

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

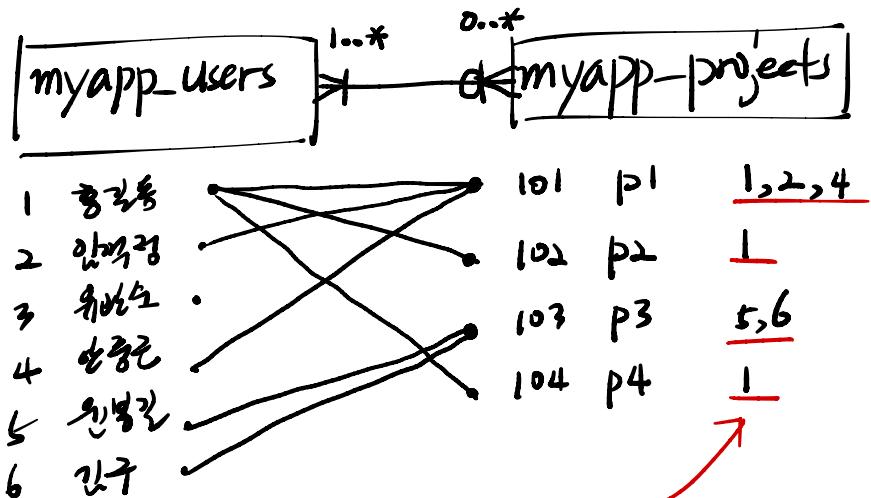
1번은 1번

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
관계 흐름을 제어하는 관계

제어?

* 데이터를 관리하는 틀

- 1 흐름
- 2 일정
- 3 주제

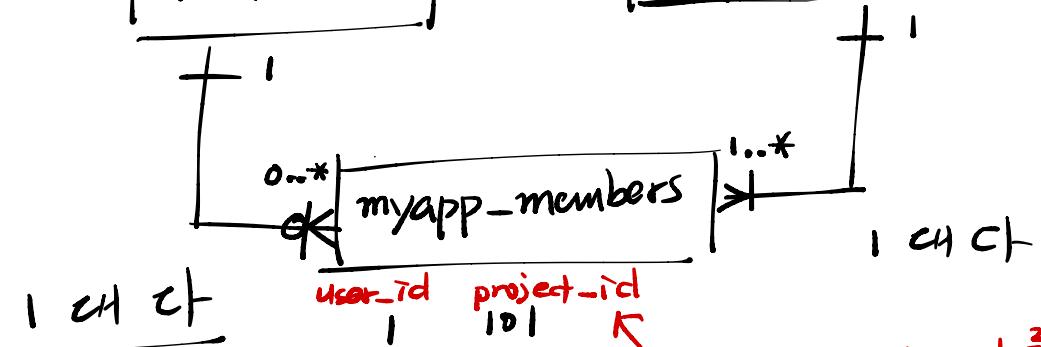
myapp-users	
1	2

- 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



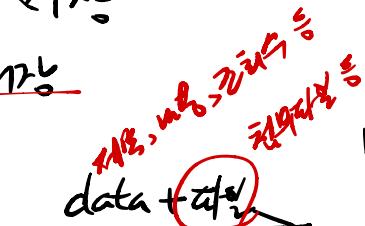
1 CH C1

	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

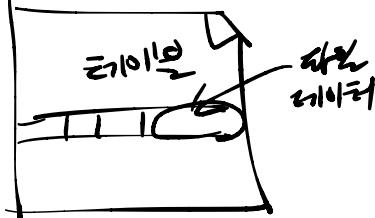
데이터의 구조가
정해져야 한다.
(구조화된 형태)



구조화된 형태
구조화된 형태

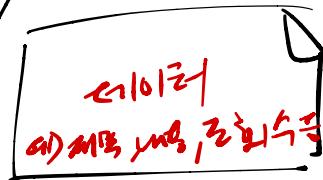


구조화된 형태
구조화된 형태



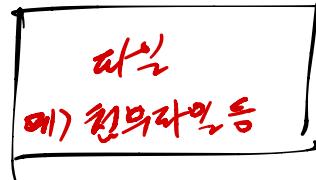
② 자료를 분리해서 저장

데이터 자체의
구조화

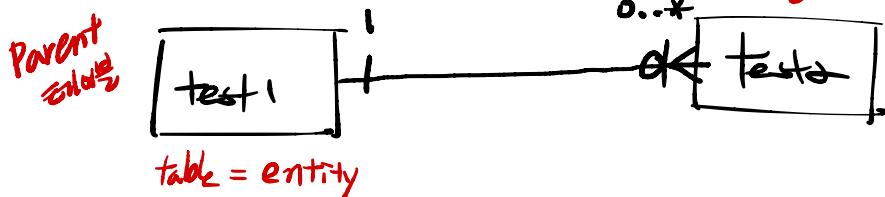


구조화된 형태
구조화된 형태

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 방식은 ~준으로
지원하는지 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

