

# **Spring Boot Seminar - 1**

**Wafflestudio Rookies seminar 2024**

**Sangmin Kim, 2024**


# Seminar - 1 목차

- 과제 0 리뷰
- RDBMS
- Spring Data JPA
- 과제 1 안내

# 과제 0 리뷰

## Kotlin 관련


32 + `if(authorizationHeader != null && authorizationHeader.substring(0, 7) == "Bearer ")`

 sangggggg 5 days ago

...

substring 은 string 문자열의 길이가 인덱스 범위보다 짧다면 IndexOutOfBoundsException 을 뱉을 수도 있어요. (어쩌다 잘못된 헤더를 넣은 요청이 들어왔을 때 현재는 401 이 아닌 500 같은 다른 메시지가 뜨게 될거예요, 테스트 해보셔도 좋습니다)

요건 `.take(7)` 로 안전하게 처리해줄 수 있습니다.



 sangggggg 2 days ago Author ...

[nit]

Suggested change

24 - `val user = UserDatabase.find { it.username == username }`

25 - `if (user == null) {`

26 -  `throw SignInUserNotFoundException()`

27 - `} else if (user.password != password) {`

28 -  `throw SignInInvalidPasswordException()`

29 - `}`

30 -

31 - `return user`

24 + `val user = UserDatabase.find { it.username == username } ?: throw`

25 +  `SignInUserNotFoundException()`

26 +

27 + `if (user.password != password) {`

28 +  `throw SignInInvalidPasswordException()`

29 + `}`

29 + `return user`

Commit suggestion



 sangggggg 2 days ago Author ...

`?:` 를 사용하시면 좀 더 간결한 코드를 작성하실 수 있습니다.



# 과제 0 리뷰

## Kotlin 관련

```
val status: Int = when (e) {  
    is AuthenticateException -> 401  
    is SignInInvalidPasswordException -> 401  
    is SignInUserNotFoundException -> 401  
    is SignUpBadPasswordException -> 400  
    is SignUpBadUsernameException -> 400  
    is SignUpUsernameConflictException -> 409  
}
```

```
41 + seminarDetails.forEach { detail ->  
42 +     when (detail.name) {  
43 +         "IOS", "ANDROID" -> mobileCnt += 1  
44 +         "REACT" -> webCnt += 1  
45 +         "DJANGO", "SPRING" -> serverCnt += 1  
46 +     }  
47 + }
```

davin111 yesterday

요렇게 enum 을 사용하면 String 의 오타를 방지할 수 있다는 당연한 장점 외에도 when 을 훨씬 유의미하게 쓸 수 있습니다.

```
seminarDetails.forEach {  
    when (it.type) {  
        SeminarDetail.Type.MOBILE -> mobileCnt++  
        SeminarDetail.Type.WEB -> webCnt++  
        SeminarDetail.Type.SERVER -> serverCnt++  
    }  
}
```

왜냐면 when 은 enum 처럼 유한한 set 를 각 분기가 exhaustive 하게(빠지는 케이스 없이) 다루도록 컴파일 타임에 강제하기 때문 입니다.

예컨대 아래처럼 Type 에 언젠가 새로운 값이 추가되면, 서버 내 수많은 코드들을 side effect 없이 변경하는 것이 중요할 수 있습니다.

```
enum class Type {  
    MOBILE,  
    WEB,  
    SERVER,  
    WATCH, // 새로 생긴 type  
}
```

이 때, 제안드린대로 코드를 작성했다면 아래와 같은 컴파일 에러가 발생해, 반드시 분기로서 WATCH 를 다루도록 강제할 수 있습니다.

```
seminarDetails.forEach {  
    when (it.type) {  
        'when' expression must be exhaustive, add necessary 'WATCH' branch or 'else' branch instead  
    }  
}
```

# 과제 0 리뷰

## Kotlin Linter 관련

- Indentation
- Spacing 등...
- 코드를 볼 때 불-편 한 것들
- 을 검사하고 고쳐주기도 하는 Linter
- 다음 과제 부터는 ktlint 를 통해 코틀린 코드를 lint 해서 제출해야 합니다.
  - 과제 README 에 설명을 추가할 예정

```
data class SignInRequest(val username:String, val password :String)
```

```
data class SignInRequest(  
    val username: String,  
    val password: String,  
)
```

# 과제 0 리뷰

## Git Remote Repository

- Feedback PR 은 머지하지 말아주세요!
  - 세미나장/조교들이 리뷰를 남기기 위해 Diff 를 확인하는 PR 입니다
- (마찬가지로) Feedback Branch 도 push 하지 말아주세요!
  - Main branch 위에서 작업하고 push 하시면 됩니다.



# 과제 0 리뷰

## Spring 관련



mscheong01 Pending

...

Spring Framework 동작상 controller 내에서 에러가 발생하면, 내부적으로 그 에러를 catch 해 exception handler 에 전달해 주게 됩니다. 따라서 여기서와 같이 explicit 하게 매번 에러를 catch 하여 handleException 함수 콜을 해주지 않으셔도 됩니다



Suggested change

```
15 - //TODO("구현해주세요")
16 - return try {
17 -     //UserService의 회원가입 호출
18 -     userService.signUp(request.username, request.password, request.image)
19 -     ResponseEntity.ok().build() //성공 시 200 0k 응답 반환
20 - } catch (e: UserException) {
21 -     //실패 시 그에 맞는 코드를 갖는 응답 반환
22 -     handleException(e)
23 - }

15 + //UserService의 회원가입 호출
16 + userService.signUp(request.username, request.password, request.image)
17 + return ResponseEntity.ok().build() //성공 시 200 0k 응답 반환
```

Commit suggestion ▼

Add suggestion to batch

```
12 fun signup(
13     @RequestBody request: SignUpRequest,
14 ): ResponseEntity<Unit> {
15 +
16 +     UserServiceImpl().signup(request.username, request.password, request.image)
```



sangggggg 2 minutes ago

Author ...

UserController 의 생성자에 Dependency Injection 으로 UserServiceImpl 이 꽂혀 있으니, 이걸 사용하면 되고 매번 인스턴스를 생성할 필요는 없습니다.  
DI 에 대한 설명은 다다음 세미나에서 좀 더 자세하게 드리도록 하겠습니다.

Suggested change

```
16 - UserServiceImpl().signup(request.username, request.password,
16 + userService.signup(request.username, request.password, request.image)
```

Commit suggestion ▼

Add suggestion to batch



sangggggg now

Author ...

사실 지금 정도 간단한 클래스 인스턴스는 그냥 매번 생성해서 써도 괜찮은거 아니냐? 성능에 큰 영향도 없을텐데?

라고 생각할 수 있습니다 (실제로도 그렇고요 ㅋㅋ),

하지만 앞으로는 DB Connection Pool 과 같이 공유되어 관리되어야 하는 인스턴스가 존재하고, 이에 의존하는 클래스들을 만들어야 하는 만큼, DI 로 주입된 객체를 쓰는게 바람직합니다. (이건 앞으로 DI 를 배우고, DB 를 배우면서 느낌을 알게 되실거 예요)



Reply...

# 과제 0 리뷰

## Spring 관련

```
8 +
9 +     if(UserDatabase.any { it.username == username }) {
10 +         throw SignUpUsernameConflictException()
11 +     }
12 +     if(username.length < 4) {
13 +         throw SignUpBadUsernameException()
14 +     }
15 +     if(password.length < 4) {
16 +         throw SignUpBadPasswordException()
17 +     }
```

Comment on lines +9 to +17



**sangggggg** now

Author ...

