Spring Data JPA 2

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference

Limiting Query Results

- **쿼리 결과 제한**
  - first나 top 키워드를 사용하여 검색 결과 중 일부만 조회

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

  - 페이징이나 슬라이싱을 함께 사용할 경우, 제한이 적용된 조회 결과에서 수행

Entity Graph

- **패치전략**
  - 기본적으로 EAGER나 LAZY 둘 중 하나를 사용
  - static하게 결정되므로 런타임 시 동적으로 전략 수정이 불가
  - 전략이 수정되면 어플리케이션 수정 및 컴파일
  - EntityGraph의 목적
    - 연관된 엔티티를 패치할 때 성능 향상
    - 객체 그래프 탐색의 형태로 데이터를 조회할 수 있음
    - 모든 연관관계를 Lazy로 설정하면 함께 조회시 추가적은 JPQL을 작성해야 하지만 이를 EntityGraph를 사용하면 더 편리함

https://www.baeldung.com/jpa-entity-graph

Entity Graph

```java
@Entity
public class User {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
  private String email;
  //...
}
```

```java
@Entity
public class Comment {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String reply;

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn
  private Post post;

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn
  private User user;
  //...
}
```

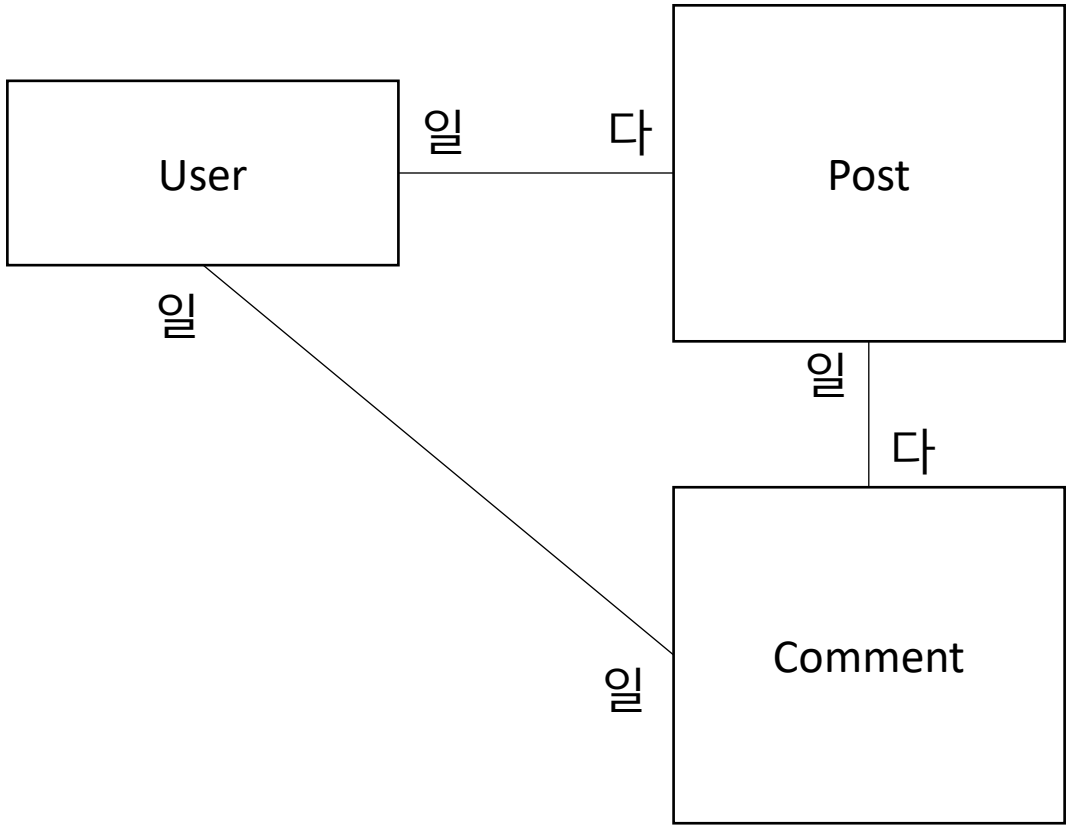
```java
@Entity
public class Post {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String subject;
  @OneToMany(mappedBy = "post")
  private List<Comment> comments = new ArrayList<>();

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn
  private User user;

  //...
}
```

```
Post  ->  user:User
      ->  comments:List<Comment>
            comments[0]:Comment -> user:User
            comments[1]:Comment -> user:User
```

