

## \* DBMS 개념

↳ Database Management System

데이터 보관과 조회를 서비스하는 프로그램

\* Database = 데이터를 다룬다 보관할 것

- 실시간 접근 가능 : 데이터 처리 요청에 즉시 응답
- 동시성 : 여러 사용자가 동시에
- 데이터의 독립성 : 애플리케이션에 비중
- 일관성 : 데이터 처리 작업이 끝난 후 데이터의 값은 유일한 상태를 유지해야 한다.
- 무결성 : 유일하지 않은 데이터의 등록과 변경, 삭제는 제한
- 보안성 : 사용자 인증과 권한 관리 ↳ 예) 기사를 작성할 수 있다.

데이터를  
보관하는  
곳



auth

(authentication)

ID/PWD 일치

유일한 사용자

(authorization)

기능의 이용권한 관리

등록 허용?  
변경 허용?

:

## \* DBMS

- 데이터베이스를 관리하는 S/W  
- 예) Oracle, MySQL, PostgreSQL, MS-SQL, Altibase, Tibero, Cubrid 등

↳ 예) 특정 테이블 데이터 쓰기

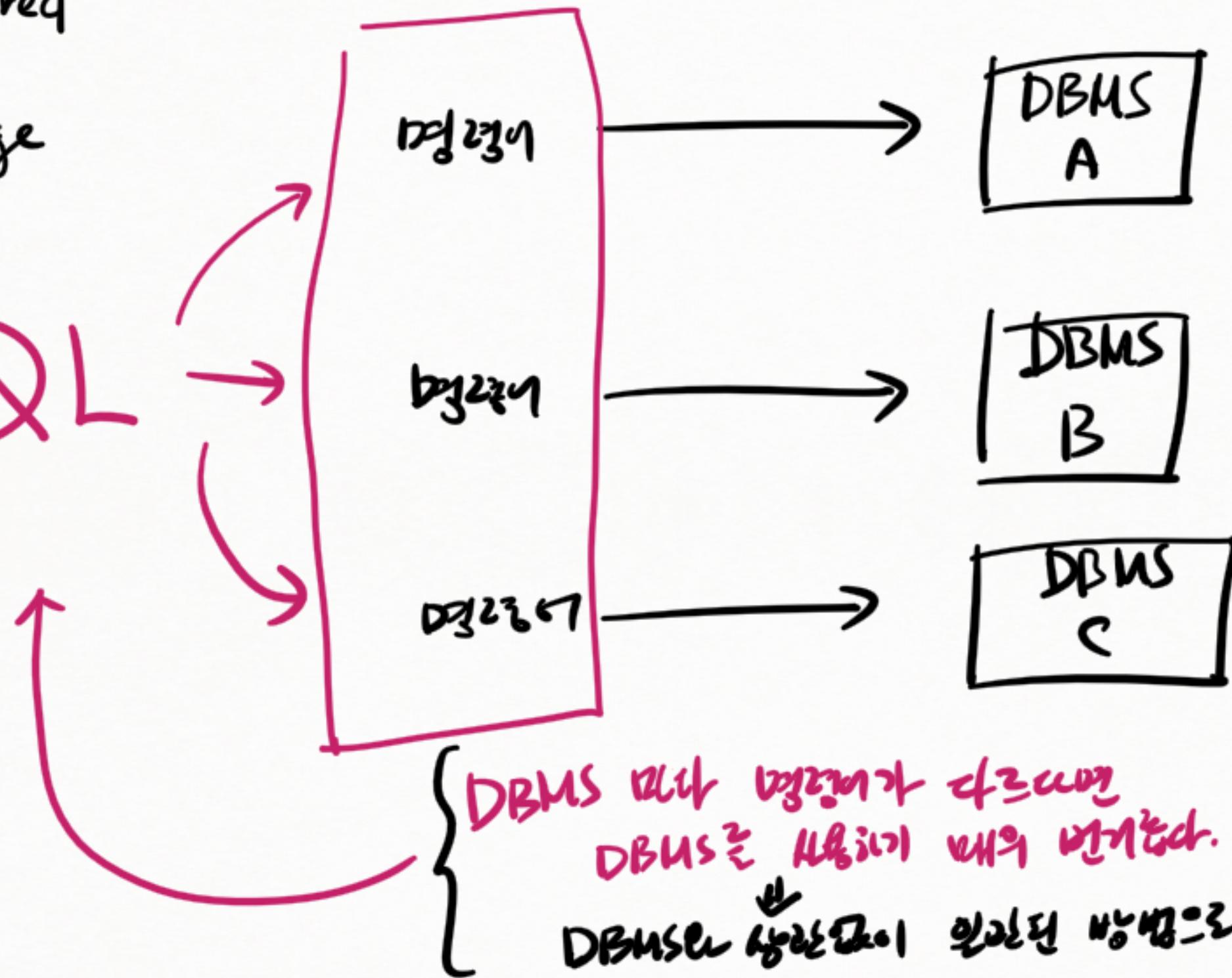
↓  
설정된 티머의 글로벌 쓰기

## \* SQL의 탄생

Structured  
Query  
Language

SQL

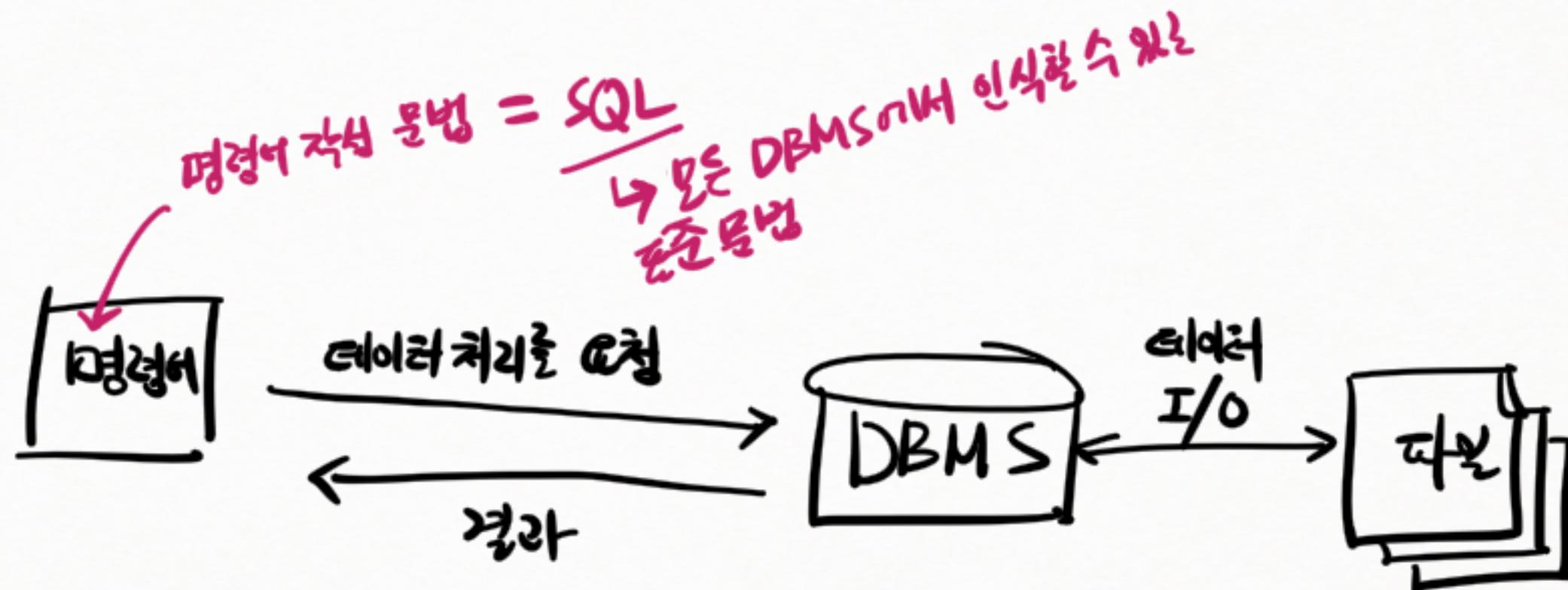
명령어  
작성  
방법을  
통해



DBMS마다 명령어가 다르므로  
DBMS를 사용하기 매우 번거롭다.

DBMS와 함께 명령어를 작성하는 방법이 매우 편리하다!

