "Email", length = 30)
    private String email;

    @Column(name = "Phone", length = 25)
    private String phone;

    @OneToMany(mappedBy = "studentId")
    private List<Enrollment> enrollments;

    public List<Enrollment> getEnrollments() {
        return enrollments;
    }

    public void setEnrollments(List<Enrollment> enrollments) {
        this.enrollments = enrollments;
    }
}
```

//Setter And Getter

```
public String getStudentId() {  
    return studentId;  
}
```

```
public void setStudentId(String studentId) {  
    this.studentId = studentId;  
}
```

```
public String getFirstName() {  
    return firstName;  
}
```

```
public void setFirstName(String firstName) {  
    this.firstName = firstName;  
}
```

```
public String getLastName() {  
    return lastName;  
}
```

```
public void setLastName(String lastName) {  
    this.lastName = lastName;  
}
```

```
public LocalDate getDateOfBirth() {  
    return dateOfBirth;  
}
```

```
public void setDateOfBirth(LocalDate dateOfBirth) {  
    this.dateOfBirth = dateOfBirth;  
}
```

```
public String getGender() {  
    return gender;  
}
```

```
public void setGender(String gender) {  
    this.gender = gender;  
}
```

```
public String getEmail() {  
    return email;  
}
```

```
public void setEmail(String email) {  
    this.email = email;  
}
```

```
public String getPhone() {  
    return phone;  
}
```

```
public void setPhone(String phone) {  
    this.phone = phone;  
}
```

//Default Constructor

```
public Student() {  
    super();  
    // TODO Auto-generated constructor stub  
}
```

//All argument Constructor

```
public Student(String studentId, String firstName, String lastName, LocalDate  
dateOfBirth, String gender,  
                String email, String phone) {  
    super();  
    this.studentId = studentId;  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.dateOfBirth = dateOfBirth;  
    this.gender = gender;  
    this.email = email;  
    this.phone = phone;  
}
```

```

//ToString method
@Override
public String toString() {
    return "Student [studentId=" + studentId + ", firstName=" + firstName + ",
lastName=" + lastName
                    + ", dateOfBirth=" + dateOfBirth + ", gender=" + gender + ",
email=" + email + ", phone=" + phone
                    + ", enrollments=" + enrollments + "]\n";
}
}

```

HibernateUtil Class:

```

package com.sms;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            return new Configuration().configure("hibernate.cfg.xml")
                .addAnnotatedClass(Student.class)
                .buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}

```

Main Class:

```

package com.sms;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

public class RetrievingRecord {

    public static void main(String[] args) {
        // Obtain a Hibernate SessionFactory
        SessionFactory factory = HibernateUtil.getSessionFactory();

        // Open a new session
        Session session = factory.openSession();

        // Begin a transaction
        Transaction transaction = session.beginTransaction();

        // Retrieve the object using the primary key
        Student student=session.load(Student.class, "S105");

        //display data using toString() method
        System.out.println(student);

        // Retrieve the same object using same session
        Student student1=session.load(Student.class, "S105");

        //display data using toString() method
        System.out.println(student);

        // Close the session
        session.close();
    }
}

```

Output


```

Hibernate:
  select
    enrollment0_.StudentId as studenti4_0_0_,
    enrollment0_.EnrollmentID as enrollme1_0_0_,
    enrollment0_.EnrollmentID as enrollme1_0_1_,
    enrollment0_.InstructorID as instruct2_0_1_,
    enrollment0_.courseID as courseid3_0_1_,
    enrollment0_.StudentId as studenti4_0_1_
  from
    Enrollment enrollment0_
  where
    enrollment0_.StudentId=?
Student [studentId=S105, firstName=Peter, lastName=Parker, dateOfBirth=2011-06-05, gender=M, email=
Student [studentId=S105, firstName=Peter, lastName=Parker, dateOfBirth=2011-06-05, gender=M, email=
Go to Settings to activate Windows.

```

In this example, as you can observe, we've fetched the identical entity during the current session. As a result, Hibernate retrieves it from the cache instead of making another database request. This ensures that the query is executed only once, effectively decreasing the volume of database queries.

II. Second level cache

Hibernate provides several second-level cache providers that can be used to store cached data in a shared cache across multiple sessions. These include:

- **Ehcache:** Ehcache is a popular open-source caching library that provides an efficient in-memory caching solution for Hibernate. It supports various features such as expiration policies, distributed caching, and persistent caching.
- **Infinispan:** Infinispan is an open-source data grid platform that provides a distributed cache for Hibernate. It supports various caching modes such as local, replicated, and distributed caching and provides features such as expiration policies, transactions, and data eviction.
- **Hazelcast:** Hazelcast is an open-source in-memory data grid that provides a distributed caching solution for Hibernate. It supports various features such as distributed caching, data partitioning, data replication, and automatic failover.
- **JBoss Cache:** JBoss Cache is a popular open-source caching library that provides a scalable and distributed caching solution for Hibernate. It supports various features such as distributed caching, data replication, and transactional caching.
- **Caffeine:** Caffeine is a high-performance in-memory caching library that provides a fast and efficient caching solution for Hibernate. It supports various features such as expiration policies, eviction policies, and asynchronous loading.

Developers can choose the appropriate caching provider based on their specific requirements such as performance, scalability, and feature set. It is also possible to

implement a custom cache provider by implementing the CacheProvider interface provided by Hibernate.

Configure the second-level cache:

To configure the second-level cache in Hibernate, you need to perform the following steps:

- **Choose a cache provider:** You can choose a cache provider that meets your application requirements. Hibernate provides several cache providers such as Ehcache, Infinispan, Hazelcast, JBoss Cache, and Caffeine.
- **Add the caching provider to the classpath:** You need to add the caching provider library to the classpath of your Hibernate application.
- **Configure Hibernate properties:** You need to configure Hibernate properties to enable the second-level cache and specify the caching provider. For example, to enable the Ehcache provider, you can add the following properties to the hibernate.cfg.xml file:

