

## 4주차

### JPA 이해

#### 1. ORM이란

**ORM : Object-Relational Mapping**

**Object :** "객체" 지향 언어(자바,파이썬)

**Relational :** "관계형" 데이터베이스(H2,MySQL)

객체 지향언어와 관계형 데이터베이스에서 사용하는 언어가 달라서 사이에서 통역을 해주는 역할  
DB언어 예시) "Insert Into", "Select \* from" 등

#### 2. JPA는?

자바의 ORM 이다.

JPA 예시) @Entity, @Id, @Column등

- 하이버네이트 (Hibernate)?

JPA 를 실제로 구현하는 구현체

### JPA 영속성 컨텍스트 이해

#### 1. 영속성 컨텍스트?

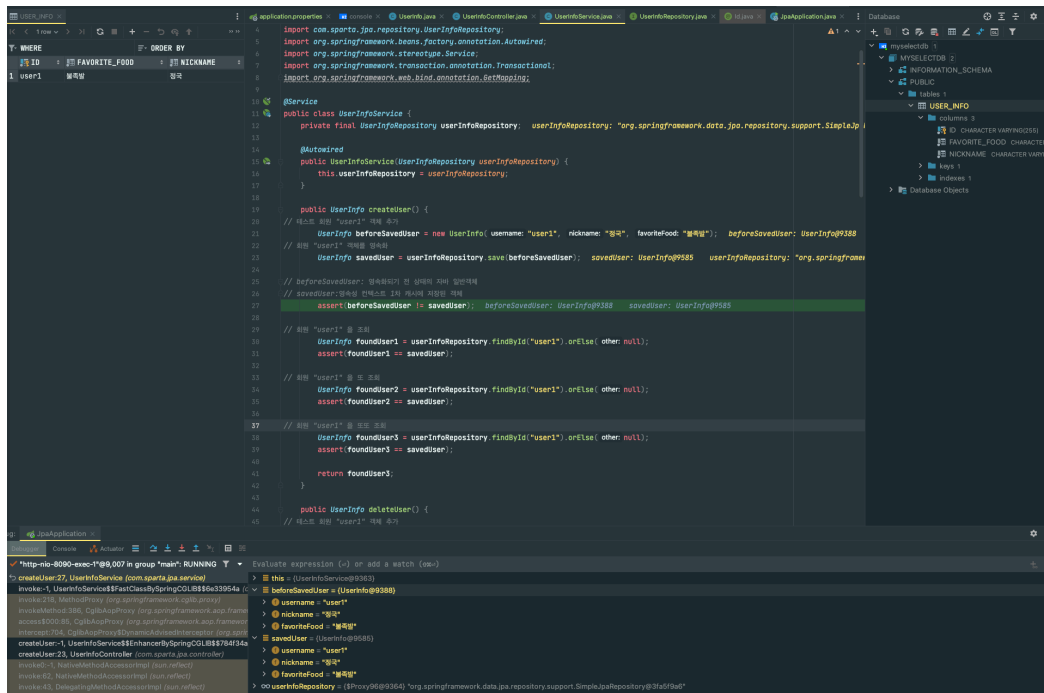
- JPA

객체 - ORM - DB

객체 - 영속성 컨텍스트 매니저(entity context manager) - DB

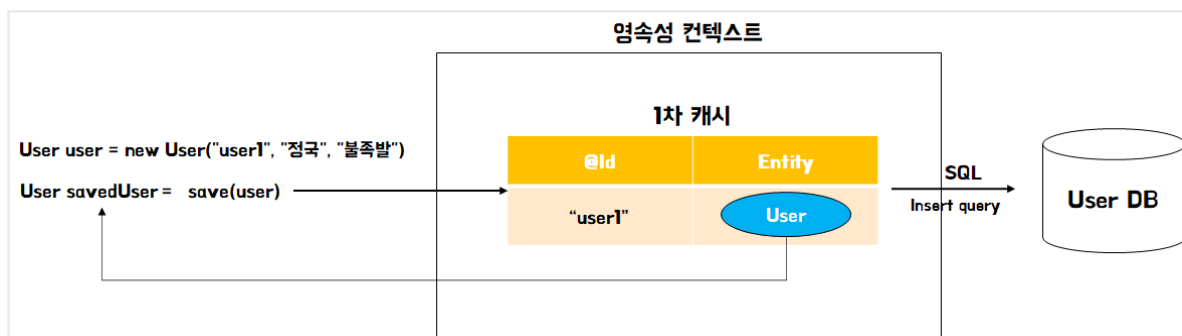
- 영속성 컨텍스트 매니저 : 객체랑 DB 사이에 소통을 효율적으로 관리
- PK (Primary Key)  
테이블에서 각 row(줄)마다 가져야 하는 유일무이한 값 (Null 허용되지 않음)
- PK는 테이블에서 각 row마다 Null이 허용되지 않는 유일무이한 값을 가진다면 자연키,  
인조키 설정 가능 (일반적으로 ID 를 PK로 설정)  
자연키 ex) USERNAME, EMAL  
인조키 ex) ID