## \* DBMS 와 SQL



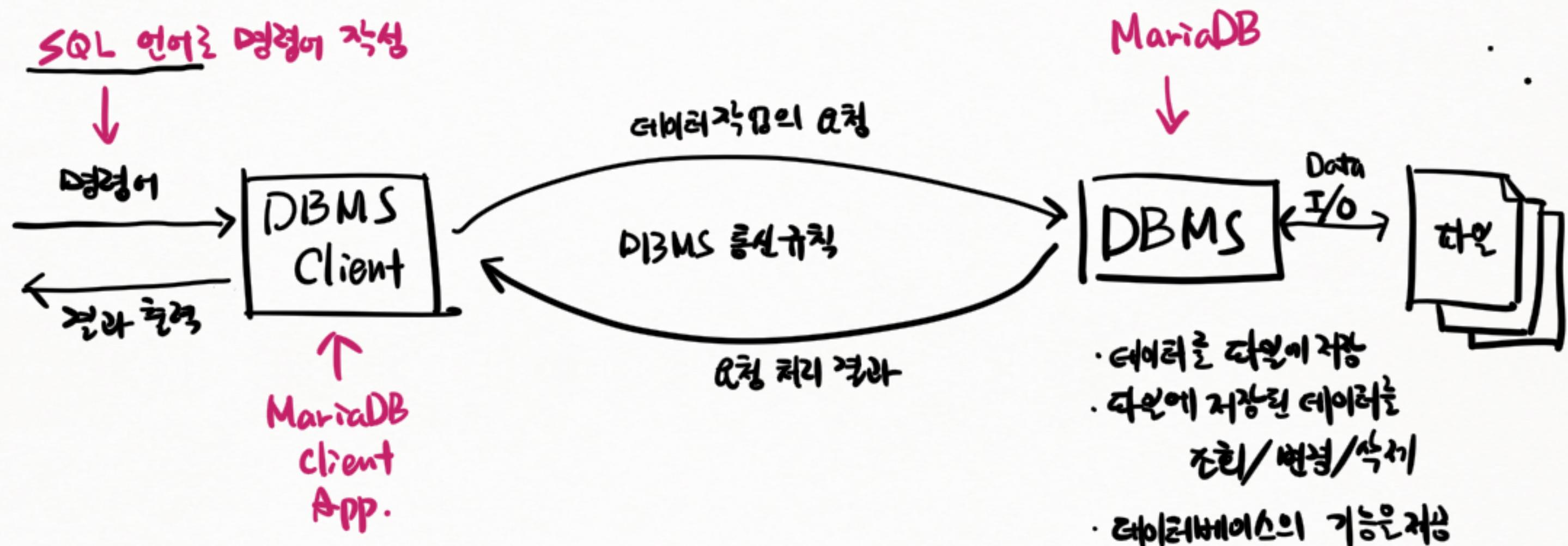
SQL 명령을 보내고 결과를 받으려면

DBMS에서 정한 규칙에 따라 통신을 해야 한다!

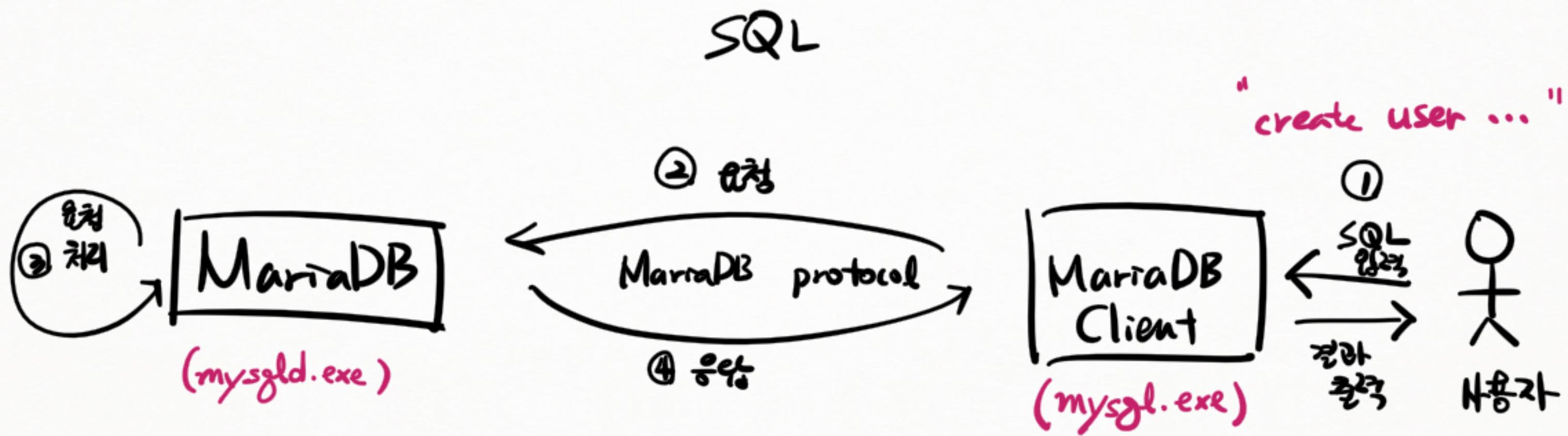
↳ 문제는 DBMS 제조사 쪽에서 통신규칙을 공개하지 않는다.