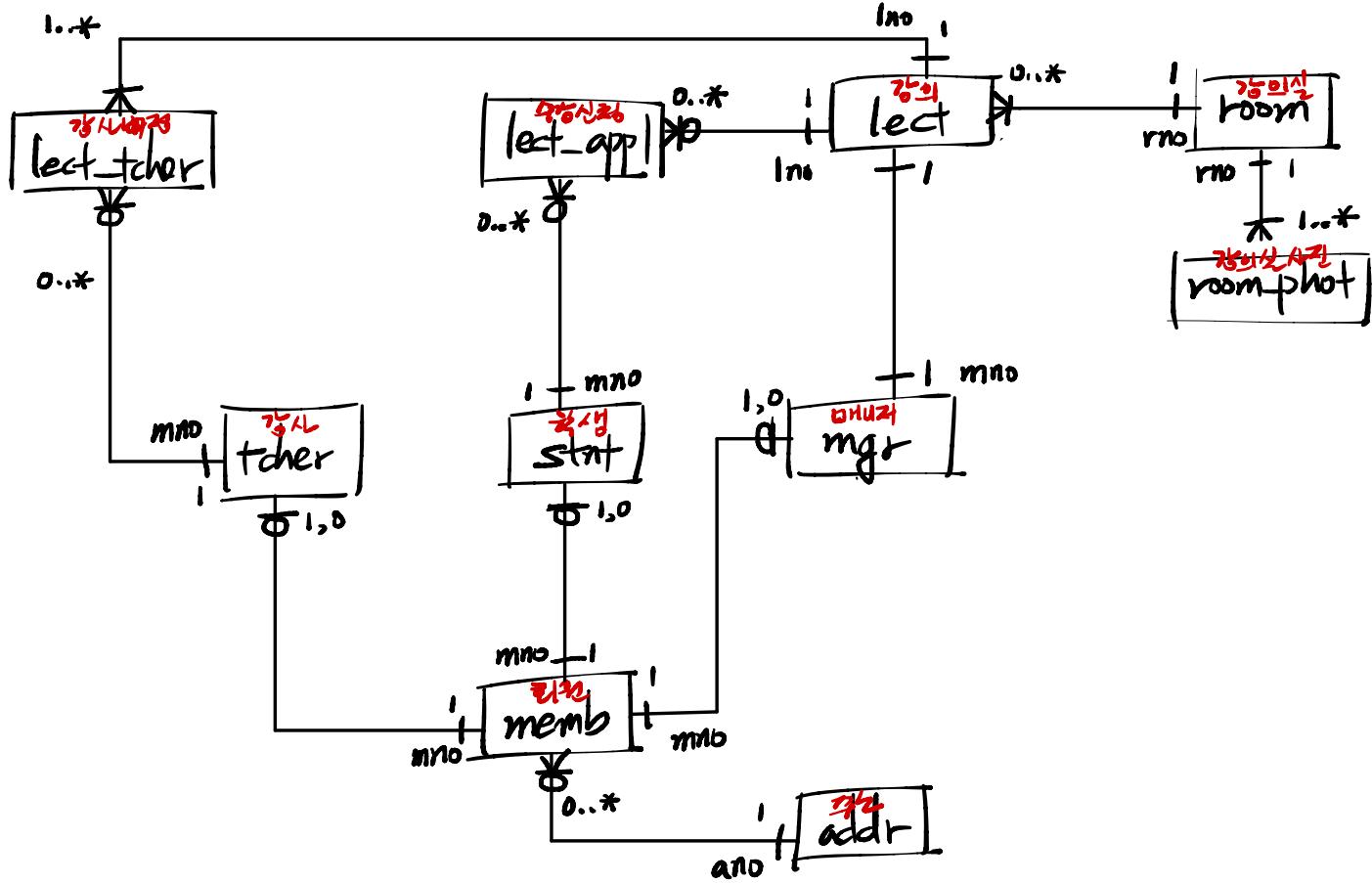
```
select * from attach_file5;
```

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

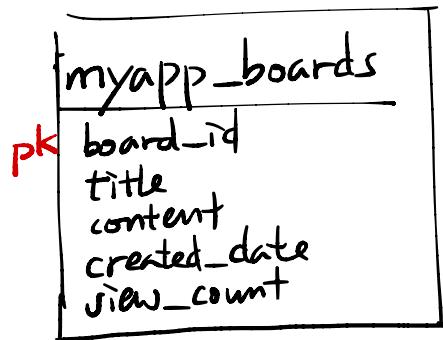
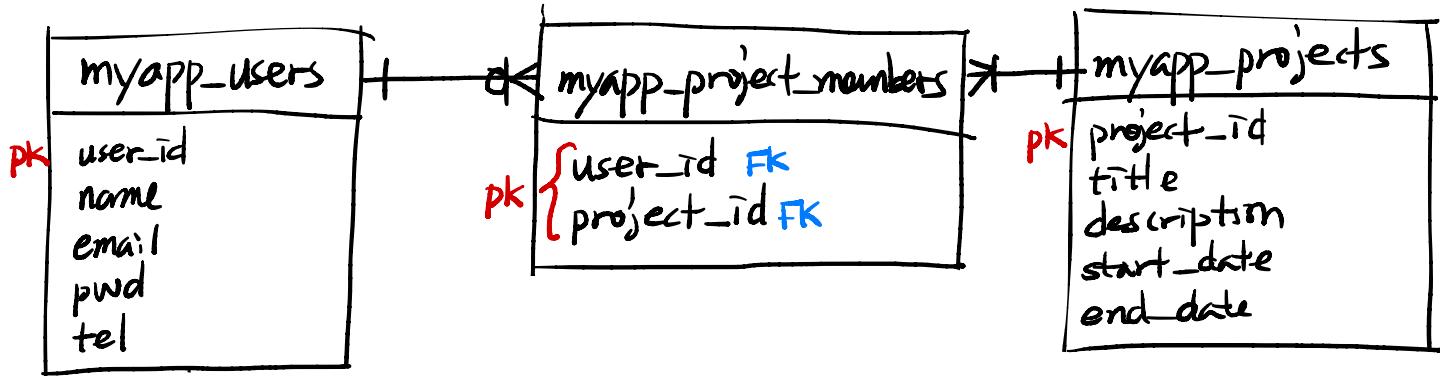
```