### Debugging 실습



## JPA 영속성 컨텍스트 1차 캐시 이해

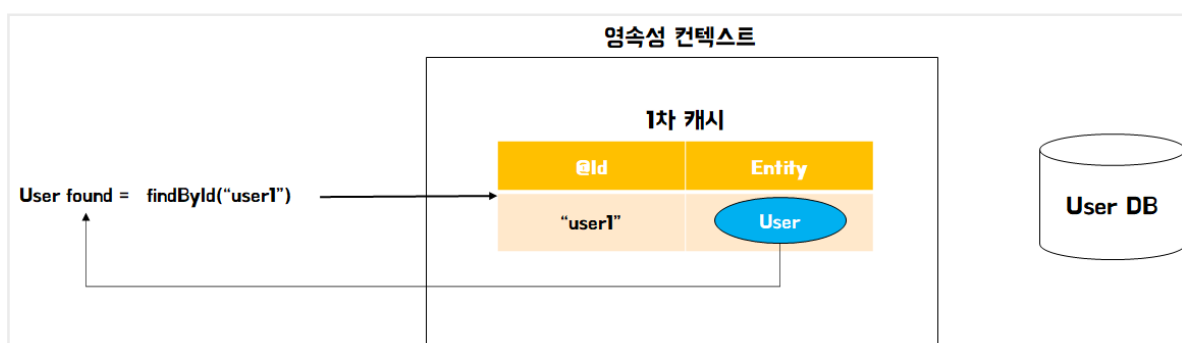
### 1. 영속성 컨텍스트 1차 캐시 - Entity 저장 시



객체를 만들어 save 를 하면 영속성 컨텍스트에 1차 캐시에 새로운 객체를 만들어 그안에 id 값이랑 Entity값을 넣어 1차캐시에서 가지고 있고 그리고 받은 데이터를 DB에 전달도함

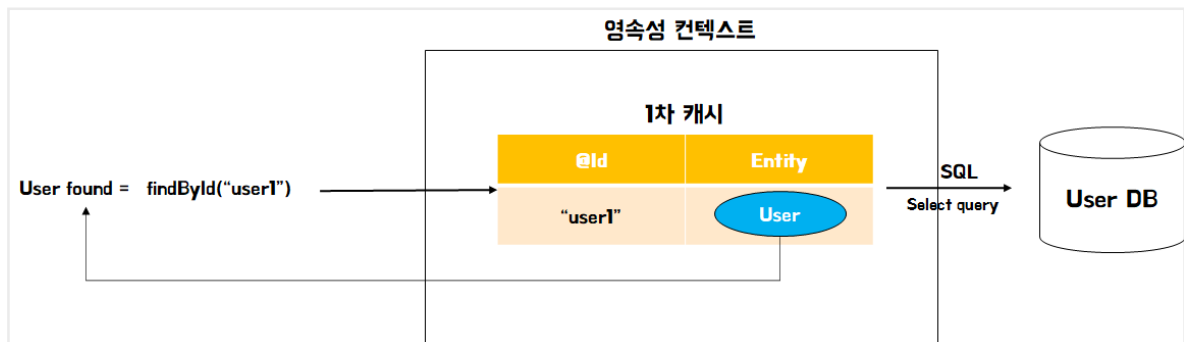
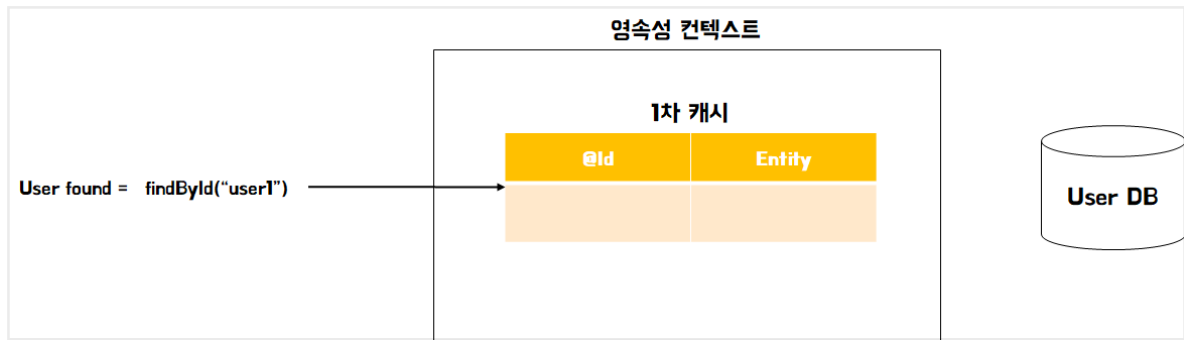
### - Entity 조회 시

