

40. DBMS ဆိုတဲ့ (၇)

* ဤအားလုံး

JDBC API

Database

[datetime]

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

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

getTimestamp() \Rightarrow java.sql.Timestamp 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	startdate	enddate	members
101	p1	-	-	-	1,3
102	p2	-	-	-	1,2
103	p3	-	-	-	1,2,3

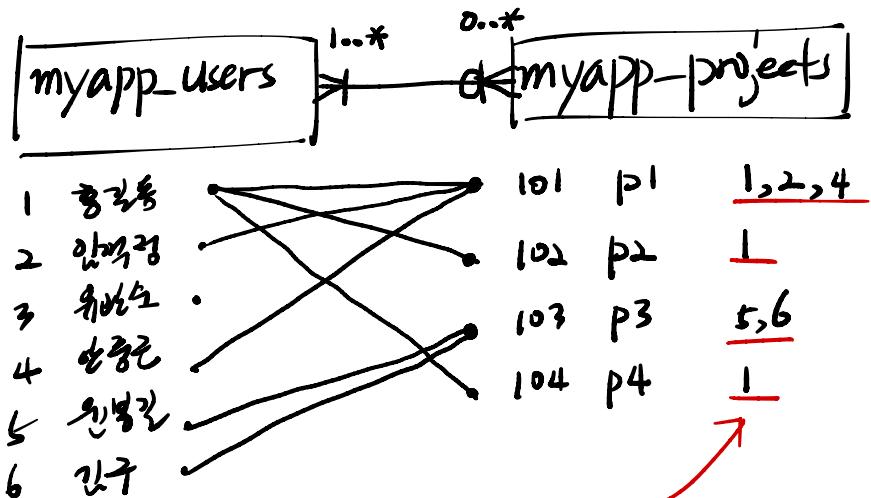
1번은 1번

프로젝트에 참여하는
사람은 반드시 있는
필수값입니다!

↑
필수값이 까지다!
↳ 필수값입니다!

41. 외부키(foreign key) 활용

1) 데이터의 데이터간에 관계를 짜는
(cf 대 cf 관계)



(cf 대 cf)
(cf 대 cf 관계)

외부 키와 함께 관계의 흐름을
제어하는 관계를 데이터에 막을 수 있다

관련된 관계를 데이터에 막을 수 있다

Foreign key 활용
+
관련된 관계
관계로 데이터를 제어할 수 있다

제어?

* 데이터를 관리하는 틀

- 1 흐름
- 2 일정
- 3 주제

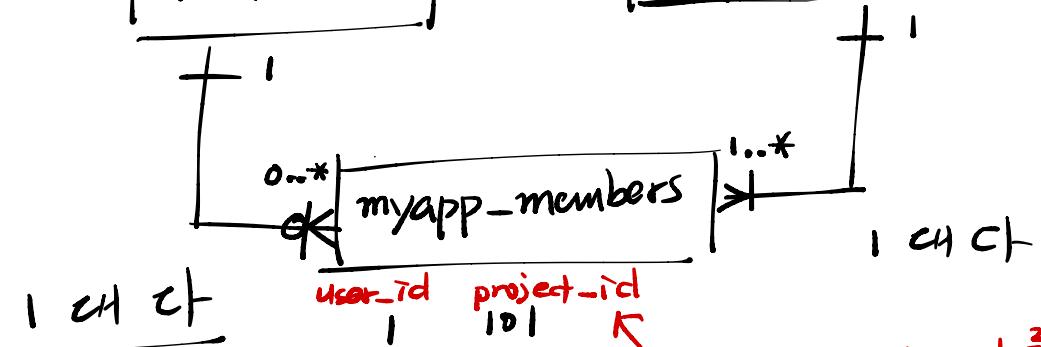
myapp-users	
1	2

- 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



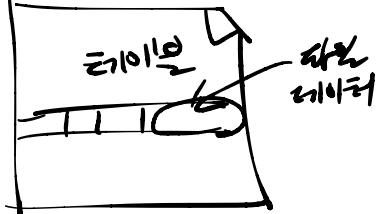
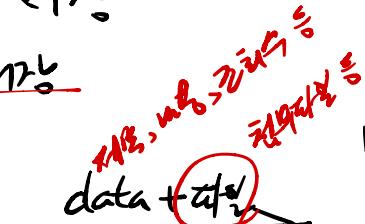
1 대 1 대

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

다른 테이블의 pk를 → 가리킨다
"Foreign key"

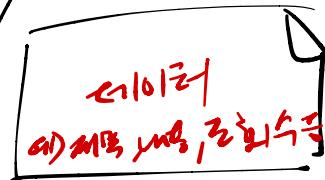
* DB의 특성 짚기

① 데이터가 구조화된



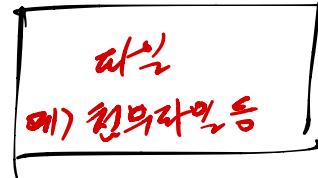
② 자료를 분리해서 저장

데이터베이스의
구조화된



구조화된
구조화된

구조화된



데이터베이스가
구조화된 자료로
구조화된 자료로



자료는 데이터
구조화된 자료



자료는 알고 쓰는
모의 구조화



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;

(가상 열을 표시) ③
select rno, name, concat(loc, '-', name) as name2
from room

✓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)

같은값을 토대로끼고 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!

↑ 같은 이름을 가진 컬럼이 있다.

```
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



board 테이블에
bno 컬럼이
없다!
or!

✓ 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 헷
(inner join)