```
<property name="hibernate.cache.use_second_level_cache">true</property>
<property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheR
egionFactory</property>
```

- **Configure entity caching:** You need to configure entity caching for specific entities in your Hibernate application. You can do this by adding the @Cacheable annotation to the entity class and specifying the cache region name. For example:

```
@Entity
@Cacheable
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE, region="myEntityCache")
public class MyEntity {
    // ...
}
```

How Does Hibernate second-level cache work:

- Hibernate second-level cache works by storing entity and query data in a shared cache that is accessible across multiple sessions. When a query or entity is fetched for the first time, it is stored in the second-level cache, and any subsequent requests for the same entity or query are served from the cache rather than querying the database again.
- Here is a high-level overview of how Hibernate second-level cache works:
 - A request is made to fetch an entity or execute a query: When a request is made to fetch an entity or execute a query, Hibernate checks the second-level cache to see if the data is already cached.
 - Cache lookup: If the data is already cached, Hibernate retrieves it from the cache and returns it to the user.
 - Database query: If the data is not cached, Hibernate queries the database to fetch the data and then stores it in the second-level cache for future use.
 - Cache eviction: When an entity is updated or deleted, Hibernate automatically removes the corresponding entry from the second-level cache to ensure data consistency.
 - Cache expiration: The second-level cache can also be configured to expire entries based on certain criteria such as time-to-live or maximum cache size.

Hibernate supports different caching strategies, including read-only, read-write, and transactional caching. The caching strategy determines how the data is stored and retrieved from the cache, and developers can choose the appropriate strategy based on their specific requirements.

Let's observe it through a practical demonstration:

Here we are using Ehcache cache provider

First add this two dependencies in your pom.xml file:

```
<!-- https://mvnrepository.com/artifact/net.sf.ehcache/ehcache -->
<dependency>
<groupId>net.sf.ehcache</groupId>
<artifactId>ehcache</artifactId>
<version>2.10.6</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-ehcache -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-ehcache</artifactId>
<version>5.4.5.Final</version>
</dependency>
```

Secondly add this two properties in your hibernate.cfg.xml file:

```
<property name="hibernate.cache.use_second_level_cache">true</property>
<property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheR
egionFactory</property>
```

Hibernate.cfg.xml File:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

    <hibernate-configuration>

        <session-factory>

<property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://localhost:3306/studentsystem</property>
<property name="connection.username">root</property>
<property name="connection.password">mysql</property>
<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
```

```

<property name="hbm2ddl.auto">update</property>
<property name="show_sql">true</property>
<property name="format_sql">true</property>
<property name="hibernate.cache.use_second_level_cache">true</property>
<property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheR
egionFactory</property>

</session-factory>

</hibernate-configuration>

```

Entity Class:

```

package com.sms;

import java.time.LocalDate;
import java.util.List;

import javax.persistence.Cacheable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.OneToOne;

import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

@Entity
@Cacheable
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
public class Student {
    @Id
    @Column(name = "StudentId", length = 10)
    private String studentId;

    @Column(name = "FirstName", length = 50)
    private String firstName;