#### 1. 1차 캐시에 조회하는 Id 가 존재할 경우



1차 캐시에 있는 값을 그대로 전달한다

#### 2. 1차 캐시에 조회하는 Id 가 없는경우



DB에서 조회하는 Id인 데이터를 찾아온 다음 1차 캐시에 넣고 그다음 1차 캐시에 있는 값을 전달한다

- '1차 캐시' 사용의 장점
  1. DB 조회 횟수를 줄임 (DB 조회를 많이 할 수록 성능이 낮아짐)
  2. '1차 캐시' 를 사용해 DB row 1개당 객체 1개가 사용됨을 보장(객체 동일성 보장)

*beforeSavedUser*: 영속화되기 전 상태의 자바 일반객체

*savedUser*: 영속성 컨텍스트 1차 캐시에 저장된 객체

*foundUser1*: DB에서 데이터 조회한객체

```
// 테스트 회원 "user1" 객체 추가
UserInfo beforeSavedUser = new UserInfo( username: "user1", nickname: "정국", favoriteFood: "불족발");
// 회원 "user1" 객체를 영속화
UserInfo savedUser = userInfoRepository.save(beforeSavedUser); savedUser: UserInfo@9712
```

```

beforeSavedUser = {UserInfo@9706}
  > f username = "user1"
  > f nickname = "정국"
  > f favoriteFood = "불족발"
savedUser = {UserInfo@9712}
  > f username = "user1"
  > f nickname = "정국"
  > f favoriteFood = "불족발"
  
```

저장시 객체 값만 저장되고 1차 캐시에서 객체가 새로생성된것 확인

```
// 회원 "user1" 을 조회
UserInfo foundUser1 = userInfoRepository.findById("user1").orElse( other: null);
```

```
✓ ≡ savedUser = {UserInfo@9712}
> f username = "user1"
> f nickname = "정국"
> f favoriteFood = "불족발"
✓ ≡ foundUser1 = {UserInfo@9712}
> f username = "user1"
> f nickname = "정국"
> f favoriteFood = "불족발"
```

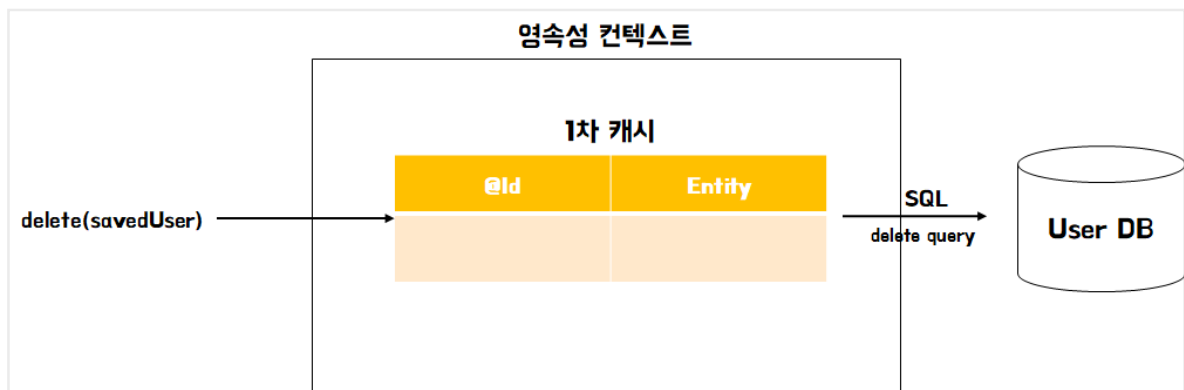
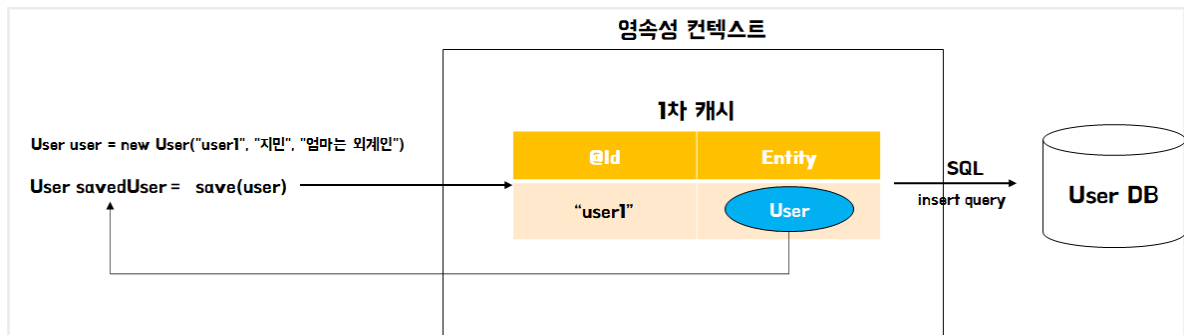
조회시 1차캐시 객체 주소 값 그대로 조회된 내용 확인

```
✓ ≡ foundUser1 = {UserInfo@9712}
> f username = "user1"
> f nickname = "정국"
> f favoriteFood = "불족발"
✓ ≡ foundUser2 = {UserInfo@9712}
> f username = "user1"
> f nickname = "정국"
> f favoriteFood = "불족발"
✓ ≡ foundUser3 = {UserInfo@9712}
> f username = "user1"
> f nickname = "정국"
> f favoriteFood = "불족발"
```