```
select * from board5;
```

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

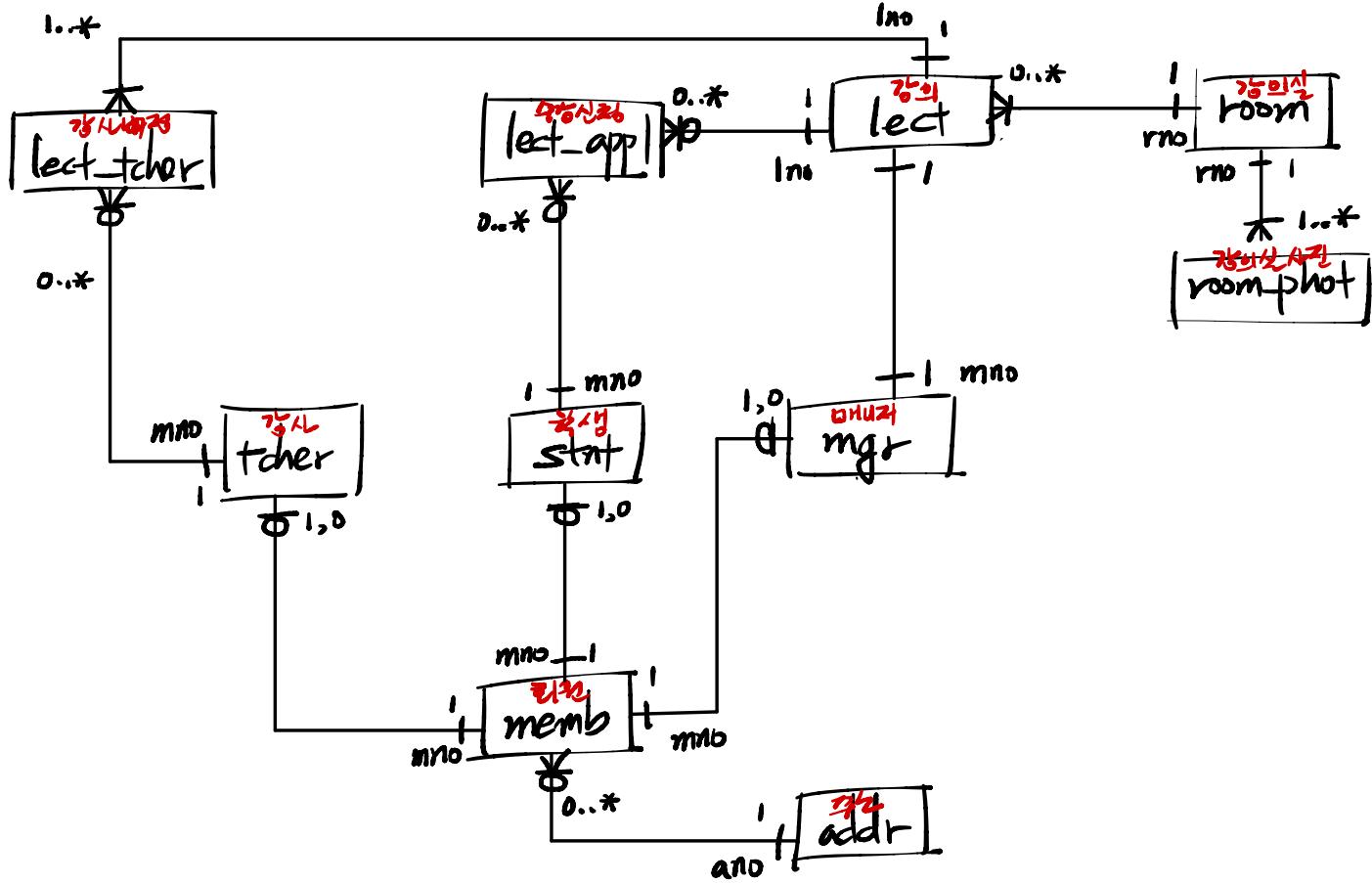
```
select * from attach_file5;
```

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

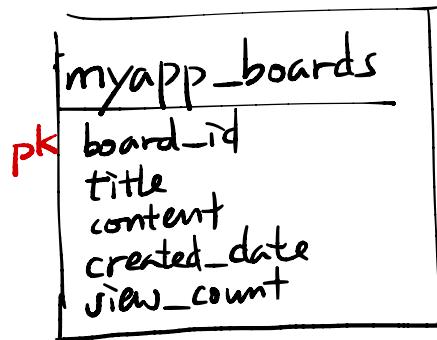
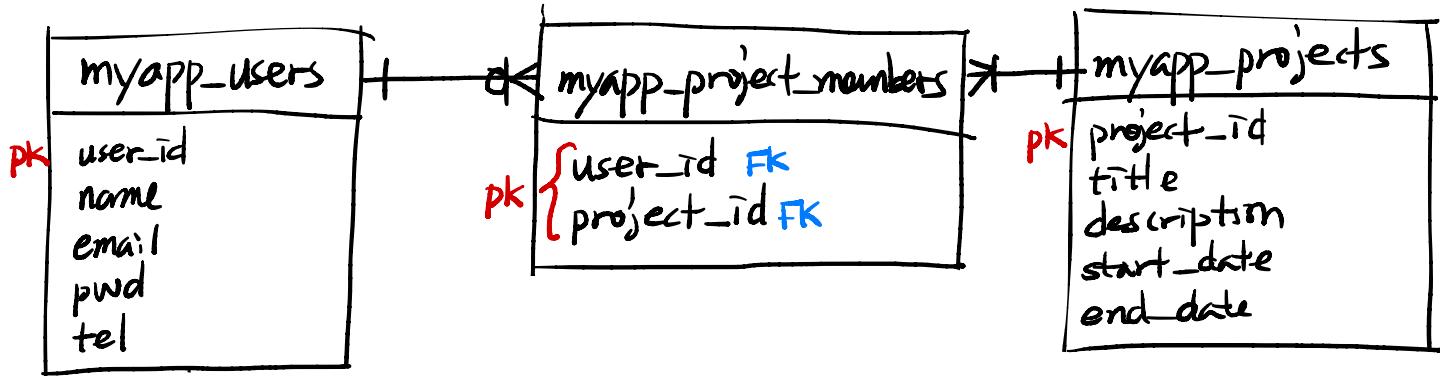
```
select no, title, content, fno, filepath, bno  
from board5 b join attach_file5 a on b.no=a.bno;
```

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

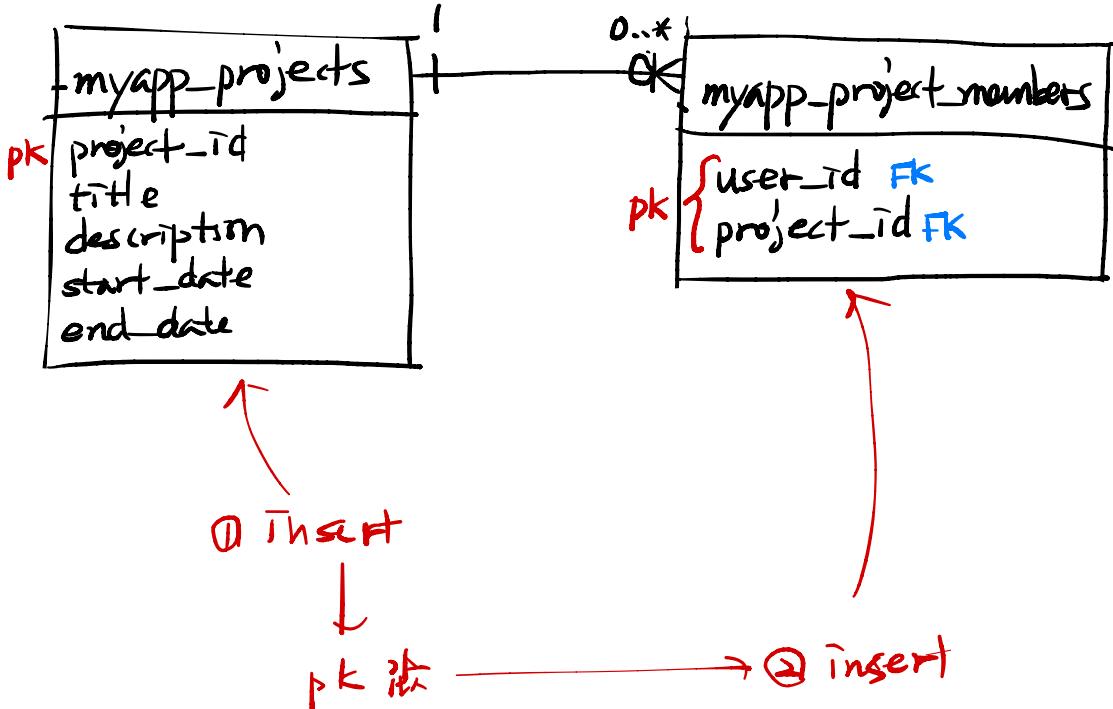
* JOIN 예제 - ERD (Entity Relationship Diagram)



* 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

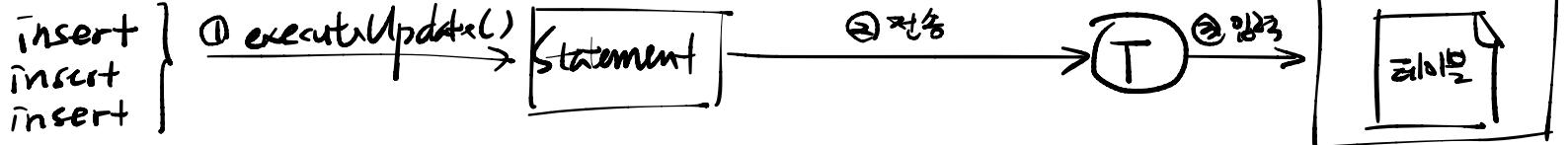
