Spring data - II

Teaching Faculty: Dr. Muhyieddin Al-Tarawneh

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Derived query methods - naming convention

- Just by looking at the corresponding method name in the code, Spring Data JPA can determine what the query should be.

- Spring Data JPA supports
  - find
  - read
  - query
  - count
  - get

EXAMPLES

- List<T> findByAgeLessThan(Integer age)
- List<T> findByNameIsNot(String name);
- List<T> findByActiveTrue();
- List<T> findByNameStartingWith(String prefix);

EXAMPLES

- List<T> findByNameEndingWith(String suffix);
- List<T> findByNameContaining(String infix);
- List<T> findByNameOrBirthDateAndActive(String name, ZonedDateTime birthDate, Boolean active);
- List<User> findByNameOrderByNameAsc(String name);

JPQL

- Java Persistence Query Language (JPQL) is an object model focused query language similar in nature to SQL.

- JPQL understands notions like inheritance, polymorphism and association.

- JPQL is a heavily-inspired-by a subset of HQL. A JPQL query is always a valid HQL query, the reverse is not true, however.

- Prevents SQL injection.

JPQL syntax

- CLAUSES:
    - SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY
- OPERATORS:
    - Navigation operator (.)
- Arithmetic operators:
    - * (multiplication), / (division), + (addition) and - (subtraction).
- Comparison operators:
    - =, <>, <, <=,>, >=, IS [NOT] NULL, [NOT] BETWEEN,
- Logical operators:
    - AND, OR, NOT.

Criteria query

- Criteria API is a programmatic approach to query instead of string-based approach as in JPQL.
- Good for Dynamic queries.

JPQL

```
Query query =
   entityManager.createQuery("select m
   from Member m where m.memberNumber
   =:number");
```

Criteria API

```
CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder
();

CriteriaQuery<Member> query=

criteriaBuilder.createQuery(Member.class);

Root<Member> memberRoot = query.from(Member.class);

query.select(memberRoot);

query.where(criteriaBuilder.equal(memberRoot.get("memberNumber"),
   number) );
```

EXAMPLES

```
@Query(value = "SELECT e FROM Employee e WHERE e.lastName = :lastname")
public List<Employee> findByLastName(String lastname);
```

Parent-Child operations (Cascade Types)

- PERSIST
  - Propagates the persist operation from a parent to a child entity.
- MERGE
  - Copies the state of the given object onto the persistent object with the same identifier.
- DETACH
  - The child entity will also get removed from the persistent context.

Parent-Child operations (Cascade Types)

- REMOVE
  - Removes the row corresponding to the entity from the database and also from the persistent context.
- REFRESH
  - The child entity also gets reloaded from the database whenever the parent entity is refreshed.
- ALL
  - Shortcut for cascade={PERSIST, MERGE, REMOVE, REFRESH}

Fetch Type

- Eager Loading
  - It fetches the child entities along with parent.
- Lazy Loading
  - It fetches the child entities lazily, that is, at the time of fetching parent entity it just fetches proxy of the child entities and when you access any property of child entity then it is actually fetched by hibernate.

```
@OneToMany(fetch = FetchType.LAZY,    mappedBy = "user")
public List<Order> orders;
```

Hibernate FETCH Strategies

- Select
- Join
- Subselect
  - BatchSize

Hibernate FETCH Strategy -- select

- Default

- N+1 Fetches
- a second SELECT [per parent N] is used to retrieve the associated collection.

- @Fetch(FetchMode.SELECT)

To Be Avoided

Hibernate FETCH Strategy -- JOIN

- associated collections are retrieved in the same SELECT.
- uses an OUTER JOIN.
- 1 Fetch
- EAGER
- @Fetch(FetchMode.JOIN)

need to watch collection sizes; can be useful strategy.

HIBERNATE FETCH STRATEGY -- JOIN

- ALWAYS Causes an EAGER fetch of the child collections.
- This is because the characteristic of a Join is ONE fetch Parent & Child TOGETHER
- This can only be accomplished by loading the child collection when the parent is fetched [~= EAGER fetch]

It can be implemented Manually to use LAZY initialization.

Hibernate FETCH Strategy -- subselect

- a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch
- 2 Fetches
    - ORM will do ONE Fetch for All Parents
    - ORM will do ONE Fetch for All child collections

Hibernate FETCH Strategy -- subselect

- @Fetch(FetchMode.SUBSELECT)
- depends on the "parent" query. If parent Query is complex, it could have performance impacts.
- If fetch=FetchType.LAZY need to "hydrate" children

Hibernate FETCH Strategy -- batch

- Optimization of Select Fetching
- Associated collections are fetched according to declared Batch Size(N/Batch Size) + 1
- @BatchSize(size=n)
- Batch fetching is often called a blind-guess optimization
- # of Fetches “unknown” UNLESS size of parent is constant

INHERITANCE

- Single Table
- Joined Tables [Table Per Subclass]
- Table per Class

Inheritance - single table

- Contains all columns for Super Class & ALL Sub Classes
- De-normalized schema
- Efficient queries
- Difficult to maintain as the number of columns increase.
- Fast polymorphic queries

Inheritance - joined table

- Table per Subclass
- Normalized schema
- Similar to OO classes
- Less efficient queries.
- Effective if hierarchy isn't too deep.

Inheritance - table Per class

- Super Class is replicated in each subclass table.
- Uses UNION instead of JOIN.
- All needed columns in each table.

Main points

- Spring provides a Transactional capability for ORM applications.
- The mechanism of transcending allows the individual to tap into Transcendental Consciousness and enlivens its qualities in activity.