조회 시 1차캐시에 조회하는 Id가 있다면 1차 캐시에 있는 데이터를 그대로 가져옴

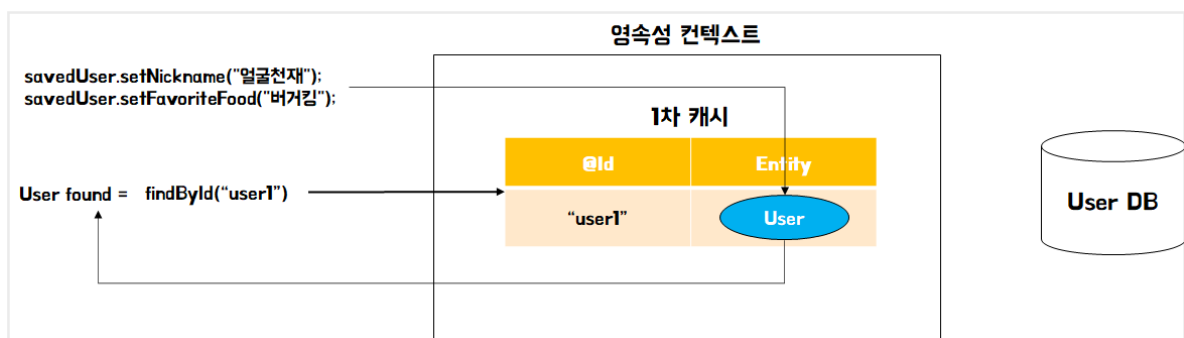
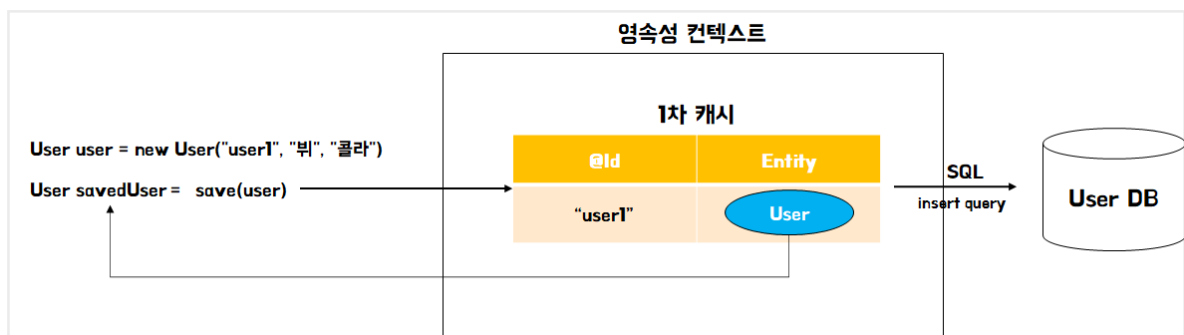
Tip. 1차 캐시는 스프링에서만 동작하나 설정에 따라 바뀔 수 있다

- Entity 삭제



1차캐시에 Id값을 찾아서 삭제하고 DB에 같은 Id값을 찾아 삭제한다, 만약 1차캐시에 같은 Id값 데이터가 없다면 DB를 바로 조회하고 Id값이 있으면 삭제한다

