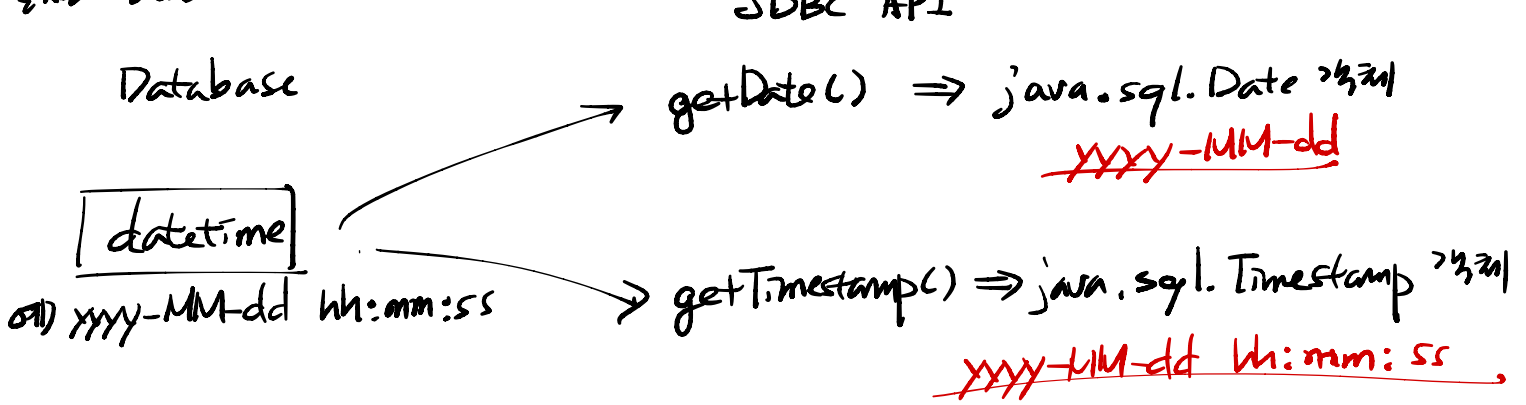
40. DBMS 활용하기(계속)

* 날짜 관련된 함수



4. 외부키(foreign key) 사용

myapp_users

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

myapp_projects

project_id	title	description	startdate	enddate	members
101	p1	-	-	-	1,3
102	p2	-	-	-	1,2
103	p3	-	-	-	1,2,3

1번회원 삭제

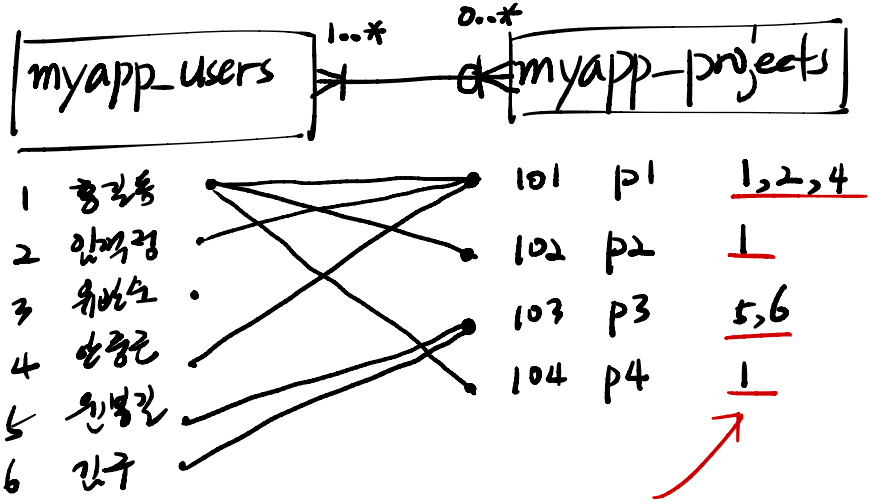
프로젝트 테이블의
멤버 변경과 무관한 변경과 된다

무결성이 깨진다!
↳ 결함 발생!

4. 외부키(foreign key) 사용

1) 테이블의 데이터간에 관계를 파악

(다 대 다 관계)

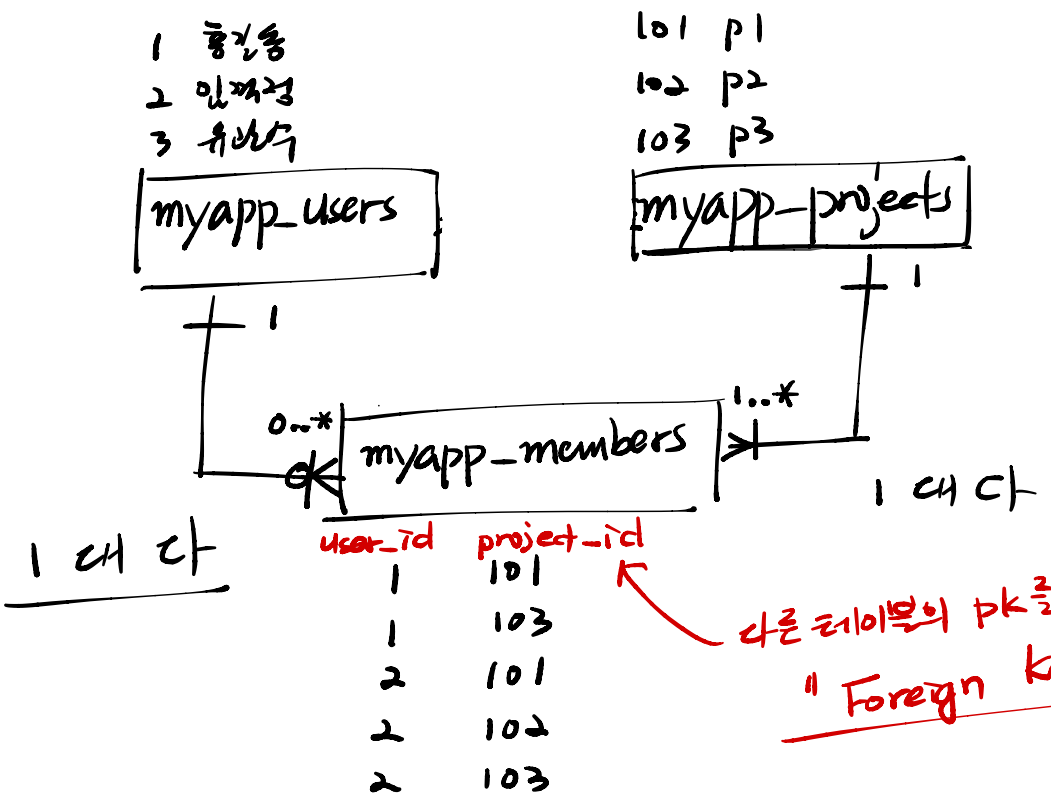


(1 대 다 / 다 대 다) 관계를 위와 같이 하나의 컬럼에 저장하면 무결한 데이터 입력을 막을 수 없다

Foreign key 제약조건 + 다대다의 경우 별도 테이블로 관리

해결?

* 다대다 관계를 다루기

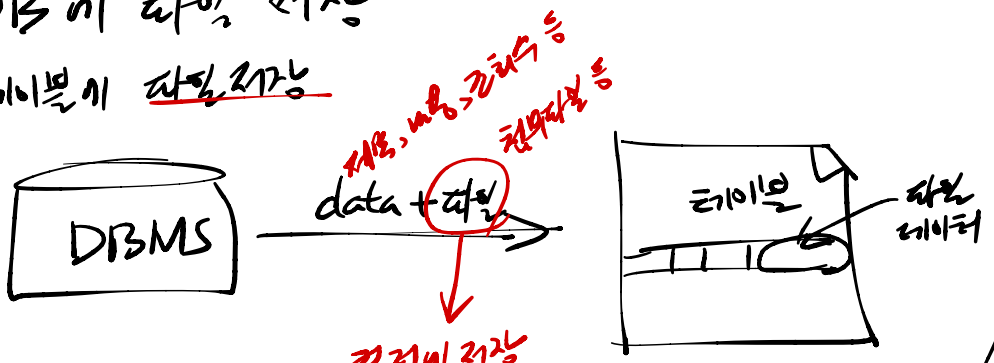


★ 외부키 (foreign key)

- 다른 테이블의 pk를 가리킴
- 없는 pk를 저장할 수 없음

* DB에 파일 저장

① 테이블에 파일저장



컬럼에 저장

↓

데이터베이스가 저장된 (blob 타입으로 저장)

⇓

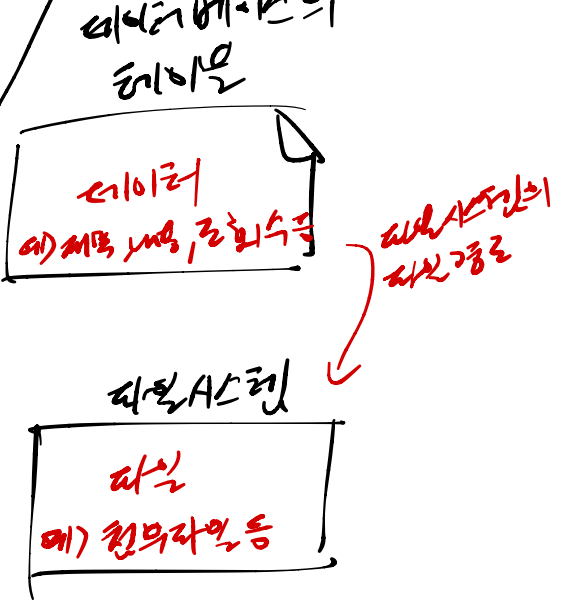
파일 안에 데이터 저장하는 방식

⇓

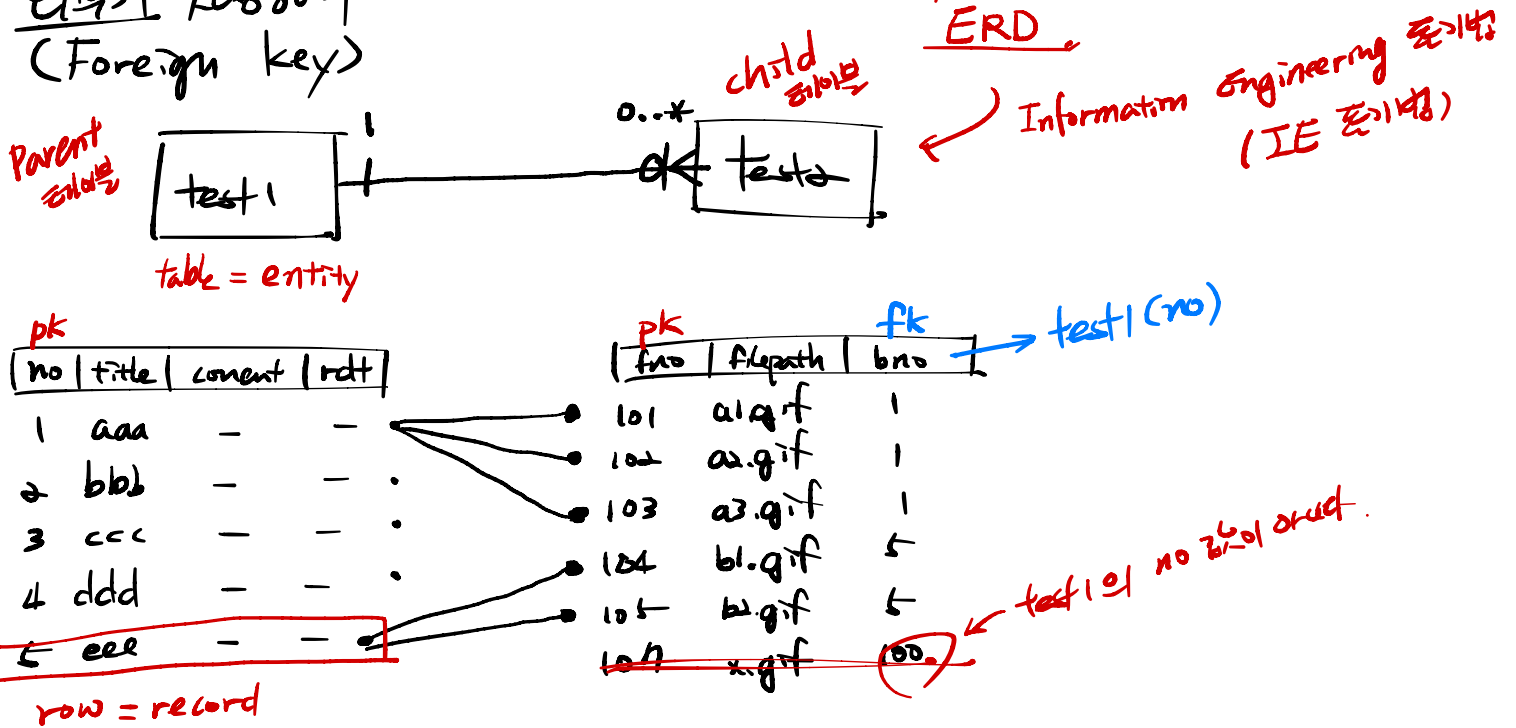
파일을 읽는 속도 매우 느려짐

하지만?

② 파일을 분리해서 저장



4. 외래키 사용하기 (Foreign key)



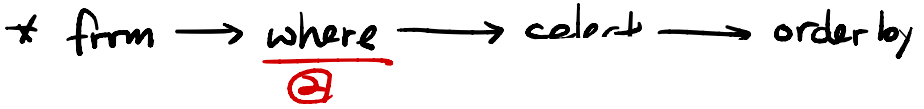
* from → where → select → order by

① 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 |
+-----+--------+------+------+
```

① ②

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




```
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
from room
order by loc asc, name2 asc;
```

* from → where → select → order by

③ ← 결과로 추출할 컬럼 표시 (가상 컬럼 준비

```
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 |
+------+----------+-------+-------+
```

✓ name2 (loc - name)

강남-501
강남-502
강남-503
종로-301
종로-302
종로-303
서초-301
서초-302
서초-501
서초-601

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

* from → where → select → order by

```
select rno, name, concat(loc,'-',name) as name2
from room
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 → 결과추출 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 안에서 지정한 컬럼의 값은 결과 데이터로 추출한다

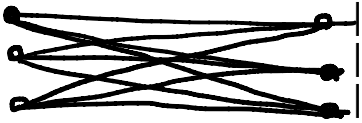
* JOIN - cross join = cartesian join

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 ⇒ 같은 이름의 컬럼 값을 기준으로 데이터를 합친다

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 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 => 한계!

```
select * from board2;
+----+---------+---------+
| no | title   | content |
+----+---------+---------+
|  1 | 제목1   | 내용    |
|  2 | 제목2   | 내용    |
|  3 | 제목3   | 내용    |
+----+---------+---------+
```

```
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)
```

삽입된 데이터끼리 조인된 적 없다.