select no, title, content, fno, filepath, bno  
from board5 b join attach_file5 a on b.no=a.bno;
```

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

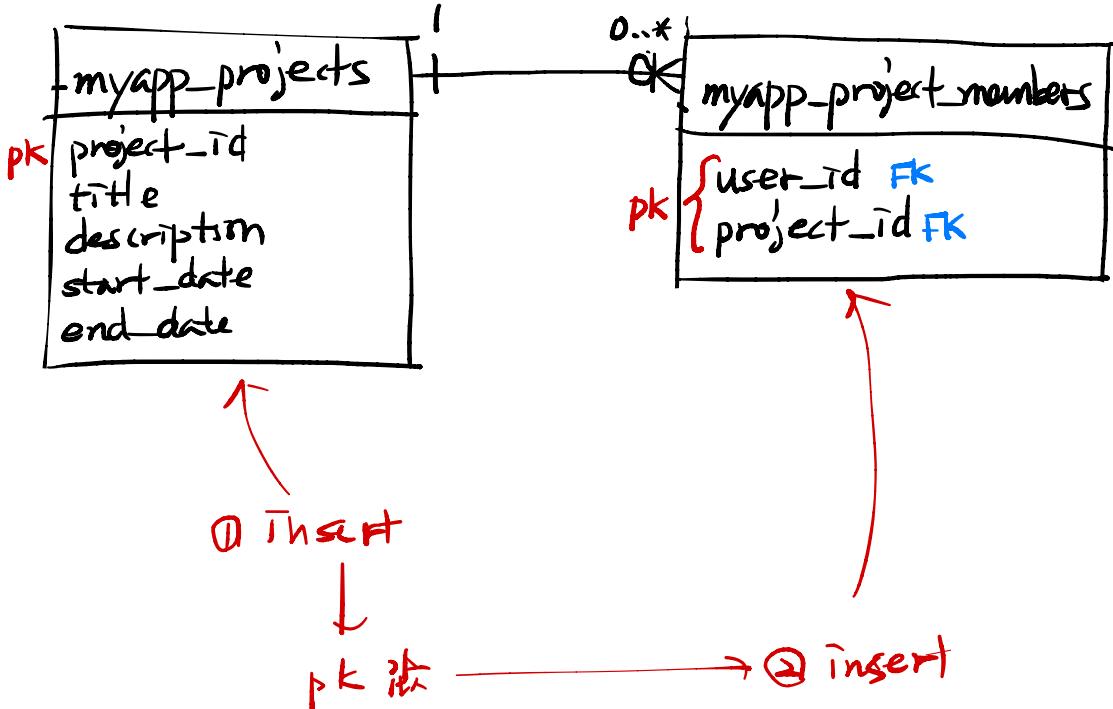
* JOIN 예제 - ERD (Entity Relationship Diagram)