#### - Entity 업데이트 실패



set설정만 진행하면 1차 캐시에만 Id"user1"에 대한 변경요청 내용이 바뀌고 실제 DB에는 변경되지 않음  
 실제 예제 코드

```

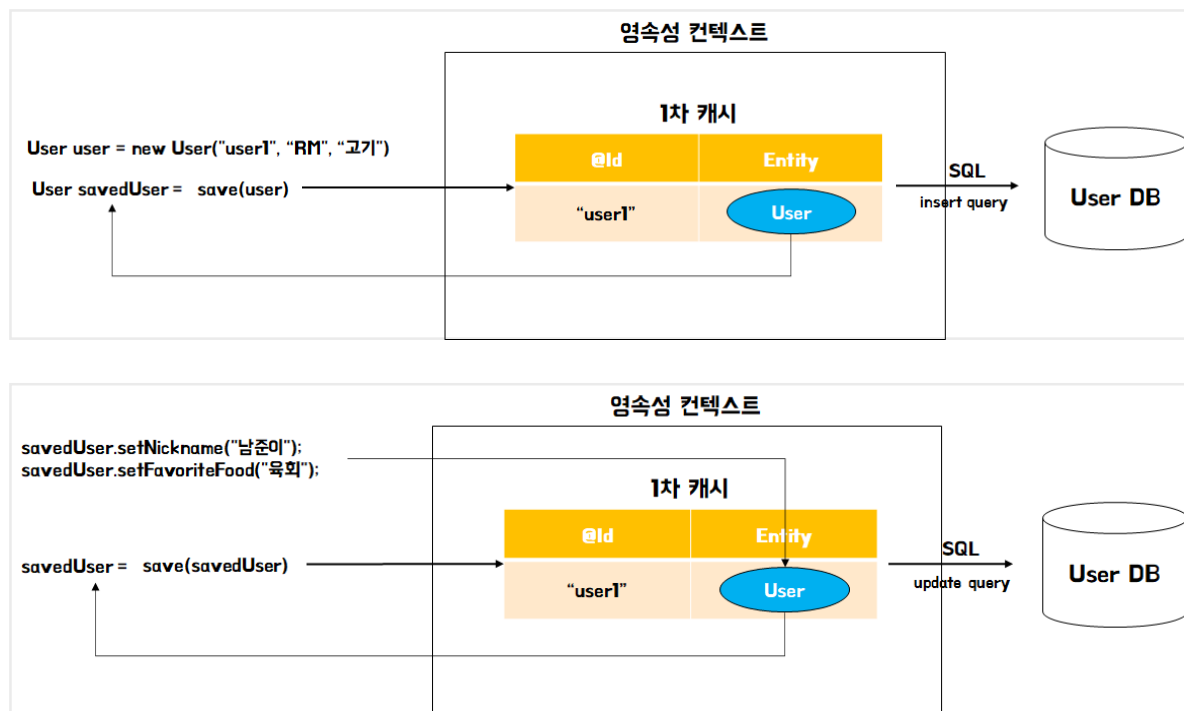
7 public UserInfo updateUserFail() {
8     // 회원 "user1" 객체 추가
9     UserInfo user = new UserInfo( username: "user1", nickname: "뽀", favoriteFood: "콜라"); user: UserInfo@9105
10    // 회원 "user1" 객체를 영속화
11    UserInfo savedUser = userInfoRepository.save(user); user: UserInfo@9105 savedUser: UserInfo@9284
12
13    // 회원의 nickname 변경
14    savedUser.setNickname("얼굴천재");
15    // 회원의 favoriteFood 변경
16    savedUser.setFavoriteFood("버거킹");
17
18    // 회원 "user1" 을 조회
19    UserInfo foundUser = userInfoRepository.findById("user1").orElse( other: null); foundUser: UserInfo@928
20    // 중요! foundUser 는 DB 값이 아닌 1차 캐시에서 가져오는 값
21    assert(foundUser == savedUser); savedUser: UserInfo@9284 foundUser: UserInfo@9284
22    assert(foundUser.getUsername().equals(savedUser.getUsername()));
23    assert(foundUser.getNickname().equals(savedUser.getNickname()));
24    assert(foundUser.getFavoriteFood().equals(savedUser.getFavoriteFood()));
25
26    return foundUser;
27 }
28
29 public UserInfo updateUser1() {
30     // 테스트 회원 "user1" 생성
31     UserInfo user = new UserInfo( username: "user1", nickname: "RM", favoriteFood: "고기");
32     // 회원 "user1" 객체를 영속화
33     UserInfo savedUser1 = userInfoRepository.save(user);
34
35     // 회원의 nickname 변경
36     savedUser1.setNickname("남준이");
37     // 회원의 favoriteFood 변경
38     savedUser1.setFavoriteFood("육회");
39
40     // user1 을 저장
41     UserInfo savedUser2 = userInfoRepository.save(savedUser1);
42 }

```

Y- WHERE ORDER BY  
 ID : FAVORITE\_FOOD : NICKNAME  
 1 user1 콜라 뽀

jp \*main\*: RUNNING Evaluate expression (=) or add a watch (ow=)  
 m.sparta.jpa.service) > this = (UserService@9083)  
 BySpringCGLIB\$\$6e33954a (c > user = (UserInfo@9105)  
 amework.cglib.proxy) > username = "user1"  
 g.springframework.aop.frame > nickname = "뽀"  
 ework.aop.framework > favoriteFood = "콜라"  
 AdvisedInterceptor (org.spr > savedUser = (UserInfo@9284)  
 hancerBySpringCGLIB\$\$c99f > username = "user1"  
 (com.sparta.jpa.controller) > nickname = "얼굴천재"  
 (sun.reflect) > favoriteFood = "버거킹"  
 (sun.reflect) > foundUser = (UserInfo@9284)  
 mpl (sun.reflect) > username = "user1"  
 4 (org.springframework.web > nickname = "얼굴천재"  
 ethod (org.springframework > favoriteFood = "버거킹"  
 HandlerMethod (org.springfr > oo userInfoRepository = (\$Proxy96@9084) \*org.springframework.data.jpa.repository.support.SimpleJpaRepository@d4e39f5"  
 ippingHandlerAdapter (org.sp