keyRS.getInt(1); 202 ← pk 값

keyRS.next(); → 203 ← pk 값

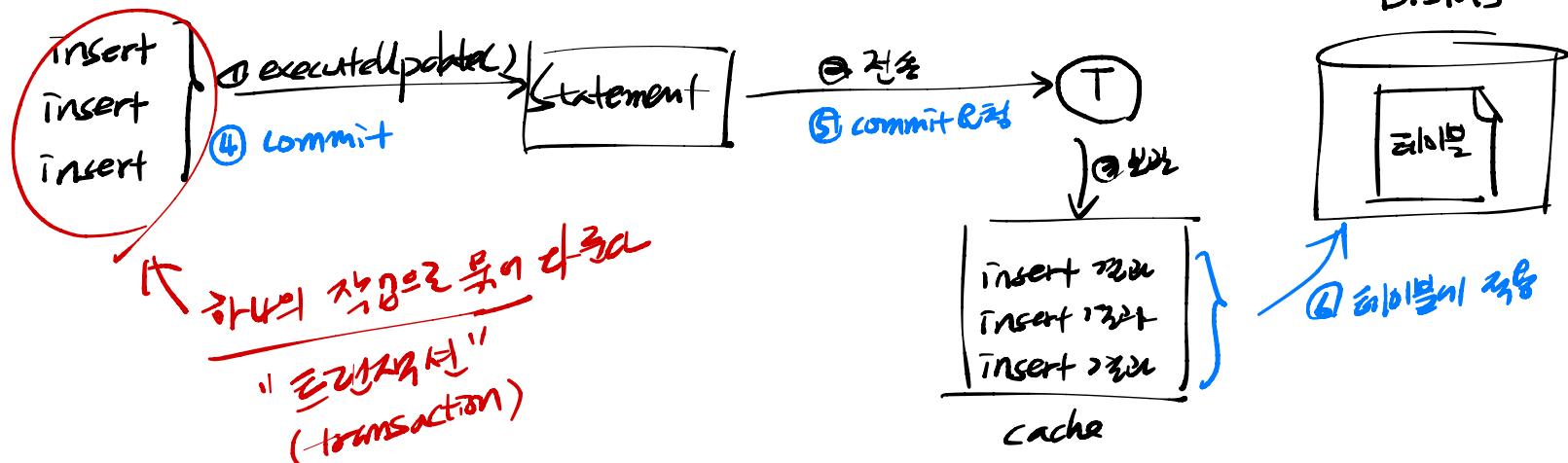
keyRS.getInt(1); 203 ← pk 값

* auto commit vs 수동 commit

1) Auto Commit = 즉시 데이터베이스 반영

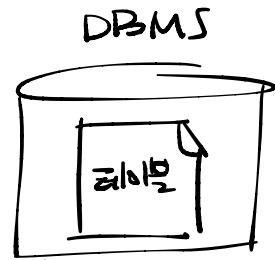
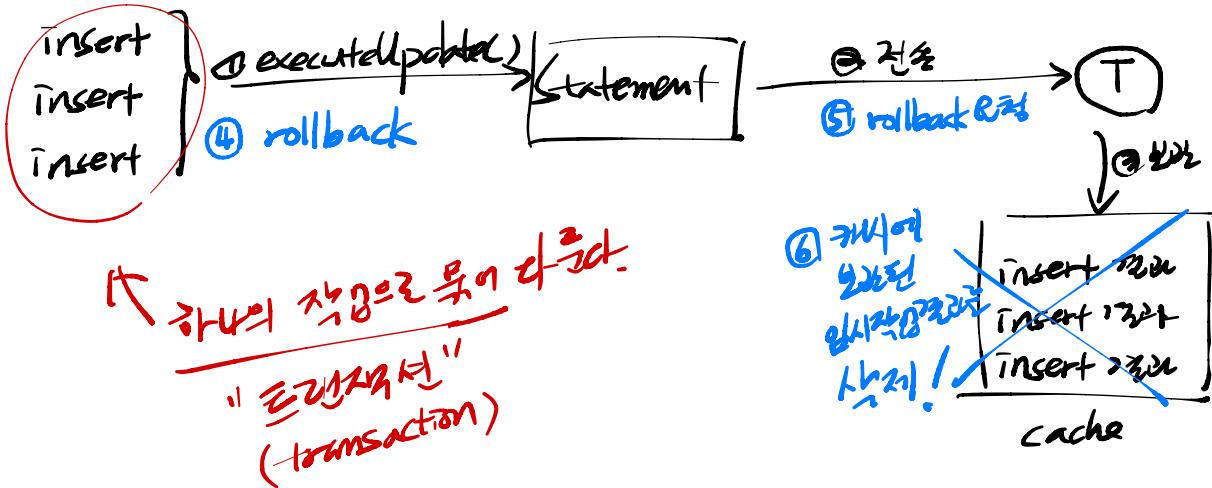


2) 수동 commit = commit

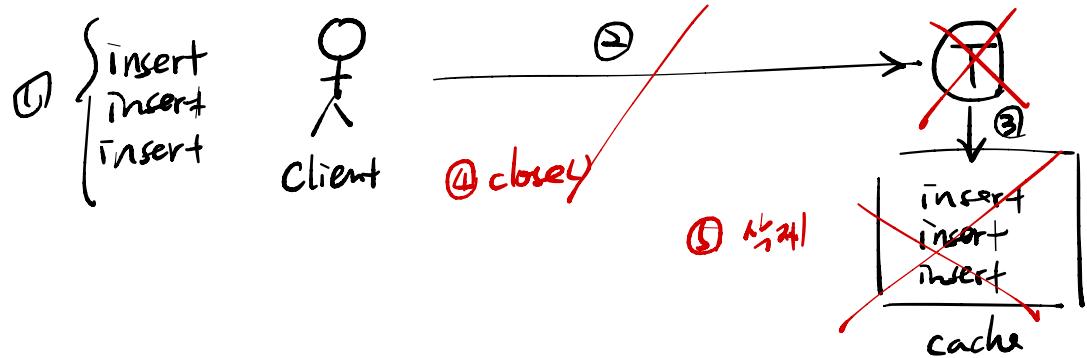


* auto commit vs 수동 commit

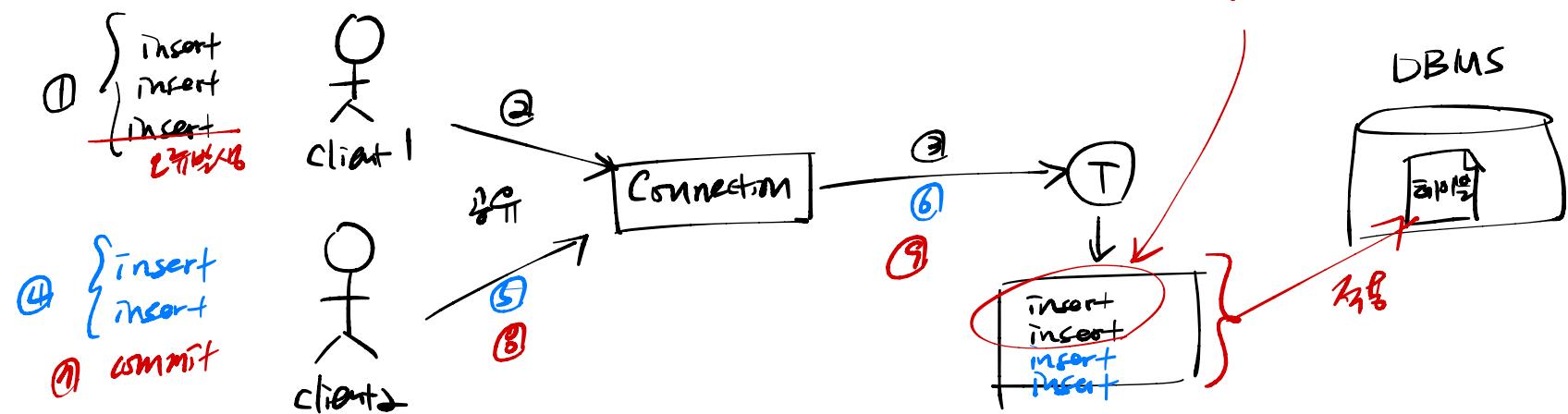
2) 수동 commit = rollback



* 쓰기지연된 DB 처리법 장단점

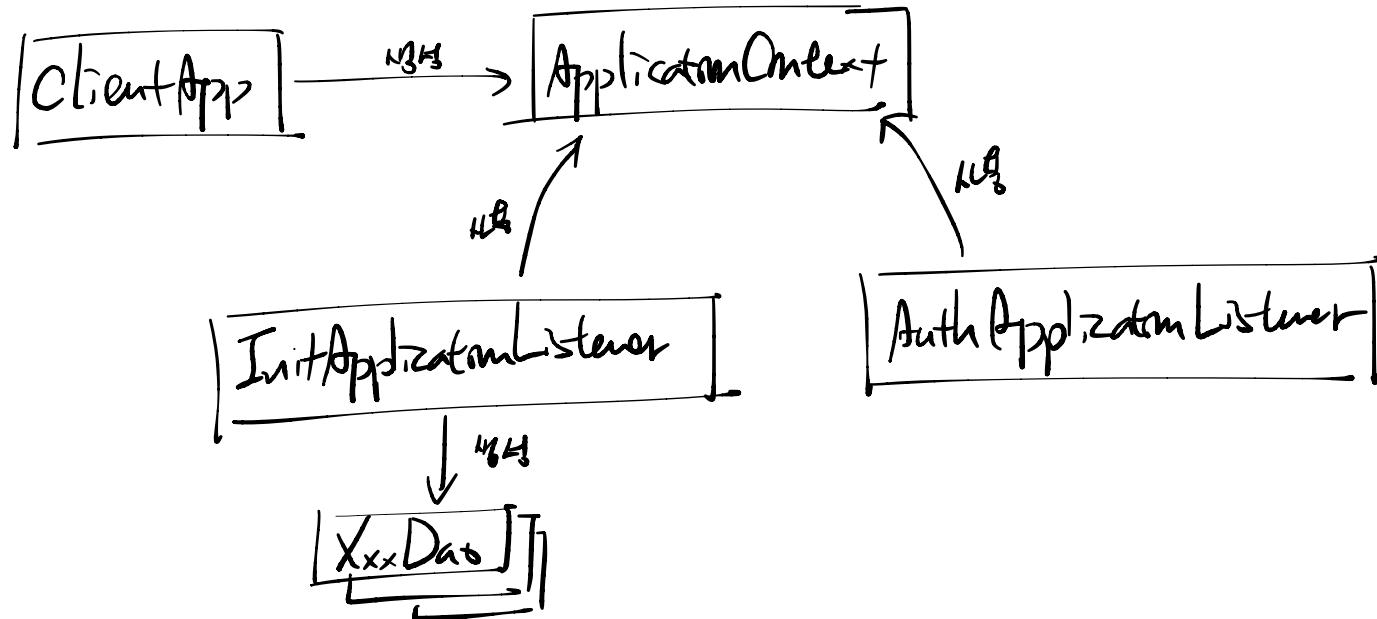


쓰기지연된 DB 처리법 장단점
기억나는 예제를 봐보자
쓰기지연된 DB 처리법은 rollback이 어렵다!
Delete는 rollback이 어렵다!
client의 rollback은 가능하다!
rollback은 가능하다!

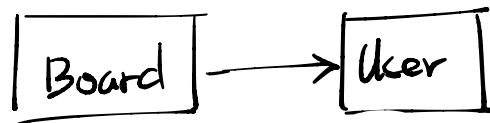