* 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

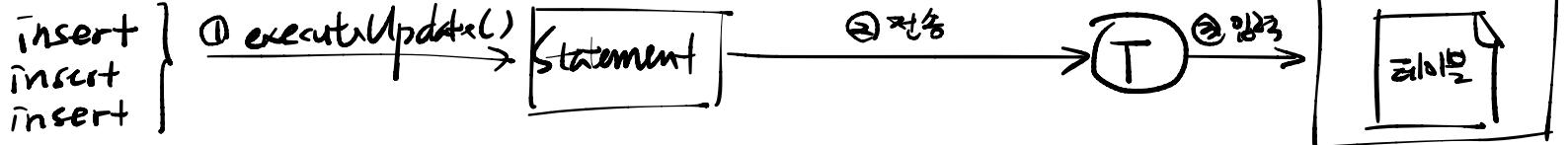
keyRS.getInt(1); 202 ← pk 값

keyRS.next(); → 203 ← pk 값

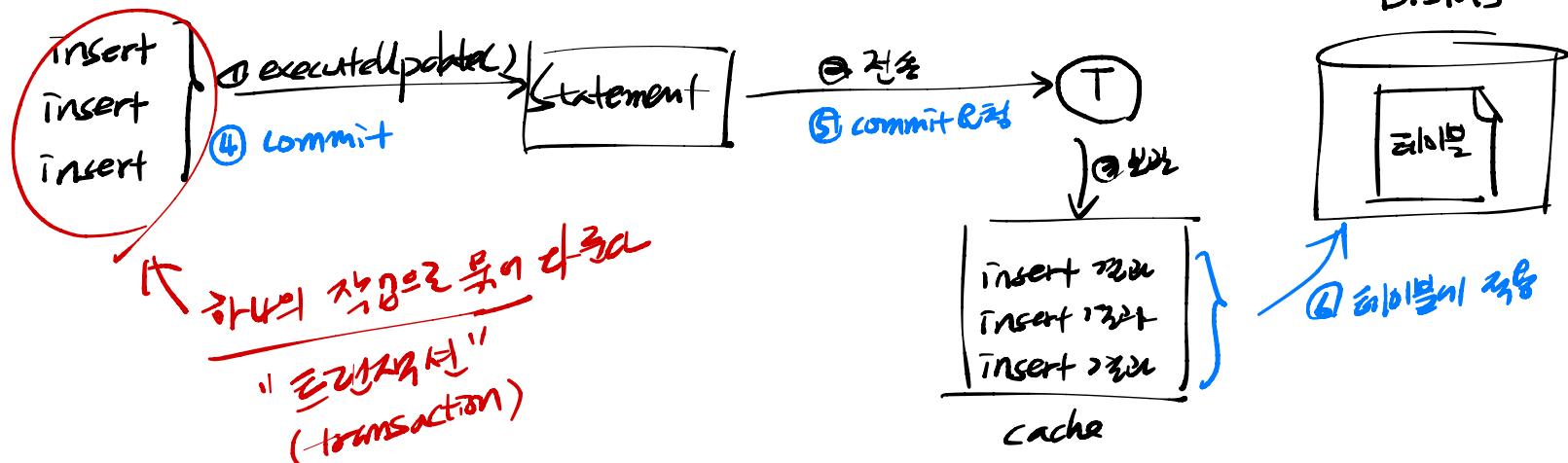
keyRS.getInt(1); 203 ← pk 값