↳ 애신 통신을 대행하는 클라이언트를 제공한다.

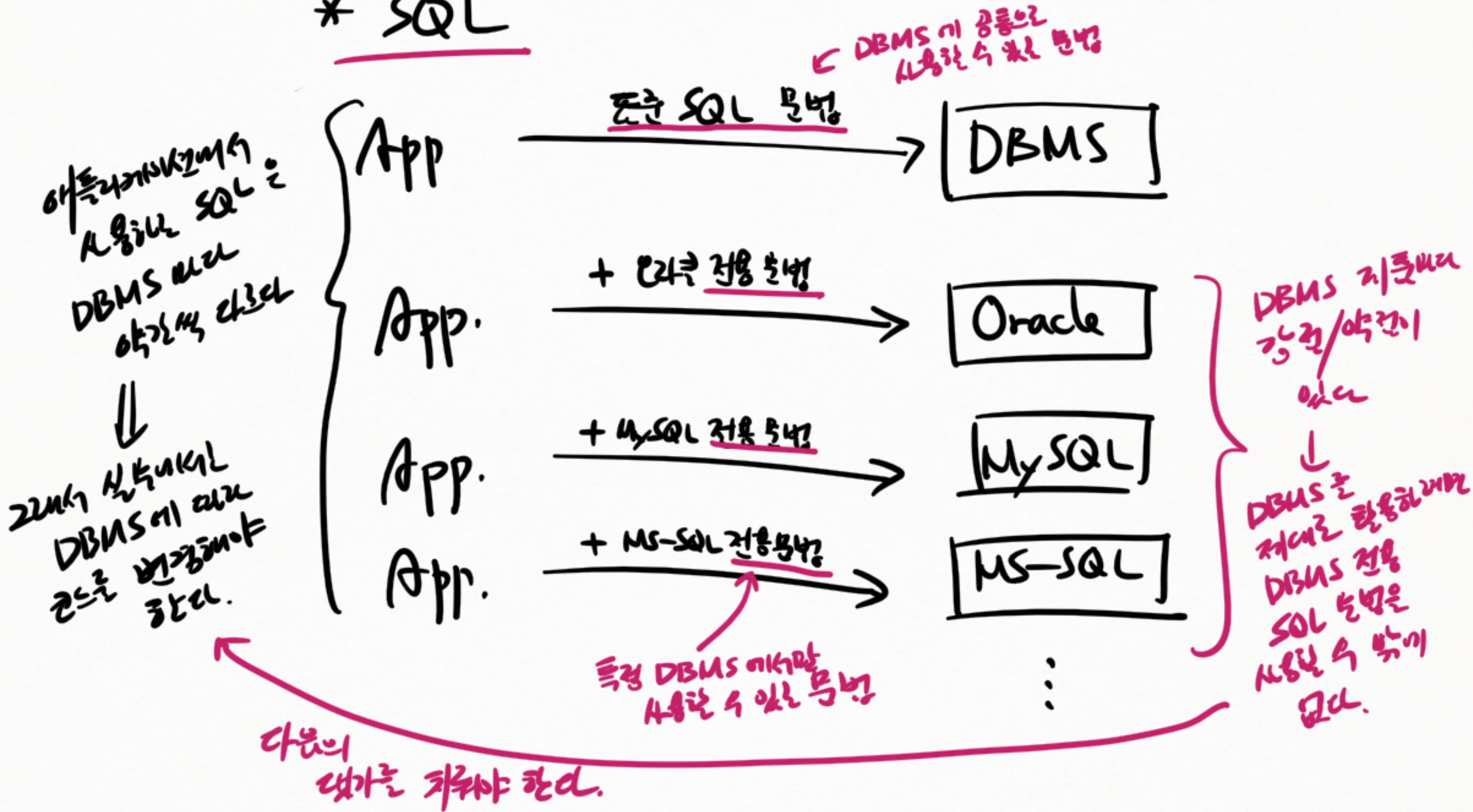
## \* DBMS 서버와 클라이언트