## - Entity 업데이트 방법 (1)



set설정 이후 save 까지 해야 DB에 저장됨

Tip. save요청을 하면 자동으로 처음 DB에 넣을 땐 insert를 해주고 DB 데이터를 바꿀땐 update를 해준다.

## 실제 예제 코드

```

public UserInfo updateUserInfo() {
    // 테스트 회원 "user1" 생성
    UserInfo user = new UserInfo(username: "user1", nickname: "RM", favoriteFood: "고기");
    // 회원 "user1" 객체를 영속화
    UserInfo savedUser1 = userInfoRepository.save(user);
    // 회원의 nickname 변경
    savedUser1.setNickname("남준이");
    // 회원의 favoriteFood 변경
    savedUser1.setFavoriteFood("육회");

    // user1 을 저장
    UserInfo savedUser2 = userInfoRepository.save(savedUser1);
    assert(savedUser1 == savedUser2);
    return savedUser2;
}

```

WHERE: ID, FAVORITE\_FOOD, NICKNAME  
ORDER BY

1 user1 육회 남준이

Running: Evaluate expression (-) or add a watch (ex=)

userInfoRepository = (\$Proxy96@9079) "org.springframework.data.jpa.repository.support.SimpleJpaRepository@e61e6f5"

### - Entity 업데이트 방법 (2)

@Transactional 을 추가하는 방법

userInfoRepository.save() 함수를 호출하지 않아도, 함수가 끝나는 시점에 변경된 부분을 알아서 업데이트 해 줌 (이를 "Dirty check" 라고 함)

Tip. @Transactional은 SQL 문장들을 따로 모아놨다가 나중에 한번에 보내는 역할

## DB 의 연관관계 이해

- JPA 가 제공하는 연관관계는 결국 DB 의 연관관계를 표현하기 위함
- 따라서 먼저 DB 의 연관관계를 이해해야 함

## 예제

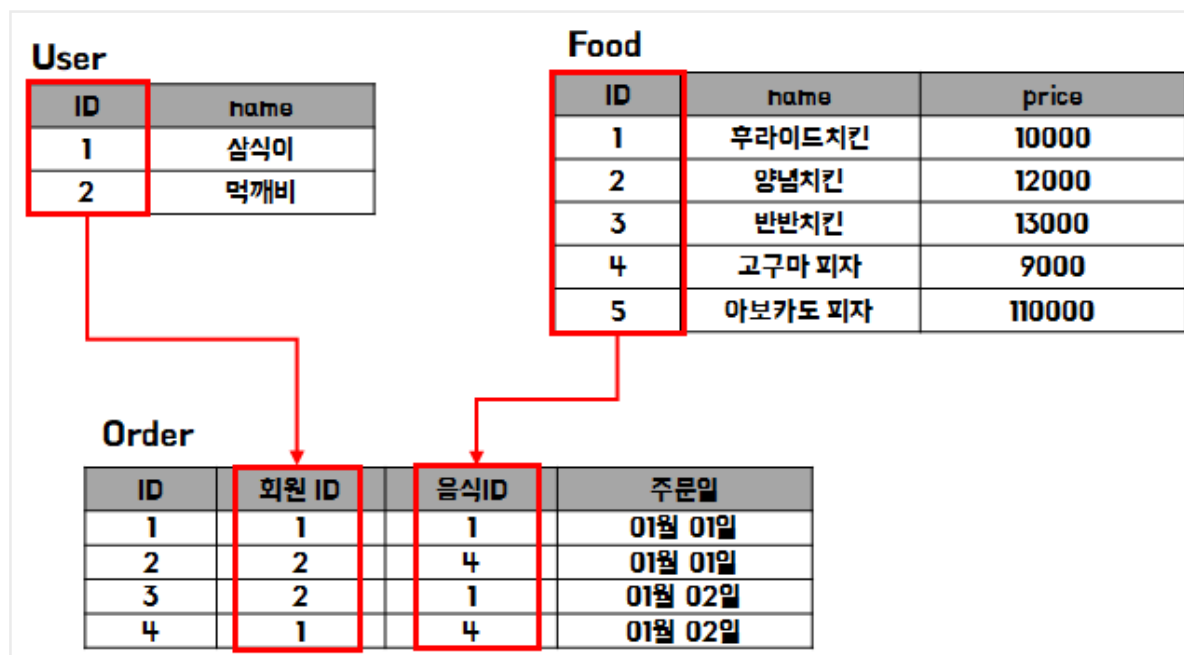
User		
ID	name	foodname
1	삼식이	후라이드치킨
2	먹깨비	
3	삼식이	양념치킨