* JOIN - join ~ using(기준컬럼, ...)

```
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 |
+-----+--------+-----+
```

여러 컬럼 값을 기준으로 조인할 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(기준컬럼, ...) ⇒ 한계!

↳ 같은 이름을 가진 컬럼이 없다.

```
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 |
+-----+----------+-----+
```

board5 테이블에는 bno 컬럼이 없다!

```
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'
```

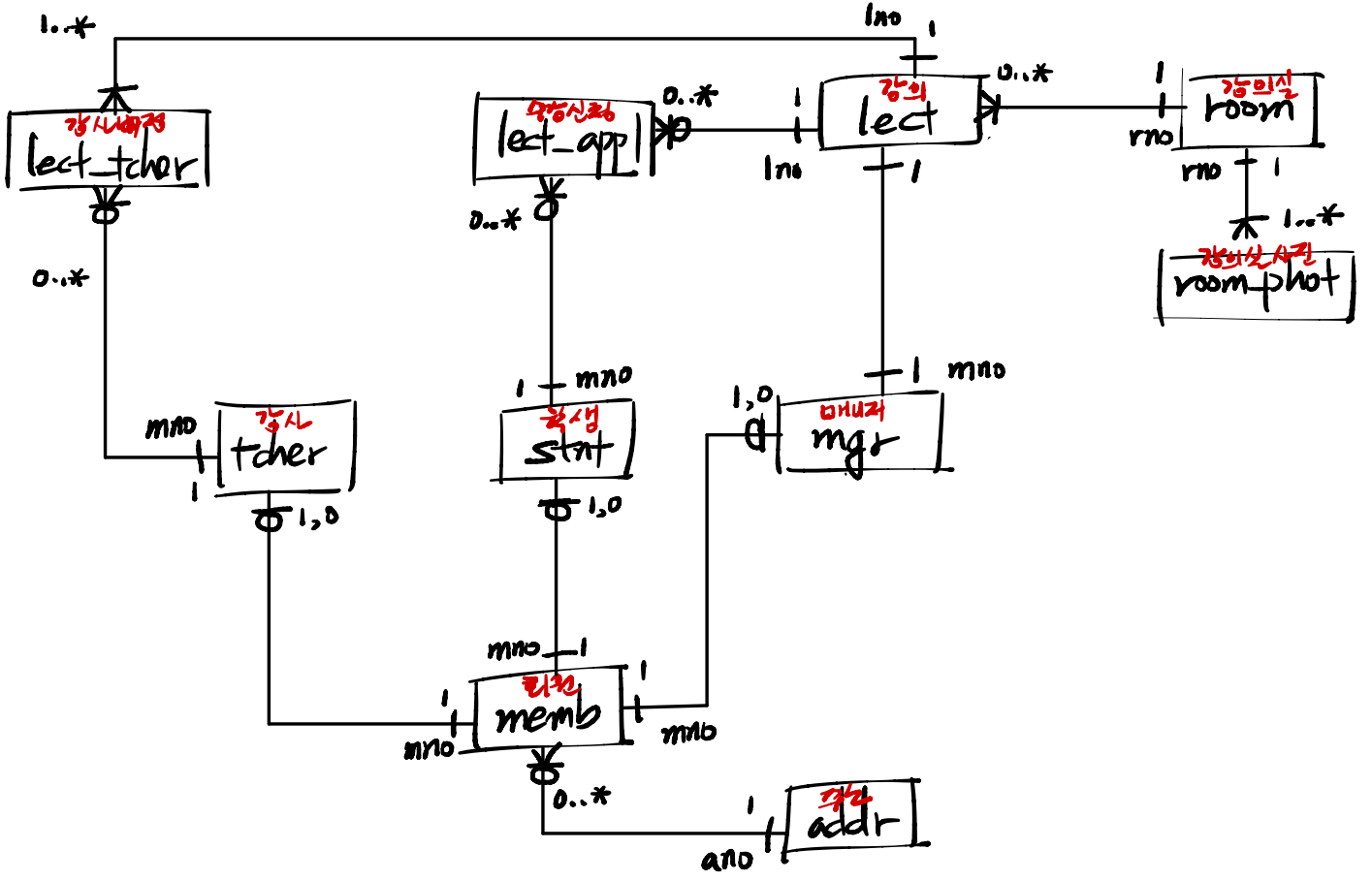
* JOIN - join ~ on 절
(inner join)

```
select * from board5;
+----+---------+---------+
| no | title   | content |
+----+---------+---------+
|  1 | 제목1   | 내용    |
|  2 | 제목2   | 내용    |
|  3 | 제목3   | 내용    |
+----+---------+---------+
```

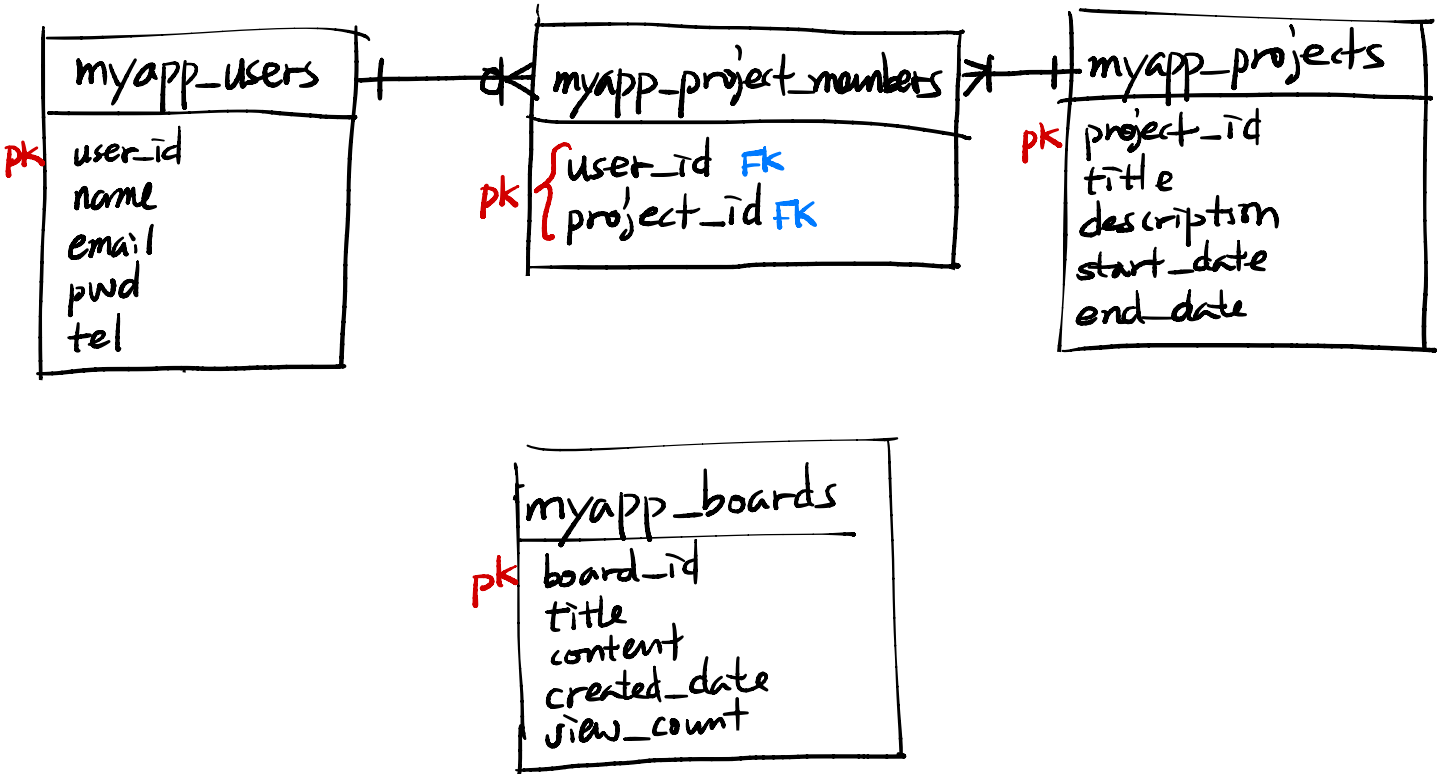
```
select * from attach_file5;
+-----+----------+-----+
| fno | filepath | bno |
+-----+----------+-----+
|   1 | a1.gif   |   1 |
|   2 | a2.gif   |   1 |
|   3 | c1.gif   |   3 |
+-----+----------+-----+
```

```
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 |
+----+---------+---------+-----+----------+-----+
```

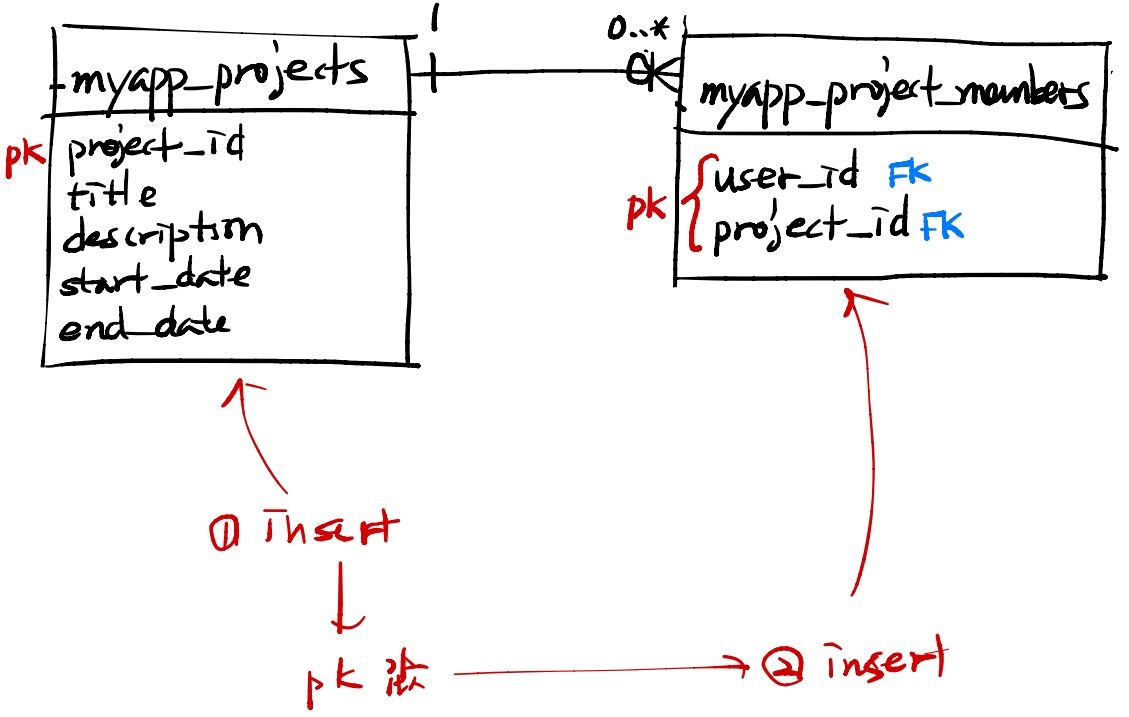
* JOIN 연습 - ERD (Entity Relationship Diagram)



* myapp ERD


myapp_users
pk
user_id
name
email
pwd
tel
myapp_project_members
pk
user_id FK
project_id FK
myapp_projects
pk
project_id
title
description
start_date
end_date
myapp_boards
pk
board_id
title
content
created_date
view_count

* 프로젝트 insert


1
0..*
myapp_projects
pk
project_id
title
description
start_date
end_date
myapp_project_members
pk
user_id FK
project_id FK
① insert
pk 값
② insert

* 자동생성된 PK 값 가져오기

```
insert into x_board (title, contents)
values ('aaa', 'xxx'),
       ('bbb', 'yyy'),
       ('ccc', 'zzz');
```

→

DBMS

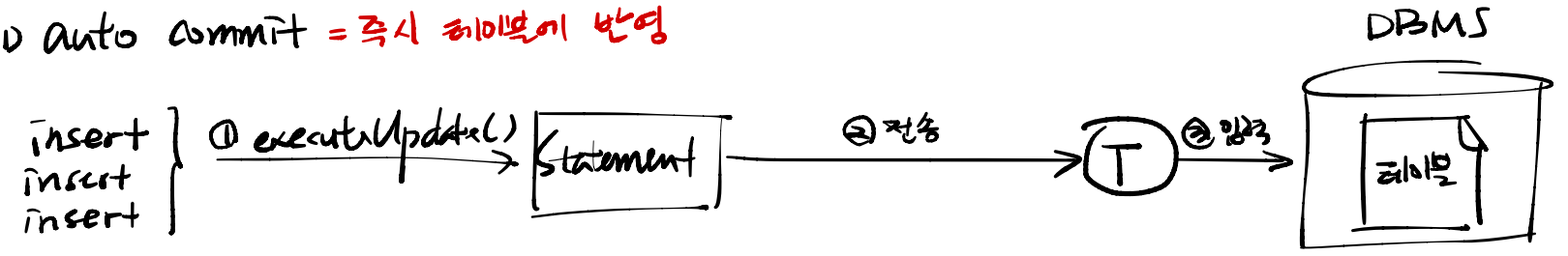
board_id	title	contents	...
202	aaa	xxx	
203	bbb	yyy	
204	ccc	zzz	

