

## 40. DBMS ဆိုတဲ့ (၇)။

\* ဤအားလုံး၏

JDBC API

Database

[datetime]

(ii) yyyy-MM-dd hh:mm:ss

getDate()  $\Rightarrow$  java.sql.Date  $\Rightarrow$   
yyyy-MM-dd

getTimestamp()  $\Rightarrow$  java.sql.Timestamp  $\Rightarrow$   
yyyy-MM-dd hh:mm:ss,

# 41. 외부키(foreign key) 활용

myapp-users

user_id	name	email	pwd	created_at
1	aaa	-	-	-
2	bbb	-	-	-
3	ccc	-	-	-

myapp-projects

project_id	Title	description	start_date	end_date	members
P01	P1	-	-	-	1, 3
P02	P2	-	-	-	1, 2
P03	P3	-	-	-	1, 2, 3

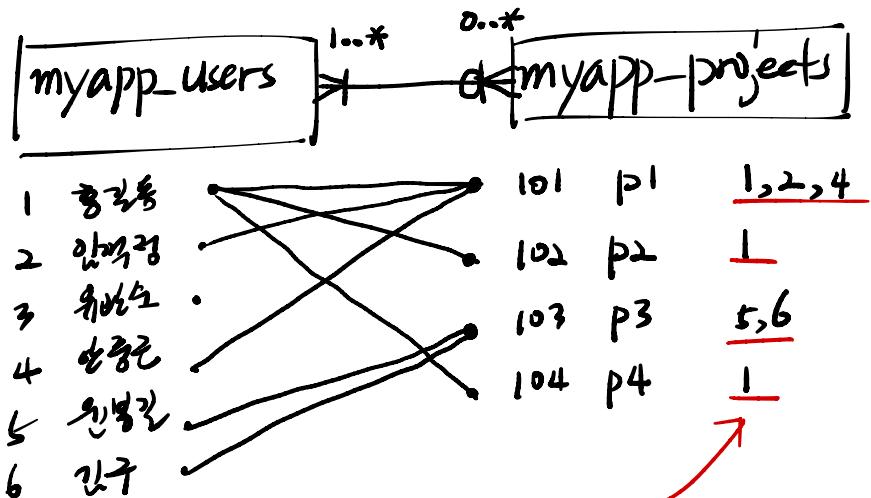
1번은 1번

프로젝트에 참여하는  
사람은 반드시 있는  
필수적인 조건이다!

↑  
필수적이기 때문이다!  
↳ 필수인 필수인!

## 41. 외부키(foreign key) 활용

1) 데이터의 데이터간에 관계를 짜는  
(cf 대 cf 관계)



(cf 대 cf)  
(cf 대 cf 관계를

외부 키와 함께 관계의 풍부함을

증가시킬 수 있어서 광범위한 막을 수 있다

Foreign key 활용  
+  
데이터의 관계  
관계형 데이터베이스에서 관계를

증가?

\* 데이터를 관리하는 틀

- 1 흐름
- 2 일정
- 3 주제

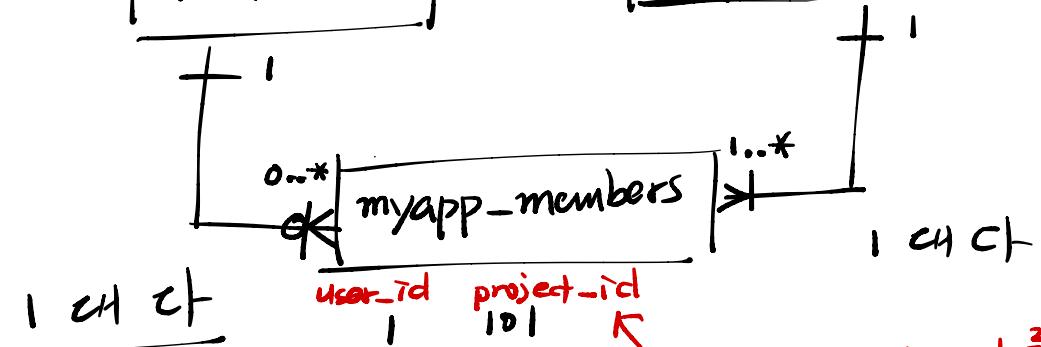
myapp-users	
1	2

- 1 흐름
- 2 일정
- 3 주제

myapp-projects	
101	p1
102	p2
103	p3

\* 외부키 (foreign key)

- 다른 테이블의 pk를 가리킨다
- 같은 pk를 사용할 수 있다



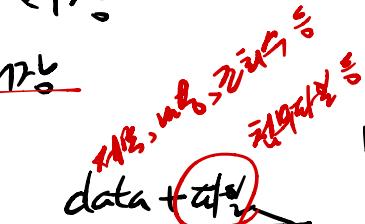
1 CH C1

	user_id	project_id
1	101	
1	103	
2		101
2		102
2		103

다른 테이블의 pk를 → 가리킨다  
"Foreign key"

\* DB의 특성 짚기

① 데이터가 구조화된



구조화된 형태

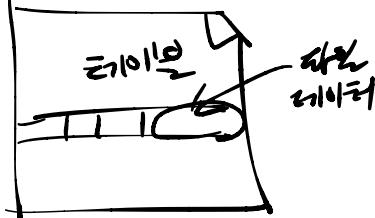
데이터의 구조가  
정해져야 한다.  
(구조화된 형태)



구조화된 형태  
구조화된 형태

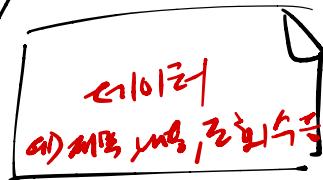


구조화된 형태  
구조화된 형태



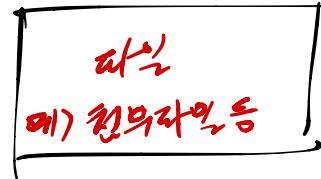
② 자료를 분리해서 저장

데이터 자체의  
구조화

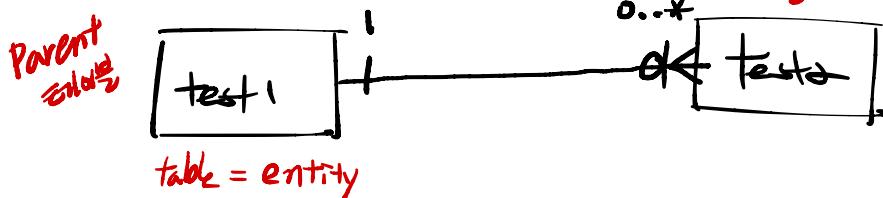


구조화된 형태  
구조화된 형태

구조화된 형태



44. child table  
(Foreign key)



ERD  
Information Engineering  $\rightarrow$  IEs  
(IE  $\rightarrow$  DB)

pk	no	title	content	rdt
1	aaa	-	-	.
2	bbb	-	-	.
3	ccc	-	-	.
4	ddd	-	-	.
5	eee	-	-	.

row = record

pk	fno	filepath	bno	fk	test1 (no)
1	101	a1.gif	1		
2	102	a2.gif	1		
3	103	a3.gif	1		
4	104	b1.gif	5		
5	105	b2.gif	5		
6	106	x.gif	00		test1의 no 00은 오류.

\* from → where → select → order by  
①

```
mysql> select * from room;
+----+-----+-----+-----+
| rno | loc      | name    | qnty   |
+----+-----+-----+-----+
| 1  | 강남      | 501    | 30     |
| 2  | 강남      | 502    | 30     |
| 3  | 강남      | 503    | 30     |
| 4  | 종로      | 301    | 30     |
| 5  | 종로      | 302    | 30     |
| 6  | 종로      | 303    | 30     |
| 7  | 서초      | 301    | 30     |
| 8  | 서초      | 302    | 30     |
| 9  | 서초      | 501    | 30     |
| 10 | 서초      | 601    | 30     |
+----+-----+-----+-----+
```

select rno, name, concat(loc, ' ', name) as name2  
① from room  
order by loc asc, name2 asc;

\* from → where → color by → order by  
②

```
mysql> select * from room;
+----+-----+-----+-----+
| rno | loc      | name    | qnty   |
+----+-----+-----+-----+
| 1  | 강남      | 501     | 30     |
| 2  | 강남      | 502     | 30     |
| 3  | 강남      | 503     | 30     |
| 4  | 종로      | 301     | 30     |
| 5  | 종로      | 302     | 30     |
| 6  | 종로      | 303     | 30     |
| 7  | 서초      | 301     | 30     |
| 8  | 서초      | 302     | 30     |
| 9  | 서초      | 501     | 30     |
| 10 | 서초      | 601     | 30     |
+----+-----+-----+-----+
```

```
select rno, name, concat(loc, ' ', name) as name2
① from room
order by loc asc, name2 asc;
```

\* from → where → select → order by

③ ↑ (가장 최종적인 결과 표시)

mysql> select \* from room;

(  
① select rno, name, concat(loc, '-', name) as name2  
from room  
order by loc asc, name2 asc;  
② ↑ (가장 최종적인 결과 표시)  
③ — (가장 최종적인 결과 표시)

rno	loc	name	qnty	name2 (loc - name)
1	강남	501	30	강남-501
2	강남	502	30	강남-502
3	강남	503	30	강남-503
4	종로	301	30	종로-301
5	종로	302	30	종로-302
6	종로	303	30	종로-303
7	서초	301	30	서초-301
8	서초	302	30	서초-302
9	서초	501	30	서초-501
10	서초	601	30	서초-601

\* from → where → select → order by

④  
③ select rno, name, concat(loc, ' ', name) as name2  
② from room  
① order by loc asc, name2 asc;

rno	loc	name	qnty	name2
1	강남	501	30	강남-501
2	강남	502	30	강남-502
3	강남	503	30	강남-503
7	서초	301	30	서초-301
8	서초	302	30	서초-302
9	서초	501	30	서초-501
10	서초	601	30	서초-601
4	종로	301	30	종로-301
5	종로	302	30	종로-302
6	종로	303	30	종로-303

\* from → where → select → order by → 결과 추출 ⑤

rno	name	name2
1	501	강남-501
2	502	강남-502
3	503	강남-503
7	301	서초-301
8	302	서초-302
9	501	서초-501
10	601	서초-601
4	301	종로-301
5	302	종로-302
6	303	종로-303

③ select rno, name, concat(loc, '-', name) as name2  
① from room  
④ order by loc asc, name2 asc;

select command  
선택한 결과의 값을  
결과 데이터로 추출한다

\* JOIN - Cross Join = cartesian join

board1

bno	title	content
1	제목1	내용
2	제목2	내용
3	제목3	내용

attach\_file1

fno	filepath	bno
101	a1.gif	1
102	a2.gif	1
103	c1.gif	3

select board1.bno, title, content, fno, filepath, attach\_file1.bno  
from board1 cross join attach\_file1;

bno	title	content	fno	filepath	bno
3	제목3	내용	101	a1.gif	1
2	제목2	내용	101	a1.gif	1
1	제목1	내용	101	a1.gif	1
3	제목3	내용	102	a2.gif	1
2	제목2	내용	102	a2.gif	1
1	제목1	내용	102	a2.gif	1
3	제목3	내용	103	c1.gif	3
2	제목2	내용	103	c1.gif	3
1	제목1	내용	103	c1.gif	3

\* JOIN - natural join  $\Rightarrow$  같은 이름의 컬럼 값을 기준으로 데이터를 합친다  
 board1 attach\_file

bno	title	content		fno	filepath	bno
1	제목1	내용	a	101	a1.gif	1
2	제목2	내용	a	102	a2.gif	1
3	제목3	내용	a	103	c1.gif	3

```
select b.bno, title, content, fno, filepath, a.bno
  -> from board1 b natural join attach_file1 a;
```

bno	title	content	fno	filepath	bno
1	제목1	내용	101	a1.gif	1
1	제목1	내용	102	a2.gif	1
3	제목3	내용	103	c1.gif	3

\* JOIN - natural join  $\Rightarrow$  하기!

```
select * from board2;
```

no	title	content	o
1	제목1	내용	o
2	제목2	내용	o
3	제목3	내용	o

```
select * from attach_file2;
```

no	filepath	bno
101	a1.gif	1
102	a2.gif	1
103	c1.gif	3

select b.no, title, content, a.no, filepath, bno  
-> from board2 b natural join attach\_file2 a;  
Empty set (0.00 sec)

부모 없는 데이터끼리 조인할 수 있다.