42. 201 / 206 22.8.861

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!

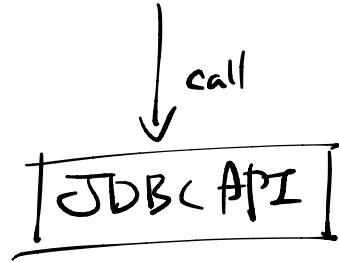
In-parameter
1. 제목
2. 내용

In-parameter
값
값

In-parameter
값
값

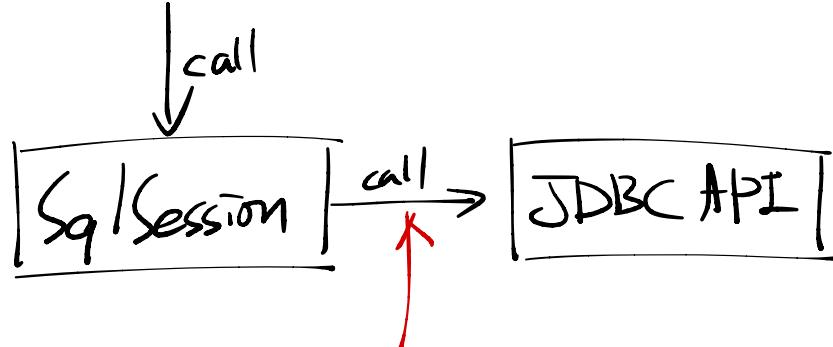
44. JDBC 관계 캡슐화

① 이전 방식



② 개선 방식

비단계적인 관계
될 수 있다.

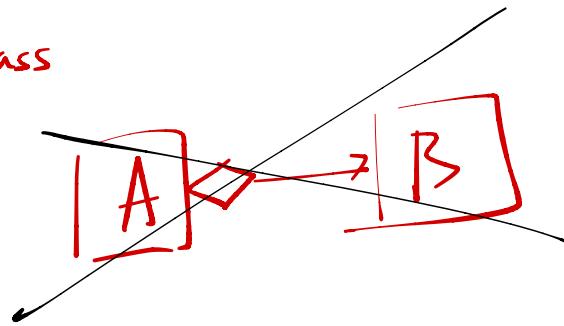
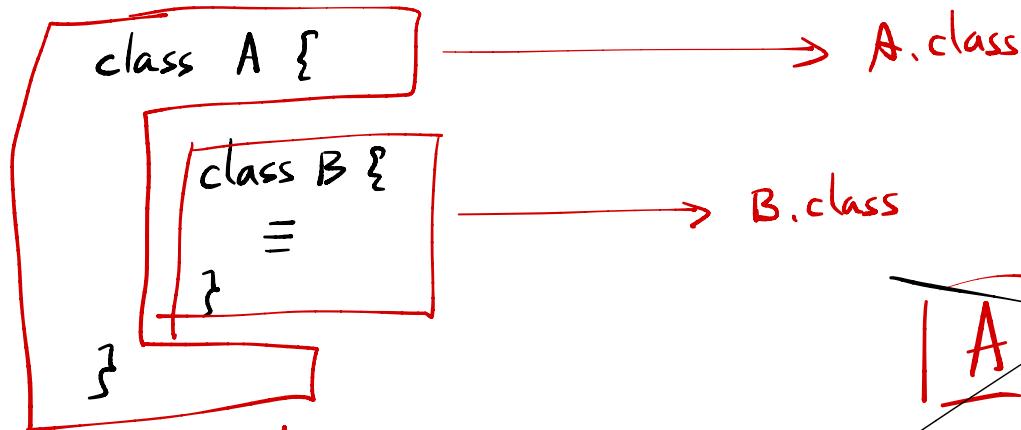


Reflection API 활용

- 초기화 대체 가능
- 동적 SQL 가능

* Reflection API

증집 클래스



전체 클래스는 증집

B를 만들기
A를 만들기 위한 대안

이면 가능

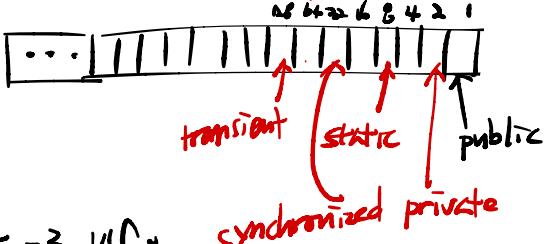
= A는 B를 만들기 위한 OBJECT!

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 : 00001000