PK

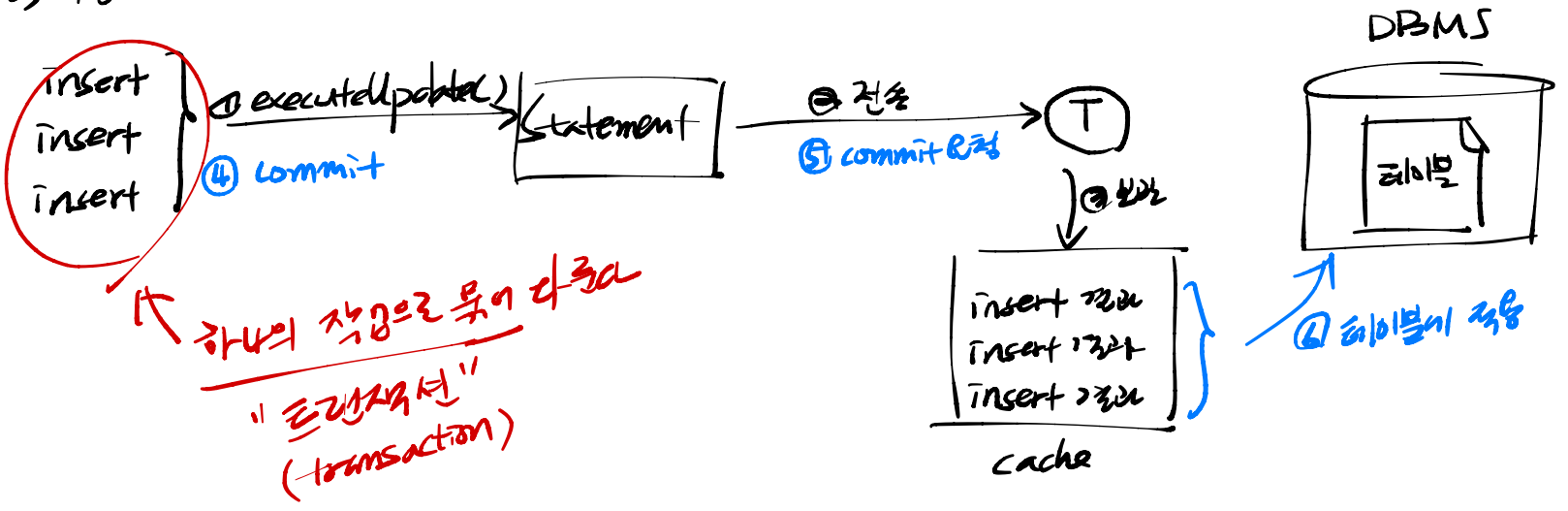
```
ResultSet keyRS = stmt.getGeneratedKeys();
keyRS.next();   → 1번 202 ← pk 레코드
keyRS.getInt(1); 202 ← pk 컬럼 값
keyRS.next();   → 1번 203 ← pk 레코드
keyRS.getInt(1); 203 ← pk 컬럼 값
```

* auto commit vs 수동 commit

1) auto commit = 즉시 테이블에 반영

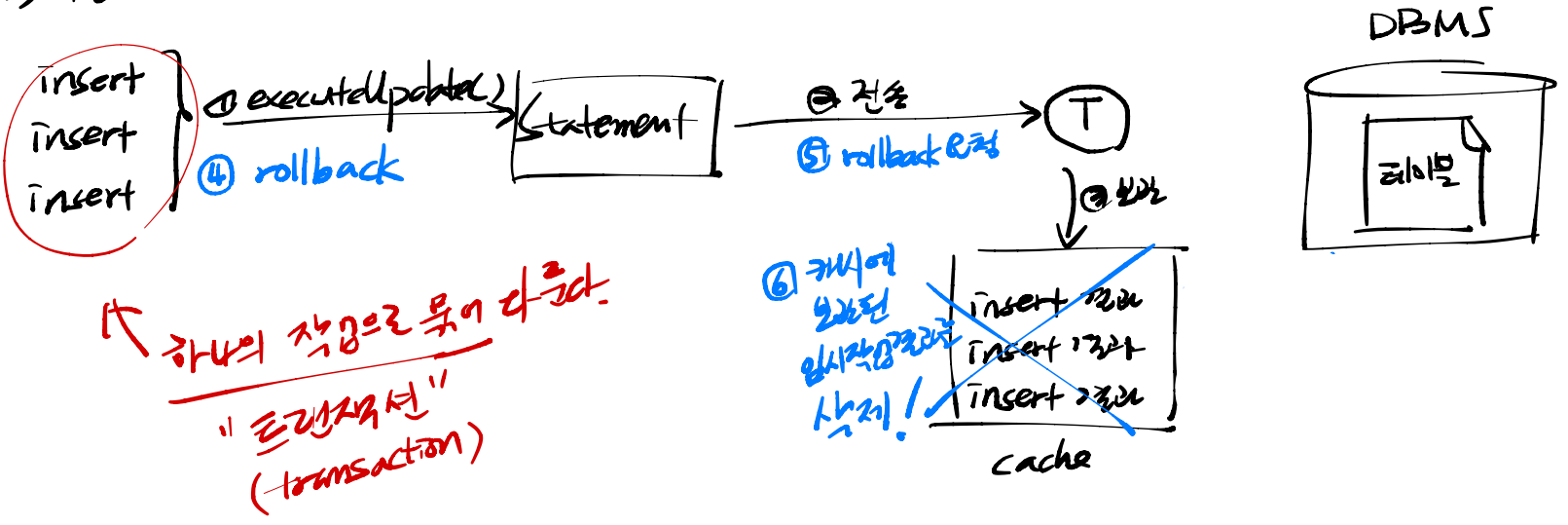


2) 수동 commit = commit



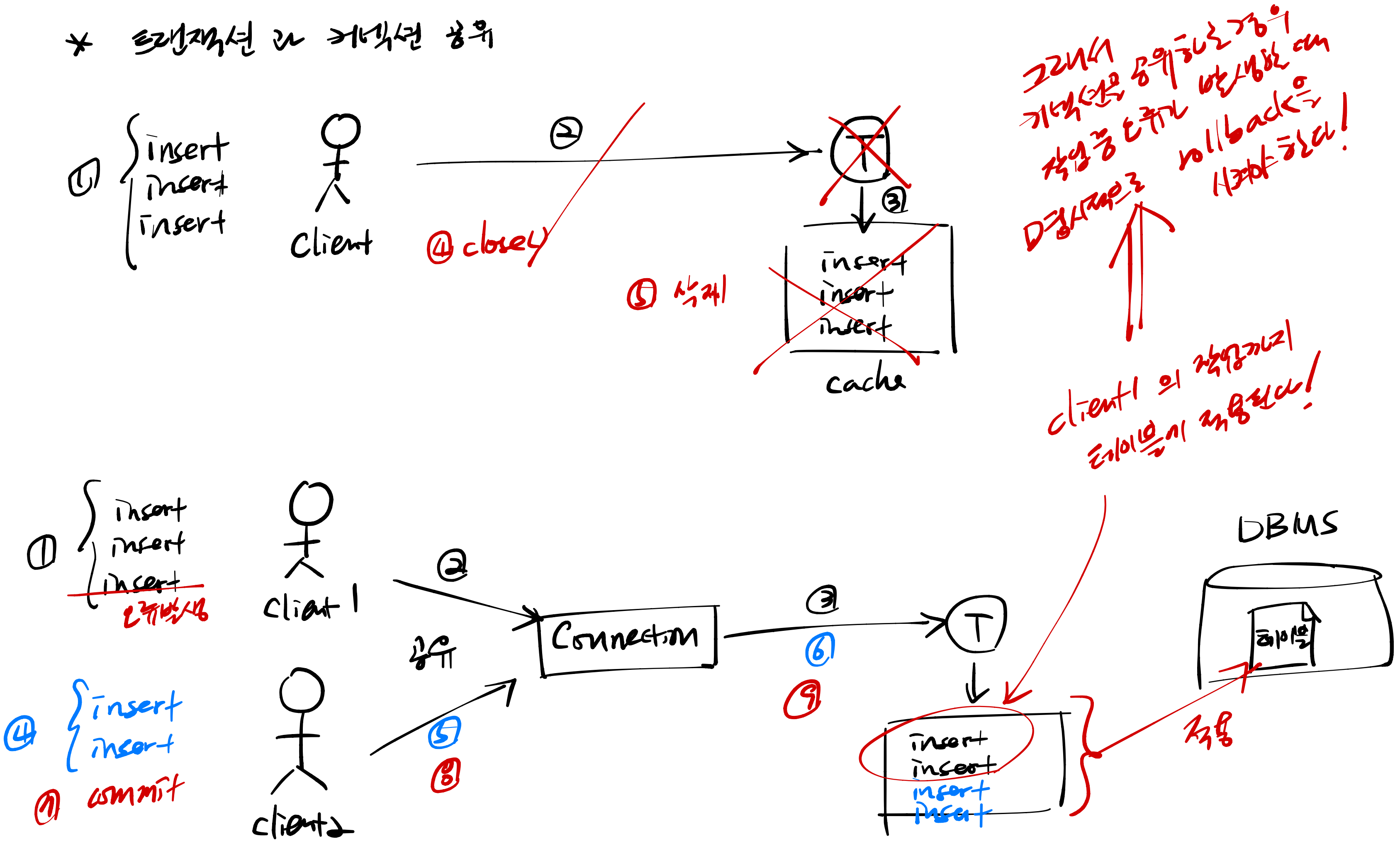
* auto commit vs 수동 commit

2) 수동 commit = rollback



↑ 하나의 작업으로 묶어 다룬다.
= 트랜잭션 (transaction)

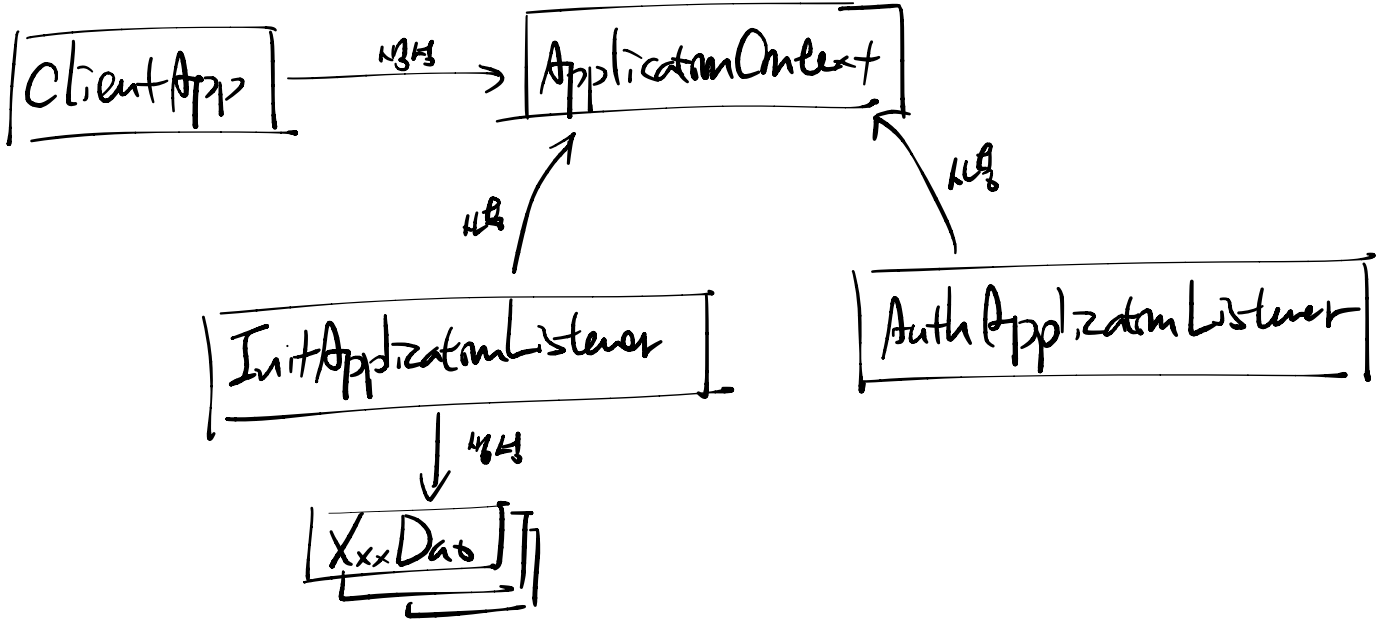
* 트랜잭션 과 커넥션 공유



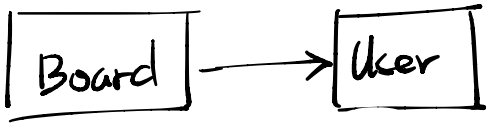
① insert insert insert
Client
②
③
④ close
⑤ 삭제
insert insert insert
cache
그래서 커넥션을 공유하는 경우 작업중 오류가 발생할 때 rollback을 DB차원으로 시켜야 한다!
client1 의 작업까지 테이블에 적용된다!
① insert insert insert
오류발생
Client1
②
공유
Connection
③
⑥
⑨
T
④ insert insert
⑦ commit
Client2
⑤
⑧
insert insert insert insert
적용
DBMS
테이블

42. 로그인/로그아웃 구현하기

① ApplicationListener 관련 객체 동작



* 게시글 + 작성자




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

43. SQL 방법 공개 방식

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

title = aaa
contents = xxx

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

* SQL injection 예 1

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

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

" insert into x_board (title, contents)
values ('aaa', 'xxx'), ('bbb', 'yyy'), ('ccc', 'zzz')"

* SQL 방법 공격 예2

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

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 실행 하기 방법 ⇒ PreparedStatement

PreparedStatement stmt = con.prepareStatement(SQL);

insert into x_board (title, contents) values (?, ?)

In-parameter = 값을 대로 준다

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

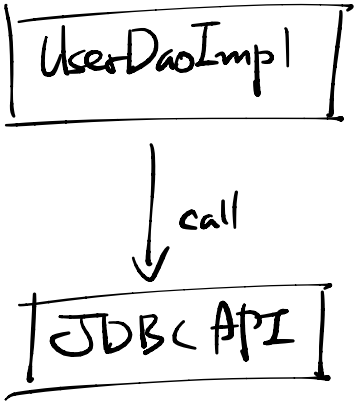
In-parameter에 들어갈 값.

SQL문과 SQL에 넣을 값을 분리!

