Spring Boot - DataBase

Anna

Pilot

Setting up MySQL

- Download and install
- Create a new user for springBoot project and give it Admin access
- Start the server
- Create a new Schema(project) and setting up tables

Steps to implement

- Add dependency in pom

spring-boot-starter-data-jpa

mysql-connector-java:

https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.29

- Configuration
  - In `resources -> application.properties` -> we should config the MySQL port

Steps to implement

- Mark Entity and Table in Java code
  - @Table(name = “student”)
  - @Entity
  - @Id
  - @Column(name = “id”)

- JPARepository & CrudRepository
  - Specify all methods
  - Hibernate

Class Goal and Demo

- Create a student using JpaRepository by Postman
- Check the data updated into Database table
- Get that student by id using JpaRepository by Postman

Homework

- Tried to connect Database on local
- Repo above goals
- Use JpaRespository to replace all CRUD **student** operations and remove previous StudentRepository

Customize SQL Query in Java Code

- Add a method findByName in Student class
  - 1. Using JpaRepository
  - 2. Write the query by hql/jpql
  - 3. Write the query using original Database query
- Delete Student demo – using Optional

Database Pagination Call

- What is Pagination?

Pagination is the process of displaying the data on multiple pages rather than showing them on a single page.

- How to implement it?
  - Using page and sizePerPage
  - Using nextToken, like DynamoDB pagination

  - HW details

Data Mapping Relations:

- 1:1 Mapping

Eg: Student and Seat (assume each student only allow one seat in system)

Student(student_id, seat_id, name, age, address, …)

Seat(seat_id, location)

- 1:M Mapping

Eg: Student and Card (One student can have multiple credit cards)

Student(student_id, name, age, address, …)

Card(card_id, student_id, number, name)

- M:M Mapping

Eg: Student and teacher (One student can have multiple teachers, One teacher can have many students)

Student(student_id, card_id, name, age, address, …)

Teacher(teacher_id, name)

Student_Teacher_Relation(student_id, teacher_id)

Cascade Type

- **CascadeType.PERSIST**: means that save() or persist() operations cascade to related entities.
- **CascadeType.MERGE**: means that related entities are merged when the owning entity is merged.
- **CascadeType.REFRESH**: does the same thing for the refresh() operation.
- **CascadeType.REMOVE**: removes all related entities association with this setting when the owning entity is deleted.
- **CascadeType.DETACH**: detaches all related entities if a “manual detach” occurs.
- **CascadeType.ALL**: cascade type all is shorthand for all of the above cascade operations.
- flush();

Fetch Type:

- Eager Loading is a design pattern in which data initialization occurs on the spot.

  fetch immediately

- Lazy Loading is a design pattern that we use to defer initialization of an object as long as it's possible. (by default)

  fetch when needed

Join Type

- inner_join
- left_join
- right_join

Reference: https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/

Potential Issue

Might cause Infinite loop since we map on both side

@JsonIgnore or override toString() method

- **Demo**

- **Homework**
  - Complete the project with Teacher entity and use student – teacher M:N Mapping and setting the proper cascade
  - Complete the regular teacher controller and using JPA repository
  - Complete the functionality of SignUpController

Goal:

- Create a teacher Anna, save it in Database Teacher table
- Create a teacher Andy, save it in Database Teacher table
- Create a student Amy, save it in Database Student table
- Build the connection between Amy – Anna, Amy – Andy via M:M mapping
- Get student Amy and check the teachers set has Anna and Andy on it
- To verify if that works, delete that teacher Anna via teacher controller, and then check that student Amy's teacher list, it should only have Andy on it

Cache:

- Why to use cache – improve performance (latency will reduce)

- How to use cache
  - @EnableCaching
  - @Cacheable(cacheNames="student") in service Get/Create methods

We don't want to populate the cache with values that we don't need often. Caches can grow quite large, quite fast, and we could be holding on to a lot of stale or unused data.

Data sync into cache in the first time, and then if there is more Create/Update/Delete operation happens, the related result will sync to cache as well.

Cache:

- @CachePut
- @CacheEvict

- But the cache above is only for demo propose

we will have multiple instances sharing the same cache through the same platform -> we need to use some cache dependency, like **Redis**

Exception Handler

- How do we handle exceptions in SpringBoot?

  @RestControllerAdvice in controller class.

  You can customize the handled exception and give different error code.

- What is **TTL**?

Time-to-live (TTL) is a value for the period of time that a packet, or data, should exist on a computer or network before being discarded.

What is AOP

- AOP: Aspect Oriented Programming based on dynamic proxy
- Cache and ExceptionHandler above are two examples of AOP

Homework (Optional)

Best Practice is to

- add cache(CRUD Operations) and test if that works
- add exception handler, returns different exception for your project.