아주 사소하고, 이번 과제에서는 더욱 사소하지만

UserDatabase 를 조회하는 것은 DB Query 를 요구하고, DBQuery 와 같은 Network IO 는 단순 코드 실행에 비해서 소요 시간이 긴 편입니다. 따라서 length 로 검증을 먼저해서, length 부터 잘못된 사용자라면 Query 를 하기 전에 미리 throw 를 해 주는 것이 사소하지만 성능상 이점이 있을 것 같아요



Reply...



# RDBMS

## 데이터를 저장하는 규칙

- 테이블의 형태로 데이터를 저장
  - 데이터 하나 하나는 행으로, 항목의 속성은 열으로 저장.
- 테이블의 형태 (데이터의 구조) 를 정의할 수 있다.
  - 항목의 속성은 숫자일수도, 문자열 일수도...
- 테이블 간의 관계를 정의할 수 있다.

SUM		=SUM(B3:B12)			
	A	B	C	D	E
1		Sales in Each Quarter			
2	Product Name	Jan'2018	April'2018	July'2018	October'2018
3	ABC Mutton	\$ 2,667.60	\$ 4,013.10	\$ 4,836.00	\$ 6,087.90
4	Crab Meat	\$ 1,768.41	\$ 1,978.00	\$ 4,412.32	\$ 1,656.00
5	Camembert Pierrot	\$ 3,182.40	\$ 4,683.50	\$ 9,579.50	\$ 3,060.00
6	Ipoh Coffee	\$ 1,398.40	\$ 4,496.50	\$ 1,196.00	\$ 3,979.00
7	Hot Pepper Sauce	\$ 1,347.36	\$ 2,750.69	\$ 1,375.62	\$ 3,899.51
8	Hot Spiced Okra	\$ 1,509.60	\$ 530.40	\$ 68.00	\$ 850.00
9	Mozzarella di Giovanni	\$ 1,390.00	\$ 4,488.20	\$ 3,027.60	\$ 2,697.00
10	Sir Rodney's Scones	\$ 1,462.00	\$ 644.00	\$ 1,733.00	\$ 1,434.00
11	Steeleye Stout	\$ 1,310.40	\$ 1,368.00	\$ 1,323.00	\$ 1,273.50
12	Veggie-spread	\$ 3,202.87	\$ 263.40	\$ 842.88	\$ 2,590.10
13	Grand Total	=SUM(B3:B12)			
14					

RDBMS(Relational Database Management System)는 관계형 데이터베이스 관리 시스템을 의미합니다. 데이터를 테이블 형식으로 저장하고, 각 테이블은 행(row)과 열(column)로 구성됩니다. 테이블 간의 관계를 이용해 데이터를 효율적으로 관리하고, SQL(Structured Query Language)이라는 언어를 통해 데이터를 생성, 읽기, 수정, 삭제할 수 있습니다.

주요 특징:

- 데이터의 구조화: 테이블에 데이터를 체계적으로 저장.
- 관계 정의: 테이블 간의 관계를 설정해 데이터의 연결성 유지.
- SQL 사용: 데이터를 조작하는 데 표준화된 SQL 사용.

대표적인 RDBMS로는 MySQL, PostgreSQL, Oracle, Microsoft SQL Server 등이 있습니다.


age ChatGPT

# RDBMS

## 데이터를 저장하는 규칙

ChatGPT >

RDBMS의 테이블 관계가 왜 필요한지 간단한 예시와 함께 설명해줘



RDBMS(관계형 데이터베이스 관리 시스템)에서 테이블 간의 관계는 데이터의 효율적 관리와 무결성을 유지하는 데 매우 중요합니다. 간단한 예시로 설명해 볼게요.

1. 중복 데이터를 줄임

만약 관계를 설정하지 않고 모든 데이터를 하나의 테이블에 넣는다면, 중복된 정보가 많아질 수 있습니다. 예를 들어, 학생과 그들이 듣는 과목 정보를 관리한다고 가정해봅시다.

• 단일 테이블 사용 시

테이블에 학생의 이름, 학번, 과목명, 교수 이름 등을 저장한다고 할 때, 학생이 여러 과목을 들으면 해당 학생의 정보가 여러 번 중복해서 기록됩니다.



학번	이름	과목명	교수
101	철수	수학	김교수
101	철수	과학	이교수
102	영희	수학	김교수
102	영희	영어	박교수

이 경우, 학생 정보가 중복되고 관리하기 어려워질 수 있습니다.

• 관계를 활용한 경우

↓

+ Message ChatGPT



ChatGPT >

• 관계를 활용한 경우

이를 두 개의 테이블로 나누어 학생 테이블과 과목 테이블을 만들고, 학생과 과목 간의 관계를 설정하면 중복 데이터를 줄일 수 있습니다.

학생 테이블:

학번	이름
101	철수
102	영희

과목 테이블:

학번	과목명	교수
101	수학	김교수
101	과학	이교수
102	수학	김교수
102	영어	박교수



두 테이블은 학번을 통해 연결됩니다. 이렇게 하면 학생의 기본 정보가 중복 저장되지 않고, 학생이 수강하는 과목 정보만 따로 관리할 수 있습니다.

2. 데이터 무결성 유지

테이블 간의 관계는 데이터 무결성을 보장하는 데도 중요한 역할을 합니다. 예를 들어, 학생 테이블에 존재하지 않는 학번이 과목 테이블에 저장되지 않도록 관계(외래키 제약 조건)를 설정하면, 잘못된 데이터 입력을 방지할 수 있습니다.