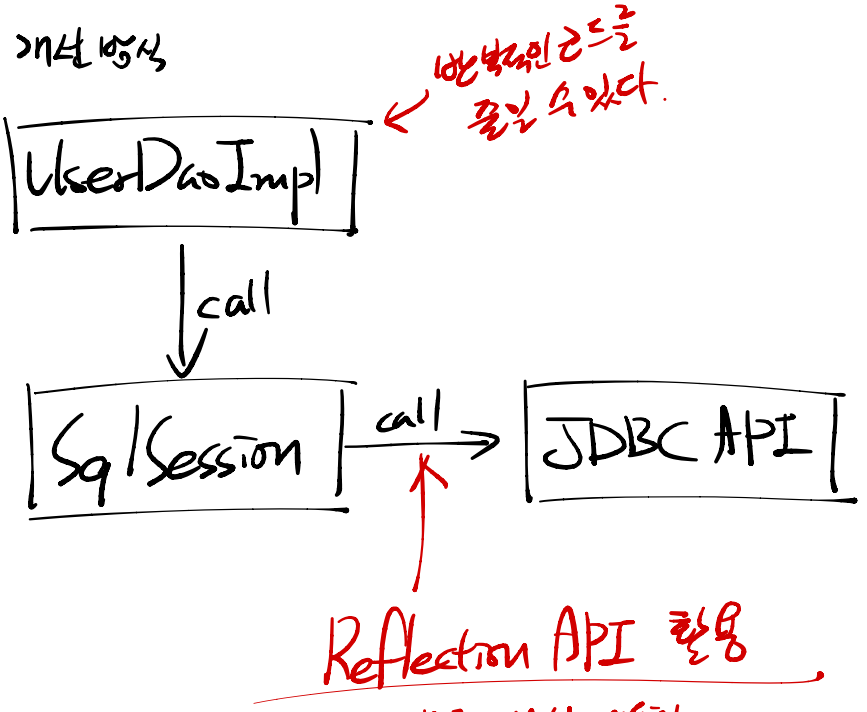
In-parameter 번호 1부터 시작

44. JDBC 코드 캡슐화

① 이전 방식

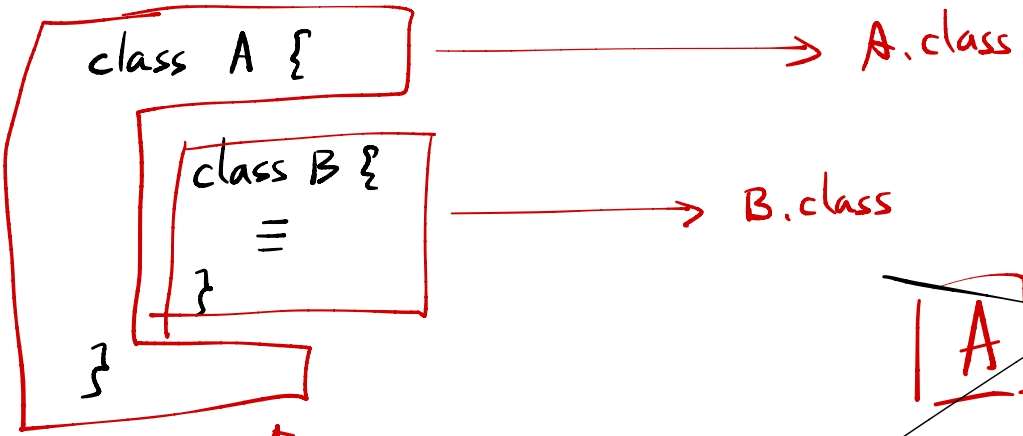


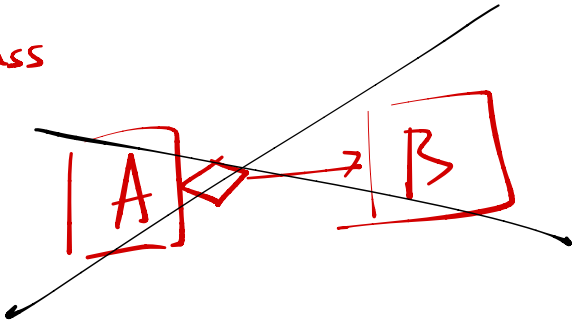
② 개선 방식



* Reflection API

중첩클래스





소스코드에서는 중첩은 B클래스가 A클래스 안에서 사용되는 의미이다

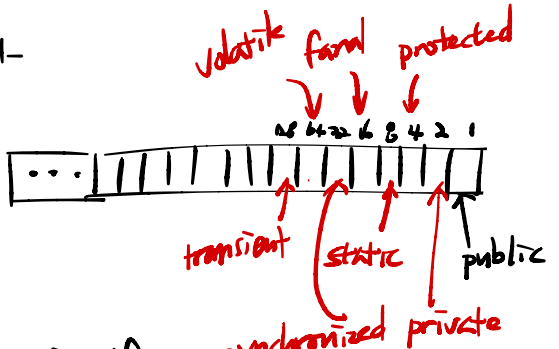
= A는 B를 포함하고 있다 OOOO!

↓

사용범위를 제한하는 문법이다

* 메서드 modifier 비트 연산-

modifier 값



예) public static final float PI = 3.14f;

↳ 00011001(modifier값)

public : 00000001
private : 00000010
protected : 00000100
static : 00001000
final : 00010000
⋮

```
  00011001
& 00000001 (public)
----------
  00000001  > 0
```

```
  00011001
& 00000010 (private)
----------
  00000000  > 0
```

* 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;}
}
```

멤버(field)

getter

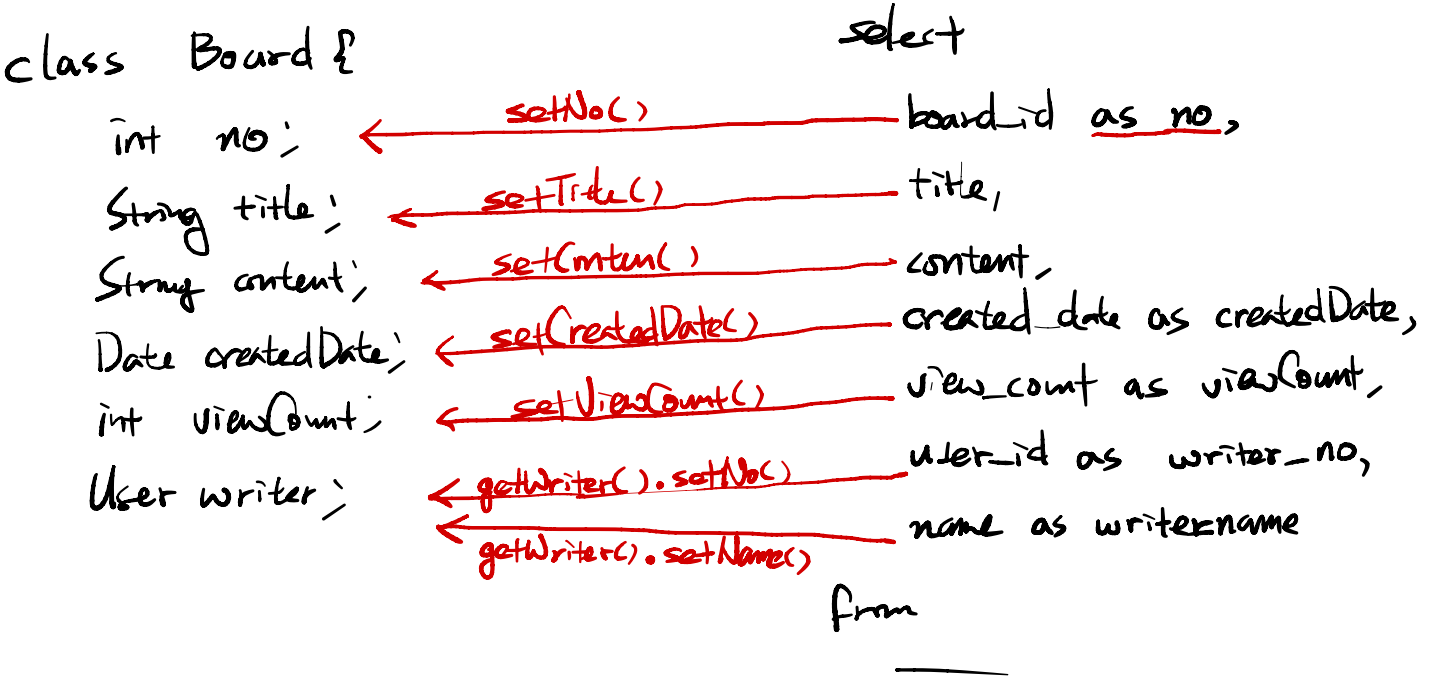
setter

property

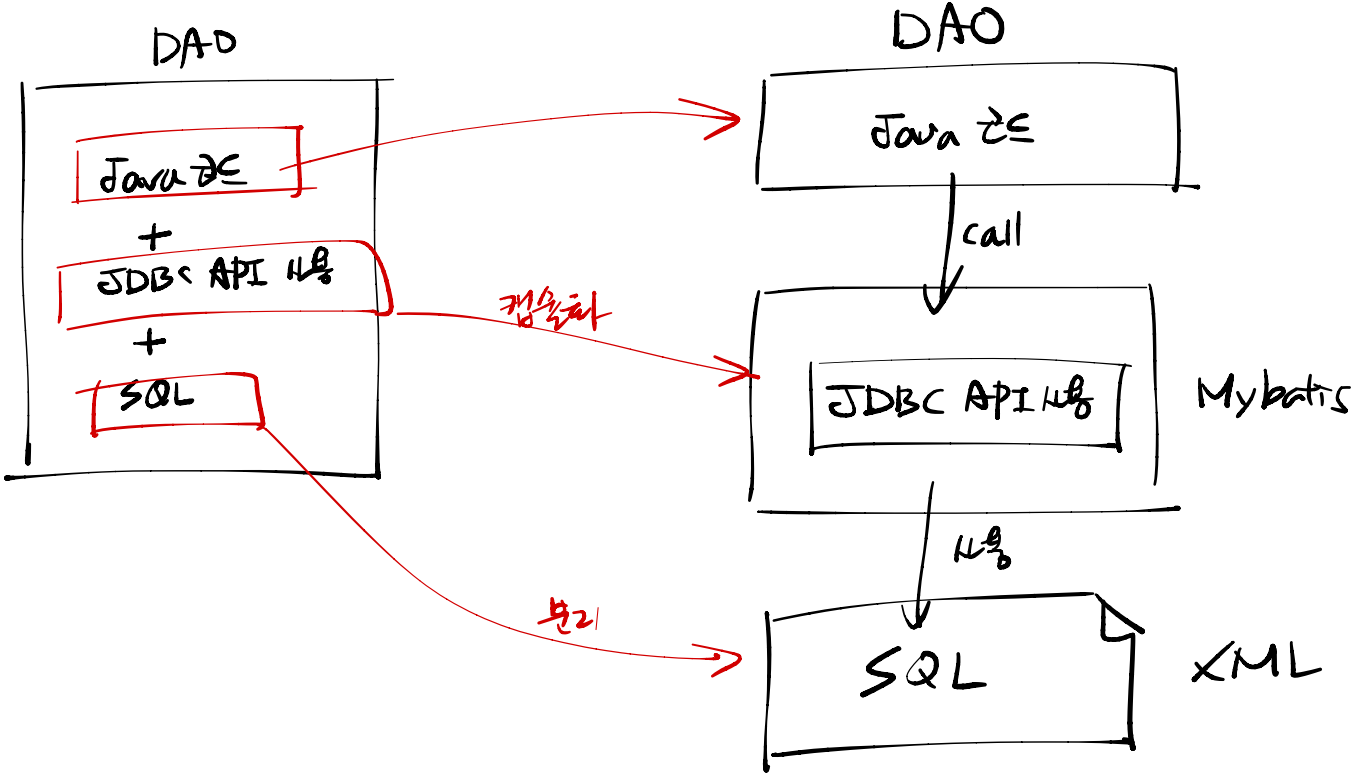
~~setCreateDate() {~}~~

"createDate" ← property name

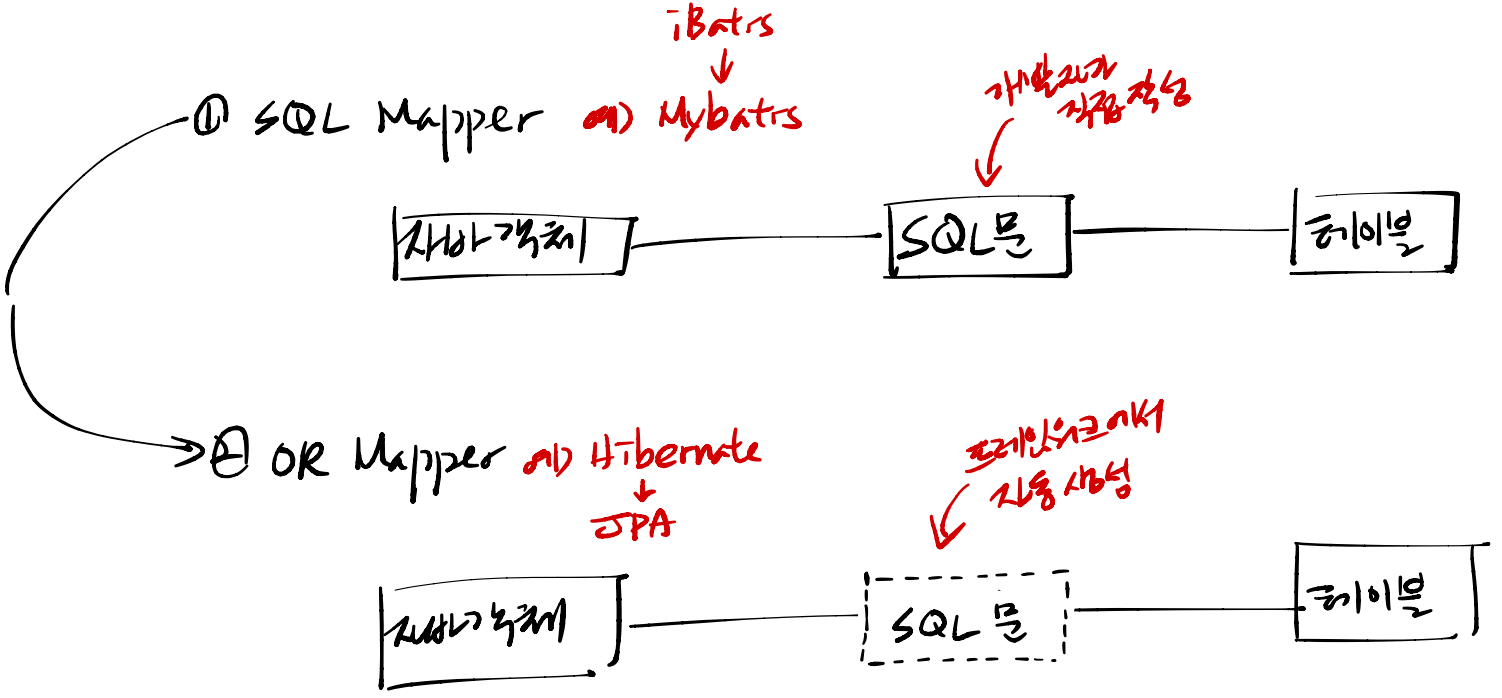
* Field & Property



45. Mybatis 장점



* 퍼시스턴스 프레임워크 (Persistence Framework)



* Mybatis 사용 순서 — ① 설정파일 작성

mybatis-config.xml

-//mybatis.org// DTD ——— //EN

규칙을 만들어 준 회사명 / 규칙을 작성한 언어

+ 국제공인규칙
- 사설규칙

규칙이름

1.0만 가능

문서를 인쇄할때 사용하는 문자인코딩

반드시 첫 줄에 와야 함.

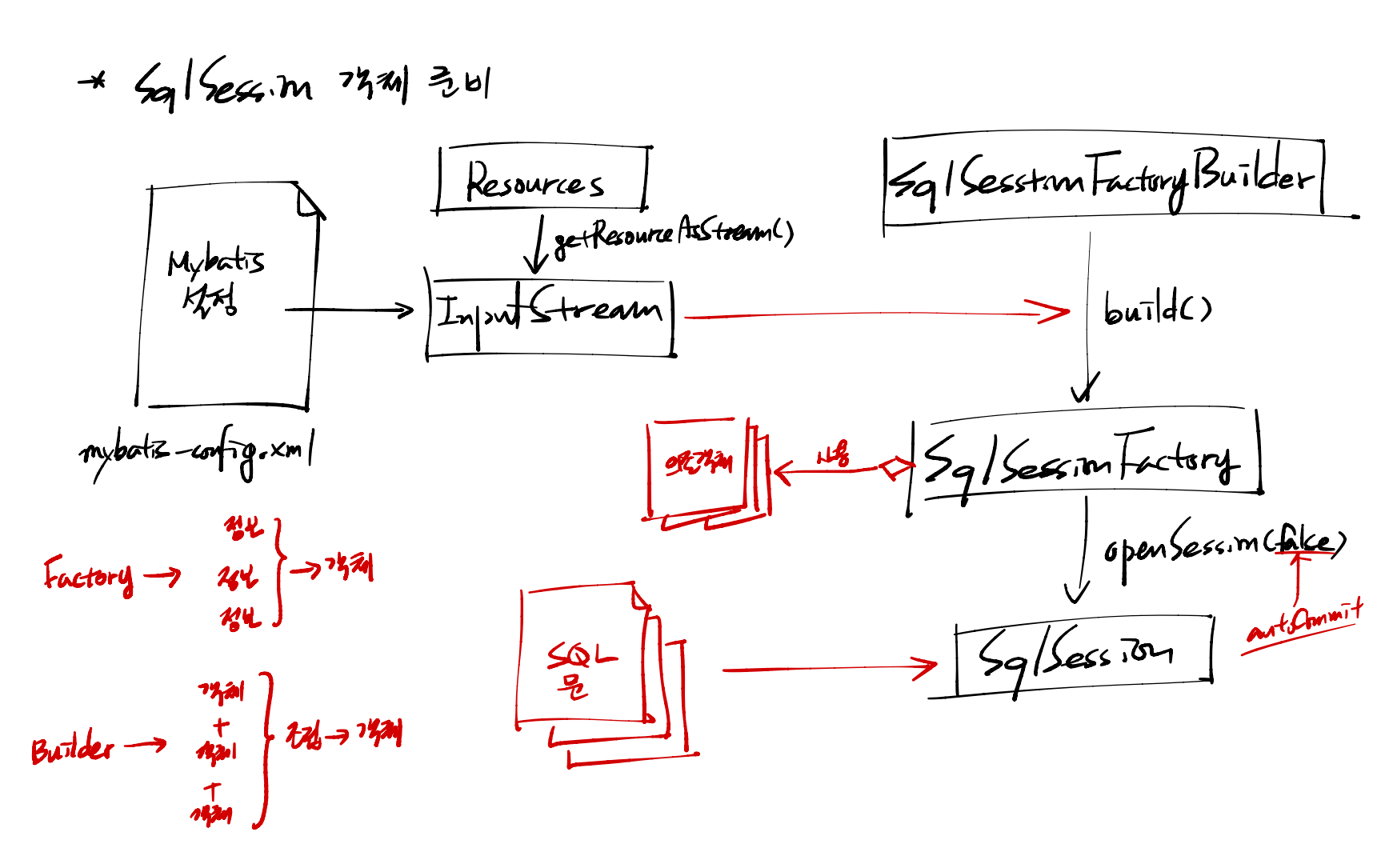
```
<?xml version="1.0" encoding="UTF-8" ?>
```
← XML 선언

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

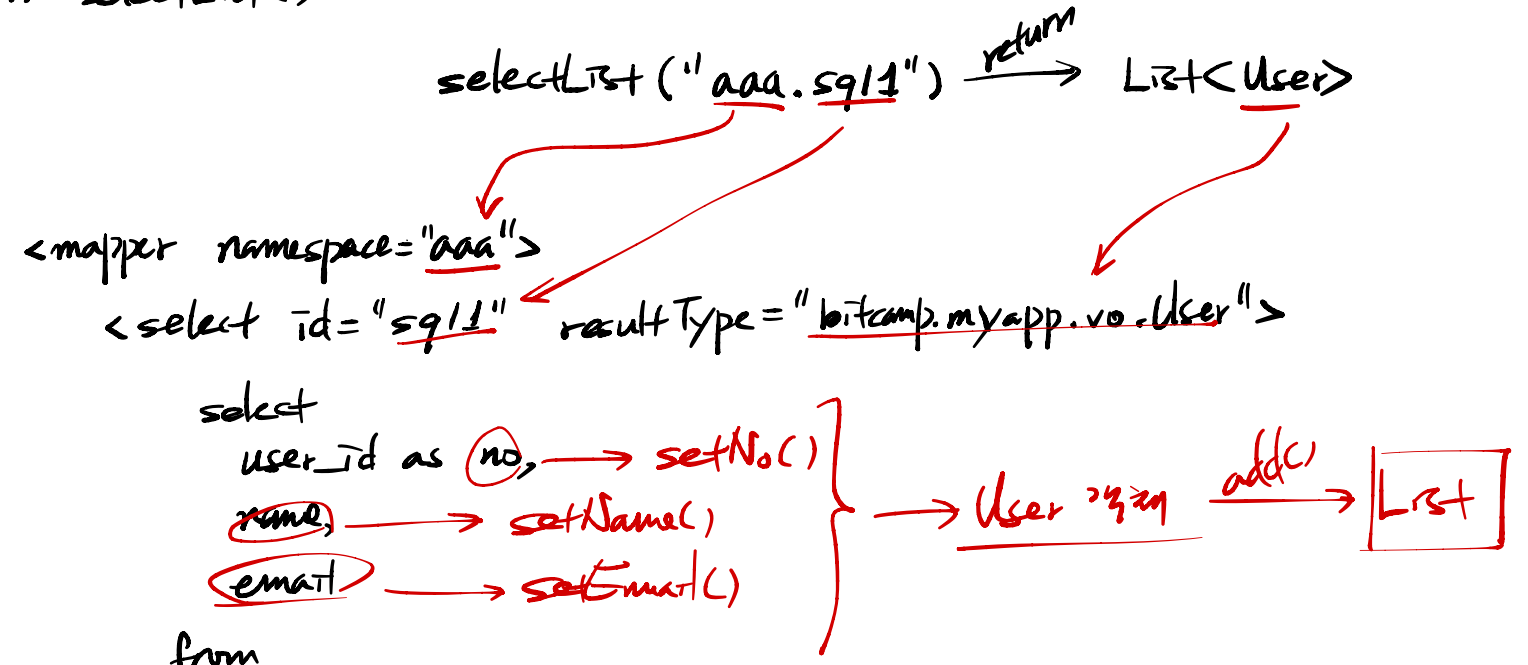
<!DOCTYPE : XML 규칙선언
configuration : root element (태그)
PUBLIC : 공개
SYSTEM (비공개) ← 특정회사에서만 사용하는 규칙일때
"https://———/—.dtd" : 규칙을 정의한 컴퓨터의 URL