* auto commit vs 수동 commit

1) Auto Commit = 즉시 데이터베이스 반영

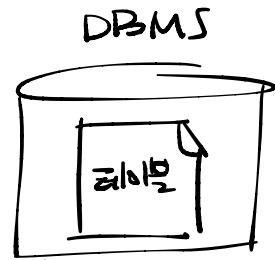
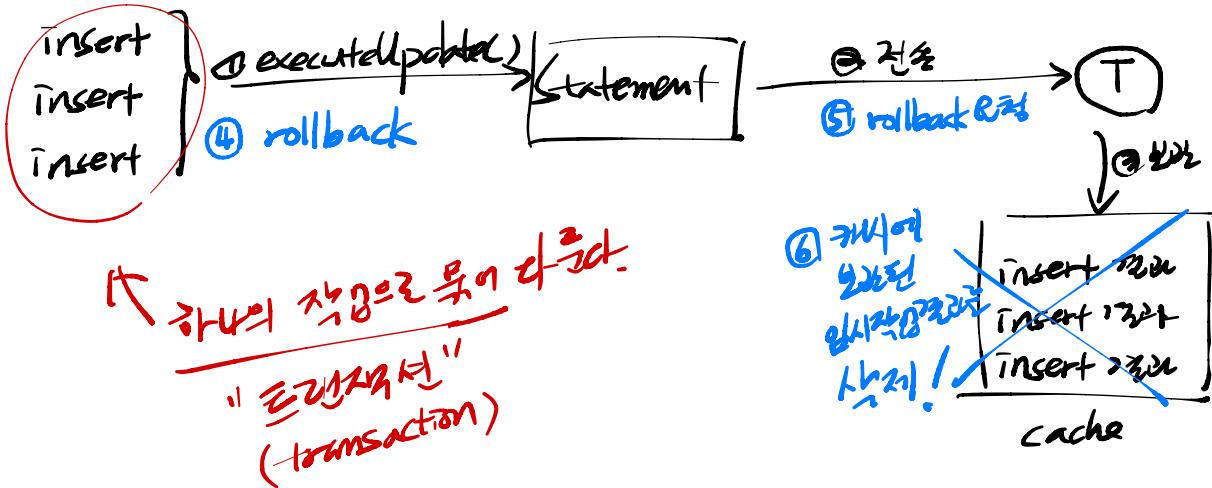


2) 수동 commit = commit

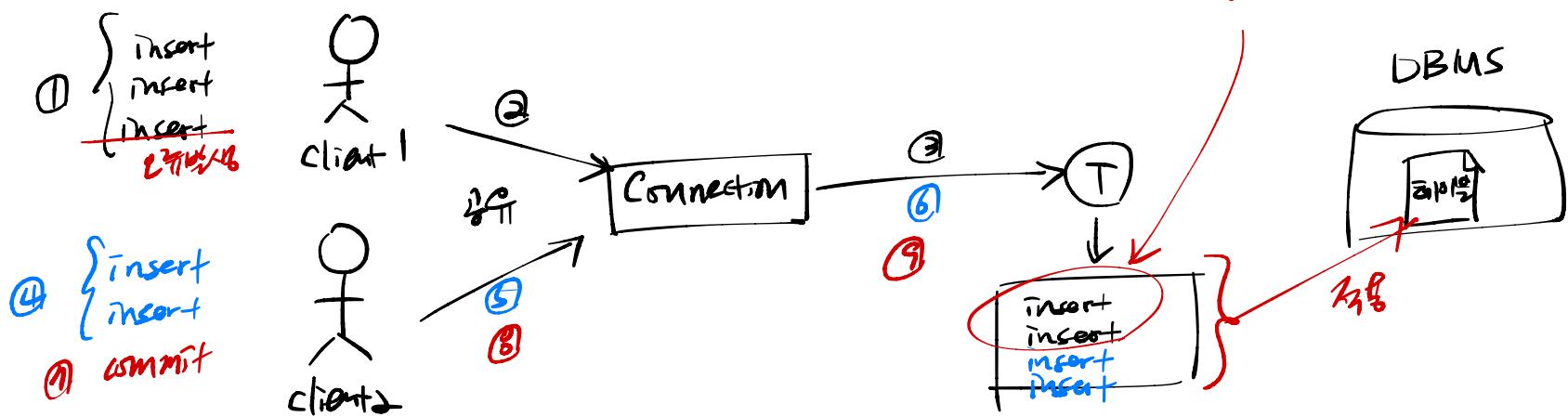
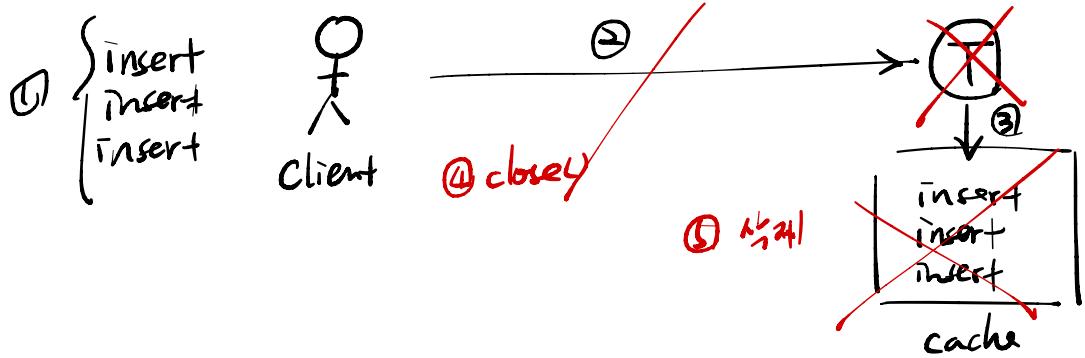