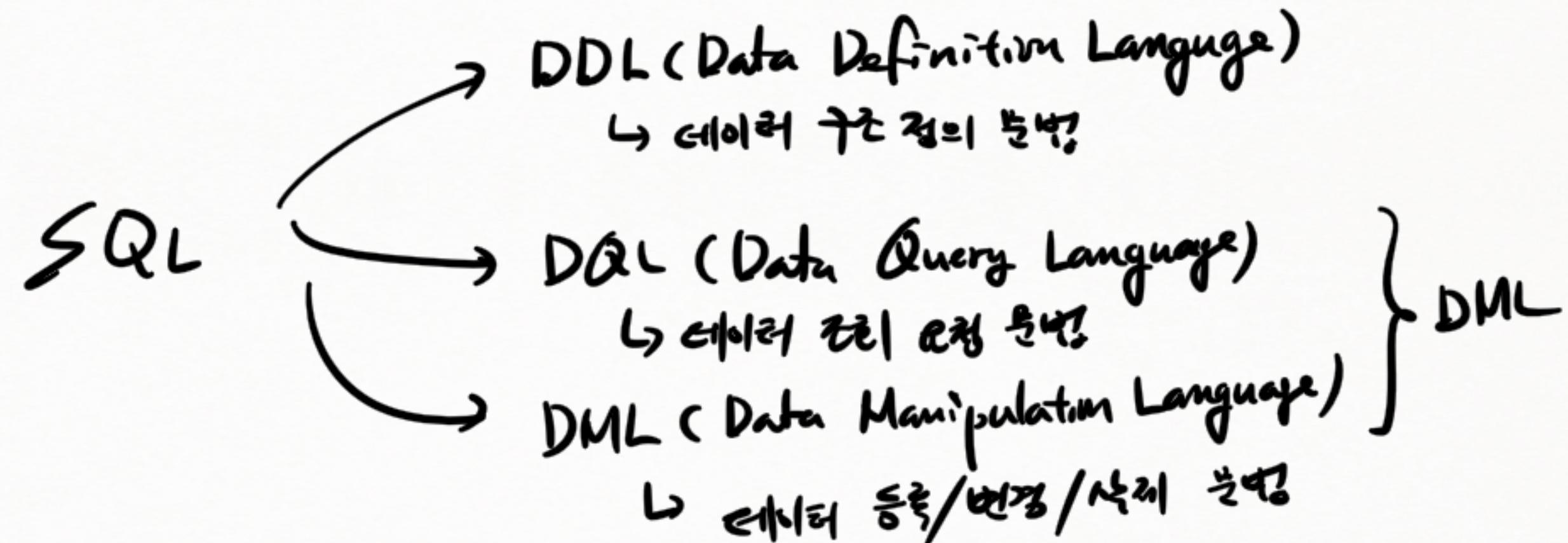
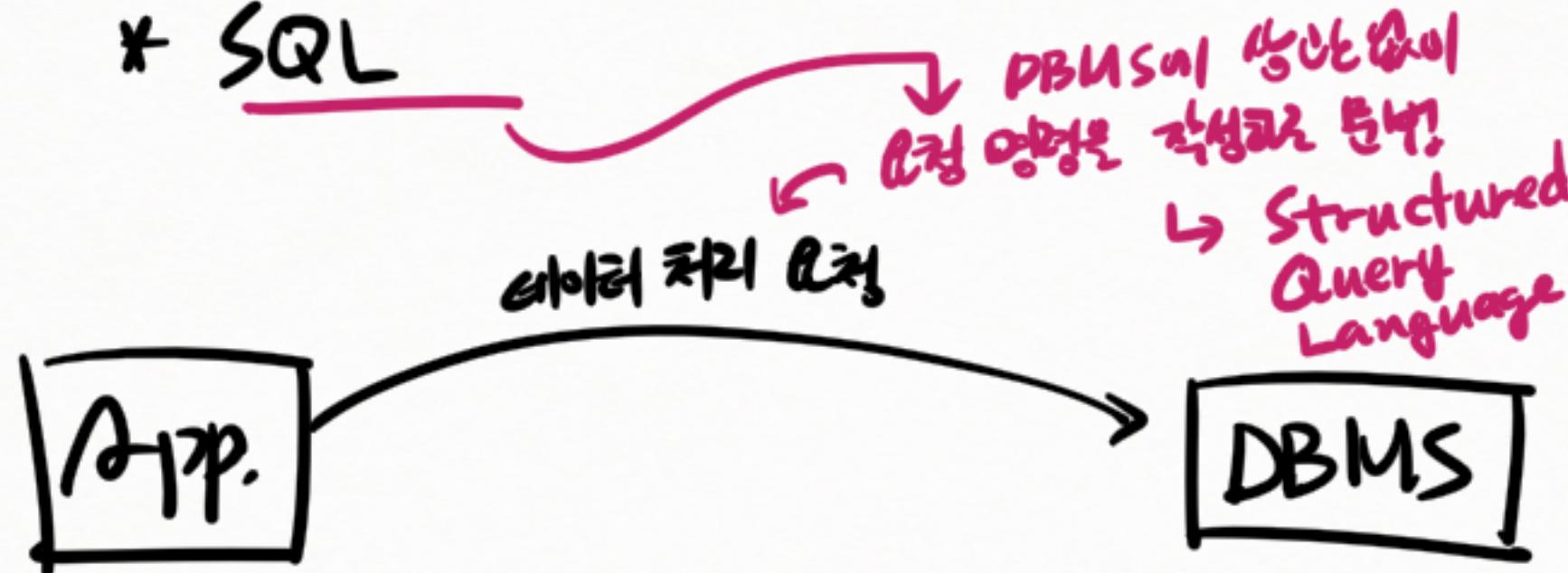
## \* DBMS 서버와 클라이언트, SQL