```
<configuration>
    |
</configuration>
```
← root element = XML 문서에 한 개만 와야만 함.

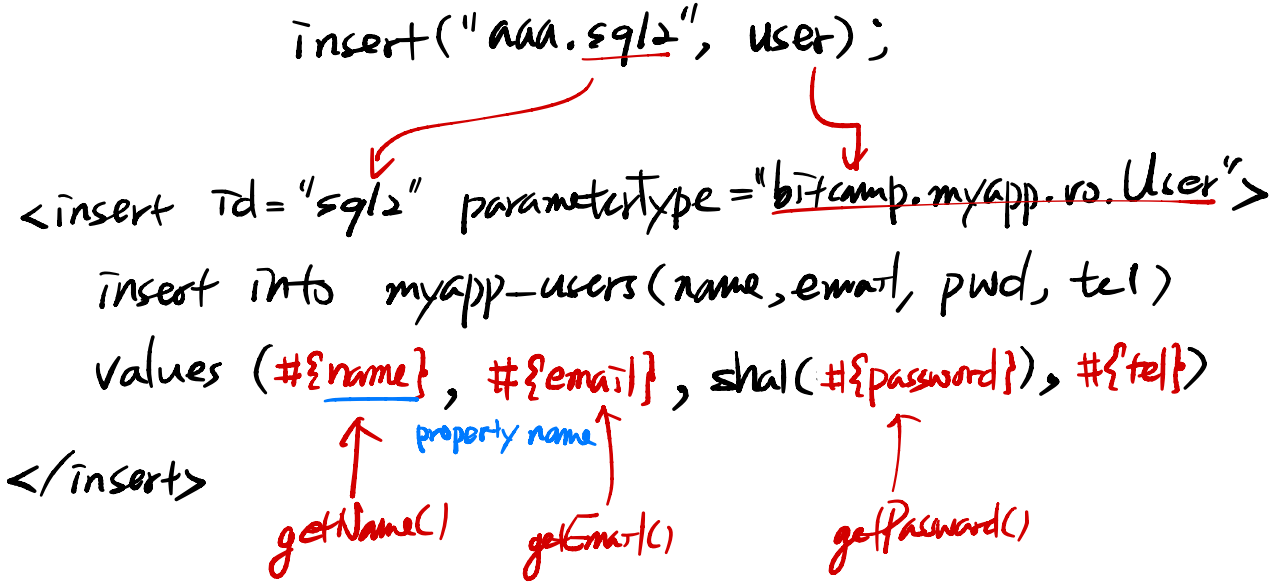
* SqlSession 객체 준비



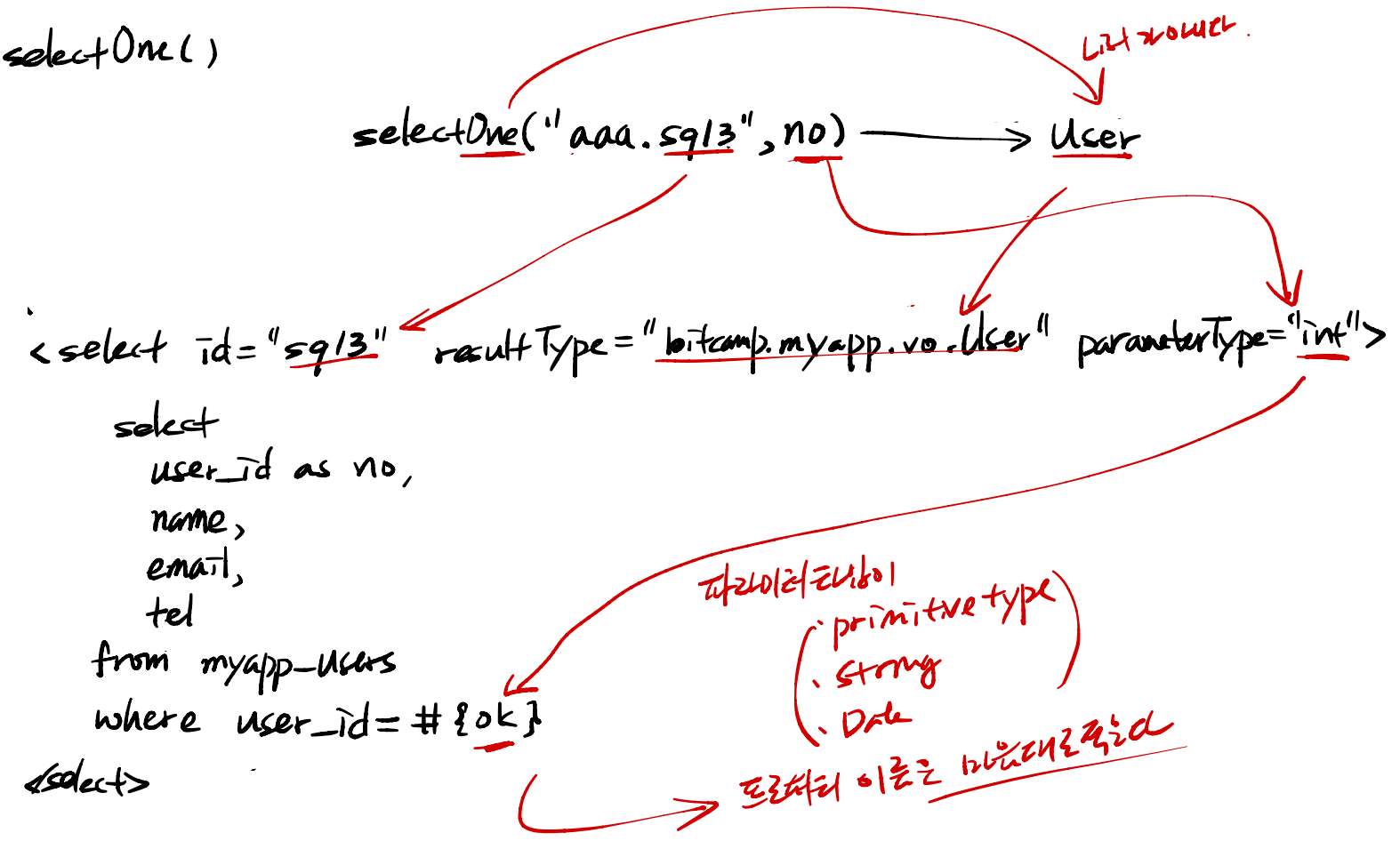
* selectList()


selectList("aaa.sql1") return List<User>
<mapper namespace="aaa">
<select id="sql1" resultType="bitcamp.myapp.vo.User">
select
user_id as no, → setNo()
name → setName()
email → setEmail()
from
order by
</select>
...
→ User 객체 add() → List

* insert()


insert("aaa.sql", user);
<insert id="sql" parameterType="bitcamp.myapp.vo.User">
insert into myapp_users(name, email, pwd, tel)
values (#{name}, #{email}, sha1(#{password}), #{tel})
</insert>
property name
getName()
getEmail()
getPassword()

* selectOne()


selectOne("aaa.sql3", no) → User
<select id="sql3" resultType="bitcamp.myapp.vo.User" parameterType="int">
select
user_id as no,
name,
email,
tel
from myapp_users
where user_id=#{ok}
</select>
파라미터타입이
primitive type
String
Date
프로퍼티 이름은 아무거나 적어도

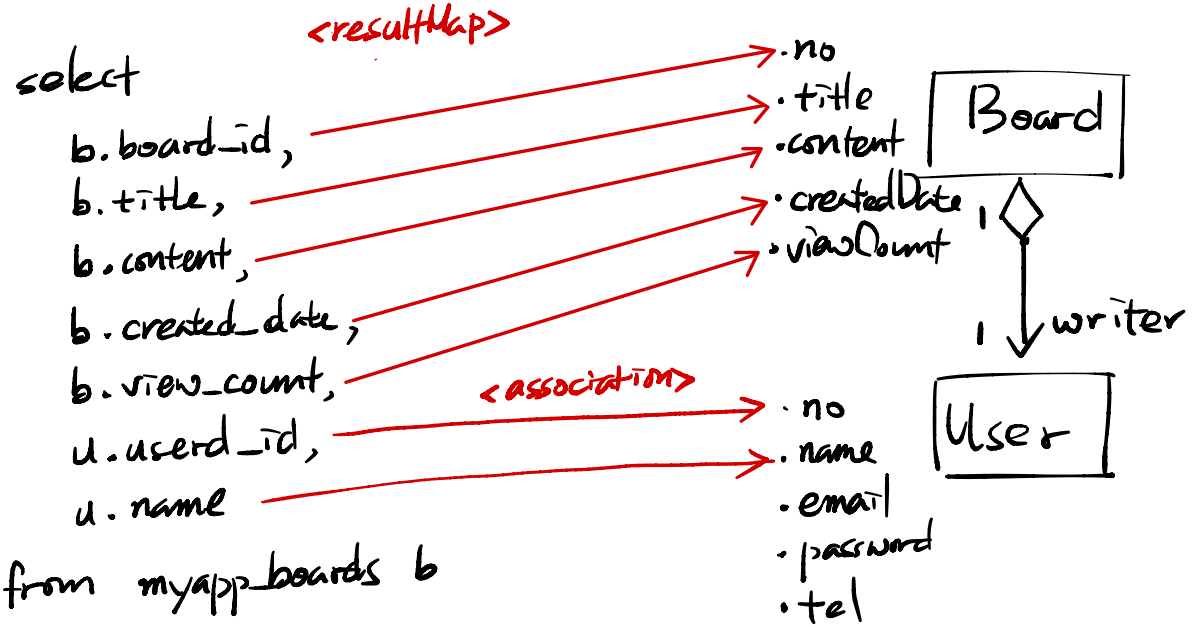
* 트랜잭션 처리

sqlSession.insert()
update()
delete()
} sqlSession.commit()
" .rollback()

UserAddCommand

try {
sqlSession.insert();
sqlSession.commit();
}
catch() { sqlSession.rollback(); }

* 조인결과를 객체에 저장하기



*<resultMap> + <association>

<resultMap> 컬럼을 객체 프로퍼티와 연결

<association> 컬럼을 포함객체의 프로퍼티와 연결하기