## \* JOIN - join ~ using ('주소', ...)

```
select * from board4;
```

bno	title	content
1	제목1	내용
2	제목2	내용
3	제목3	내용

```
select * from attach_file4;
```

fno	title	bno
1	a1.gif	1
2	a2.gif	1
3	c1.gif	3

이런 조인 방식을 ~준으로  
이연 조인 방식을 지정할 수 있다.

```
select b.bno, b.title, content, a.fno, a.title, a.bno
from board4 b join attach_file4 a using (bno);
```

bno	title	content	fno	title	bno
1	제목1	내용	1	a1.gif	1
1	제목1	내용	2	a2.gif	1
3	제목3	내용	3	c1.gif	3

\* JOIN - join ~ using (bno, ...)  $\Rightarrow$  헷갈기!

헷갈려는 이유는 가진 것과 같지 않아.

```
select * from board5;
```

no	title	content
1	제목1	내용
2	제목2	내용
3	제목3	내용

```
select * from attach_file5;
```

fno	filepath	bno
1	a1.gif	1
2	a2.gif	1
3	c1.gif	3



✓ select no, title, content, fno, filepath, bno  
  -> from board5 b join attach\_file5 a using (bno);  
ERROR 1054 (42S22): Unknown column 'bno' in 'from clause'

board 테이블에  
bno 컬럼이  
없어!  
or!

\* JOIN - join ~ on 헷  
(inner join)

```
select * from board5;
```

no	title	content
1	제목1	내용
2	제목2	내용
3	제목3	내용

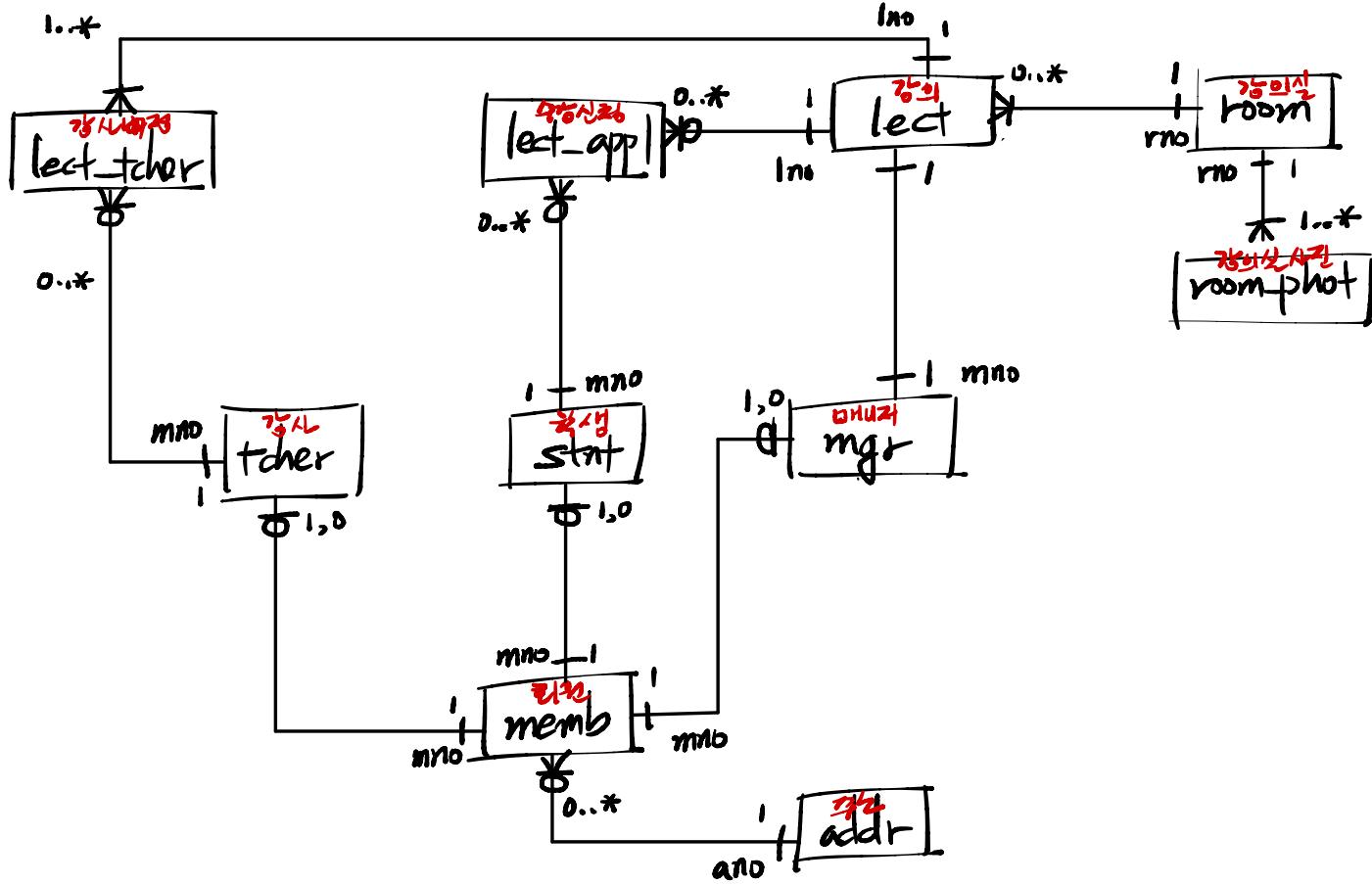
```
select * from attach_file5;
```

fno	filepath	bno
1	a1.gif	1
2	a2.gif	1
3	c1.gif	3

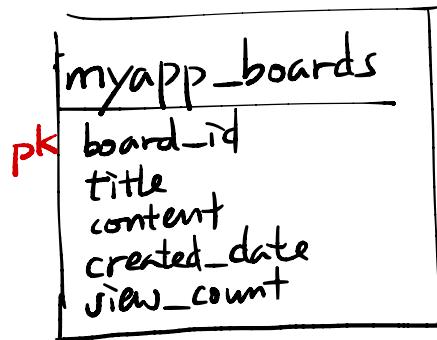
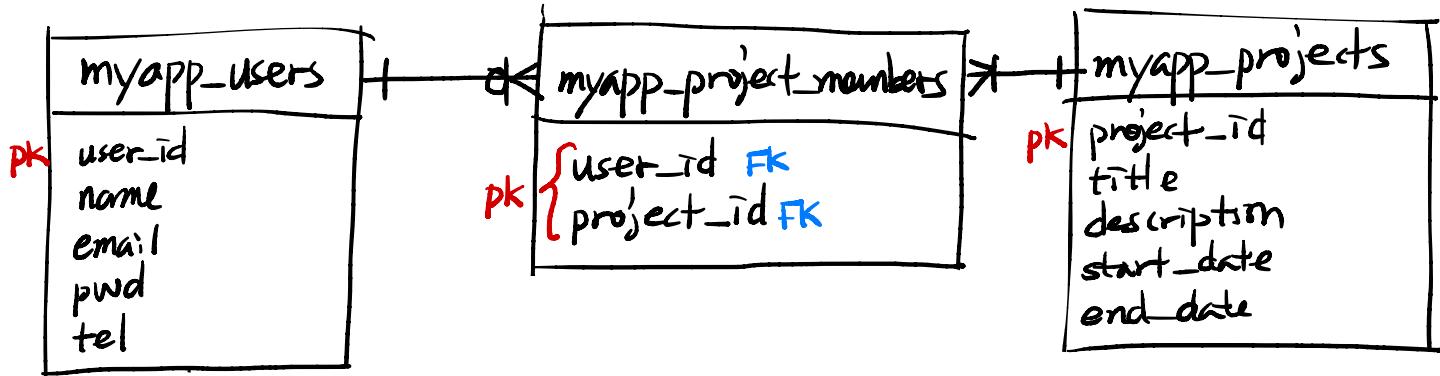
```
select no, title, content, fno, filepath, bno  
from board5 b join attach_file5 a on b.no=a.bno;
```

no	title	content	fno	filepath	bno
1	제목1	내용	1	a1.gif	1
1	제목1	내용	2	a2.gif	1
3	제목3	내용	3	c1.gif	3

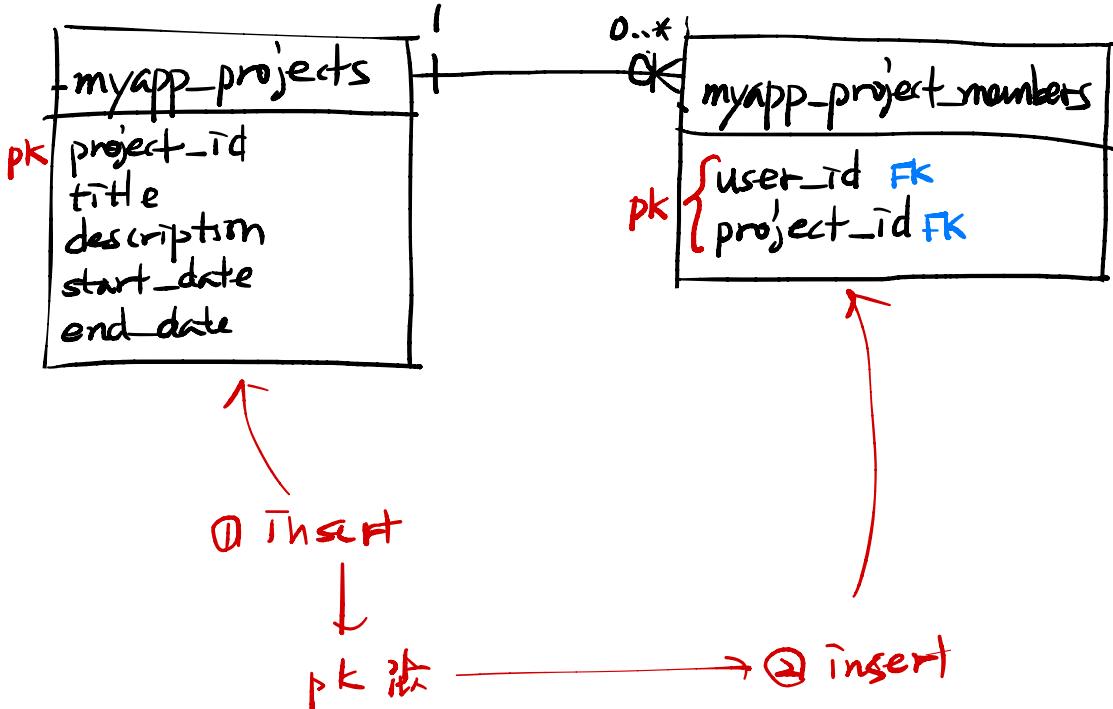
## \* JOIN 예제 - ERD (Entity Relationship Diagram)



## \* myapp ERD



\* 例えで insert



\* 자동생성된 PK 값 가져오기

insert into x\_board (title, contents)  
values ('aaa', 'xxx'),  
('bbb', 'yyy'),  
('ccc', 'zzz');