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

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

* 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()~~ { - }

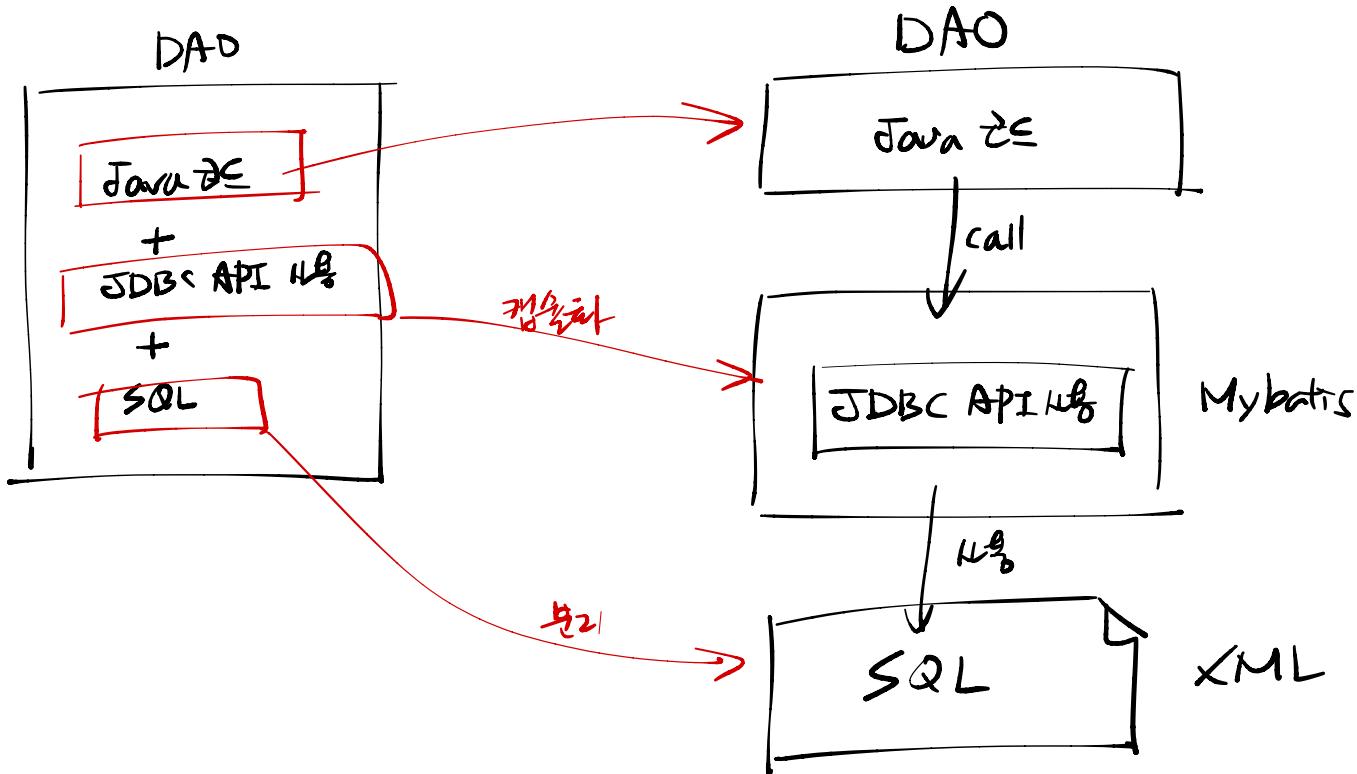
"createDate" ← property name

* Field E- Property

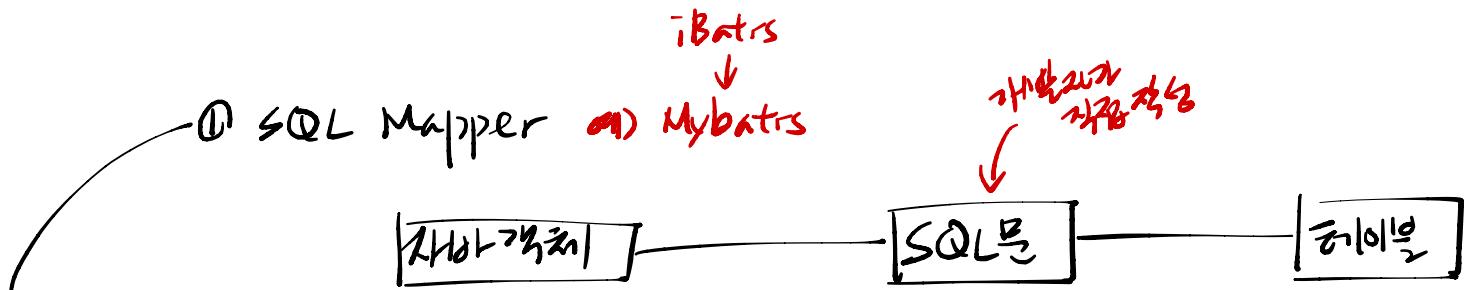
```
class Board {  
    int no;           ← setNo()  
    String title;    ← setTitle()  
    String content;  ← setContent()  
    Date createdDate; ← setCreatedDate()  
    int viewCount;   ← setViewCount()  
    User writer;     ← getWriter().setNo()  
                      ← getWriter().setName()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