https://www.baeldung.com/jpa-entity-graph

Entity Graph

```
Post  ->  user:User
      ->  comments:List<Comment>
            comments[0]:Comment -> user:User
            comments[1]:Comment -> user:User
```



- 어노테이션으로 FetchType.LAZY 혹은 FetchType.EAGER를 지정했으므로 패치 타입이 고정
- 런타임때 변경시킬 방법이 없음

https://www.baeldung.com/jpa-entity-graph

Entity Graph

- **NamedEntityGraph**
  - attributeNodes: 함께 조회될 엔티티 그래프를 나열
  - NamedAttributeNode: taget 엔티티가 조회될 때, 함께 조회될 연관 엔티티

```
@NamedEntityGraph(
      name = "post-entity-graph",
      attributeNodes = {
} @NamedAttributeNode("subject"),
            @NamedAttributeNode("user"),
            @NamedAttributeNode("comments"),

)
@Entity
public class Post {

   @OneToMany(mappedBy = "post")
   private List<Comment> comments = new ArrayList<>();

   //...
}
```

https://www.baeldung.com/jpa-entity-graph

Entity Graph

- **NamedEntityGraph**
  - 앞선 예제에서 더 나아가 Post에 속한 Comment를 작성한 User도 함께 조회(subgraphs 사용)

```
@NamedEntityGraph(
    name = "post-entity-graph-with-comment-users",
    attributeNodes = {
        @NamedAttributeNode("subject"),
        @NamedAttributeNode("user"),
        @NamedAttributeNode(value = "comments", subgraph = "comments-subgraph"),
    },
    subgraphs = {
        @NamedSubgraph(
            name = "comments-subgraph",
            attributeNodes = {
                @NamedAttributeNode("user")
            }
        )
    }
)
@Entity
public class Post {

  @OneToMany(mappedBy = "post")
  private List<Comment> comments = new ArrayList<>();
  //...
}
```

Post를 조회할 때,
글쓴이
  댓글
    댓글의 글쓴이
를 한 번에 조회

://www.baeldung.com/jpa-entity-graph

Entity Graph

- **Defining an Entity Graph with the JPA API**
  - 어노테이션을 이용하여 명시적으로 엔티티 그래프를 구성하는 것이 아닌 JPA API를 이용하여 동적으로 구성 가능
  - Post에 subject와 user 추가

```
EntityGraph<Post> entityGraph = entityManager.createEntityGraph(Post.class);

entityGraph.addAttributeNodes("subject");
entityGraph.addAttributeNodes("user");
```

https://www.baeldung.com/jpa-entity-graph

Entity Graph

- **Defining an Entity Graph with the JPA API**
  - comments와 연관된 user를 포함하는 subgraph생성

```
entityGraph.addSubgraph("comments")
      .addAttributeNodes("user");
```