DBMS

board_id	title	contents	...
202	aaa	xxx	
203	bbb	yyy	
204	ccc	zzz	

PK

ResultSet keyRS = stmt.getGeneratedKeys();

keyRS.next(); → 202 ← pk 값

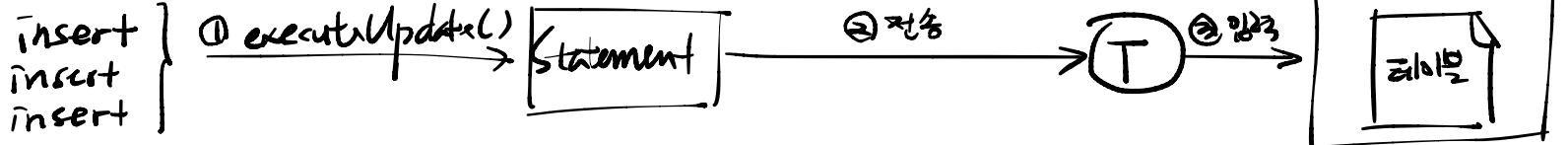
keyRS.getInt(1); 202 ← pk 값

keyRS.next(); → 203 ← pk 값

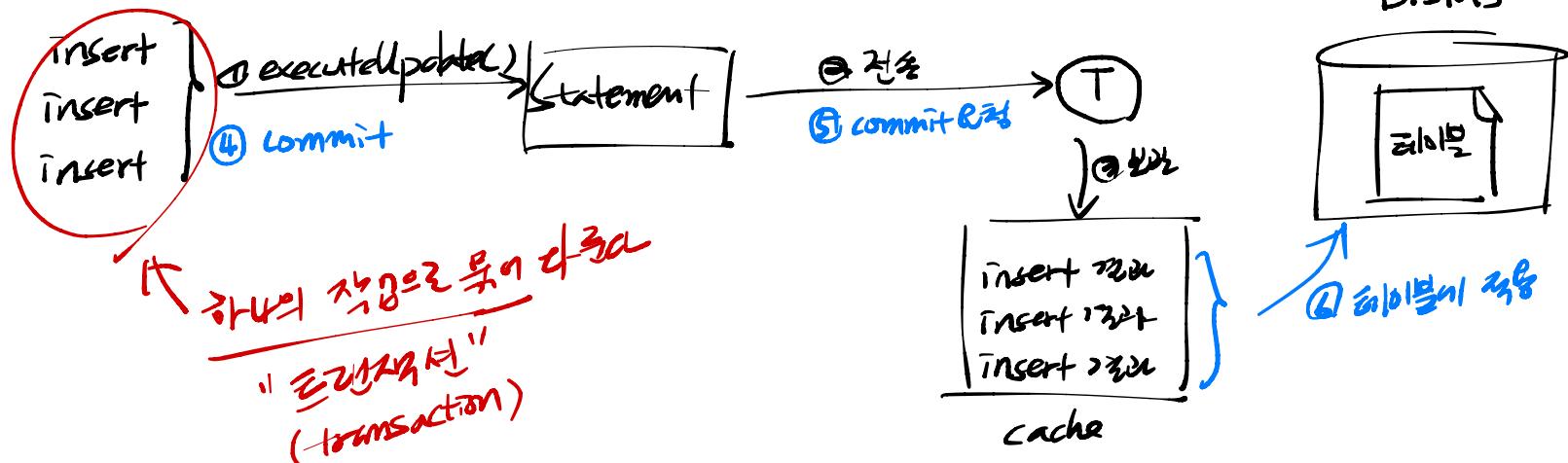
keyRS.getInt(1); 203 ← pk 값

## \* auto commit vs 수동 commit

1) Auto Commit = 즉시 데이터베이스 반영

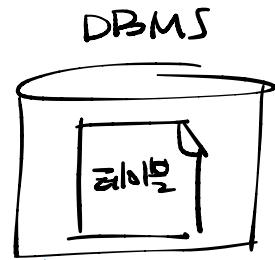
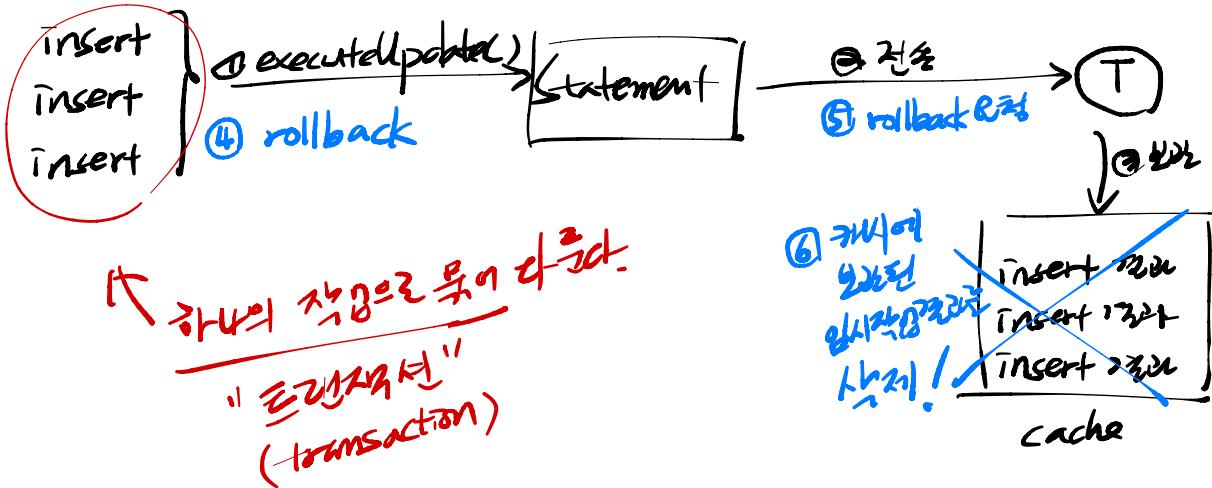


2) 수동 commit = commit

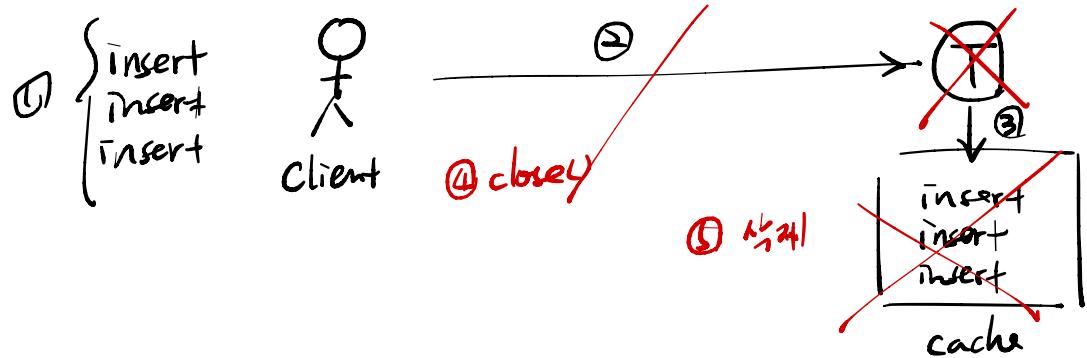


\* auto commit vs 수동 commit

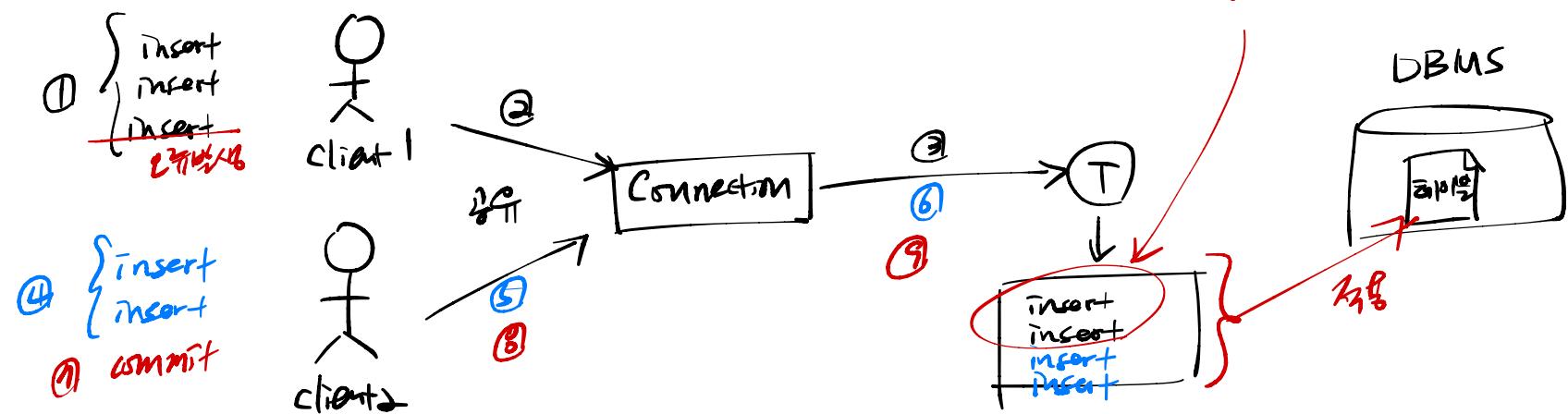
2) 수동 commit = rollback



\* 쓰기지연된 DB 처리법 장단점



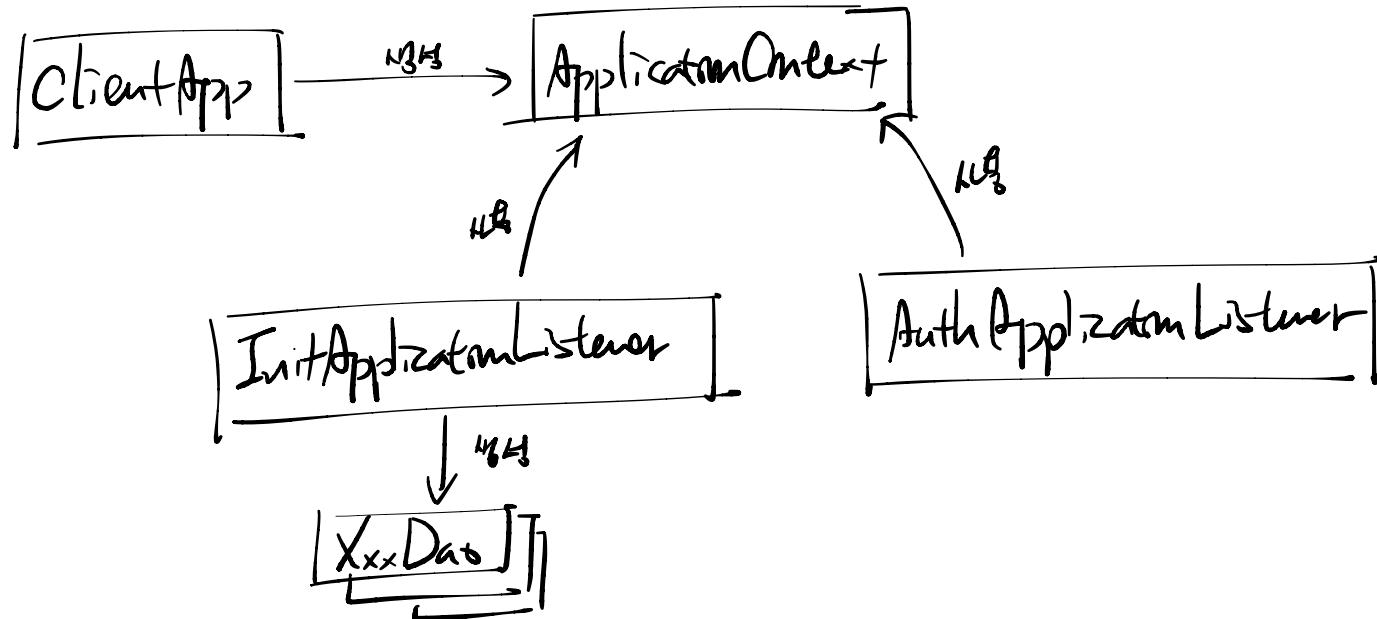
쓰기지연된 DB 처리법 장단점  
기억나는 예제를 찾았습니다  
쓰기지연된 DB 처리법은 rollback이 어렵다!  
Delete는 rollback이 어렵다!  
client의 rollback은 가능하다!  
rollback은 가능하다!