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 (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

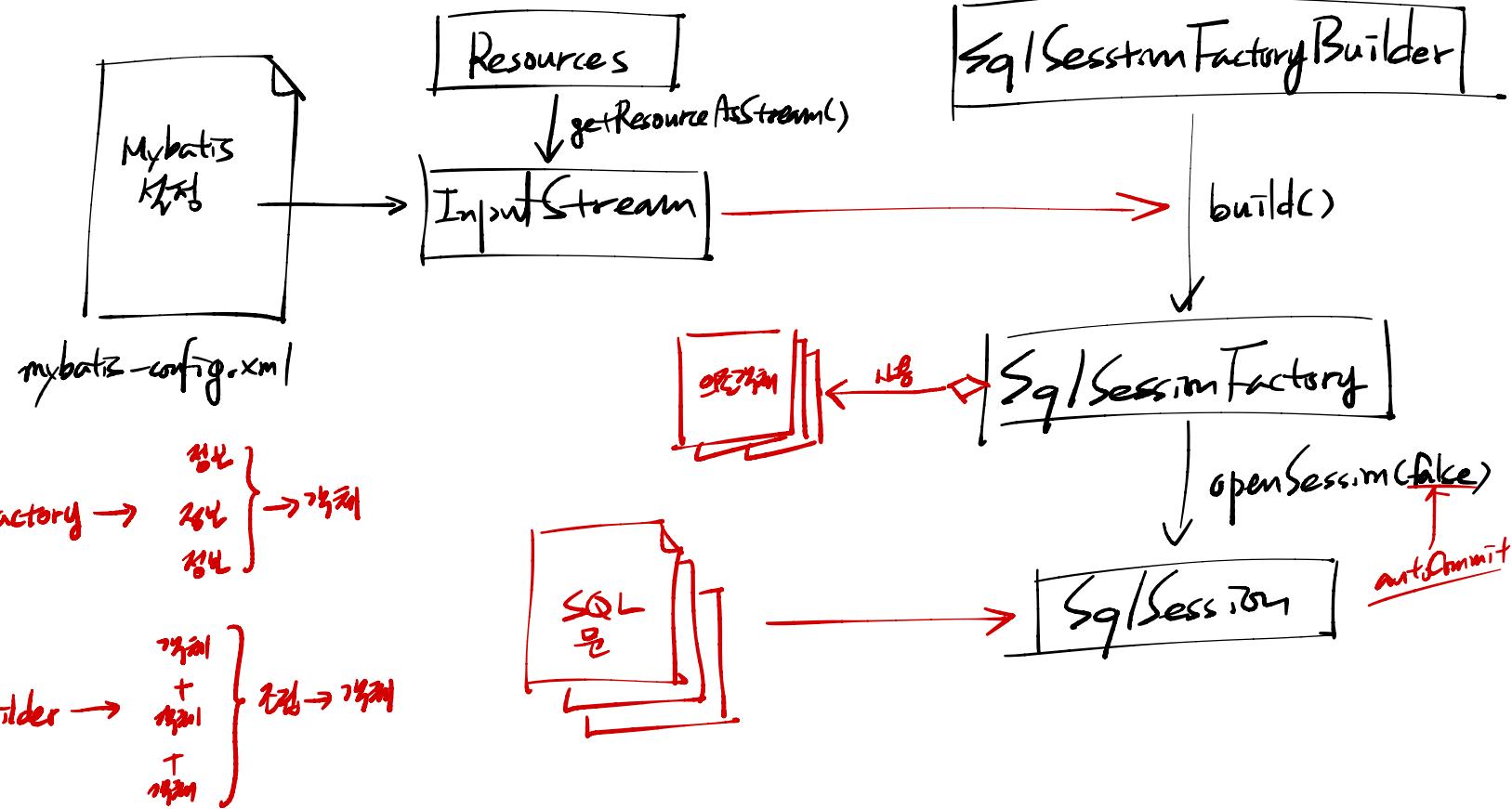
기준을 정의한 설정파일의 URL

<configuration> ← root element = XML 문서의 루트 요소

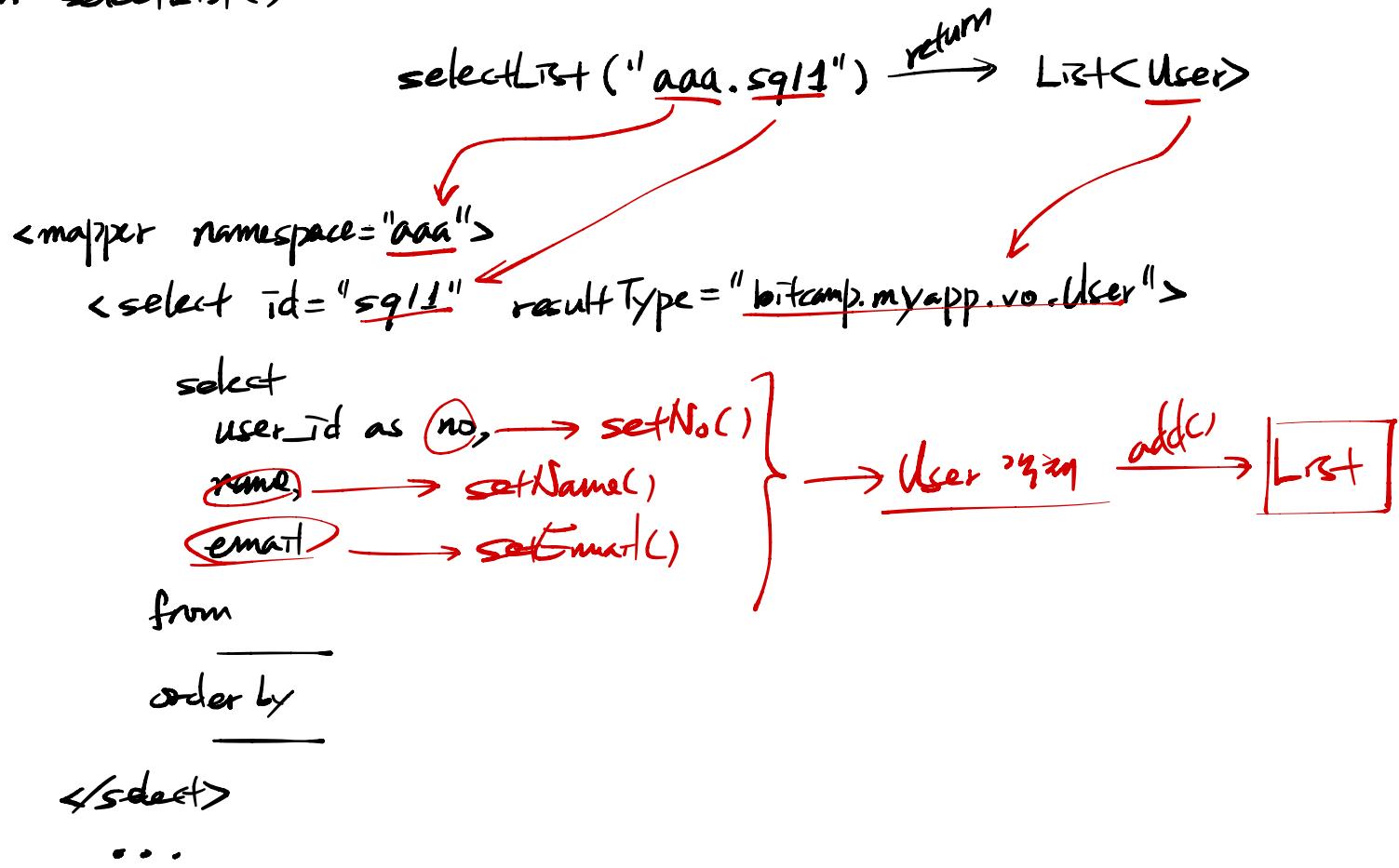
|

</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.v0.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 결과를 위한 모델

<resultMap>

select

b.board_id,

b.title,

b.content,

b.created_date,

b.view_count,

u.user_id,

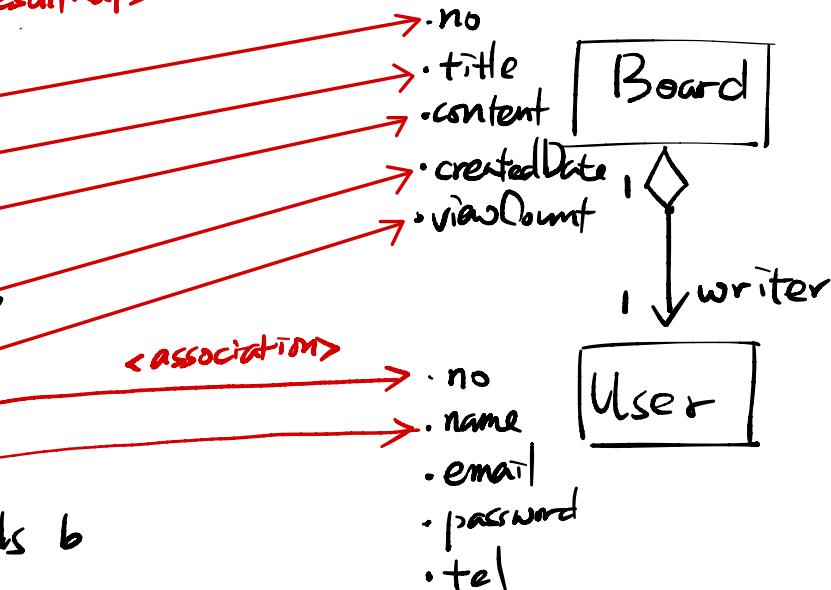
u.name

from myapp_boards b

inner join myapp_users u

on b.user_id = u.user_id

where



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

<resultMap id="BoardMap" type="bitcamp.mybatis.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.mybatis.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

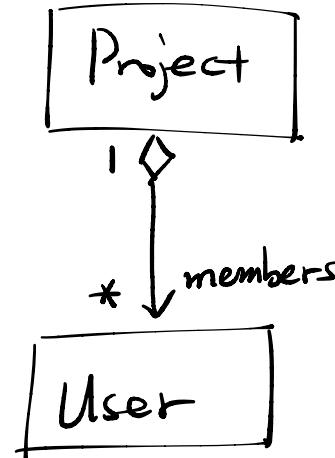
→ User 테이블

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}

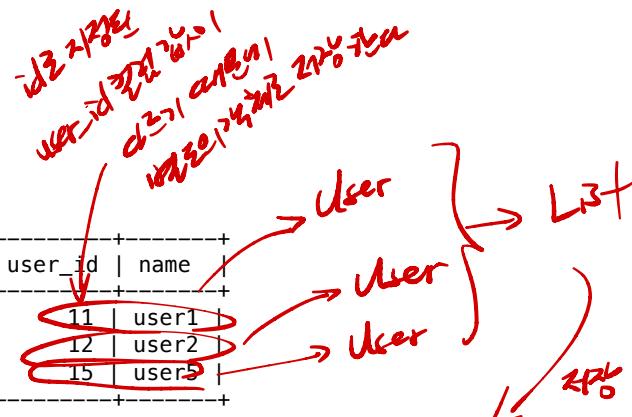


project_id	title	description	start_date	end_date	user_id	name
101	프로젝트1	설명	2024-01-01	2024-02-15	11	user1
101	프로젝트1	설명	2024-01-01	2024-02-15	12	user2
101	프로젝트1	설명	2024-01-01	2024-02-15	15	user5

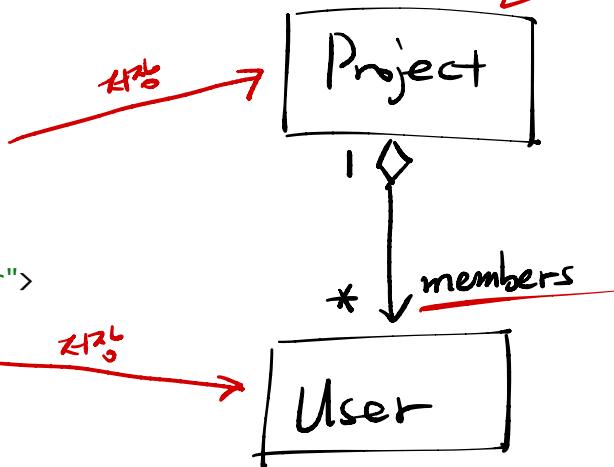
* 결과값을 1개로 하는 단 사용할 규칙 2개의

id 체크하는
중복되는
한개는
가지면 2개로 한다.

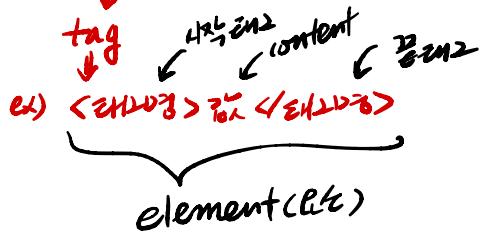
project_id	title	description	start_date	end_date	user_id	name
101	프로젝트1	설명	2024-01-01	2024-02-15	11	user1
101	프로젝트1	설명	2024-01-01	2024-02-15	12	user2
101	프로젝트1	설명	2024-01-01	2024-02-15	15	user3



```
<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



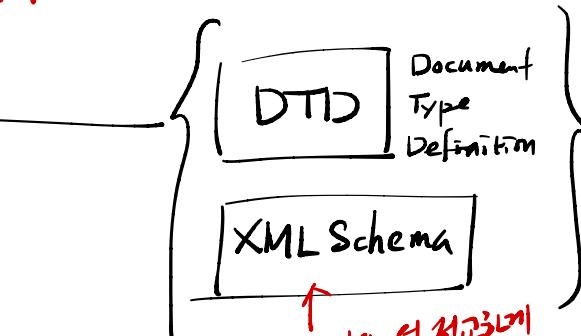
XML은 가로, 세로 모두 가능



XML의 규칙 → 자각-법, 태그는 괄호로 묶는다
→ well-formed XML



XML 문서의
설정 및 표기



DTD와의 차이점
구조화된 문서 형식

DTD(도큐먼트 타입 정의)

* XML Element

* XML Element