↓

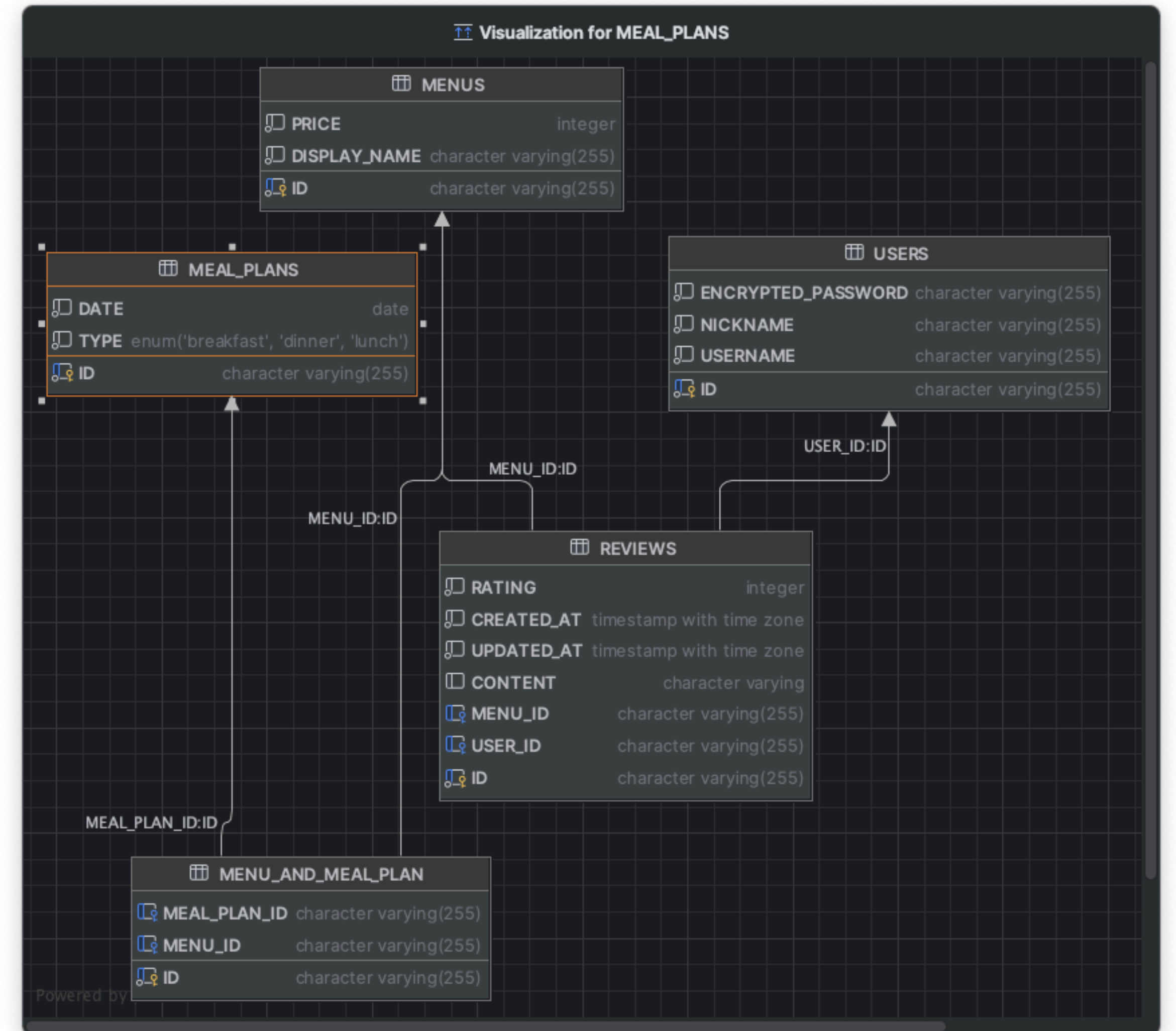
+ Message ChatGPT



# RDBMS

## 데이터를 저장하는 규격

- 테이블 간의 관계 예시) 미니 식사 DB
- 시스템에 맞는 DB Schema를 “잘” 모델링 하여야 한다.
- “잘못” 모델링 하면?
  - 비효율적으로 데이터를 저장하거나
  - 비효율적으로 데이터를 조회하거나
  - 차후 확장이 어렵다거나 등등...