* auto commit vs 수동 commit

2) 수동 commit = rollback

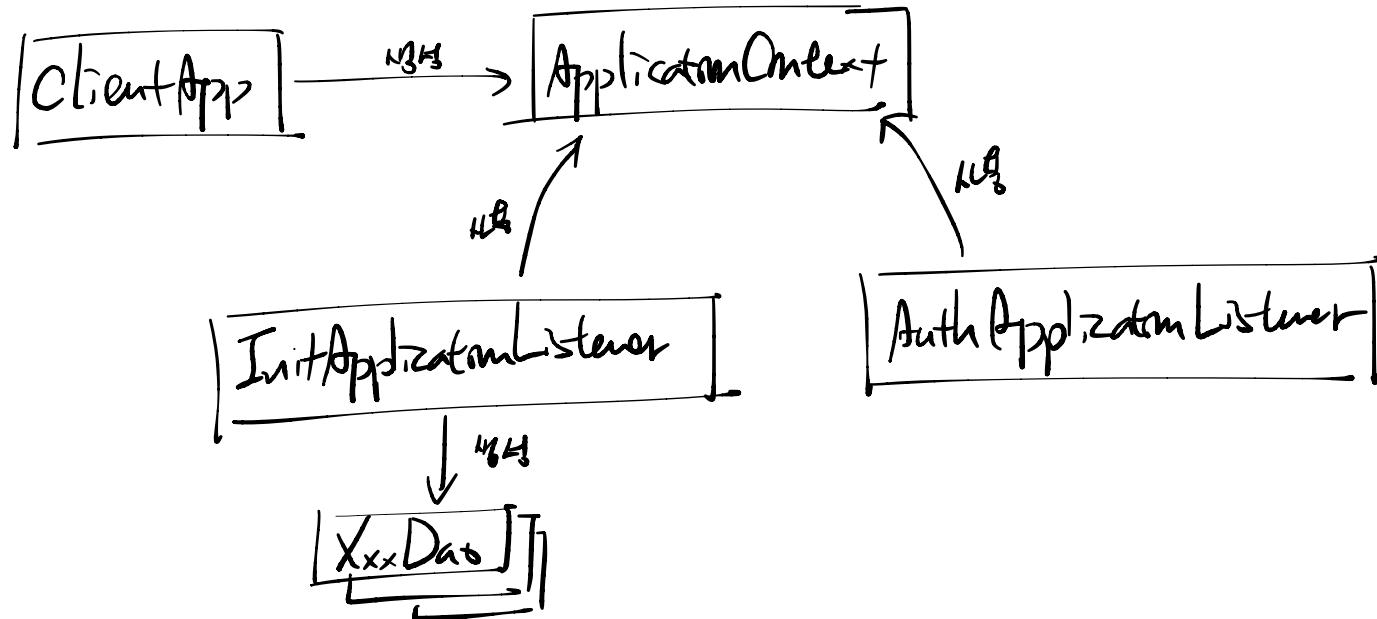


* 쓰기지연된 DB 처리법 장단점

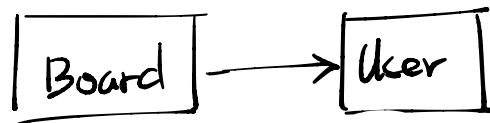


42. 201 / 206 22.8.861

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 햄버거 공격 예제

"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', '
 '" +
 contents + "'")'"

* 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!