문제점: 회원 중복

Food			
ID	name	price	userID
1	후라이드치킨	10000	1
2	양념치킨	12000	1
3	반반치킨	13000	
4	고구마 피자	9000	
5	아보카도 피자	110000	
6	후라이드치킨	10000	2

문제점: 메뉴 중복

결론 : '주문'을 위한 테이블이 필요 → Order 테이블 추가



- 회원 1명은 주문 N개를 할 수 있다.
  - 회원 : 주문 = 1 : N 관계
- 음식 1개는 주문 N개에 포함될 수 있다.
  - 음식 : 주문 = 1 : N 관계
- 결론적으로
  - 회원 : 음식 = N : N 관계

## JPA 연관관계

- JPA 연관관계 설정방법

JPA 의 경우는 Entity 클래스의 필드 위에 연관관계 어노테이션 (@) 을 설정해 주는 것만으로 연관관계가 형성

ex)'음식 배달 서버'를 개발한다고 가정



# 연관관계

Default view

Aa 관계	코드 선언	Entity	예
일대다 (1:N)	@OneToMany	Order (1) : Food (N)	배달 주문 1개에 음식 여러개 선택 가능
다대일 (N:1)	@ManyToOne	Owner (N) : Restaurant(1)	음식점 주인 여러명이 하나의 음식점을 소유 가능
일대일 (1:1)	@OneToOne	Order (1) : Coupon (1)	배달 주문 1개 주문 시, 쿠폰 1개만 할인 적용 가능
다대다 (N:N)	@ManyToMany	User (N) : Restaurant(N)	고객은 음식점 여러개 찜 가능 음식점은 고객 여러명에게 찜 가능

## ▼ 15) JPA 코드 구현

중요) 항상 Entity 본인 중심으로 관계를 생각!

- 주문 (Order) 코드

```
@Entity
public class Order {
    @OneToMany
    private List<Food> foods;

    @OneToOne
    private Coupon coupon;
}
```

- 음식점주 (Owner)

```
Java ▼
@Entity
public class Owner {
    @ManyToOne
    Restaurant restaurant;
}
```

복사

- 고객 (User)

```
@Entity
public class User {
    @ManyToMany
    List<Restaurant> likeRestaurants;
}
```

## Spring Data JPA 이해

- JPA 를 편리하게 사용하기 위해 스프링에서 JPA 를 Wrapping
- 스프링 개발자들이 JPA 를 사용할 때 필수적으로 생성해야 하나, 예상 가능하고 반복적인 코드들 → Spring Data JPA 가 대신 작성

- Repository 인터페이스만 작성하면, 필요한 구현은 스프링이 대신 진행

Ex)

- Spring Data JPA) 상품 Repository 생성

```
public interface ProductRepository extends JpaRepository<Product, Long> {  
}
```

- Spring Data JPA) 기본 제공해 주는 기능

```
// 1. 상품 생성  
Product product = new Product(...);  
productRepository.save(product);  
  
// 2. 상품 전체 조회  
List<Product> products = productRepository.findAll();  
  
// 3. 상품 전체 개수 조회  
long count = productRepository.count();  
  
// 4. 상품 삭제  
productRepository.delete(product);
```

- ID 외의 필드에 대한 추가 기능은 interface 만 선언해 주면, 구현은 Spring Data JPA 가 대신!!

```
public interface ProductRepository extends JpaRepository<Product, Long> {  
    // (1) 회원 ID 로 등록된 상품들 조회  
    List<Product> findAllByUserId(Long userId);  
  
    // (2) 상품명이 title 인 관심상품 1개 조회  
    Product findByTitle(String title);  
  
    // (3) 상품명에 word 가 포함된 모든 상품들 조회  
    List<Product> findAllByTitleContaining(String word);  
  
    // (4) 최저가가 fromPrice ~ toPrice 인 모든 상품들을 조회  
    List<Product> findAllByLpriceBetween(int fromPrice, int toPrice);  
}
```

Tip. 아래 링크에 추가기능 interface선언 방법 자세히 나옴 (영문...)