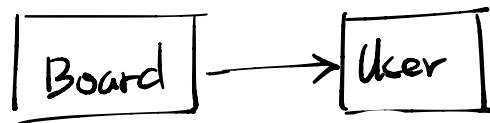
DBMS

42. 201 / 206 22.8.861

① ApplicationListener چیست چه کاری دارد



\* گلاین + سلکت



```
class Board {  
    int no;  
    :  
    User writer;  
    :  
}
```

## 43. SQL 향상된 구조

"insert into x\_board(title, contents)  
values ('" + title + "', '" + contents + "')"

↓  
title = aaa  
contents = xxx

"insert into x\_board(title, contents) values('aaa','xxx')"

## \* SQL 햄버거 만들기

"insert into x\_board(title, contents)  
values ('" + title + "', '" + contents + "')"

↓  
title=aaa  
contents=~~xxx~~ xxx'), ('bbb', 'yy'), ('ccc', 'zzz')

"insert into x\_board(title, contents)  
values('aaa', 'xxx'), ('bbb', 'yy'), ('ccc', 'zzz')"

## \* SQL 향입 공격 예 2

```
"insert into x_board(title, contents)  
values ('" + title + "', '" + contents + "')"
```

11

```
title = aaa  
contents = xxx'), (select user_id from myapp_users where user_id=1),  
          (select pwd from myapp_users where user_id=1),  
          ('xxx', 'xxx
```

```
"insert into x_board(title,contents)  
values('aaa','  
'")
```

\* SQL 4b9b 8>ny ikgf1 ⇒ PreparedStatement

PreparedStatement stmt = con.prepareStatement(SQL);

insert into x\_board(title, contents) values(?, ?)

stmt.setString(1, '제목');  
stmt.setString(2, '내용');

In-parameter  
SQL문  
SQL의 내용을 값으로 넣기!

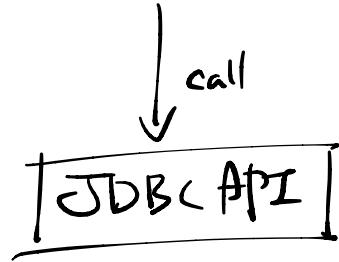
In-parameter  
1. 제목  
2. 내용

In-parameter  
값  
값

In-parameter  
값  
값

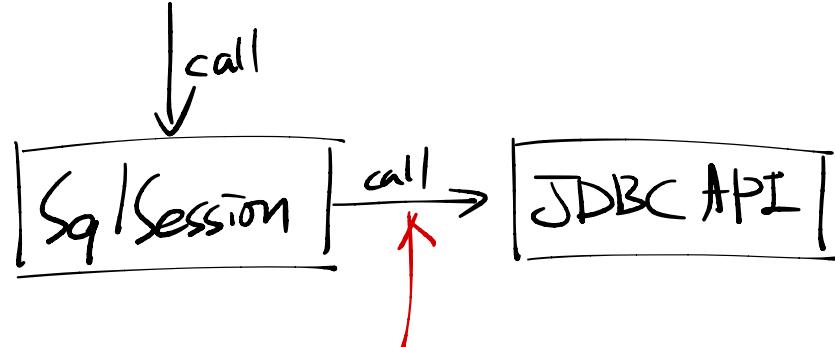
## 44. JDBC 관계 캡슐화

① 이전 방식



② 개선 방식

비단계적인 관계  
될 수 있다.

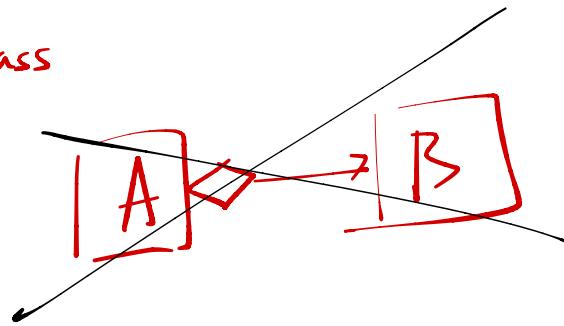
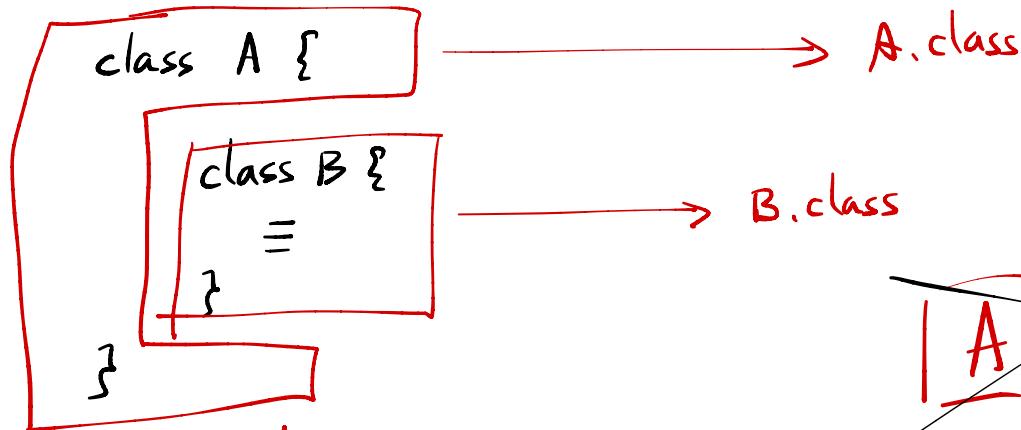


Reflection API 활용

- 초기화 대체 가능화
- 동적 SQL 가능화

## \* Reflection API

증집 클래스



전체 클래스는 증집

B를 만들기  
A를 만들기 위한 대안

이면 가능

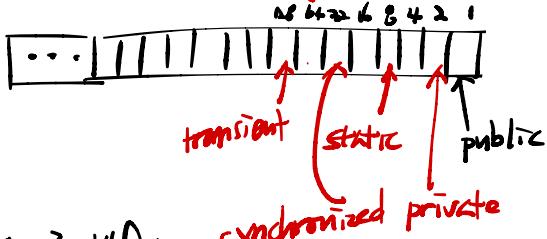
= A는 B를 만들기 위한 OBJECT!

내부에서  $\frac{1}{2}$  개의 다른 블록이 있다

\* modifier 를 이용한 연산자-

modifier 티스

volatile final protected



(1) public static final float PI = 3.14f;

↳ 00011001(modifier 티스)

public: 00000001

private: 00000010

protected: 00000100

static : 00001000

final : 00010010

$$\begin{array}{r} 00011001 \\ \& 00000010 \text{ (public)} \\ \hline 00001001 > 0 \end{array}$$

$$\begin{array}{r} 00011001 \\ \& 00000010 \text{ (private)} \\ \hline 00000000 > 0 \end{array}$$

## \* Field & Property

```
public class User implements Serializable {  
    private int no;  
    private String name;  
    private String email;  
  
    public int getNo() {return no;}  
    public void setNo(int no) {this.no = no;}  
  
    public String getName() {return name;}  
    public void setName(String name) {this.name = name;}  
  
    public String getEmail() {return email;}  
    public void setEmail(String email) {this.email = email;}  
}
```

~~setCreateDate()~~ { - }

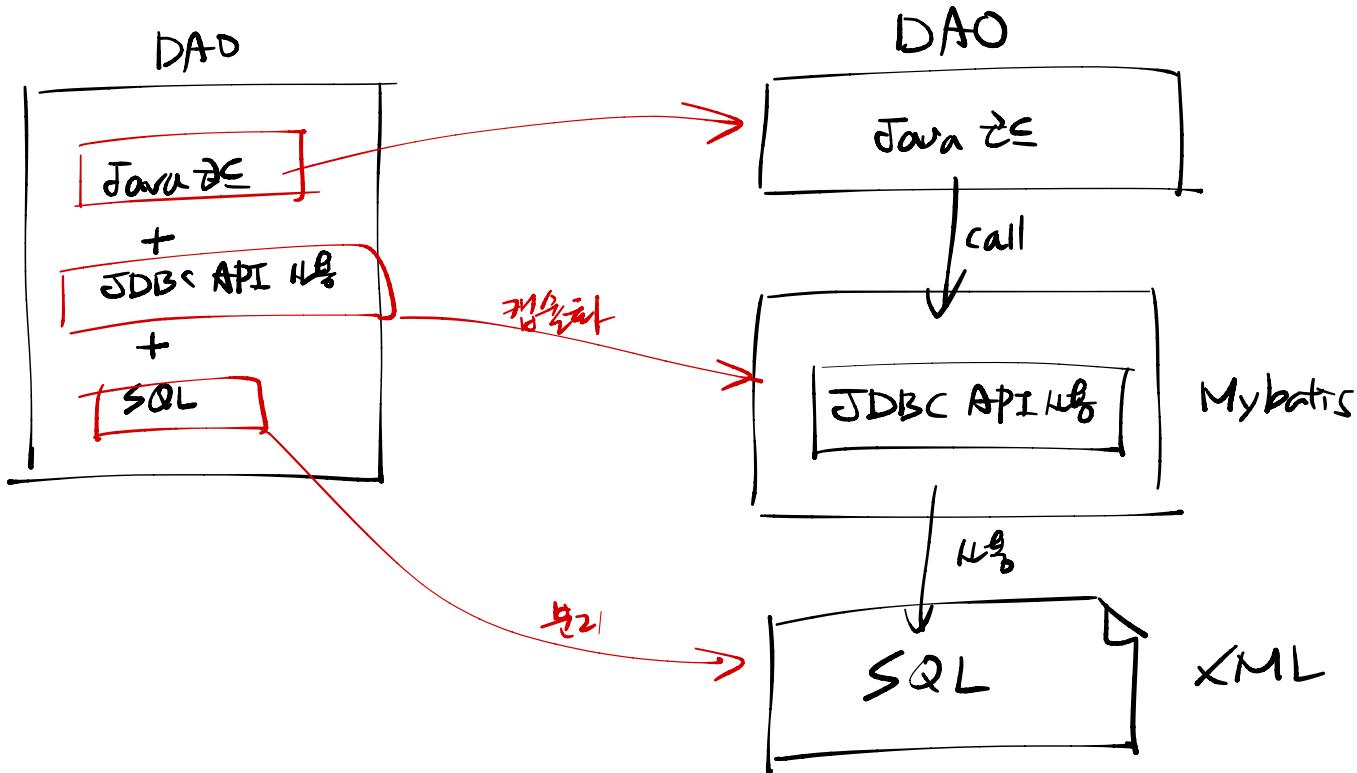
"createDate" ← property name

## \* Field E- Property

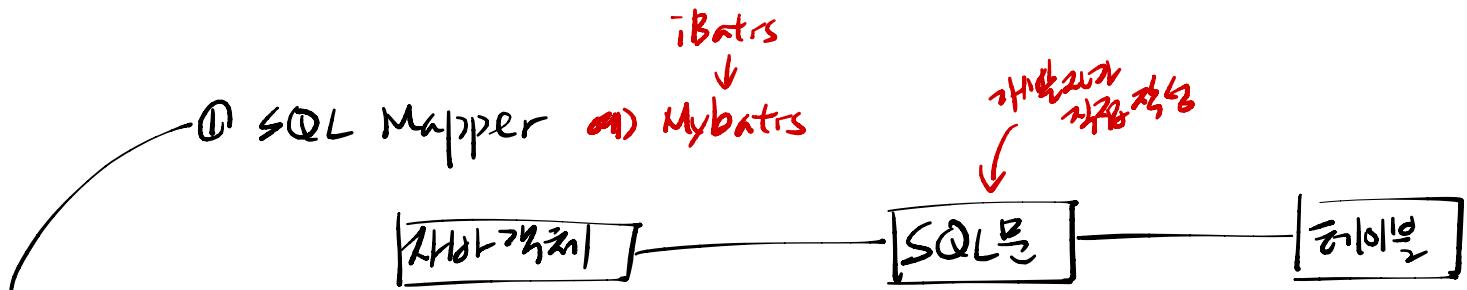
```
class Board {  
    int no;           ← setNo()  
    String title;    ← setTitle()  
    String content;  ← setContent()  
    Date createdDate; ← setCreatedDate()  
    int viewCount;   ← setViewCount()  
    User writer;     ← getWriter().setNo()  
                      ← getWriter().setName()  
} //
```

select  
 board\_id as no,  
 title,  
 content,  
 created\_date as createdDate,  
 view\_count as viewCount,  
 user\_id as writer-no,  
 name as writername  
from