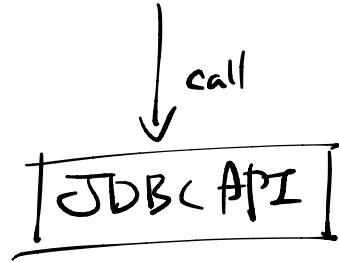
In-parameter
1. 제목
2. 내용

In-parameter
값
값

In-parameter
값
값

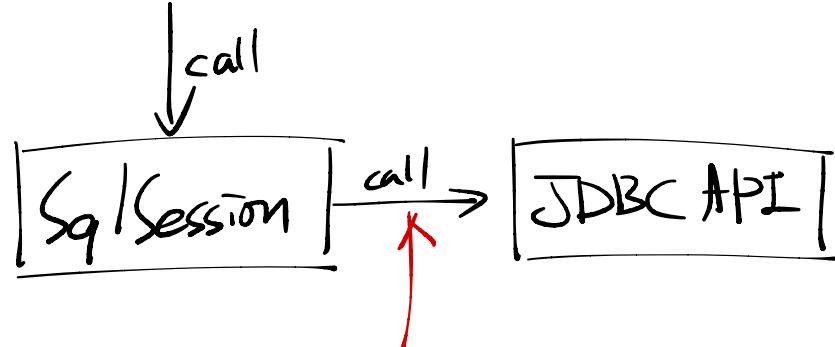
44. JDBC 관계 캡슐화

① 이전 방식



② 개선 방식

비단계적인 관계
될 수 있다.