```
<resultMap id="BoardMap" type="bitcamp.myapp.vo.Board">
    <id column="board_id" property="no" />
    <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>
```

<id> PK 컬럼인 경우

column="board_id" 컬럼명

property="no" 객체(프로퍼티)명

```
class Board {
    User writer;
}
```

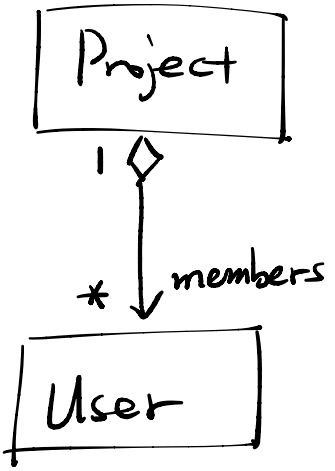
javaType="bitcamp.myapp.vo.User"

* 조인 결과를 객체에 저장하기 II

```
select
    p.project_id,
    p.title,
    p.description,
    p.start_date,
    p.end_date,
    pm.user_id,
    u.name
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 객체

→ User 객체



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

* 결과 값을 저장할 때 사용할 규칙 정의

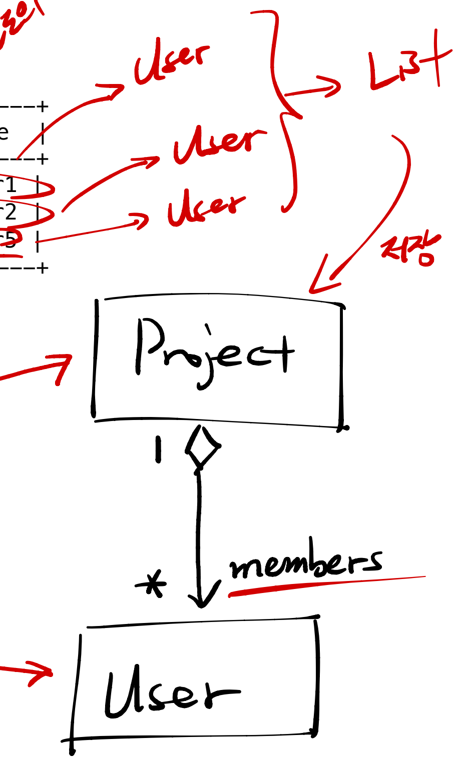
id 컬럼 값이 같으면 한 개의 값만 객체에 저장한다.