## 45. Mybatis 29



## \* 퍼시스疼스 프레임워크 (Persistence Framework)



→ ② OR Mapper and Hibernate

JPA



\* Mybatis XML 구조 - ① 구조분석, 조건



mybatis-config.xml

기준을 관리하는 XML 파일  
- //mybatis.org//DTD config 3.0//EN  
+ 규칙 정의 가능  
- 내용 제한  
기준이 있는  
기준을 적용하는

1.0 버전

규칙 정의 가능  
내용 제한 가능

내용 제한 가능  
제한

<?xml version="1.0" encoding="UTF-8" ?> ← XML 문서

<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD config 3.0//EN" "https://-----/.dtd">

XML 문서

root element  
(루트)

SYSTEM (내부)

↑ 특정 파일이나 URL로 링크되는 경로

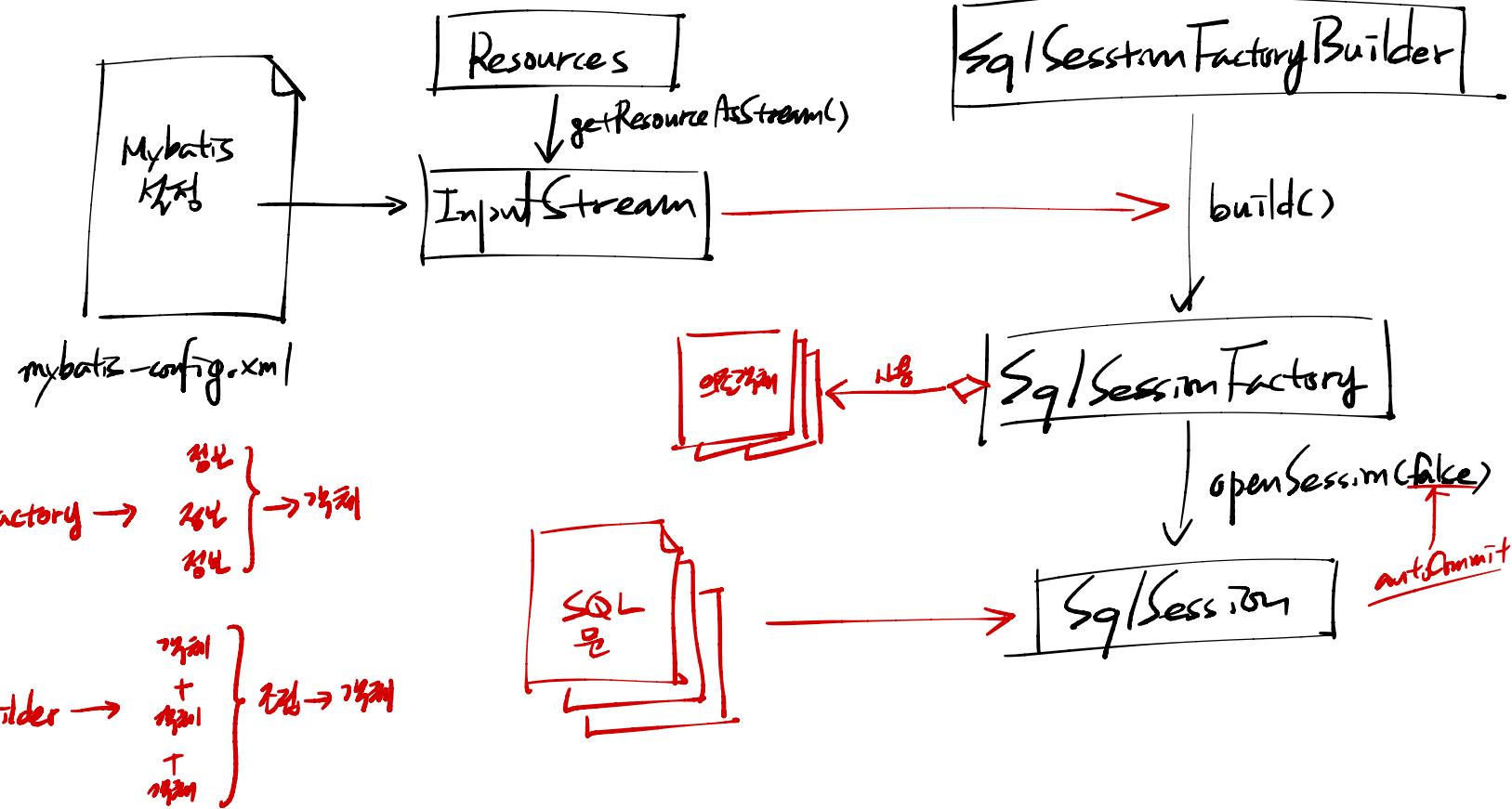
기준을 정의한 설정파일의 URL

<configuration> ← root element = XML 문서의 루트 요소

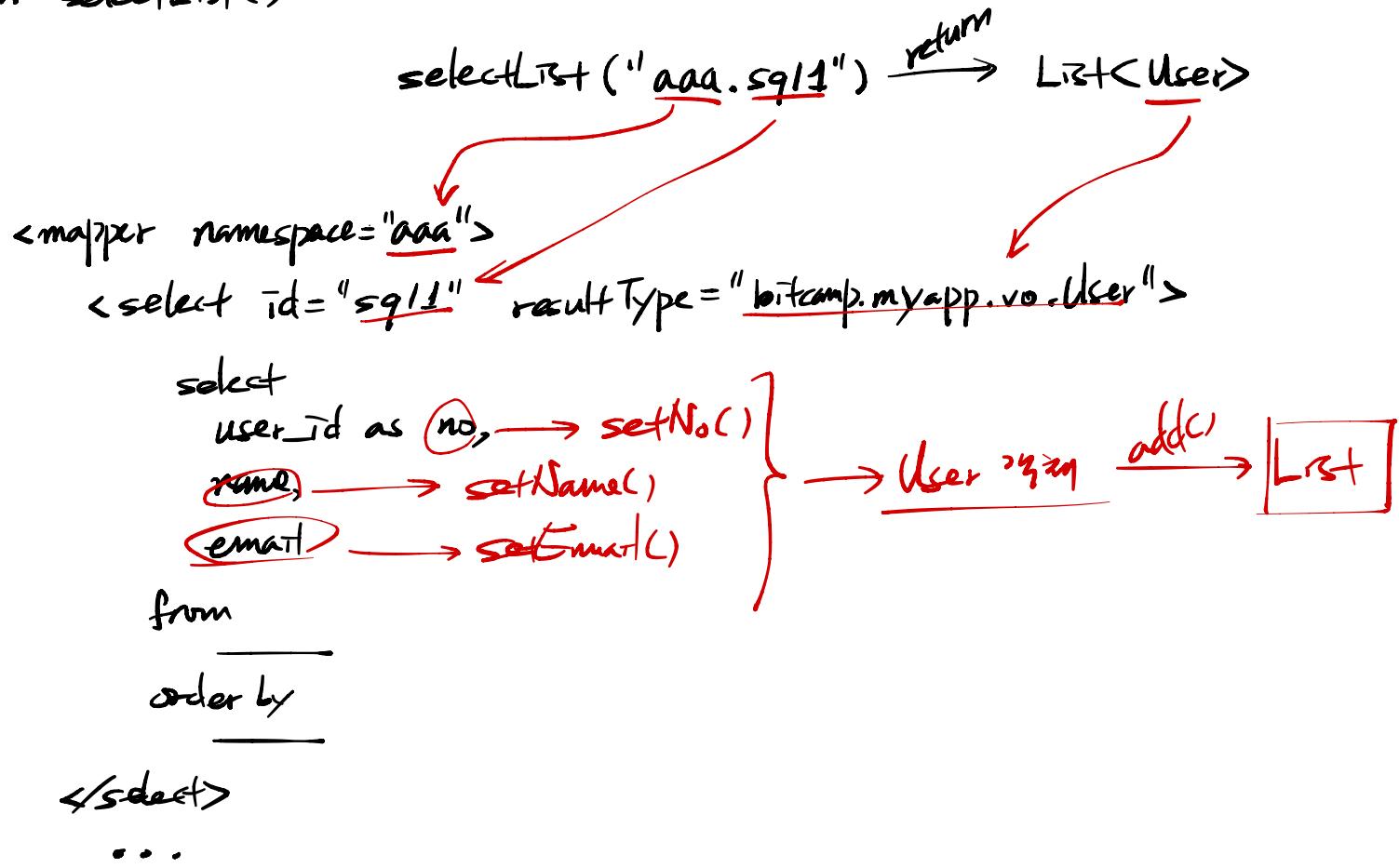
|

</configuration>

\* SqlSession.m 73번 3부



\* selectList()



\* insert()

```
insert("aaa.sql2", user);  
  
<insert id="sql2" parameterType="bitcamp.myapp.v0.User">  
    insert into myapp-users(name, email, pwd, tel)  
    values (#{name}, #{email}, sha1(#{password}), #{tel})  
</insert>
```

↑ property name  
getName()  
↑ property name  
getEmail()  
↑  
getPassword()

\* selectOne()

selectOne("aaa.sq13", no) → User

List browser

<select id="sq13" resultType="bitcamp.myapp.vo.User" parameterType="int">

select

user\_id as no,

name,

email,

tel

from myapp-users

where user\_id = #ok#

<select>

attribute

(primitive type)  
. String  
. Date

selected value button id

\* Entity Transaction

SqlSession.insert()  
update()  
delete()

} SqlSession.commit()  
" " .rollback()

---

[UserAddCommand]

try {  
 SqlSession.insert();  
 SqlSession.commit();  
}  
  
catch() { ~~SqlSession.rollback();~~ }

\* SQL 퀼문을 힙정렬 모양

<resultMap>

select

b.board\_id,

b.title,

b.content,

b.created\_date,

b.view\_count,

u.user\_id,

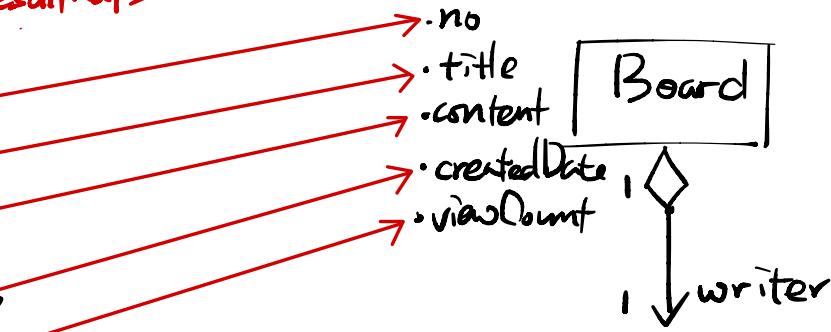
u.name

from myapp\_boards b

inner join myapp\_users u

on b.user\_id = u.user\_id

where



<association>

- no
- name
- email
- password
- tel

\*<resultMap> + <association>  
↑ ↗ 결과를 풀어서 처리하는 과정  
↗ 결과를 풀어서 처리하는 과정

<resultMap id="BoardMap" type="bitcamp.myapp.vo.Board">

<id column="board\_id" property="no"/> class Board {  
↑ PK 컬럼인 경우 ↑ ↗ 값명 ↗ 값명 사용자에게 드러내기 명 User writer;  
<result column="title" property="title"/> }  
=

<association property="writer" javaType="bitcamp.myapp.vo.User">  
<id column="writer\_no" property="no"/>  
<result column="writer\_name" property="name"/>  
</association>

</resultMap>

## \* 2단계로는 허용하지 않으나 II

select

p.project\_id,  
p.title,  
p.description,  
p.start\_date,  
p.end\_date,  
pm.user\_id,  
u.name