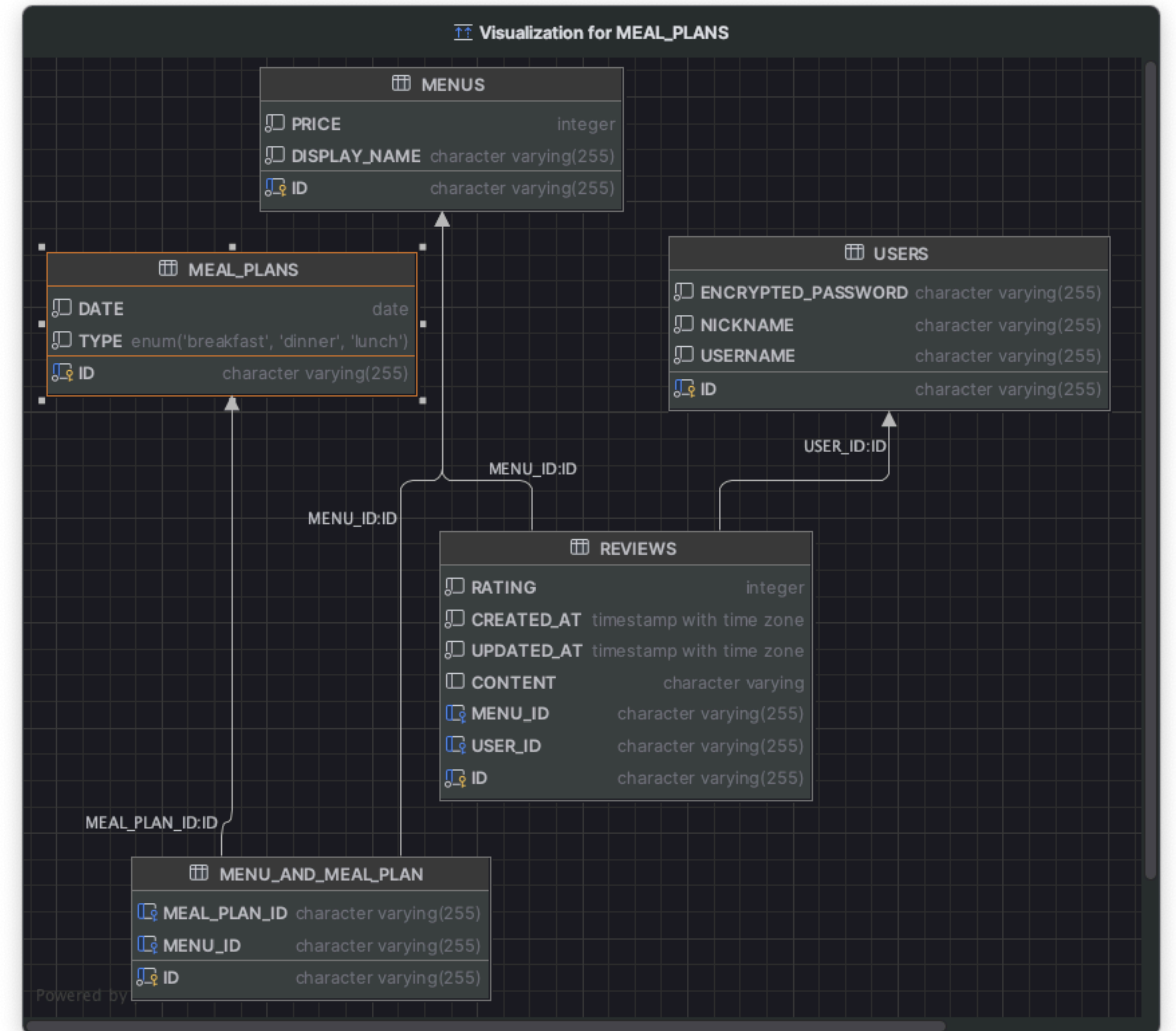




# RDBMS

## 1:N 관계 N:M 관계

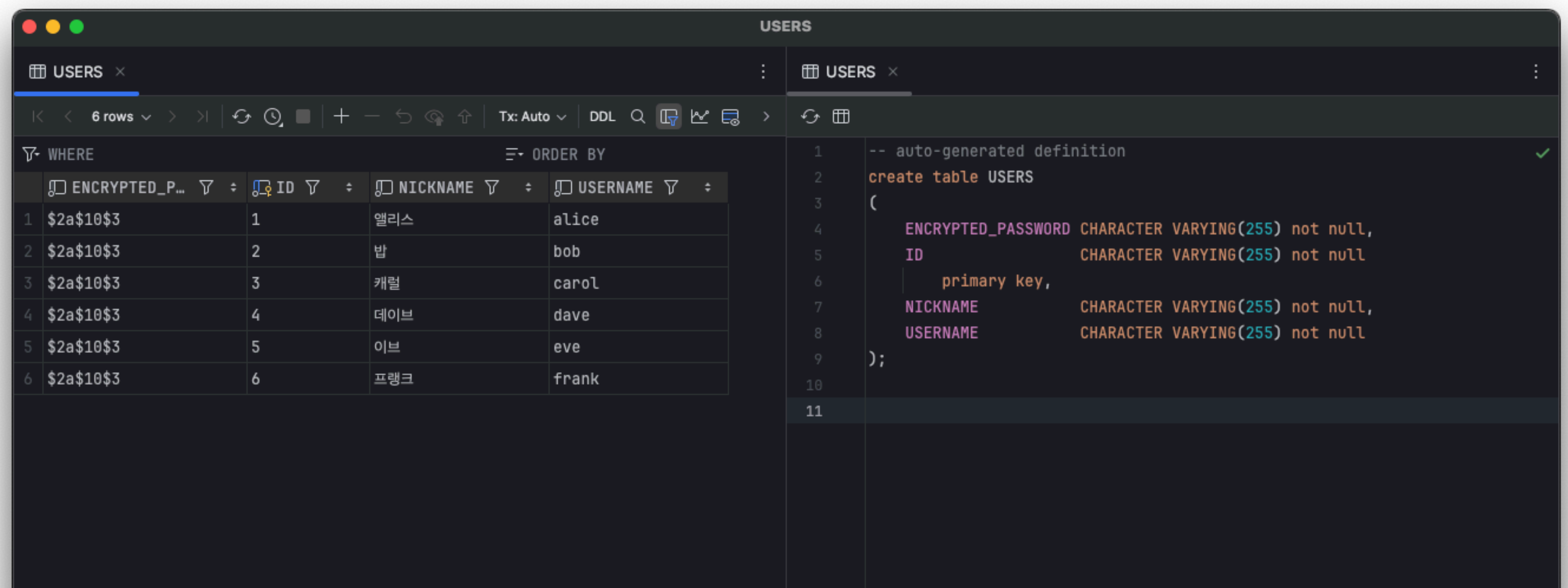
- 사용자와 리뷰는 1:N 관계
  - 사용자가 남긴 리뷰의 관계
  - Q) 데이터 구조에서 어떻게 표현?
- 식단표와 메뉴는 N:M 관계
  - 오늘 아침에 제육/된장찌개 나왔다
  - 저번주 아침에 된장찌개 나왔다
  - Q) 데이터 구조에서 어떻게 표현?



# RDBMS

## SQL

- DDL - 테이블 내부의 데이터 규격을 정의
- 행으로 저장된 개별 데이터, 열로 표현되는 데이터의 속성들
- Column 은 타입을 가진다. CHARACTER VARYING, INTEGER, BOOLEAN ....



# RDBMS

## SQL

- DML - 테이블 내부의 데이터를 조회 / 변경
  - SELECT - 조회
  - INSERT - 삽입
  - DELETE - 삭제
  - UPDATE - 변경

```
1 ✓ SELECT id, content FROM REVIEWS
2 WHERE RATING > 3
3 ORDER BY CREATED_AT DESC;
```

Output TESTDB.PUBLIC.REVIEWS x Tx

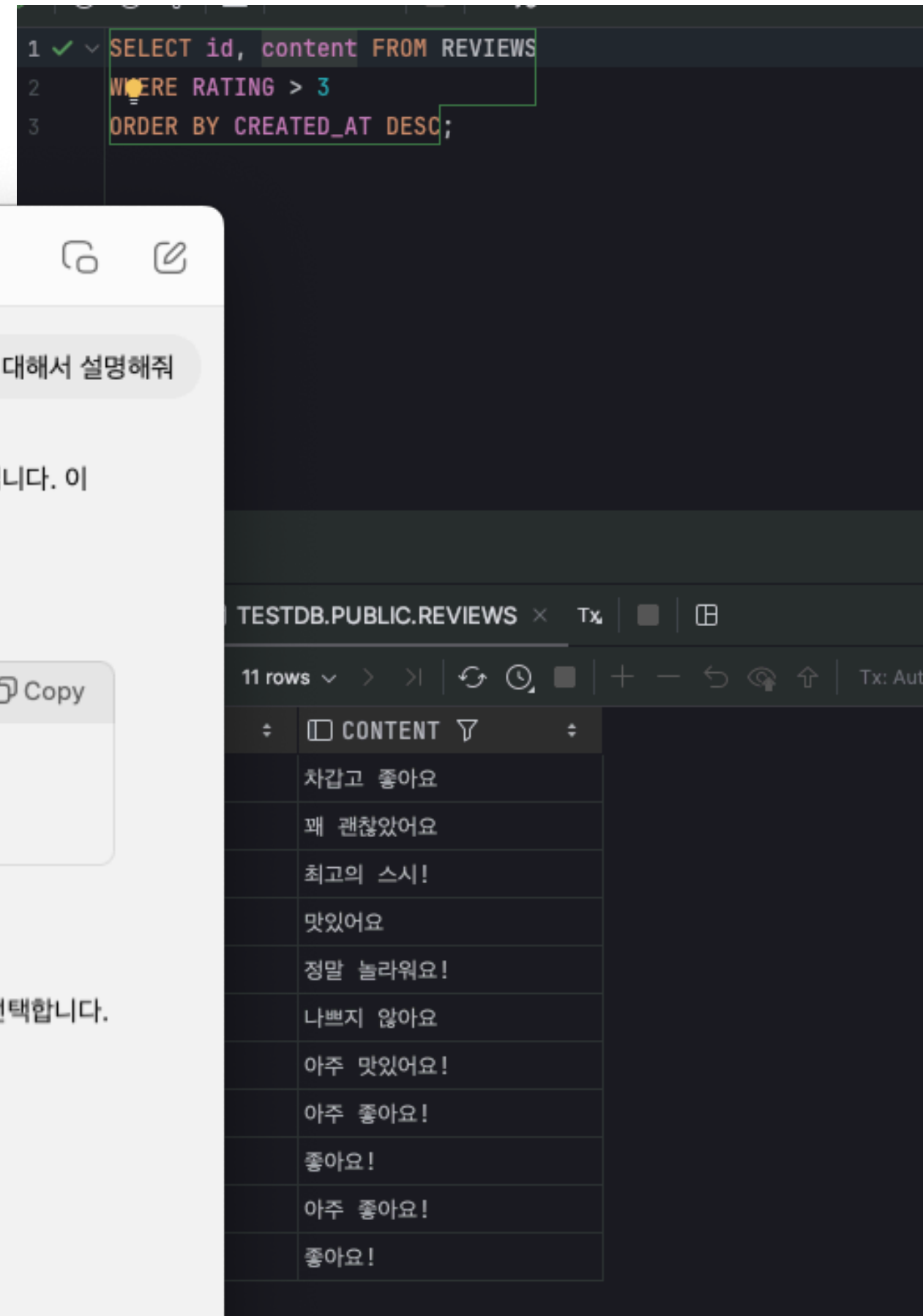
11 rows

	ID	CONTENT
1	16	차갑고 좋아요
2	14	꽤 괜찮았어요
3	13	최고의 스시!
4	11	맛있어요
5	10	정말 놀라워요!
6	8	나쁘지 않아요
7	7	아주 맛있어요!
8	1	아주 좋아요!
9	2	좋아요!
10	4	아주 좋아요!
11	5	좋아요!