  - 조회

```
Map hints = new HashMap()
hints.get("javax.persistence.fetchgraph", entityGraph);
Order order = em.find(Post.class, postId, hints);
```

https://www.baeldung.com/jpa-entity-graph

Locking

- **Lock 모드 설정**
  - 여러 트랜잭션에 의해 레코드가 수정될 경우, 동시성 문제로 인해 데이터 유실 혹은 일관성이 깨지는 문제가 발생

```
interface UserRepository extends Repository<User, Long> {

  // Plain query method
  @Lock(LockModeType.READ)
  List<User> findByLastname(String lastname);
}
```

```
public enum LockModeType {
   READ,
   WRITE,
   OPTIMISTIC,
   OPTIMISTIC_FORCE_INCREMENT,
   PESSIMISTIC_READ,
   PESSIMISTIC_WRITE,
   PESSIMISTIC_FORCE_INCREMENT,
   NONE;

   private LockModeType() {
   }
}
```

https://www.baeldung.com/jpa-pessimistic-locking

Locking

- **Lock 종류**
  - 낙관적 잠금
    - 트랜잭션간 동시성 문제는 거의 발생되지 않을껄?
    - 커밋 전, 각 트랜잭션은 다른 트랜잭션이 나의 트랜잭션에서 수정된 사항을 변경하지 않았는지 확인
    - 만약 수정에서 충돌이 발생한다면 Rollback
    - 읽기, 쓰기에 제약을 두는 것이 아닌 충돌 감지 수준의 처리
  - 비관적 잠금
    - 보나마나 동시성 문제가 발생할꺼야
    - write 전, 먼저 권한을 얻음
    - 만약 권한을 얻지 못하면 수정할 수 없음
  - 낙관적 락은 커밋 전에 충돌을 감지. 만약, 충돌이 자주 발생하면? → 충돌 감지+ 예외처리 + Rollback → 차라리 엄격하게 접근 권한을 관리하자(비관적 락)

https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#locking

Locking

- **Lock 종류**
  - Exclusive lock (배타적 잠금)
    - 쓰기 잠금
    - 특정 트랜잭션에서 데이터를 변경하고자 할 때(write) 해당 테이블 or 레코드를 다른 트랙잭션에서 읽거나 쓰지 못하게 막음
  - Shared lock (공유 잠금)
    - 읽기 잠금
    - 특정 트랜잭션에서 데이터를 읽고자 할 때(read), 다른 트랜잭션도 함께 읽을 수 있도록 허용
    - 단, 변경은 허용하지 않음

https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#locking

Locking

- **Optimistic locking**
  - JPA에서는 @Version이나 timestamp를 이용하여 낙관적 락을 구현
  - 특정 필드에 @Version이 붙은 필드를 추가(엔티티 클래스에 하나의 @Version 명시)

```
@Entity(name = "Person")
public static class Person {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "`name`")
    private String name;

    @Version
    private long version;

    //Getters and setters are omitted for brevity

}
```

@Version을 붙일 수 있는 타입

- int or Integer
- short or Short
- long or Long
- java.sql.Timestamp

https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#locking

Locking

- **LockModeType**

Type	LockModeType	LockMode
OPTIMISTIC	NONE	락 사용 안함
	OPTIMISTIC	Write와 더불어 Read 에서도 긍정적 락 수행
	OPTIMISTIC_FORCE_INCREMENT	엔티티가 수정되지 않더라도 자동으로 버전 증가
	READ	OPTIMISTIC과 같음
	WRITE	OPTIMISTIC_FORCE_INCREMENT와 같음
PESSIMISTIC	PESSIMISTIC_READ	shared lock
	PESSIMISTIC_WRITE	Row Exclusive Lock
	PESSIMISTIC_FORCE_INCREMENT	Row Exclusive Lock + 버전 자동 증가

https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#locking

Customizing Individual Repositories

- **데이터 JPA가 제공해주는 것이 충분하지 않을 때**
  - Ex) QueryDSL 사용

```
interface CustomizedUserRepository {
  void someCustomMethod(User user);
}

class CustomizedUserRepositoryImpl implements CustomizedUserRepository {

  public void someCustomMethod(User user) {
    // Your custom implementation
  }
}

interface UserRepository extends CrudRepository<User, Long>, CustomizedUserRepository {

  // Declare query methods here
}
```