```

<book>
  <title> ... </title>
  <content> ... </content>
  <authors>
    <author> ... </author>
    <author> ... </author>
  </authors>
</book>
  
```

root element

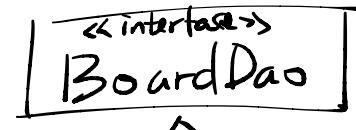
element (tag)

isbn (attribute)
issued_date (attribute)

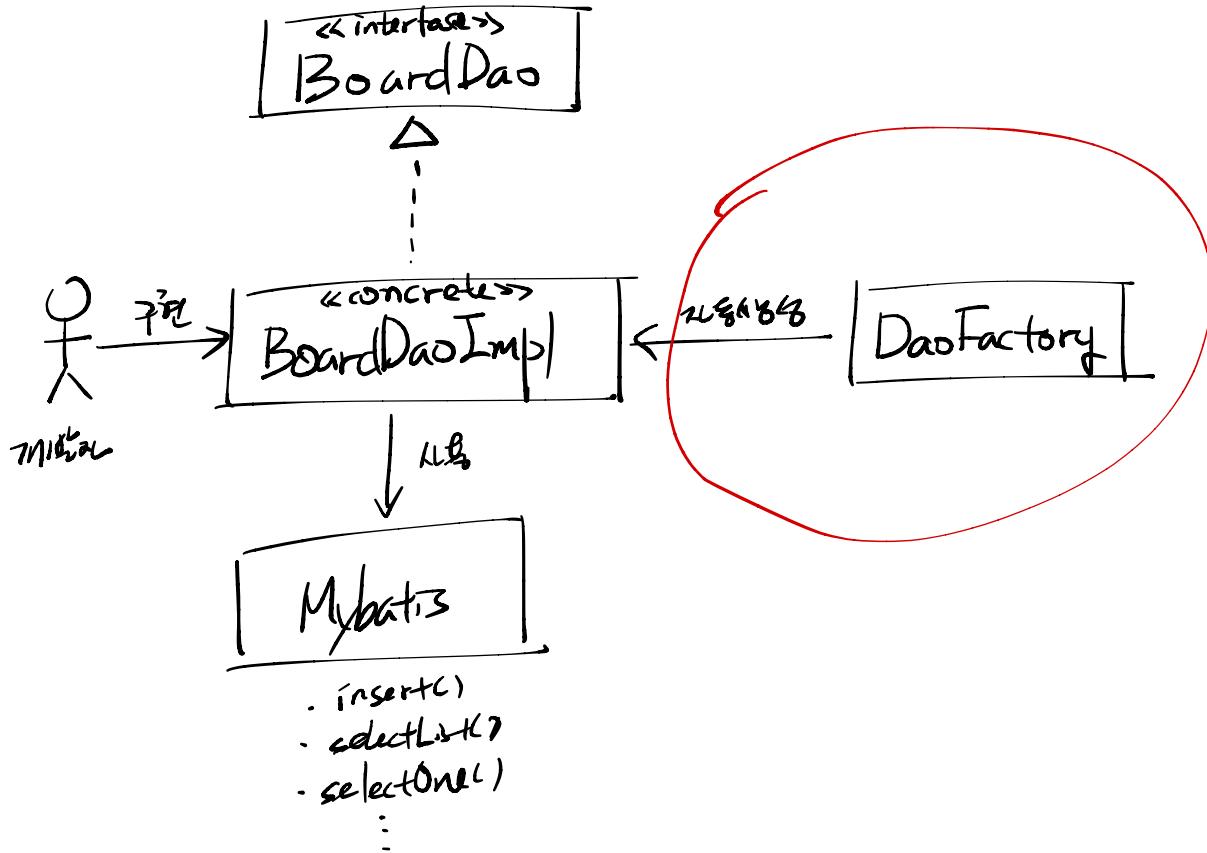
title
content
authors
author

46. DAO 주제의 알고 멤버

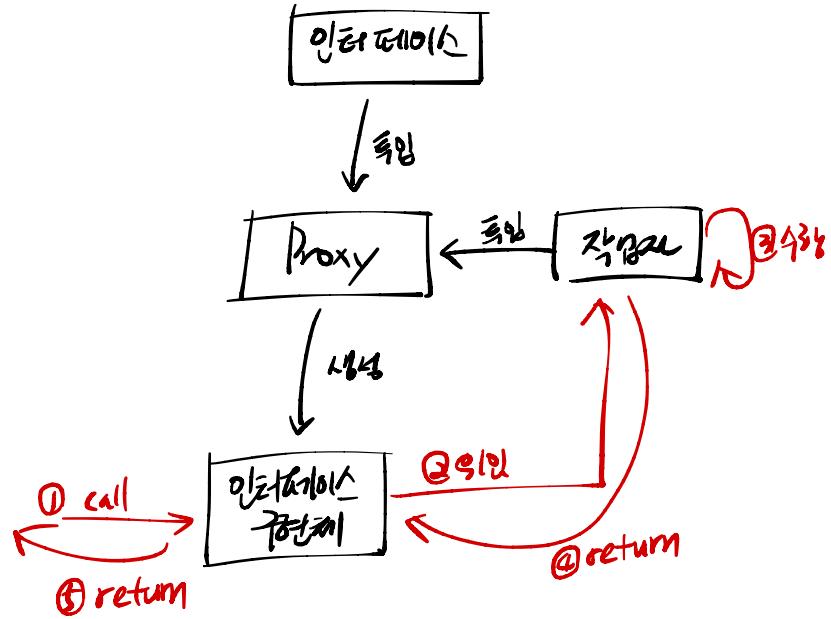