id로 지정된 user_id 컬럼 값이 다르기 때문에 별도의 객체로 저장한다

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

```xml
<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 구조화시켜 저장
X
데이터 끼리 상호작용을 할 수 있다

tag
시작태그 content 끝태그
ex) <태그명> 값 </태그명>
element(요소)

XML 태그를 가르킬 때 사용하는 문법

XML 작성 규칙 → 시작-끝, 태그이름규칙 준수 등
준수 = well-formed XML

XPath

XML

XSL

DTD Document Type Definition

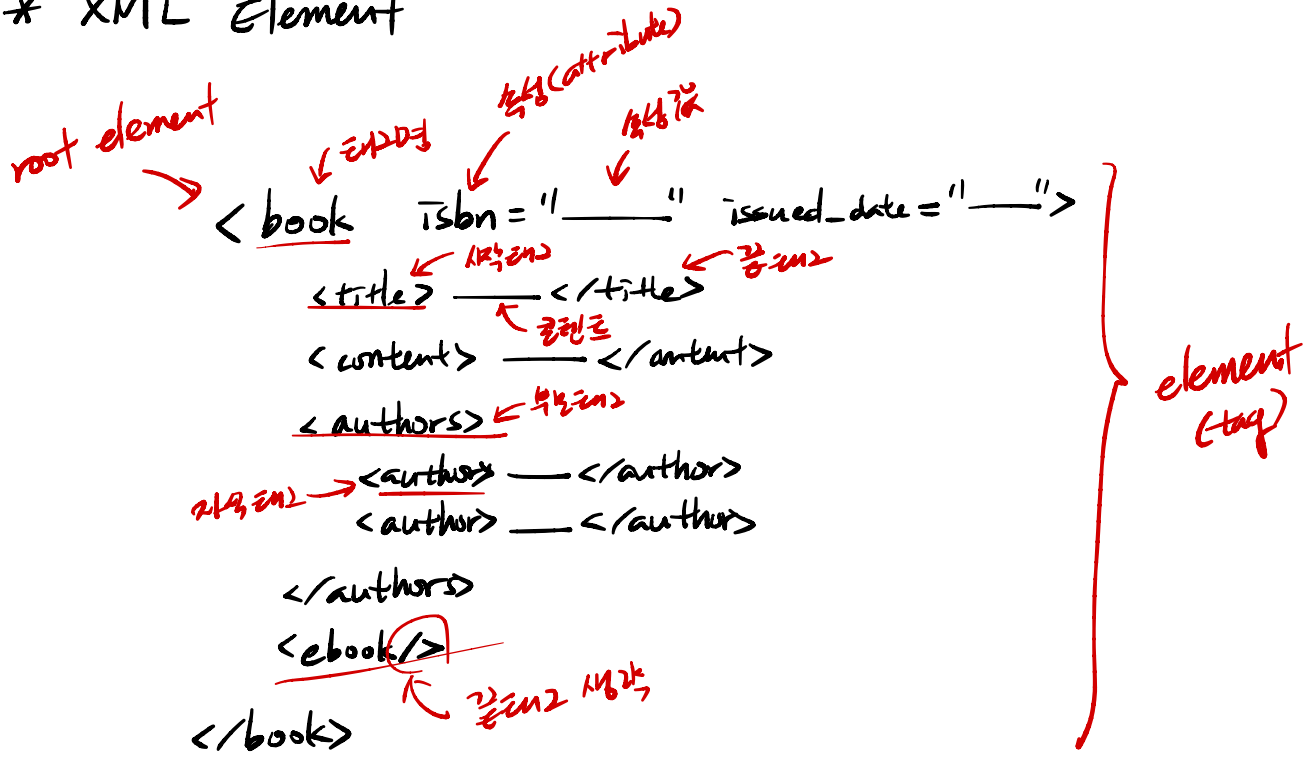
XML Schema

태그 내용 규칙

XML 태그의 스타일 지정 문법

DTD 보다 더 정교하게 제약을 할 수 있는 문법이다

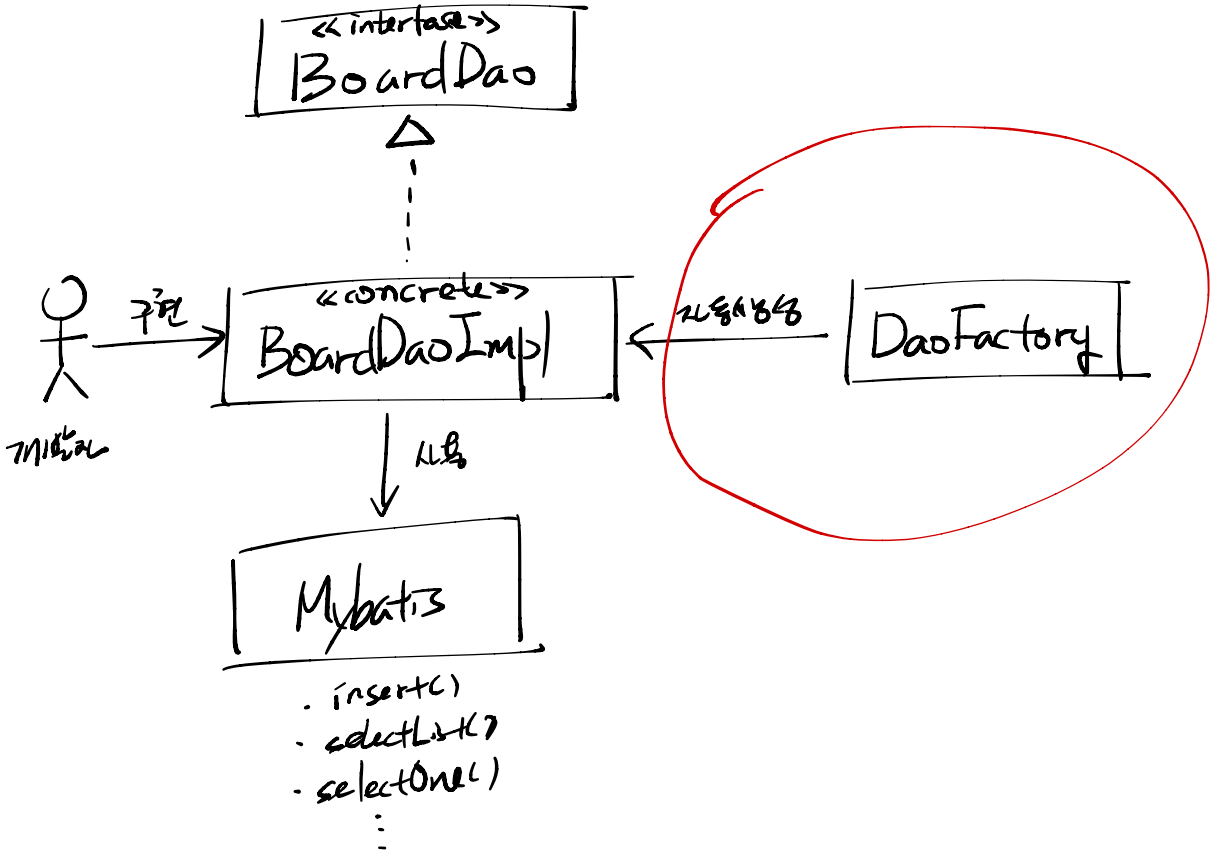
* XML Element


root element
태그명
속성(attribute)
속성값
<book isbn="____" issued_date="____">
시작태그
끝태그
<title> ____ </title>
콘텐트
<content> ____ </content>
<authors> 부모태그
자식태그
<author> ____ </author>
<author> ____ </author>
</authors>
<ebook/>
끝태그 시작
</book>
element (tag)

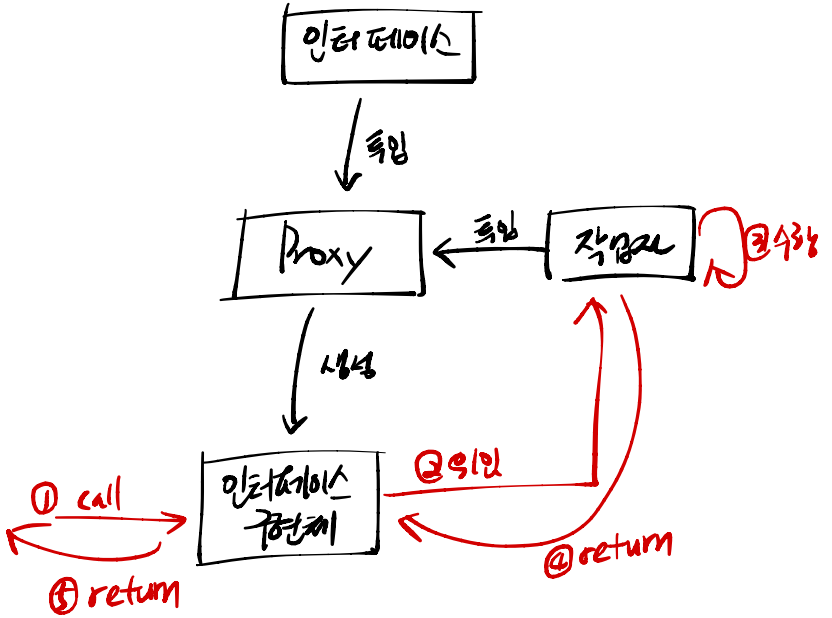
46. DAO 구현체 자동 생성

① 일반 방식

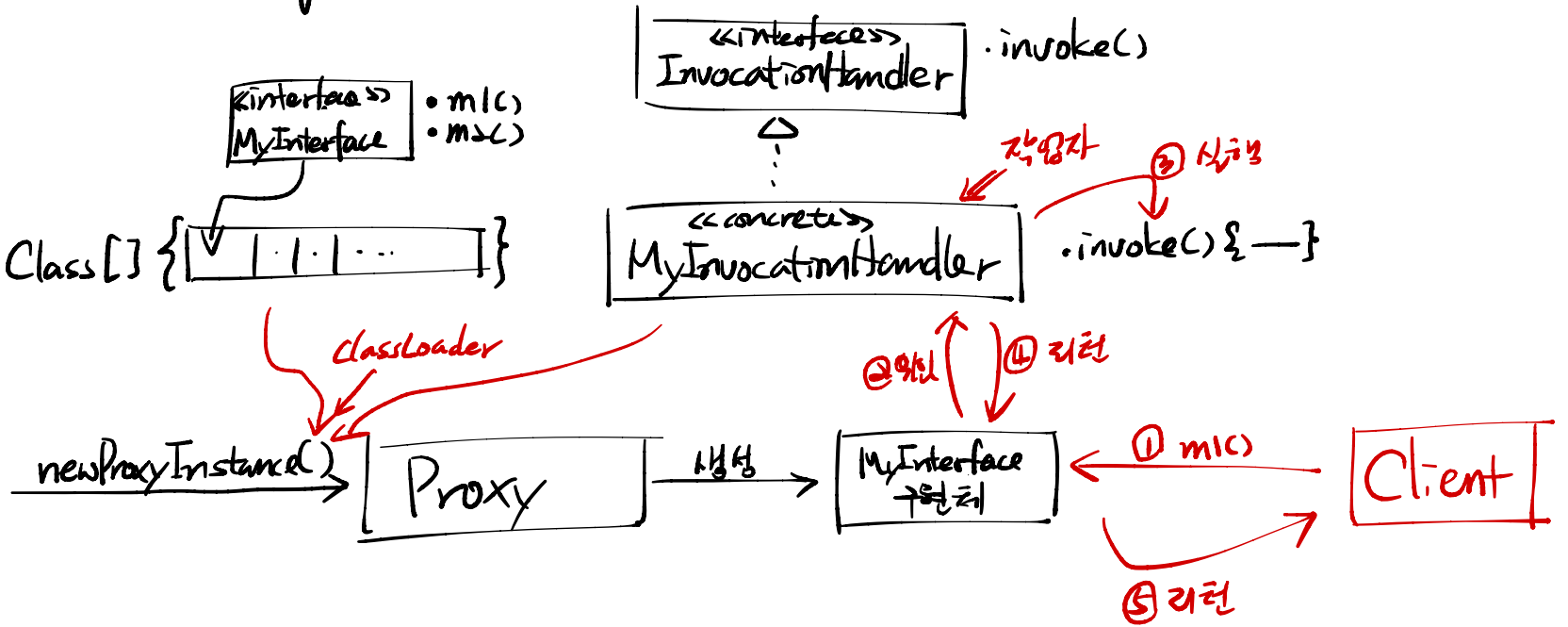
② 개선 방식