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods>

사용법 예시

```
public interface UserRepository extends JpaRepository<User, String> {
    void findByFavoriteFood(String food);
}
```

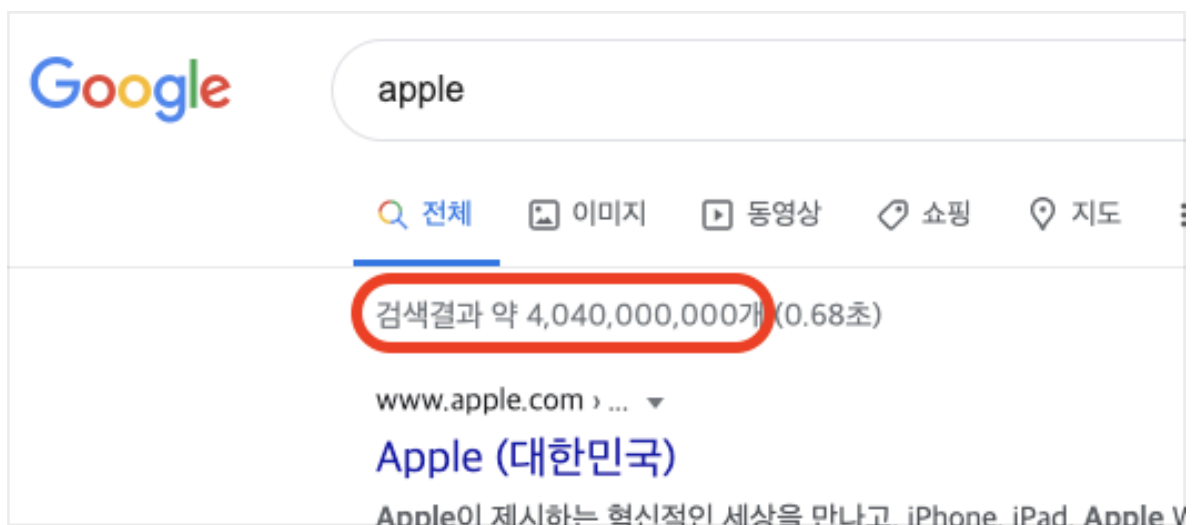
위에 FavoriteFood 가 User클래스에 멤버변수 이름 favoriteFood 랑 같아야함

```
@Setter
@Getter // get 함수를 일괄적으로 만들어줍니다.
@NoArgsConstructor // 기본 생성자를 만들어줍니다.
@Entity // DB 테이블 역할을 합니다.
public class User {
    // nullable: null 허용 여부
    // unique: 중복 허용 여부 (false 일때 중복 허용)
    @Id
    @Column(name = "id", nullable = false, unique = true)
    private String username;

    @Column(nullable = false, unique = false)
    private String nickname;

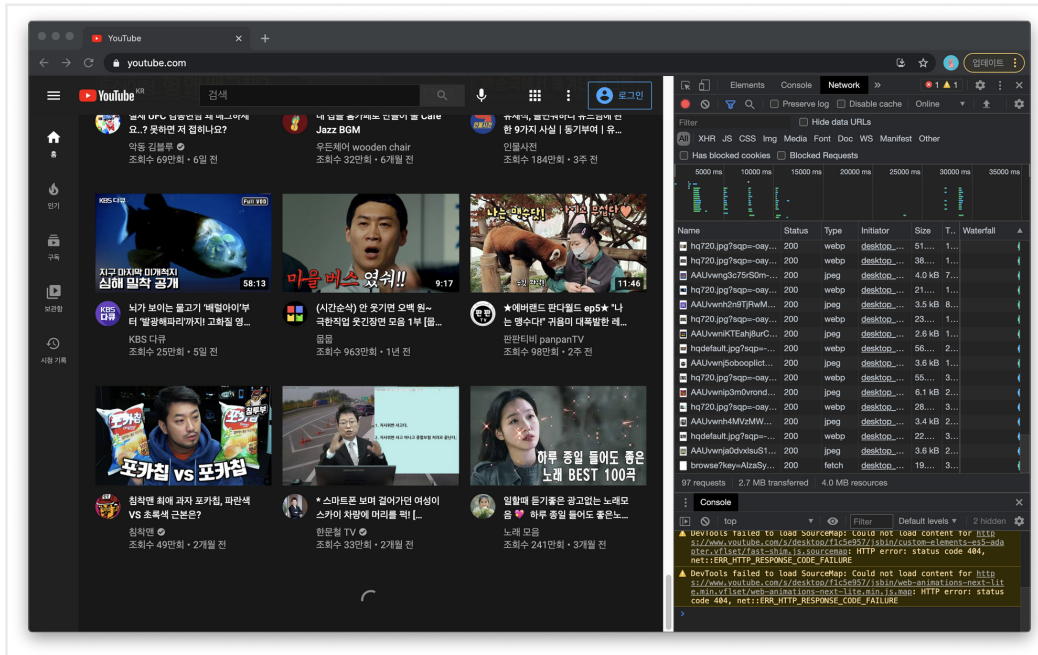
    @Column(nullable = false, unique = false)
    private String favoriteFood;
```

페이징 및 정렬 설계





페이징,페이지네이션: 몇개씩 끊어서 보이게 하는것



Infinite Scroll : 페이지네이션을 한페이지에서 계속 끊기지 않게 해서 보여주는 기능