→ Project 테이블

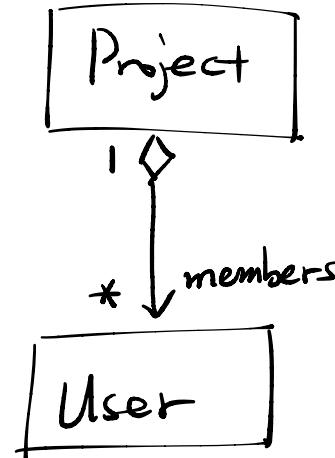
→ User 테이블

from

myapp\_projects p  
left outer join myapp\_project\_members pm  
on p.project\_id=pm.project\_id  
left outer join myapp\_users u  
on pm.user\_id=u.user\_id

where

p.project\_id=#{no}

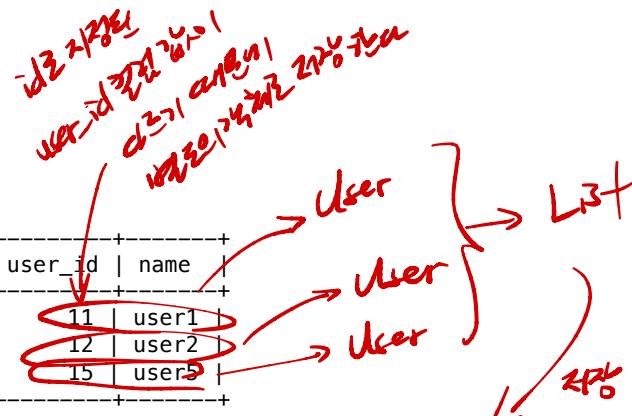


project_id	title	description	start_date	end_date	user_id	name
101	프로젝트1	설명	2024-01-01	2024-02-15	11	user1
101	프로젝트1	설명	2024-01-01	2024-02-15	12	user2
101	프로젝트1	설명	2024-01-01	2024-02-15	15	user5

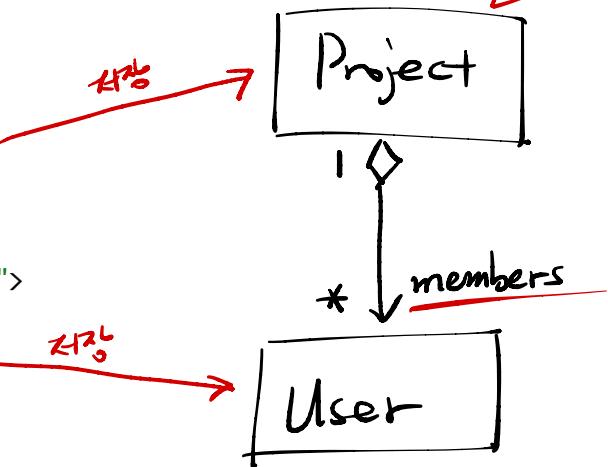
\* 결과값을 1개로 하는 단 사용할 규칙 2개의

id 체크하는  
중복되는  
한개는  
가지면 2개로 한다.

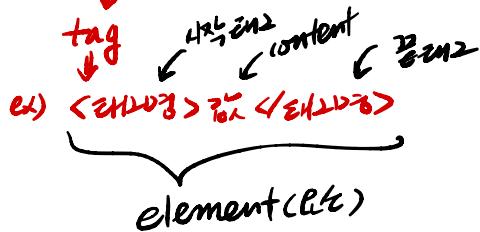
project_id	title	description	start_date	end_date	user_id	name
101	프로젝트1	설명	2024-01-01	2024-02-15	11	user1
101	프로젝트1	설명	2024-01-01	2024-02-15	12	user2
101	프로젝트1	설명	2024-01-01	2024-02-15	15	user3



```
<resultMap id="ProjectMap" type="bitcamp.myapp.vo.Project">
  <id column="project_id" property="no"/>
  <result column="title" property="title"/>
  <result column="description" property="description"/>
  <result column="start_date" property="startDate"/>
  <result column="end_date" property="endDate"/>
  <collection property="members" ofType="bitcamp.myapp.vo.User">
    <id column="user_id" property="no"/>
    <result column="name" property="name"/>
  </collection>
</resultMap>
```



\* XML (Extensible Markup Language) — 데이터를 구조화 시킬 수 있는  
데이터에 따라 사용자에게 맞는 형식으로 출력하는 언어



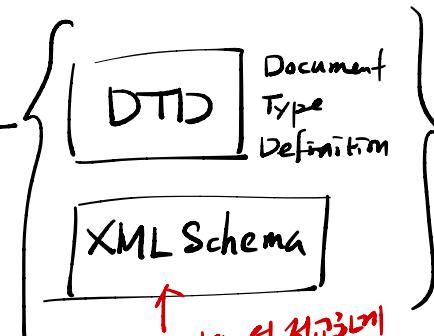
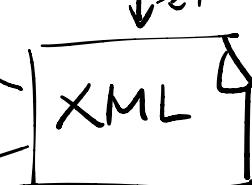
XML은 가로, 세로 모두 가능



XML의 규칙 → 자식-부모, 태그 등  
정상 = well-formed XML



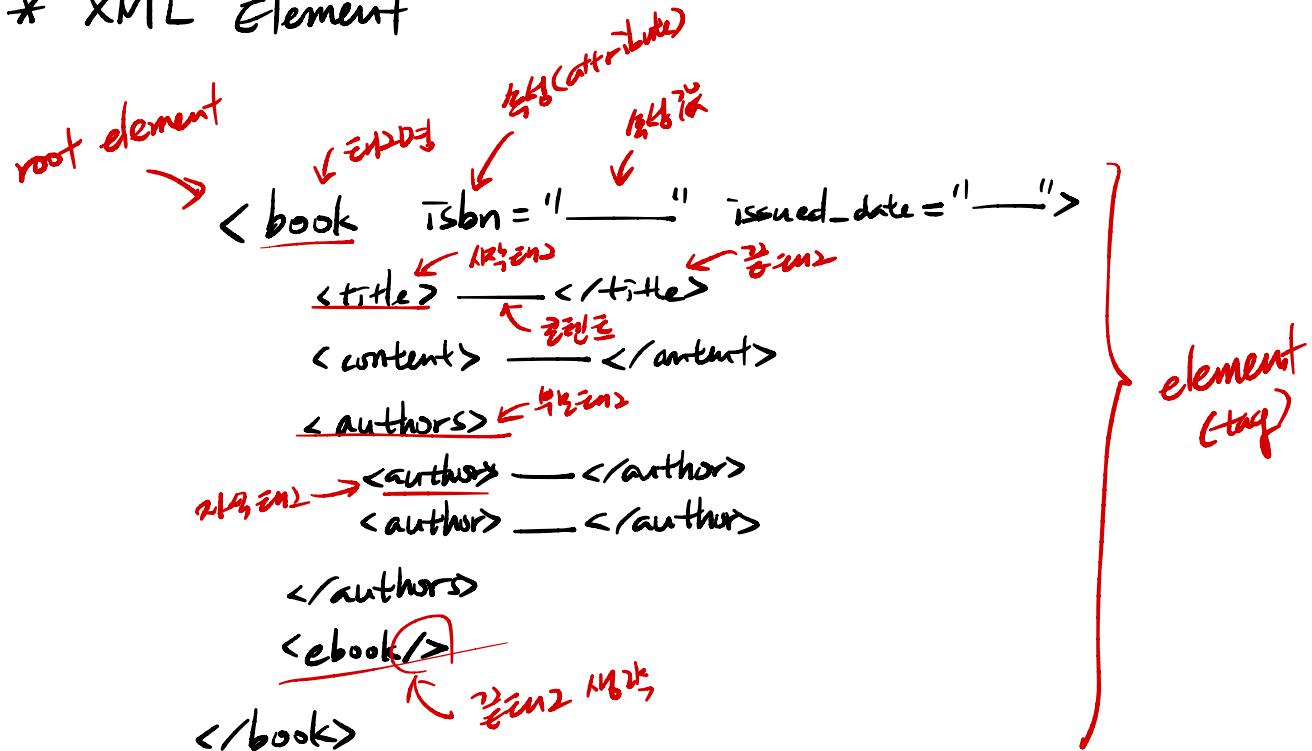
XML 문서의  
설계 및 표기



DTD와의 차이점  
구조화된 문서 형식

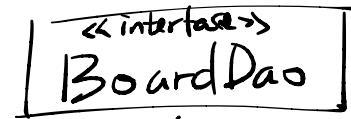
DTD(도큐먼트 타입 정의)