Reflection API 활용

- 초기화 대체 가능화
- 동적 SQL 가능화

* 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

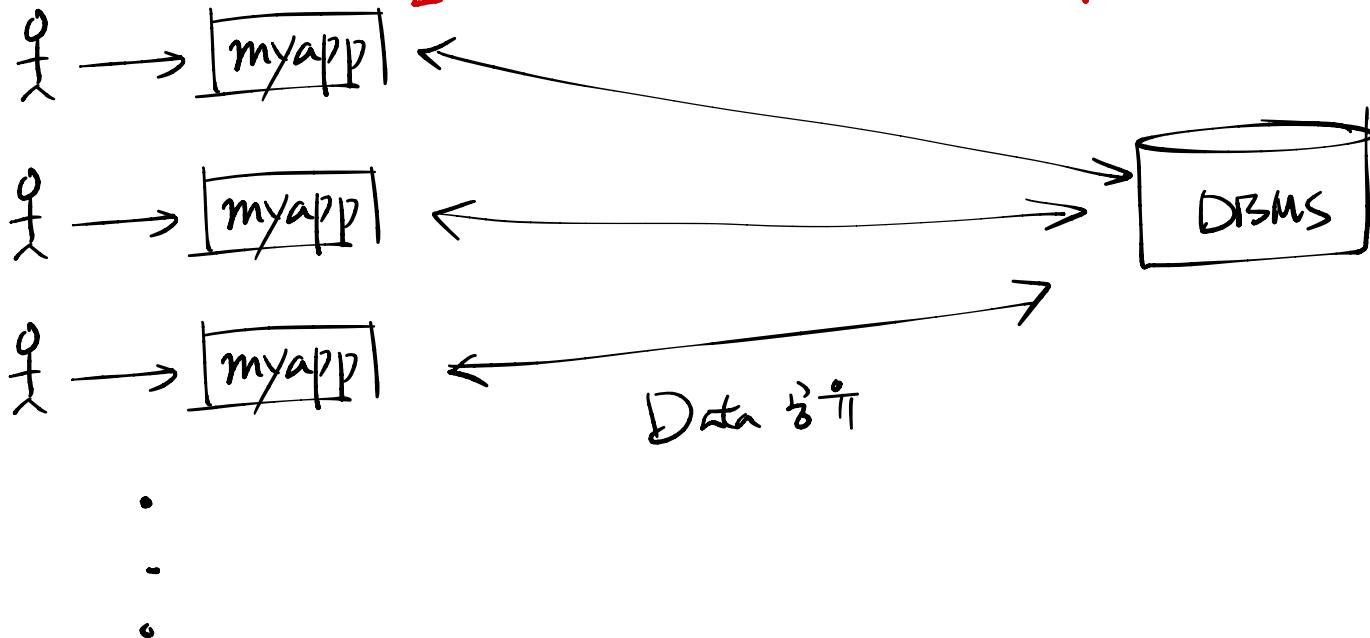
* Field E- Property

```
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

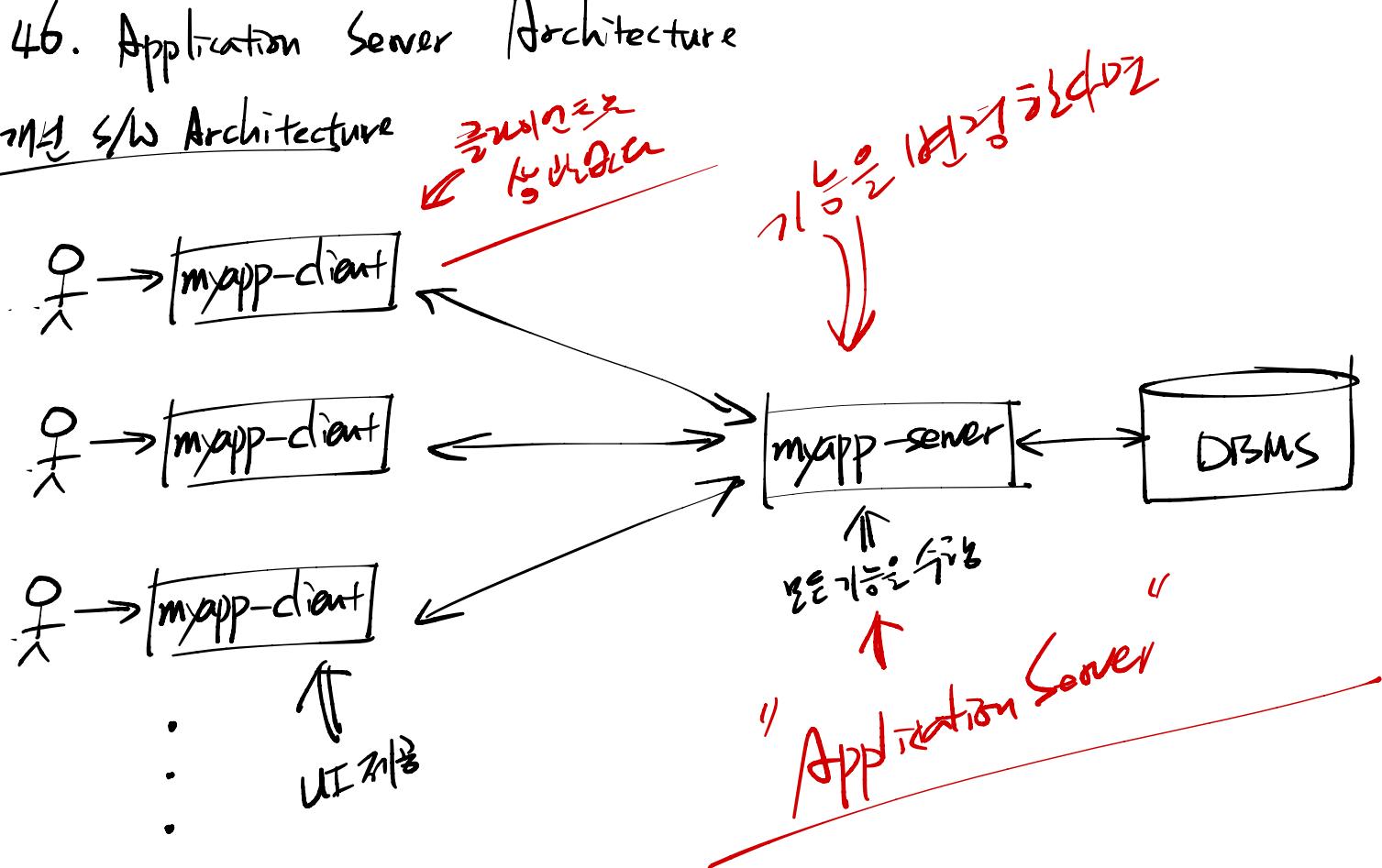
46. Application Server Architecture

① 단일 S/W Architecture



46. Application Server Architecture

② 개발 S/W Architecture



4b. Application Server Architecture