# RDBMS

## SQL SELECT



리뷰 중에서 (FROM)

점수가 3점 초과인 행의 (WHERE)

식별자 와 내용을 (SELECT)

생성된 날짜 역순으로 정렬해줘  
(ORDER BY)

# RDBMS

## SQL SELECT

```
SELECT RATING, CONTENT FROM USERS
JOIN PUBLIC.REVIEWS R 1<->1..n: on USERS.ID = R.USER_ID
WHERE USERS.ID = 1;
```

	RATING	CONTENT
5	아주 좋아요!	
5	아주 좋아요!	
5	아주 맛있어요!	
5	최고의 스시!	

유저 중에서 유저 ID 가 1인 사람이  
작성한 리뷰를 다 보여줘

ChatGPT >

추가적인 질 및 기능

- ORDER BY:** 조회한 데이터를 정렬합니다.
  - 예: ORDER BY age DESC는 age 열을 기준으로 내림차순으로 정렬합니다.
- GROUP BY:** 같은 값을 가진 행들을 그룹화하여 집계할 때 사용합니다.
  - 예: SELECT department, COUNT(\*) FROM employees GROUP BY department는 각 부서별로 직원 수를 계산합니다.
- HAVING:** GROUP BY에 의해 그룹화된 결과에 조건을 적용할 때 사용됩니다.
  - 예: HAVING COUNT(\*) > 5는 5명 이상의 직원이 있는 부서만 선택합니다.
- JOIN:** 여러 테이블에서 데이터를 결합할 때 사용합니다.
  - 예: SELECT e.name, d.department\_name FROM employees e JOIN departments d ON e.department\_id = d.id는 직원 이름과 부서 이름을 함께 조회합니다.

예시

sql

Copy

```
SELECT name, age
FROM employees
WHERE age > 30
ORDER BY age DESC;
```

+ Message ChatGPT

# RDBMS

## SQL SELECT

sql

Copy code

```
SELECT
    u.username,
    m.display_name AS menu_name,
    COUNT(r.id) AS review_count
FROM
    reviews r
JOIN
    users u ON r.user_id = u.id
JOIN
    menus m ON r.menu_id = m.id
GROUP BY
    u.username, m.display_name
HAVING
    COUNT(r.id) > 1;
```

sql

Copy code

```
SELECT
    mp.type AS meal_type,
    m.display_name AS menu_name,
    AVG(r.rating) AS avg_rating
FROM
    reviews r
JOIN
    menus m ON r.menu_id = m.id
JOIN
    menu_and_meal_plan mm ON m.id = mm.menu_id
JOIN
    meal_plans mp ON mm.meal_plan_id = mp.id
WHERE
    mp.type = 'lunch' -- Replace with the actual meal type: 'breakfast', 'lunch'
GROUP BY
    mp.type, m.display_name;
```

# Spring Data JPA

## RDBMS 를 Spring 에서 “잘” 사용하기

- 문제점)
  - 쿼리를 짜는 것이 너무 실수하기 쉽다. (IDE 없이 코드를 짜기 쉽지 않은 것과 같음)
  - 잘 갖춰진 오류 방지 시스템이 있는 도구 -> Java, Kotlin 과 같은 프로그래밍 언어
    - 컴파일러가 수많은 오류를 잡아준다.
  - 쿼리를 Java / Kotlin 으로 짜고 싶다.
    - Query Builder
  - 더 나아가서 데이터 행 하나하나를 Java Instance 처럼 관리하고 싶다
    - ORM (Object Relation Mapper)

# Spring Data JPA

## RDBMS 를 Spring 에서 “잘” 사용하기

- JPA (Java Persistent API)
- Java 진영의 ORM 기술 표준
- 장점
  - 생산성
  - 성능
  - 벤더 무관 사용 가능 (MySQL, PostgreSQL 등등...)

# Spring Data JPA

## Spring Data JPA 의 생산성

- Spring Data + JPA
  - 저장소에 저장되는 Entity
  - 저장소에 접근하는 Repository
  - 각각을 Kotlin 에서 접근한다면
  - 대응하는 SQL 로 소통한다

```
interface UserRepository : JpaRepository<UserEntity, String> {  
    fun findByUsername(username: String): UserEntity?  
    fun existsByUsername(username: String): Boolean  
}
```

```
val menuEntity = menuRepository.findByIdOrNull(menuId) ?: throw MenuNotFoundException()  
val authorEntity =  
    userRepository.findByIdOrNull(author.id) ?: throw AuthenticateException()  
val reviewEntity = ReviewEntity(  
    author = authorEntity,  
    menu = menuEntity,  
    content = content,  
    rating = rating,  
    createdAt = Instant.now(),  
    updatedAt = Instant.now(),  
).let {  
    reviewRepository.save(it)  
}
```

```
Hibernate:  
select  
    me1_0.id,  
    me1_0.display_name,  
    me1_0.price  
from  
    menus me1_0  
where  
    me1_0.id=?
```

```
Hibernate:  
insert  
into  
    reviews  
    (user_id, content, created_at, menu_id, rating, updated_at, id)  
values  
    (?, ?, ?, ?, ?, ?, ?)
```



# Spring Data JPA

## Spring Data JPA 의 생산성

- Entity
  - Entity Class = SQL Table Schema
  - Entity Instance = SQL Single Row Data
  - RDBMS 의 스키마를 Entity Class 로 관리한다.
  - 새로운 Row Insert = Entity Instance 생성 후 영속화
  - 보통 INSERT, DELETE, UPDATE 쿼리와 관련이 있다.

```
@Entity(name = "users")
class UserEntity(
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    val id: String = "",
    @Column(name = "username", nullable = false)
    val username: String,
    @Column(name = "nickname", nullable = false)
    val nickname: String,
    @Column(name = "encrypted_password", nullable = false)
    val encryptedPassword: String,
)
```

```
create table USERS
(
    ENCRYPTED_PASSWORD CHARACTER VARYING(255) not null,
    ID                  CHARACTER VARYING(255) not null
    primary key,
    NICKNAME            CHARACTER VARYING(255) not null,
    USERNAME            CHARACTER VARYING(255) not null
);
```

# Spring Data JPA

## Spring Data JPA 의 생산성

- Entity with Relationship
  - 앞서 언급한 테이블 간의 관계를 JPA 가 쉽게 쓰게해준다.
  - DB 에서는 그냥 key 에 불과했는데...
- @OneToMany
  - User 의 Review 가져올래 -> `user.reviews`
- @ManyToOne
  - Review 의 작성자 가져올래 -> `review.author`

```
@Entity(name = "reviews")
class ReviewEntity(
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    val id: String = "",
    @Column(columnDefinition = "TEXT")
    var content: String,
    @Column(name = "rating", nullable = false)
    var rating: Int,
    @ManyToOne
    @JoinColumn(name = "menu_id")
    var menu: MenuEntity,
    @ManyToOne
    @JoinColumn(name = "user_id")
    var author: UserEntity,
    @Column(name = "created_at", nullable = false)
    var createdAt: Instant,
    @Column(name = "updated_at", nullable = false)
    var updatedAt: Instant,
)
```

```
return Review(
    id = entity.id,
    content = entity.content,
    rating = entity.rating,
    user = User.fromEntity(entity.author),
    createdAt = entity.createdAt,
    updatedAt = entity.updatedAt,
)
```