## \* XML Element

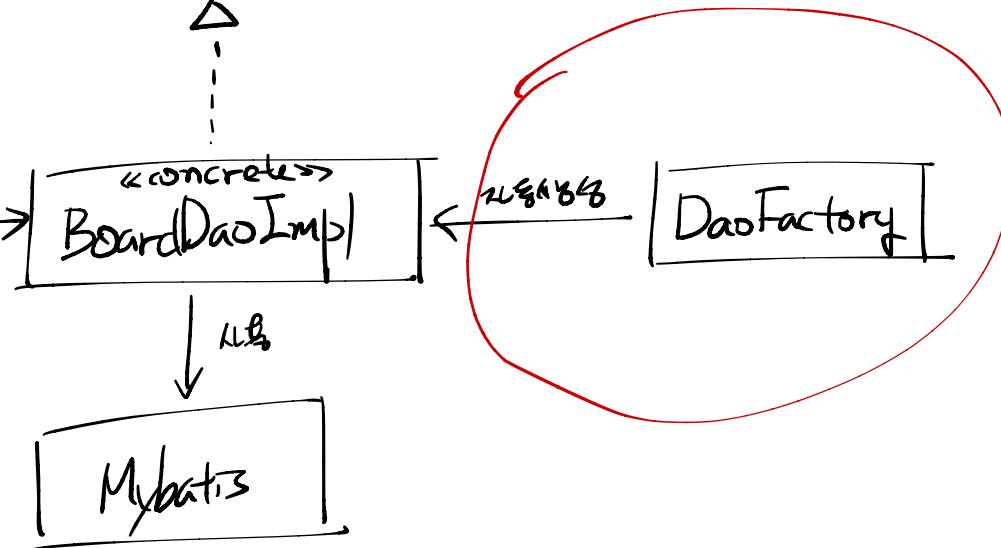


## 46. DAO 주제의 캐스팅

① 대상 타입

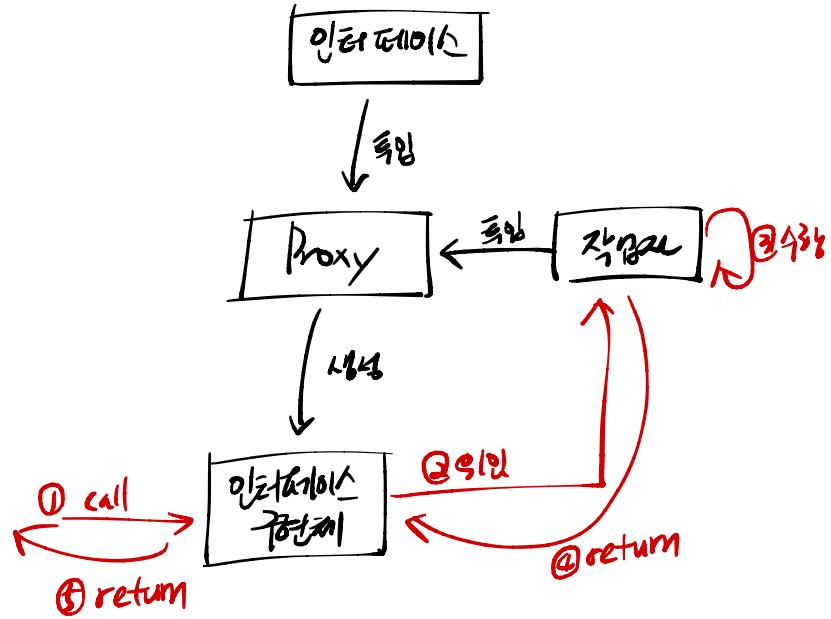


② 대상 타입

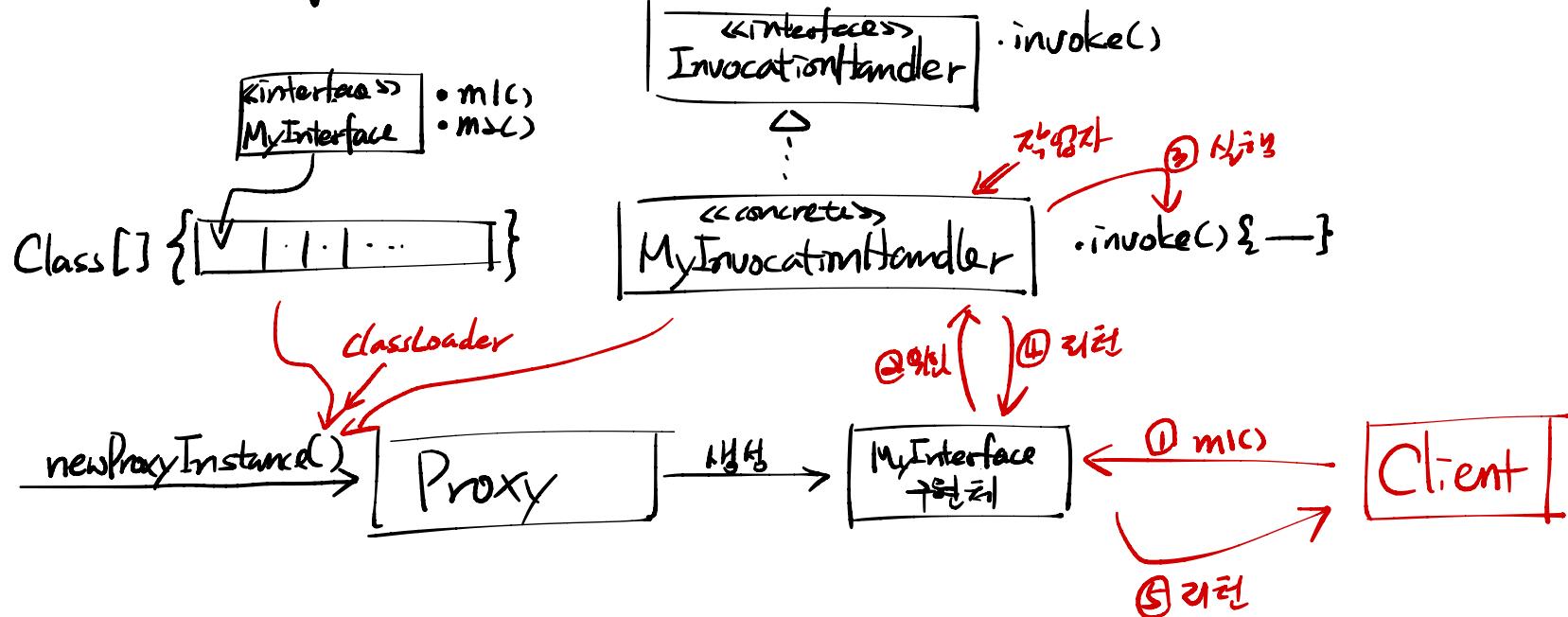


- insert()
- selectList()
- selectOne()
- :