① 디자인 패턴



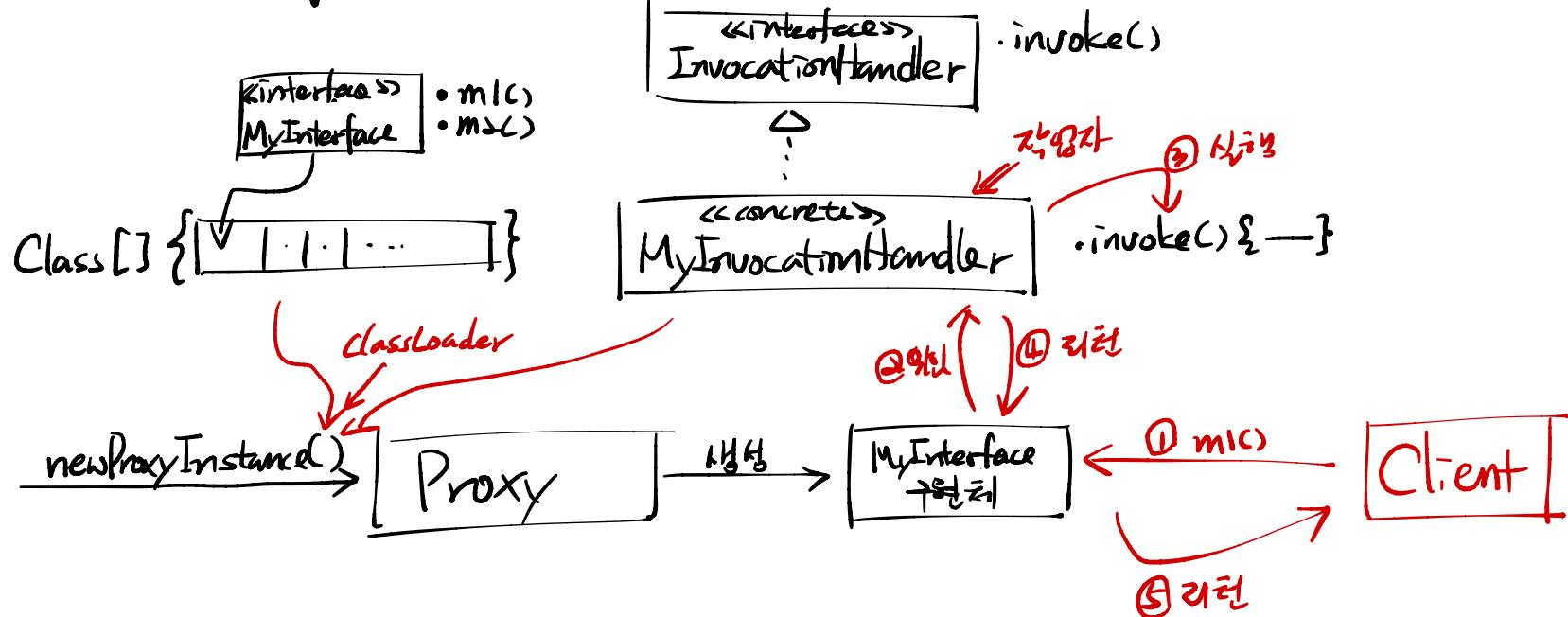
② 디자인 패턴



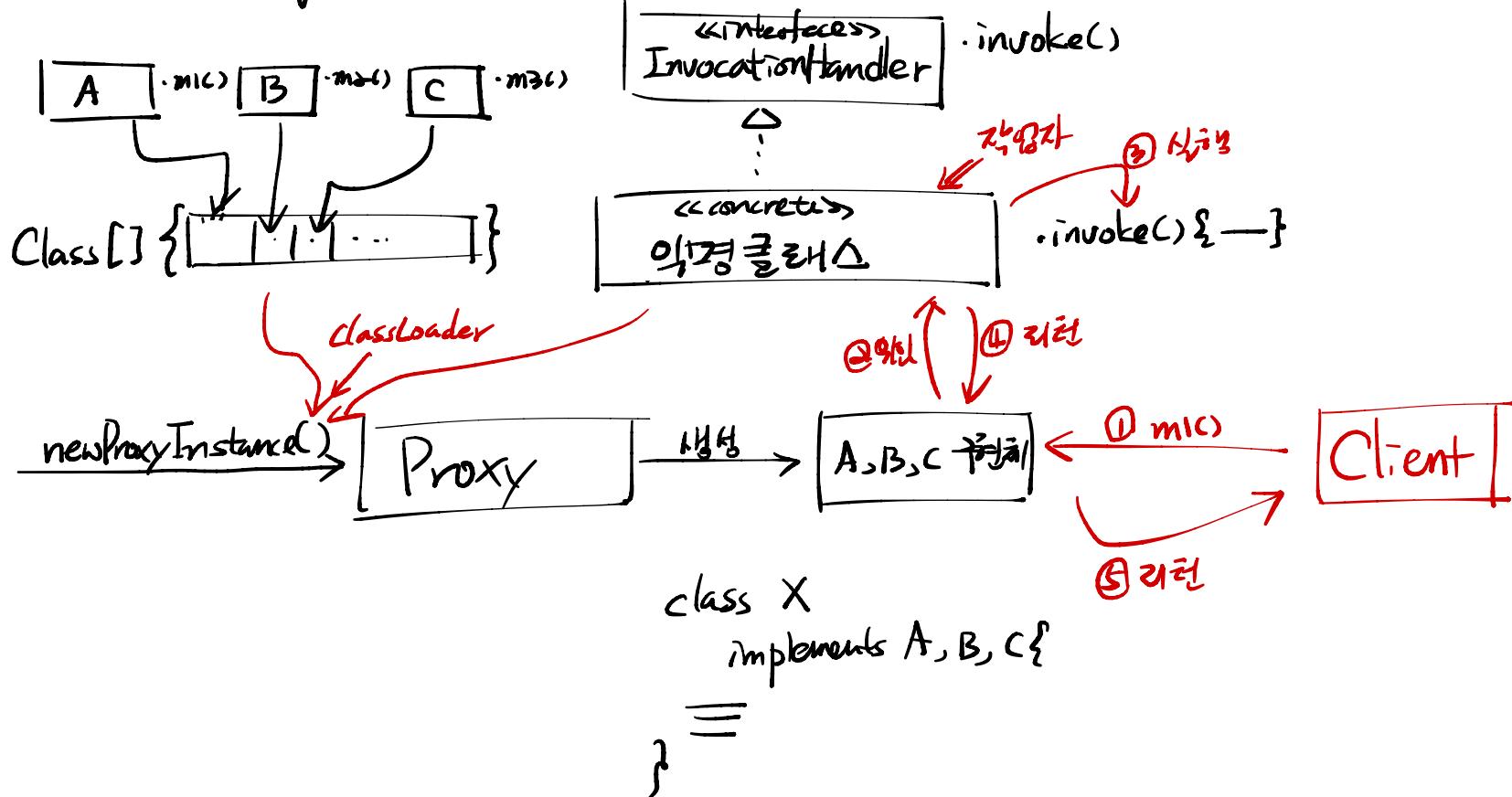
* DaoFactory 구현 원리



* DaoFactory ↗



* DaoFactory 퀵 II



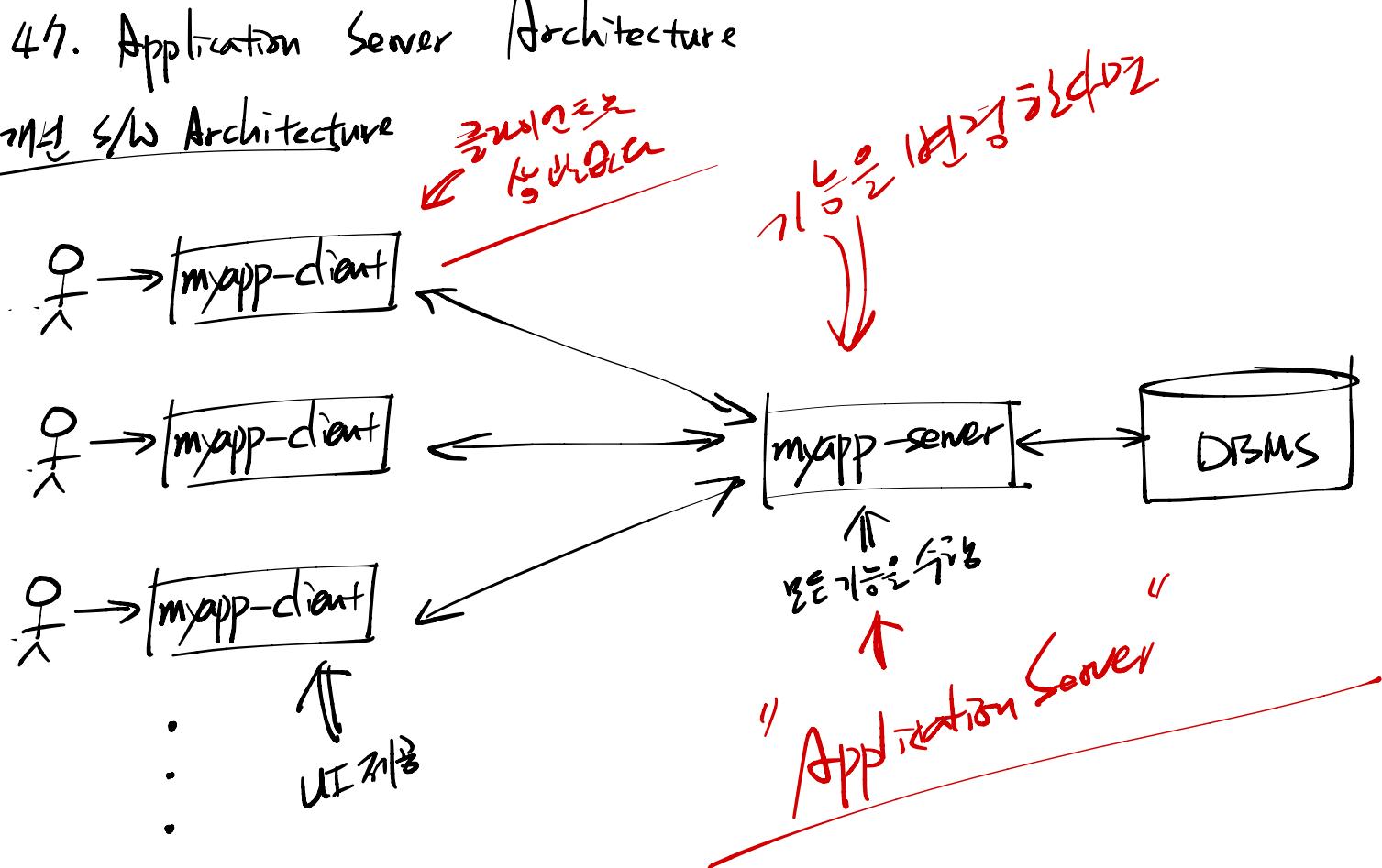
47. Application Server Architecture

① 2-tier S/W Architecture



4.7. Application Server Architecture

② 개발 S/W Architecture



47. Application Server Architecture