# Spring Data JPA

## Spring Data JPA 의 생산성

- Repository
  - 조건에 부합하는 Entity 를 가져오는 인터페이스
  - SELECT 쿼리를 날리는 것과 유사하다

```
interface UserRepository : JpaRepository<UserEntity, String> {  
    fun findByUsername(username: String): UserEntity?  
    fun existsByUsername(username: String): Boolean  
}
```

```
Hibernate:  
    select  
        ue1_0.id,  
        ue1_0.encrypted_password,  
        ue1_0.nickname,  
        ue1_0.username  
    from  
        users ue1_0  
    where  
        ue1_0.id=?
```

# Spring Data JPA

## Spring Data JPA 의 성능

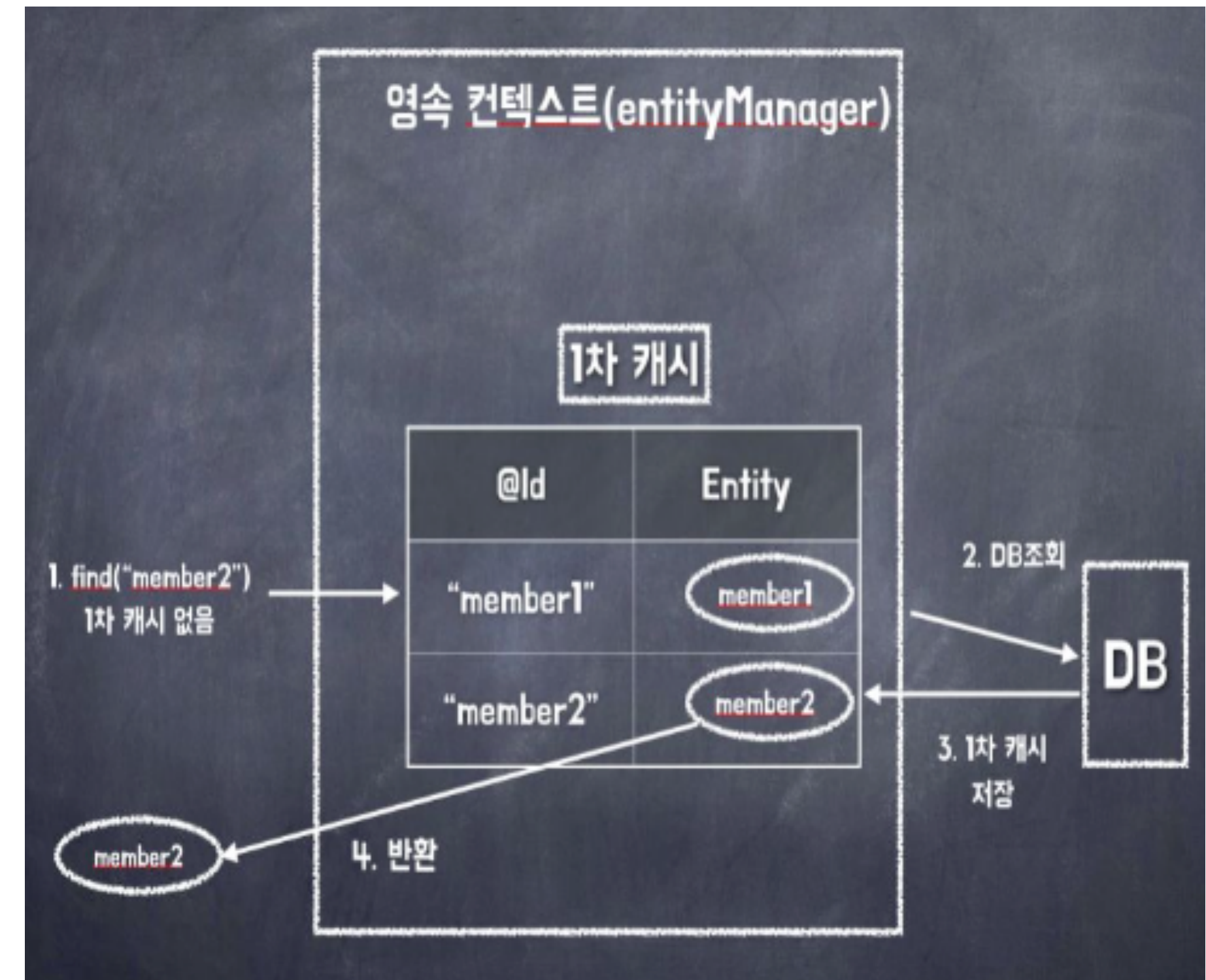
- DB 쿼리 -> Network IO
  - Network IO <<<<< Memory Read/Write
- 해결하는 가장 좋은 방법은 재활용
  - Memory 에 Cache Layer 를 두면 효율적으로 사용 가능
  - Write 를 하나하나 보내지 않고 한꺼번에 보내면 효율적으로 사용 가능
- Cache 관리 / 데이터 변경 감지 등등... (앞으로 말할 트랜잭션 까지)
- 이런 일은 JPA 영속성 컨텍스트에서 담당한다



# Spring Data JPA

## Spring Data JPA 의 성능

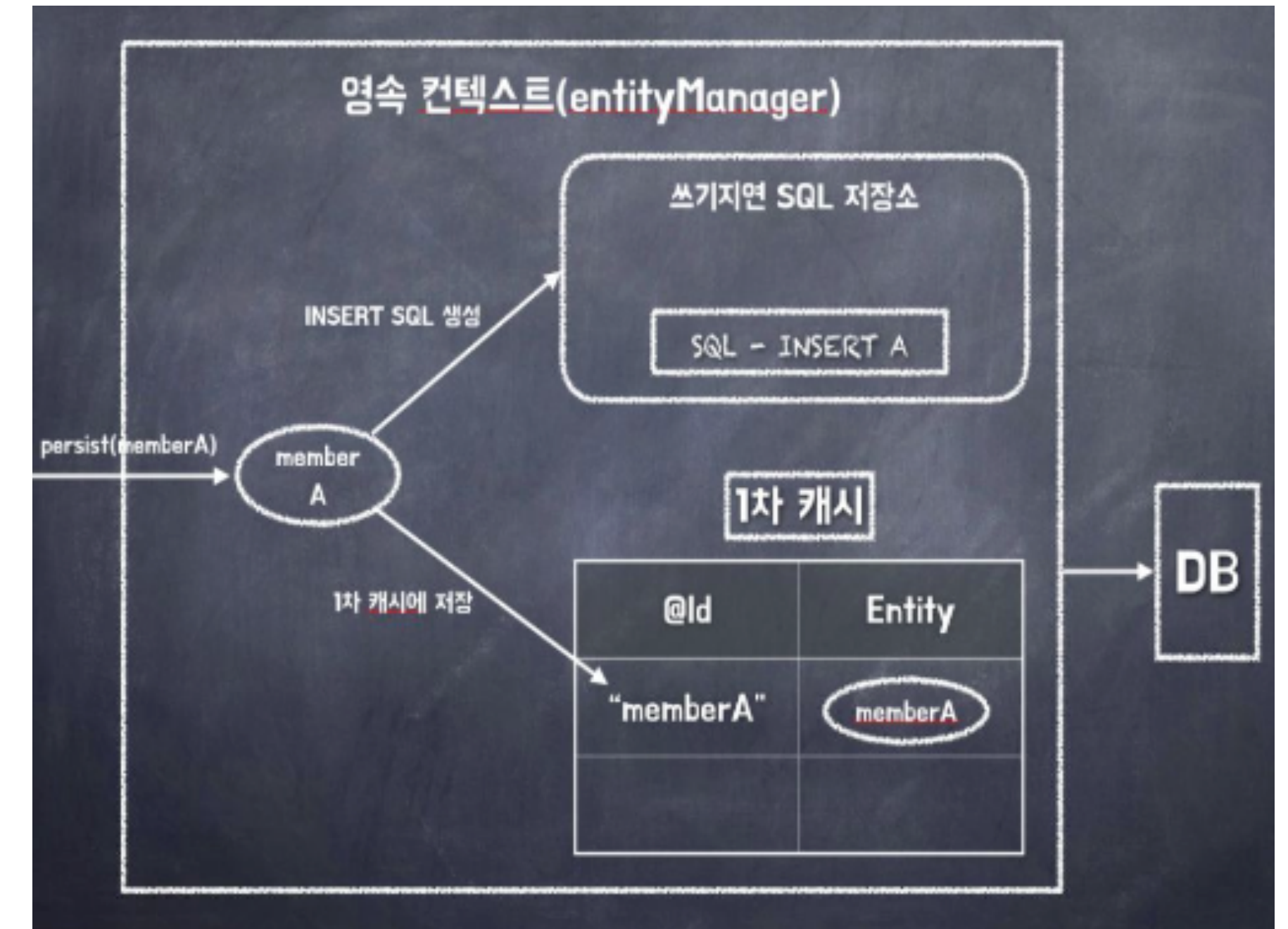
- 영속 컨텍스트와 캐싱
  - Find 시점에 캐시에 없다면 DB 조회
  - 이후 연속된 접근은 캐시에서 반환
- Q) 이러면 다른 서버에서 업데이트 해버리면 캐시가 원본과 달라지지 않나요..?