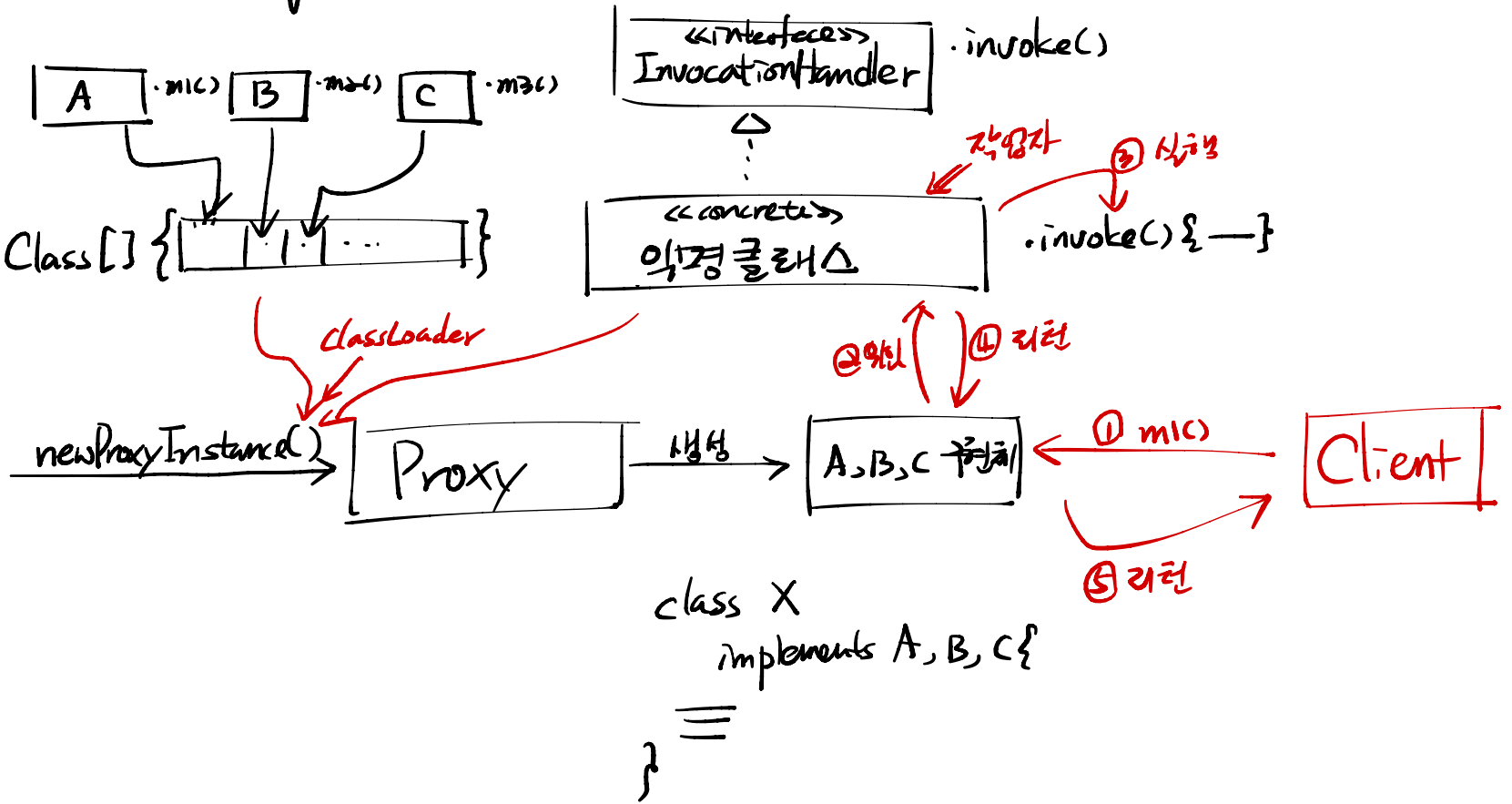
* DaoFactory 구현 원리



* DaoFactory 구현

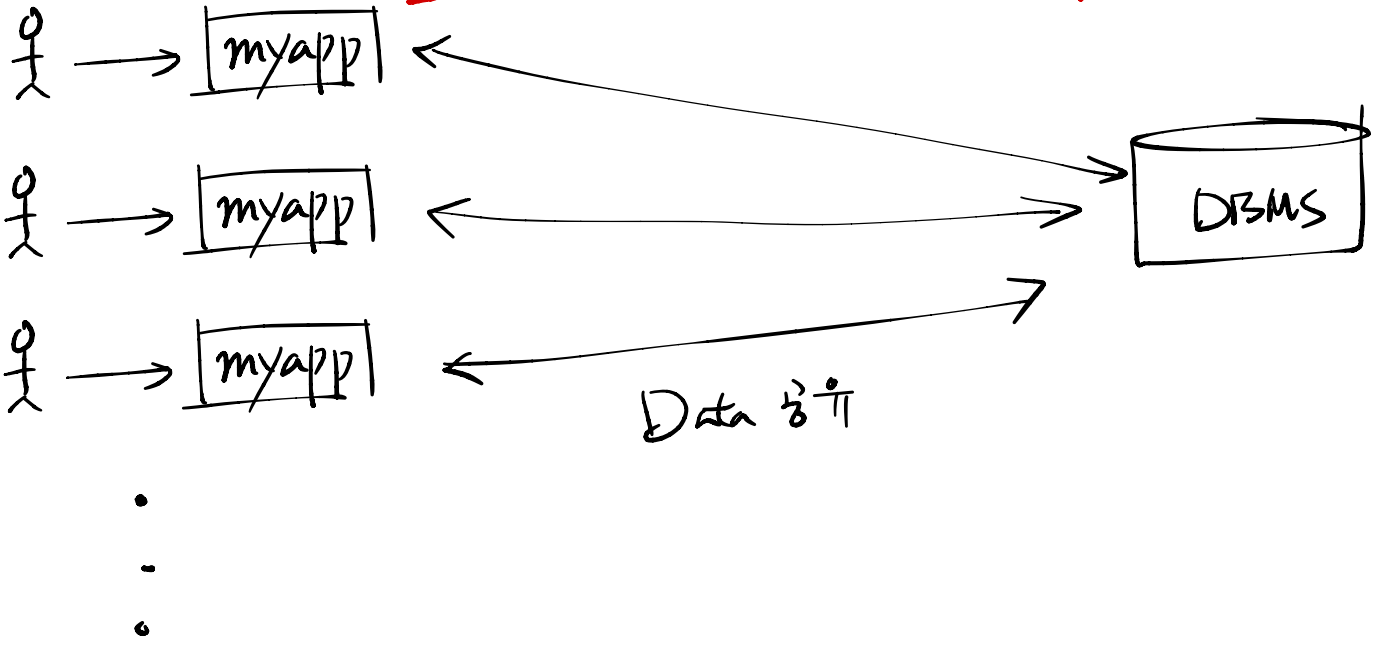


* DaoFactory 구현 II



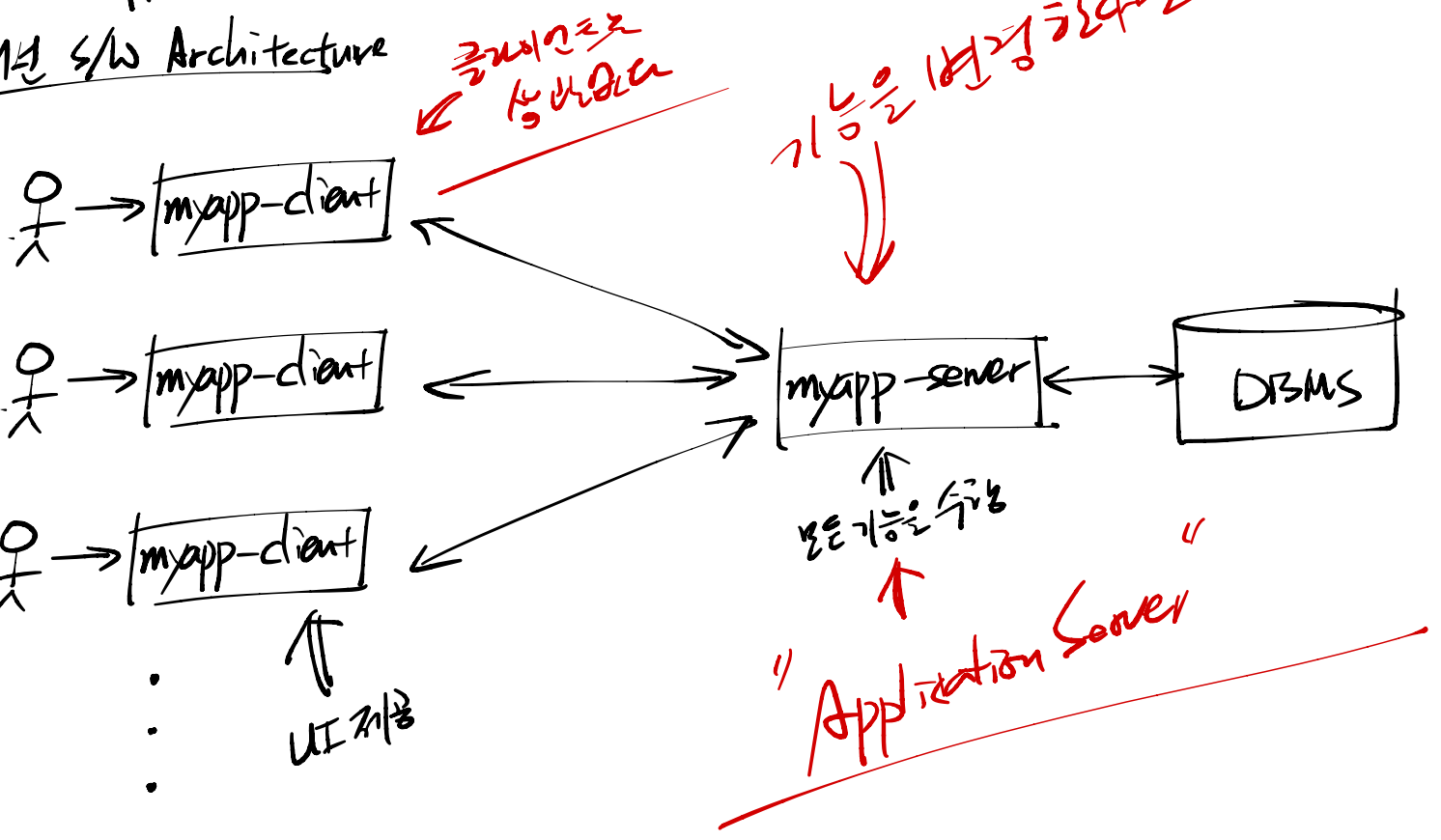
4장. Application Server Architecture

① 현재 S/W Architecture

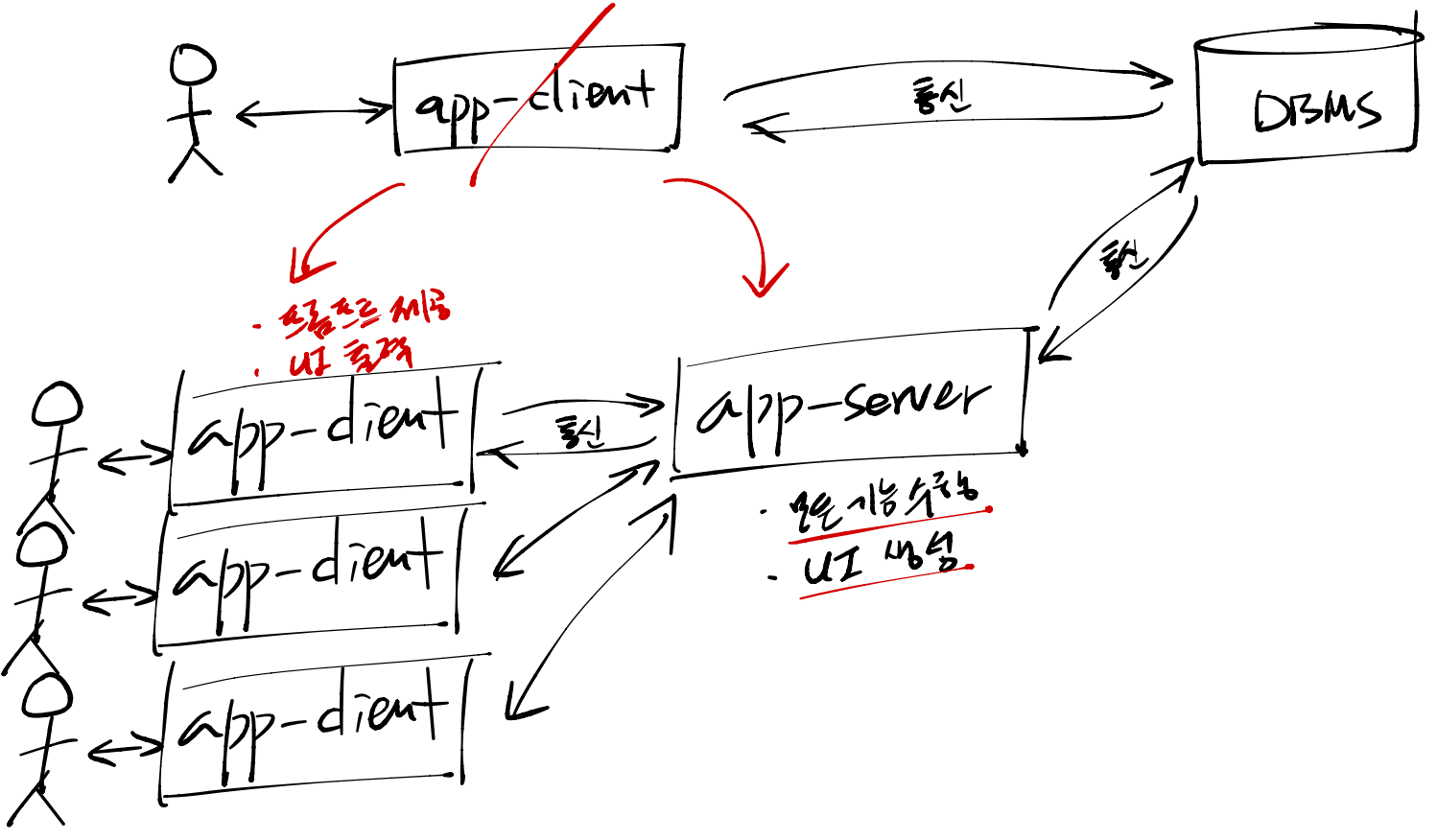


47. Application Server Architecture

전통적 S/W Architecture

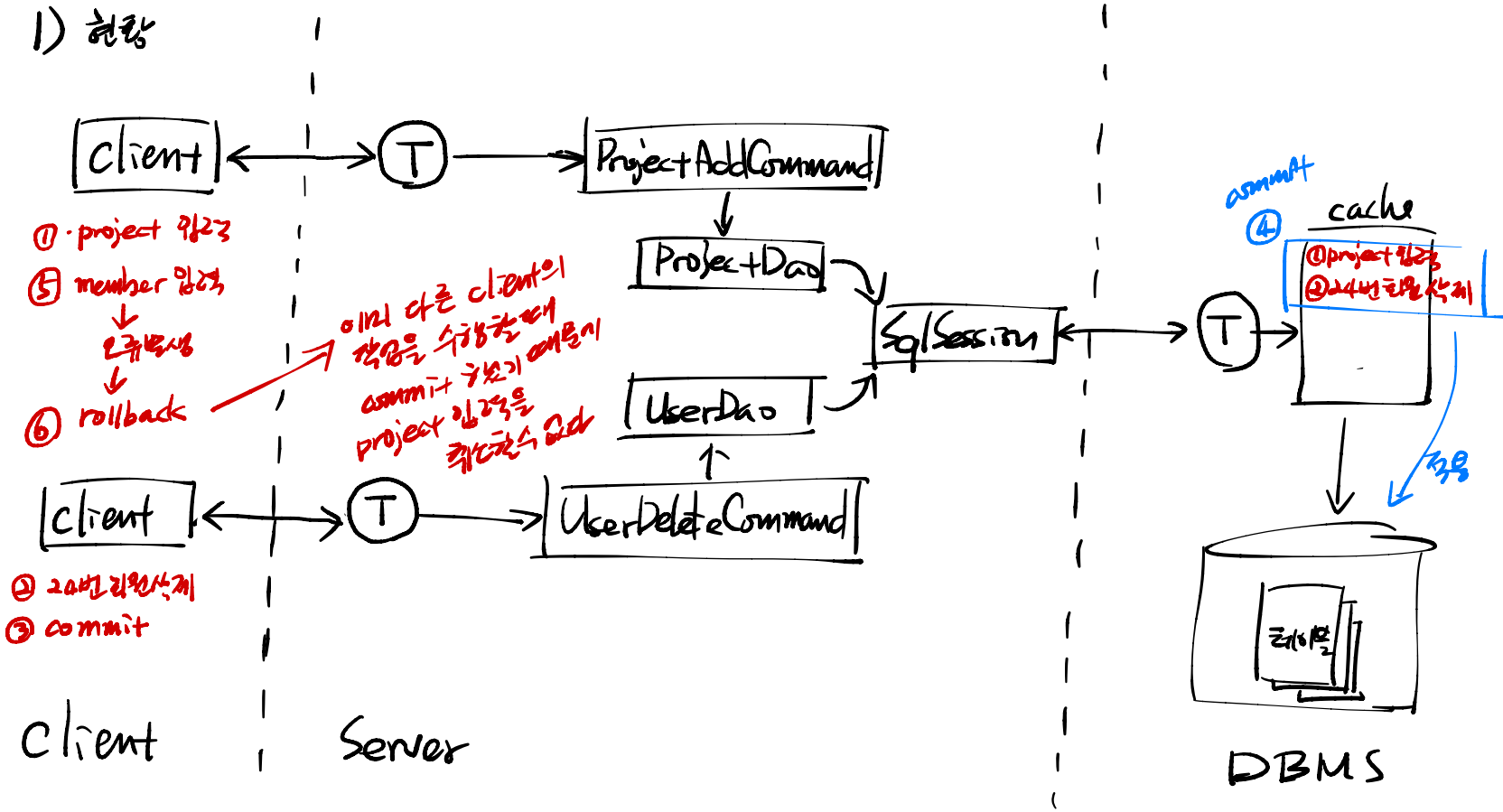


4장. Application Server Architecture

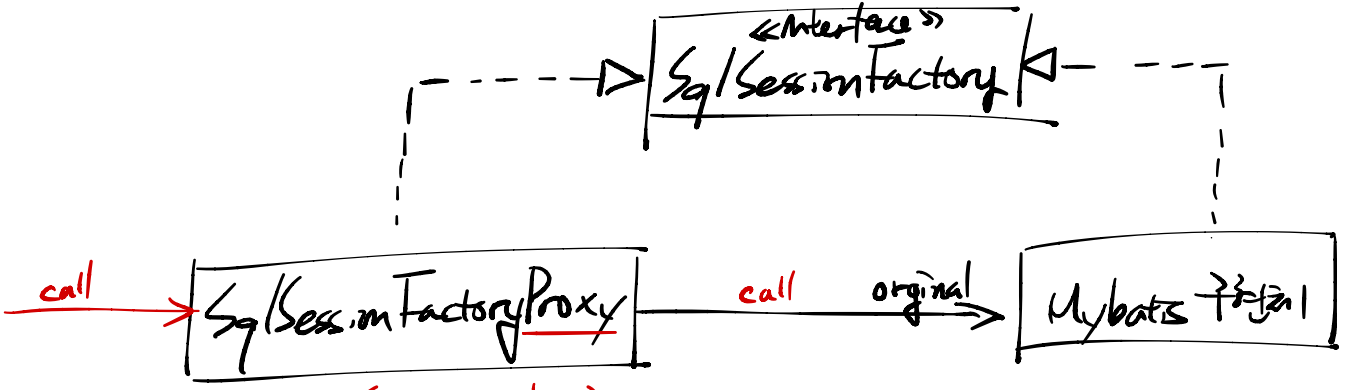


48. DB 커넥션 공유와 트랜잭션 처리

1) 현황

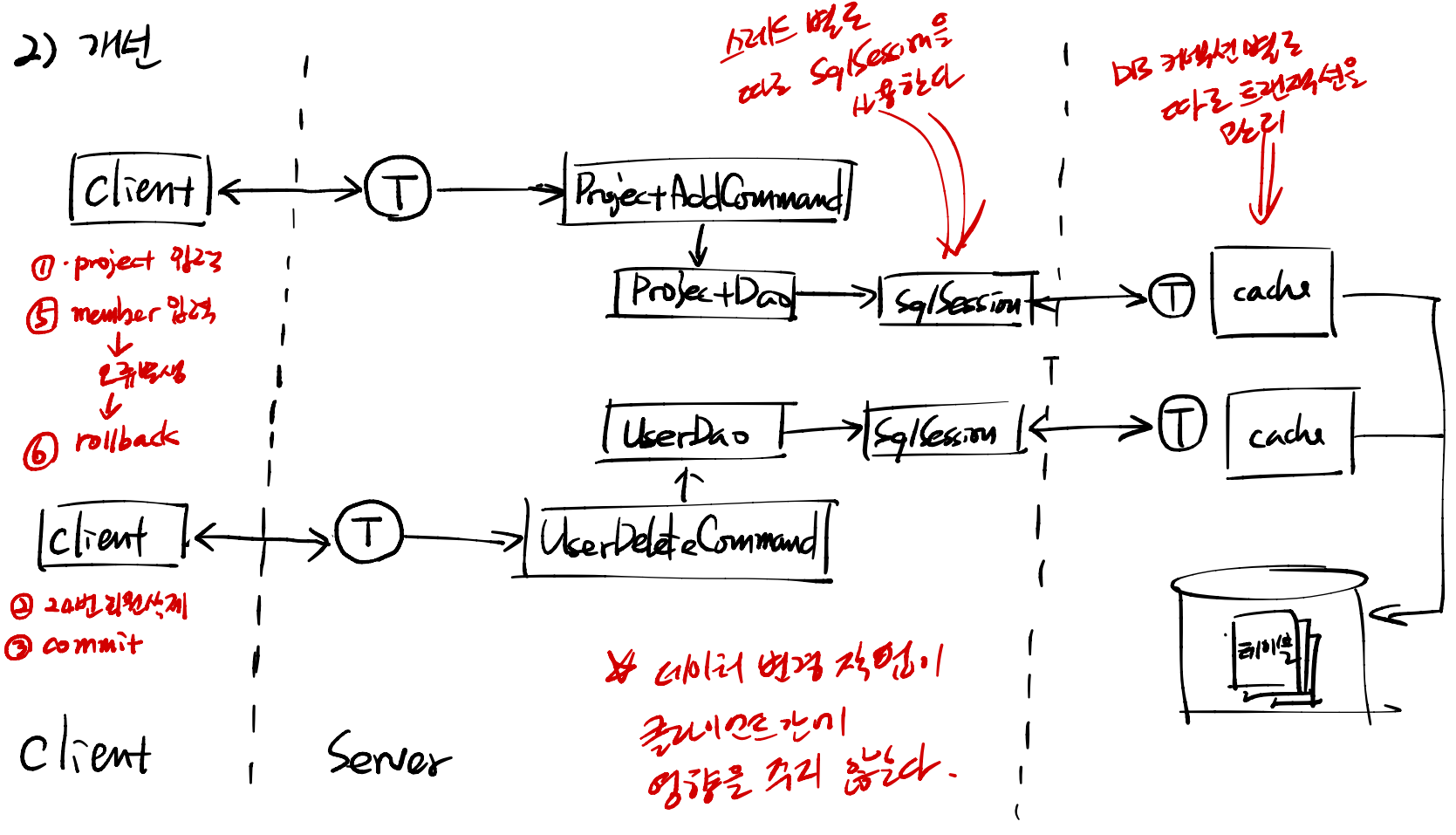


* 멀티스레드에서 스레드 전용 SqlSession을 공유할 때처



- openSession(boolean) 변경
  - ↓
  - 스레드 별로 하나 SqlSession을 사용
  - 스레드 별 관리 〃

2) 개선


트랜잭션 범위
따로 SqlSession을
사용한다
DB 커넥션 범위
따로 트랜잭션을
관리
client
T
ProjectAddCommand
① project 입력
⑤ member 입력
오류발생
⑥ rollback
ProjectDao
SqlSession
T
cache
UserDao
SqlSession
T
cache
client
T
UserDeleteCommand
② 고객센터삭제
③ commit
※ 데이터 변경 작업이
클라이언트간의
영향을 주지 않는다.
client
Server