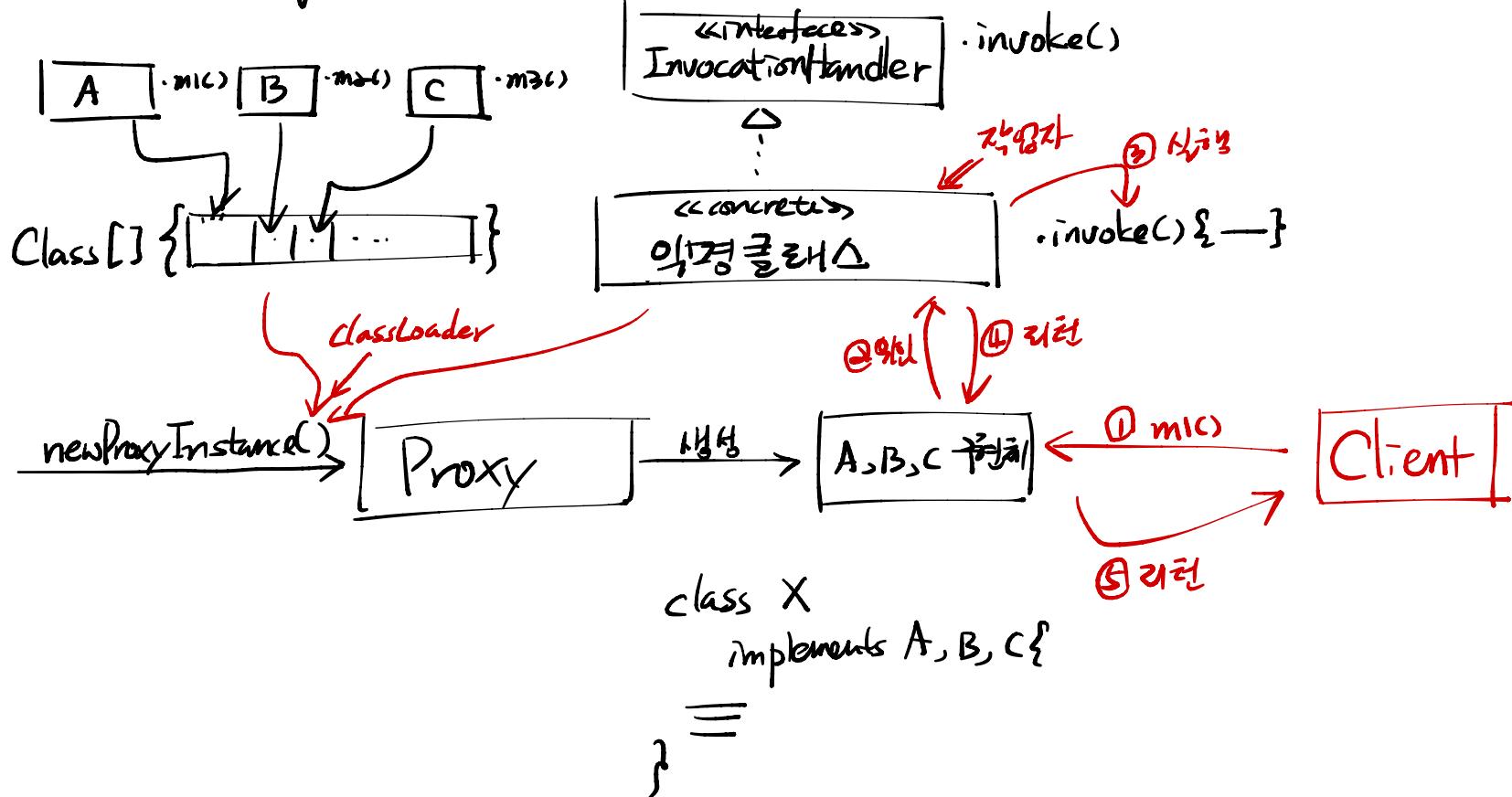
## \* DaoFactory 구현 원리



## \* DaoFactory ↗

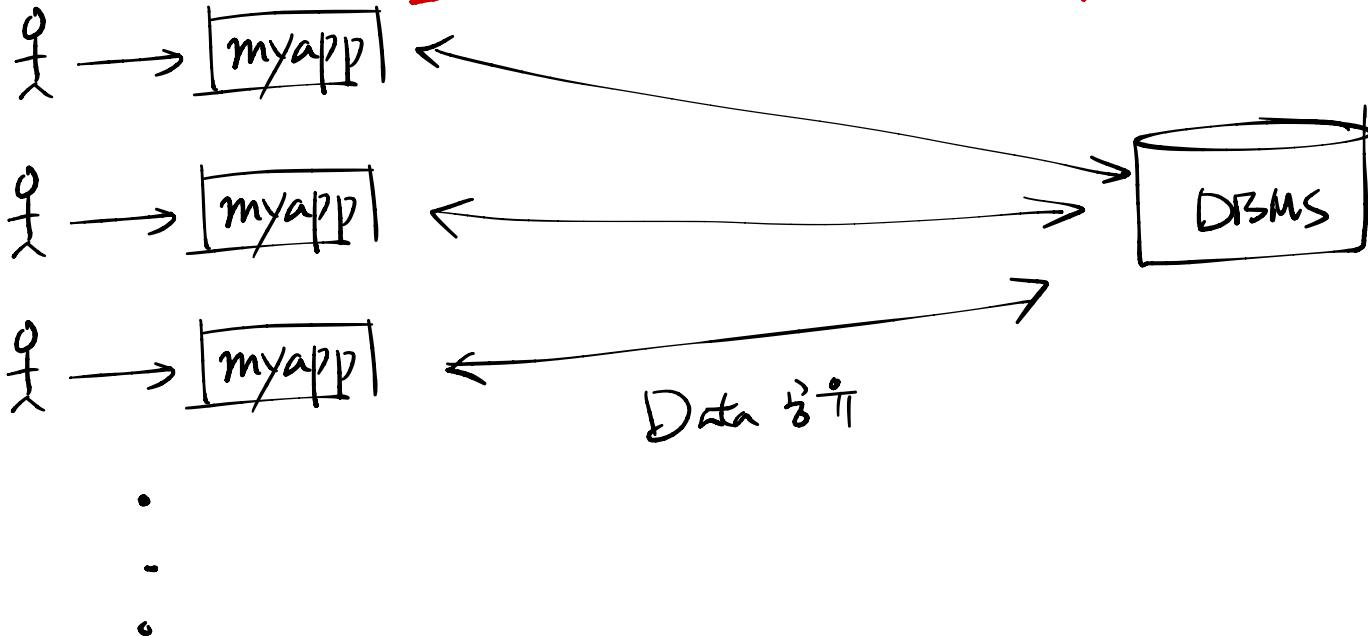


## \* DaoFactory 퀵 II



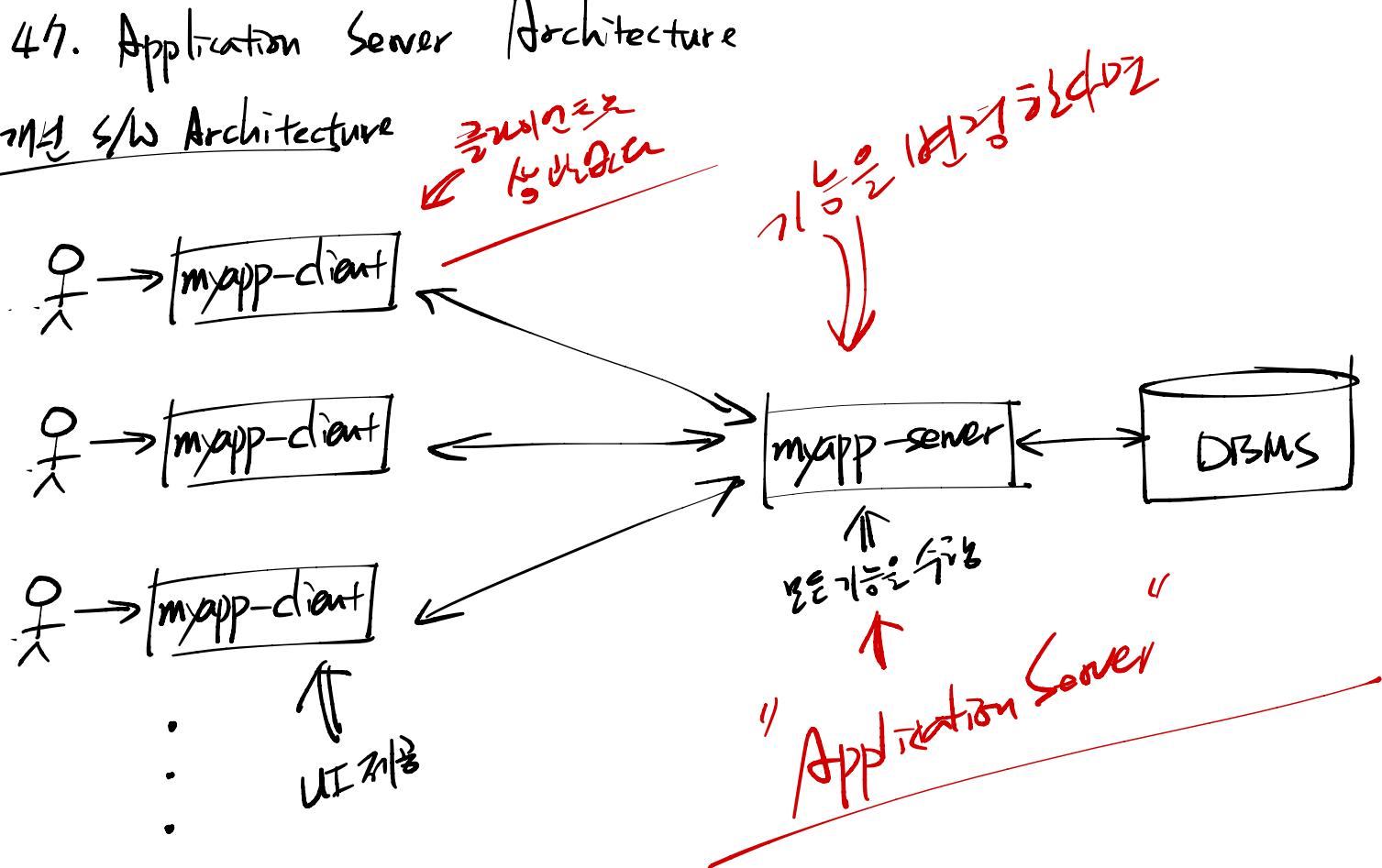
## 47. Application Server Architecture

### ① 2-tier S/W Architecture

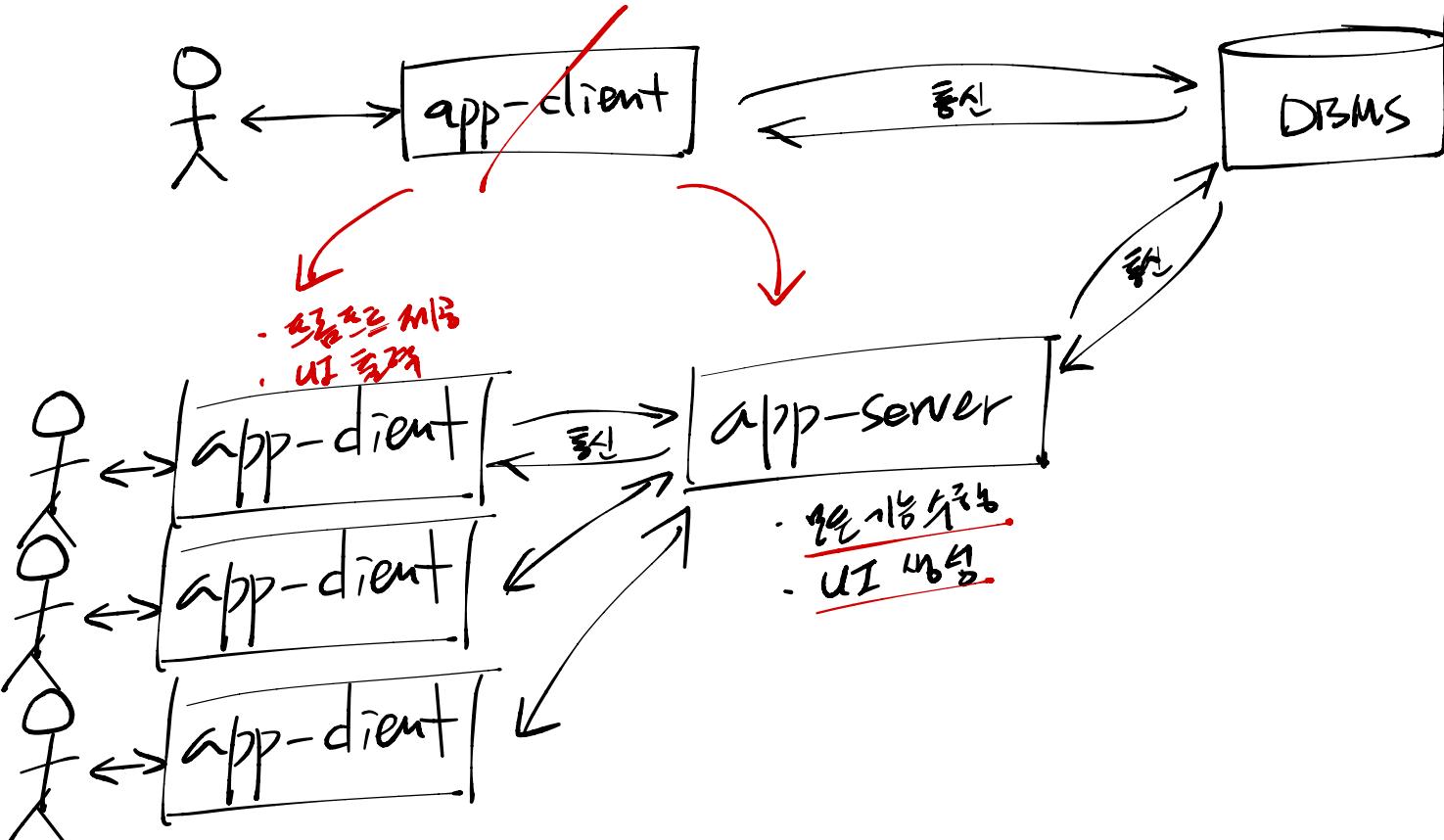


## 4.7. Application Server Architecture

### ② 개발 S/W Architecture

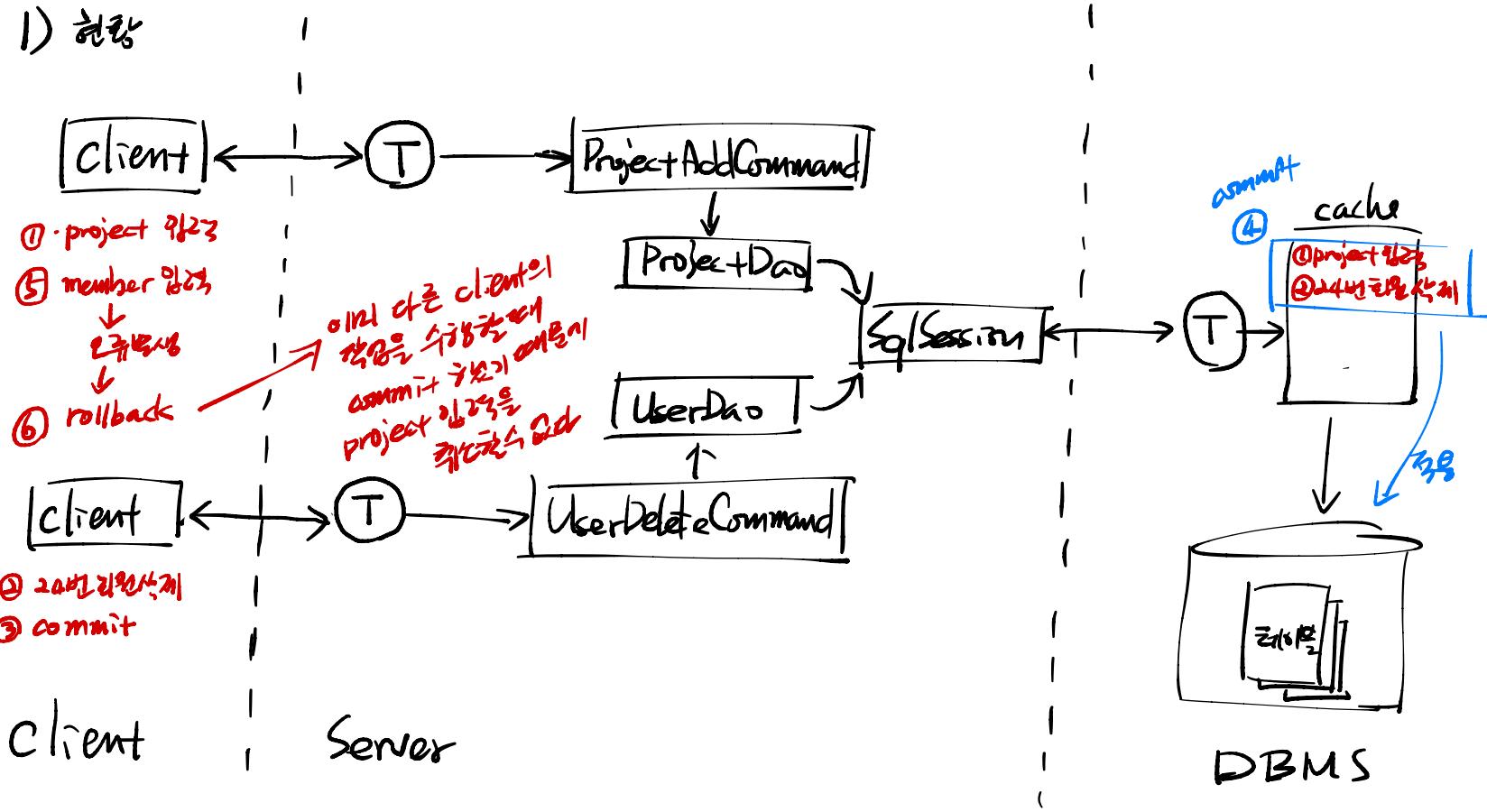


## 47. Application Server Architecture

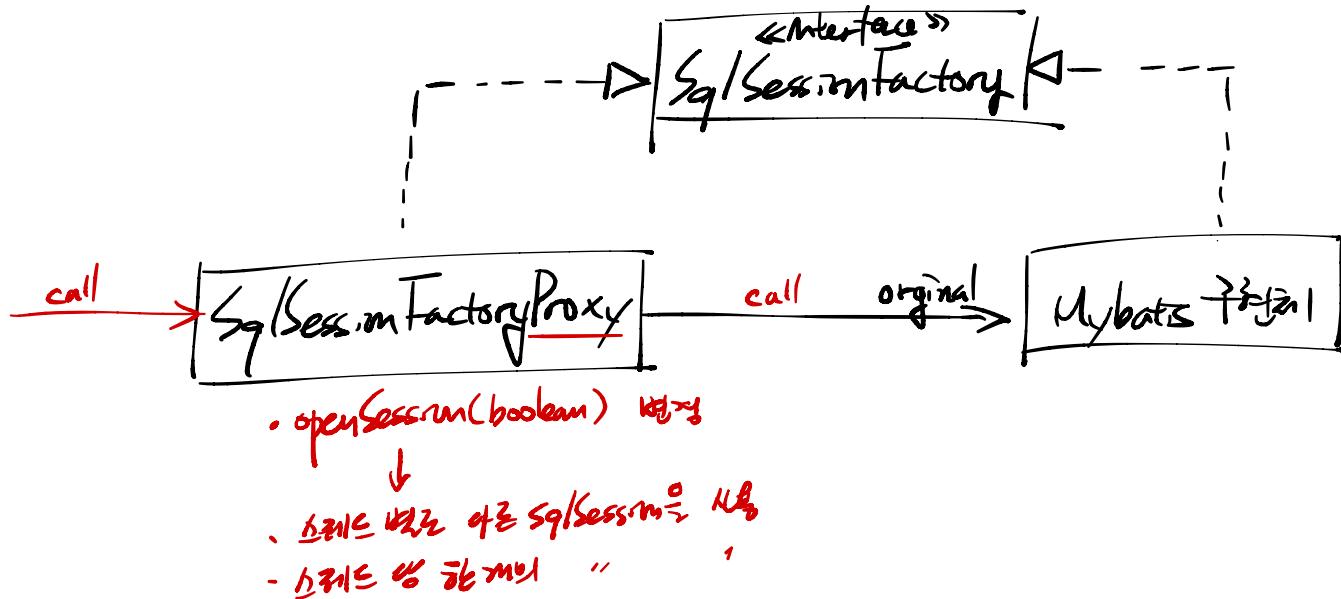


43. DB가 1번 했던 것과 2번 했던 것과 3번 했던 것과 같은지, 어떤지

1) 예상



\* 디자인 패턴에서의 투명성 적용 SqlSession 을 공유하는 구조



2) 거버넌스