# Spring Data JPA

## Spring Data JPA 의 성능

- 영속 컨텍스트와 쓰기 지연
  - INSERT, UPDATE 는 천천히 해줘도 된다
  - 그러면 한번에 뭉쳐서 보내면 IO 가 줄어들고 좋겠네
  - -> 쓰기를 모아두고 한번에 처리하게 하자.
  - Q) 이러면 Data Integrity 가 깨지지 않나요...?
    - Q1) 내 Spring Server 의 Integrity
    - Q2) DB 자체의 Integrity

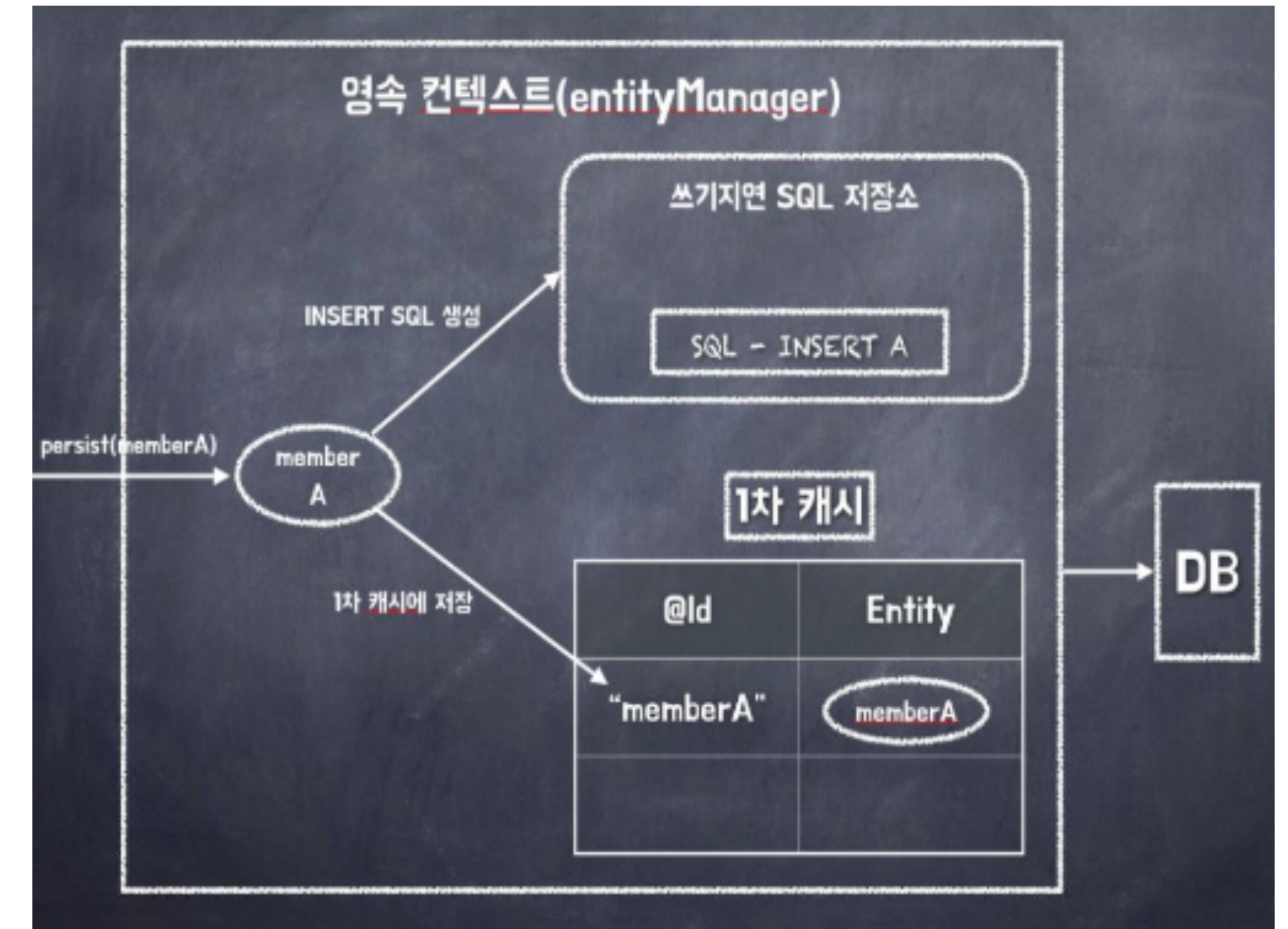




# Spring Data JPA

## Spring Data JPA 의 성능

- 영속 컨텍스트와 쓰기 지연
  - INSERT, UPDATE 는 천천히 해줘도 된다
  - 그러면 한번에 뭉쳐서 보내면 IO 가 줄어들고 좋겠네
  - > 쓰기를 모아두고 한번에 처리하게 하자.
  - Q) 이러면 Data Integrity 가 깨지지 않나요...?
    - Q1) 내 Spring Server 의 Integrity
    - Q2) DB 자체의 Integrity



# Spring Data JPA

## JoinColumn 과 성능

- 앞서 말한 문제: IO 를 줄이자
- 어떻게 보자면: Query 를 줄이자
- Query 가 늘어나는 대표적인 문제
  - 유저 A의 리뷰를 보고싶어 w/o join
    - A 이름의 유저 ID 찾고 -> 1회
    - 찾은 ID 로 Review 찾고 -> 1회
  - 유저 A의 리뷰를 보고싶어 w join
    - SELECT content FROM users JOIN reviews WHERE user.name = 'A'; -> 1회

# Spring Data JPA

## JoinColumn 과 성능

- 는 다음 시간에...