```

```
@Column(name = "LastName", length = 25)
private String lastName;
```

```
@Column(name = "DateOfBirth")
private LocalDate dateOfBirth;
```

```
@Column(name = "Gender", length = 25)
private String gender;
```

```
@Column(name = "Email", length = 30)
private String email;
```

```
@Column(name = "Phone", length = 25)
private String phone;
```

```
@OneToMany(mappedBy = "studentId")
private List<Enrollment> enrollments;
```

```
public List<Enrollment> getEnrollments() {
    return enrollments;
}
```

```
public void setEnrollments(List<Enrollment> enrollments) {
    this.enrollments = enrollments;
}
```

```
//Setter And Getter
public String getStudentId() {
    return studentId;
}
```

```
public void setStudentId(String studentId) {
    this.studentId = studentId;
}
```

```
public String getFirstName() {
    return firstName;
}
```

```
public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public LocalDate getDateOfBirth() {
    return dateOfBirth;
}

public void setDateOfBirth(LocalDate dateOfBirth) {
    this.dateOfBirth = dateOfBirth;
}

public String getGender() {
    return gender;
}

public void setGender(String gender) {
    this.gender = gender;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPhone() {
    return phone;
}
```

```

    public void setPhone(String phone) {
        this.phone = phone;
    }

    //Default Constructor
    public Student() {
        super();
        // TODO Auto-generated constructor stub
    }

    //All argument Constructor
    public Student(String studentId, String firstName, String lastName, LocalDate
dateOfBirth, String gender,
        String email, String phone) {
        super();
        this.studentId = studentId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
        this.gender = gender;
        this.email = email;
        this.phone = phone;
    }

    //ToString method
    @Override
    public String toString() {
        return "Student [studentId=" + studentId + ", firstName=" + firstName + ",
lastName=" + lastName
        + ", dateOfBirth=" + dateOfBirth + ", gender=" + gender + ",
email=" + email + ", phone=" + phone
        + ", enrollments=" + enrollments + "]";
    }
}

```


Main Class:

```
package com.sms;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

public class RetrievingRecord {

    public static void main(String[] args) {
        // Obtain a Hibernate SessionFactory
        SessionFactory factory = HibernateUtil.getSessionFactory();

        // Open a new session
        Session session = factory.openSession();

        // Begin a transaction
        Transaction transaction = session.beginTransaction();

        // Retrieve the object using the primary key
        Student student=session.load(Student.class, "S105");

//        //display data using toString() method
//        System.out.println(student);

        // Close the session
        session.close();

        // Open a another session
        Session session1 = factory.openSession();

        // Retrieve the same object using another session
        Student student1=session1.load(Student.class, "S105");

        //display data using toString() method
        System.out.println(student);

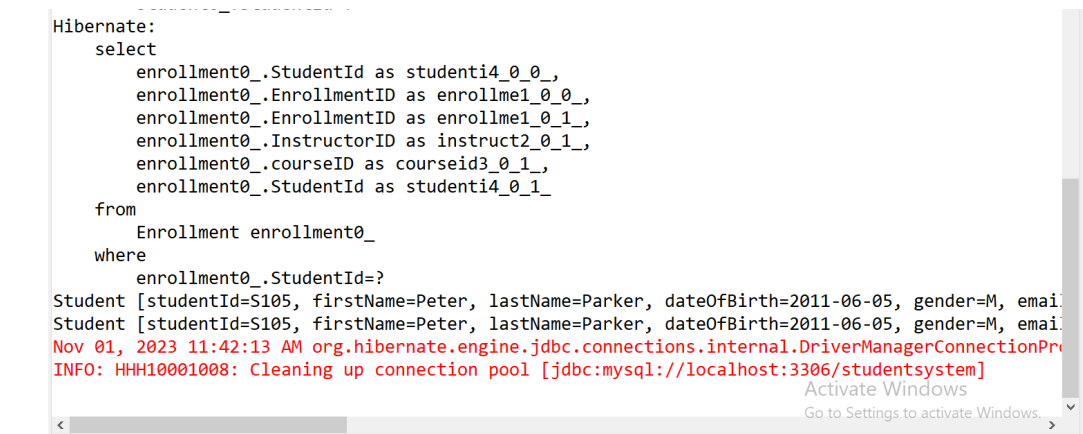
        // Close the session
        session1.close();
    }
}
```

```

        //close the session Factory
        factory.close();
    }
}

```

Output:



```

Hibernate:
select
    enrollment0_.StudentId as studenti4_0_0_,
    enrollment0_.EnrollmentID as enrollme1_0_0_,
    enrollment0_.EnrollmentID as enrollme1_0_1_,
    enrollment0_.InstructorID as instruct2_0_1_,
    enrollment0_.courseID as courseid3_0_1_,
    enrollment0_.StudentId as studenti4_0_1_
from
    Enrollment enrollment0_
where
    enrollment0_.StudentId=?
Student [studentId=S105, firstName=Peter, lastName=Parker, dateOfBirth=2011-06-05, gender=M, email=
Student [studentId=S105, firstName=Peter, lastName=Parker, dateOfBirth=2011-06-05, gender=M, email=
Nov 01, 2023 11:42:13 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionPro
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/studentsystem]

```

In this program, we have employed the SessionFactory cache. We've retrieved the same objects using different session objects, yet Hibernate only accesses the database once because the object is globally cached.

Food for thought..

The trainer can ask the students the following question to engage them in a discussion.

How does the second-level cache in Hibernate handle concurrent access to cached objects?

Exercise:

Use ChatGPT to explore more cache provider in hibernate:

Put the below problem statement in the message box and see what ChatGPT says.

I'm currently exploring Hibernate caching, and I've already worked with the EhCache provider. I'm interested in expanding my knowledge to other cache providers, so I'd appreciate it if you could recommend tutorials or resources for that purpose?