# Spring Seminar 과제 1

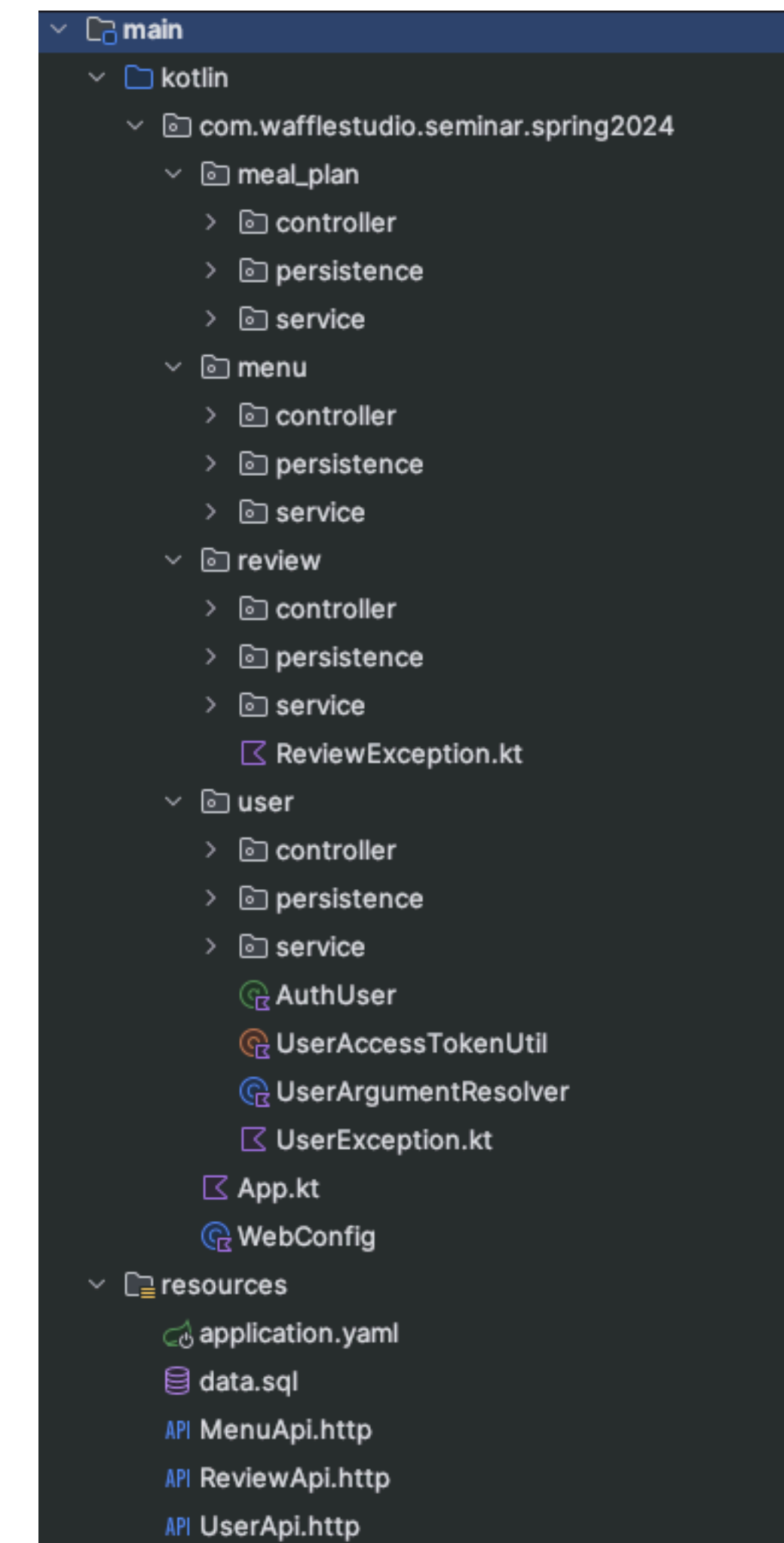
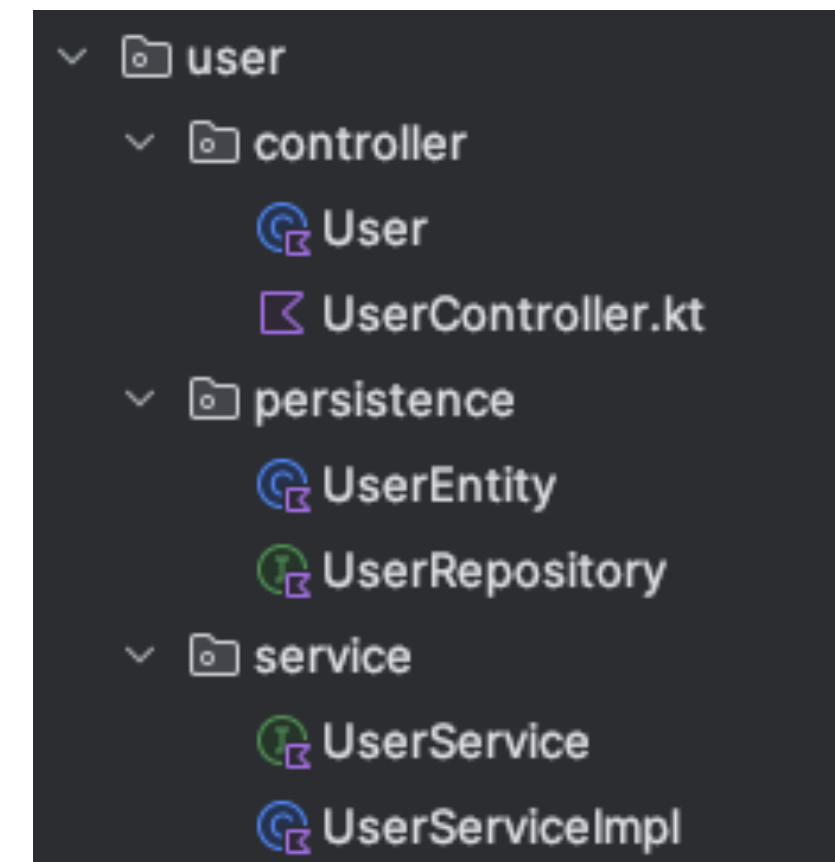
## 미니 식사 서버 만들기

- 저번 과제의 User 관리 코드 개선하기
  - 부족한 보안 관련 코드 수정 (패스워드 평문 작성, 변조가 어려운 AccessToken 발급)
- JPA 를 활용하여
  - “오늘 점심 식단” 가져오기
  - 평점 순으로 메뉴 정렬하기
  - 식단 리뷰 남기기

# Spring Seminar 과제 1

## 미니 식사 서버 만들기

- 권장 패키지 구조
- Domain (도메인 영역)
  - Controller (외부와의 통신 담당)
  - Persistence (DB와의 통신 담당)
  - Service (도메인 로직 담당)



# Spring Seminar 과제 1

## 미니 식사 서버 만들기

- DB Schema
- 권장 Schema (-> Entity) 를 그대로 가져다 쓰셔도 되고, 직접 Schema 를 만들어 보아도 됩니다.
- 직접 만들 때 어려운 부분은 식단표 : 메뉴의 N:M 관계를 DB 에 잘 모델링 하는 것