## \* SQL



## \* SQL



## \* SQL

① DDL → 데이터를 저장하는 다음 DB 개체를 정의한다

②  $\begin{cases} DQL \\ DML \end{cases}$  → DB 개체의 데이터를 다룬다  $\Rightarrow \begin{cases} \text{Table} \\ \text{View} \\ \text{Procedure} \\ \text{Function} \\ \vdots \end{cases}$

## \* SQL 테스트 준비

## ① 사용자 추가

```
> create user 'study'@'%' identified by '1111'; <
```

## ② 태이러를 저항할 태이러 베이스 생성

③ 데이터베이스를 사용할 사용자의 권한을 지정

\* DDL (Data Definition Language)

## ① 테이블 생성

레이블 쟁쟁

create table test01 (

name varchar(50) not null,  
← 커럼 타입 ← 옵션

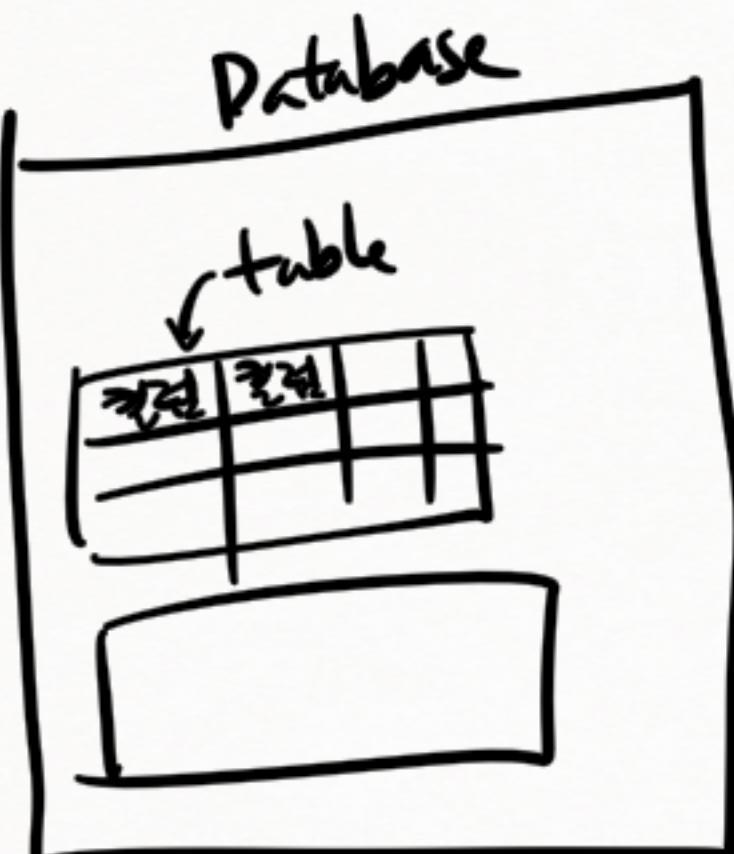
**컬럼명** kor int not null,

eng int not null,

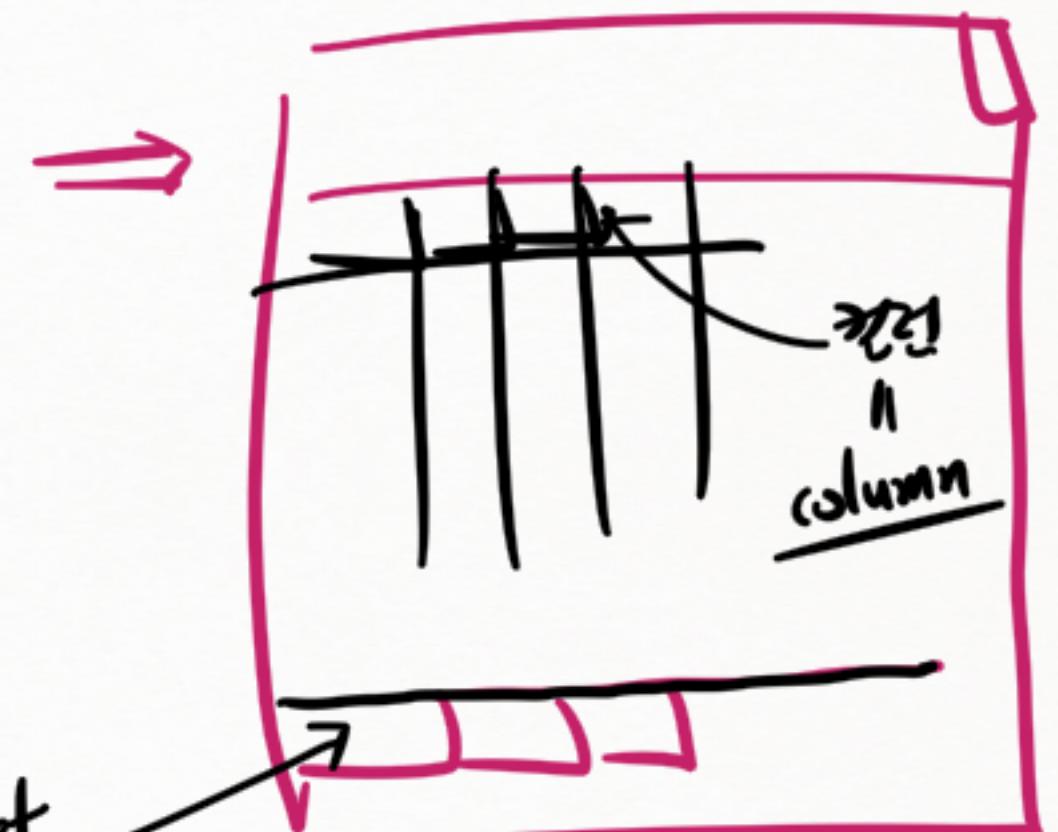
math int not null,

sum int not null,

```
aver float not null);
```

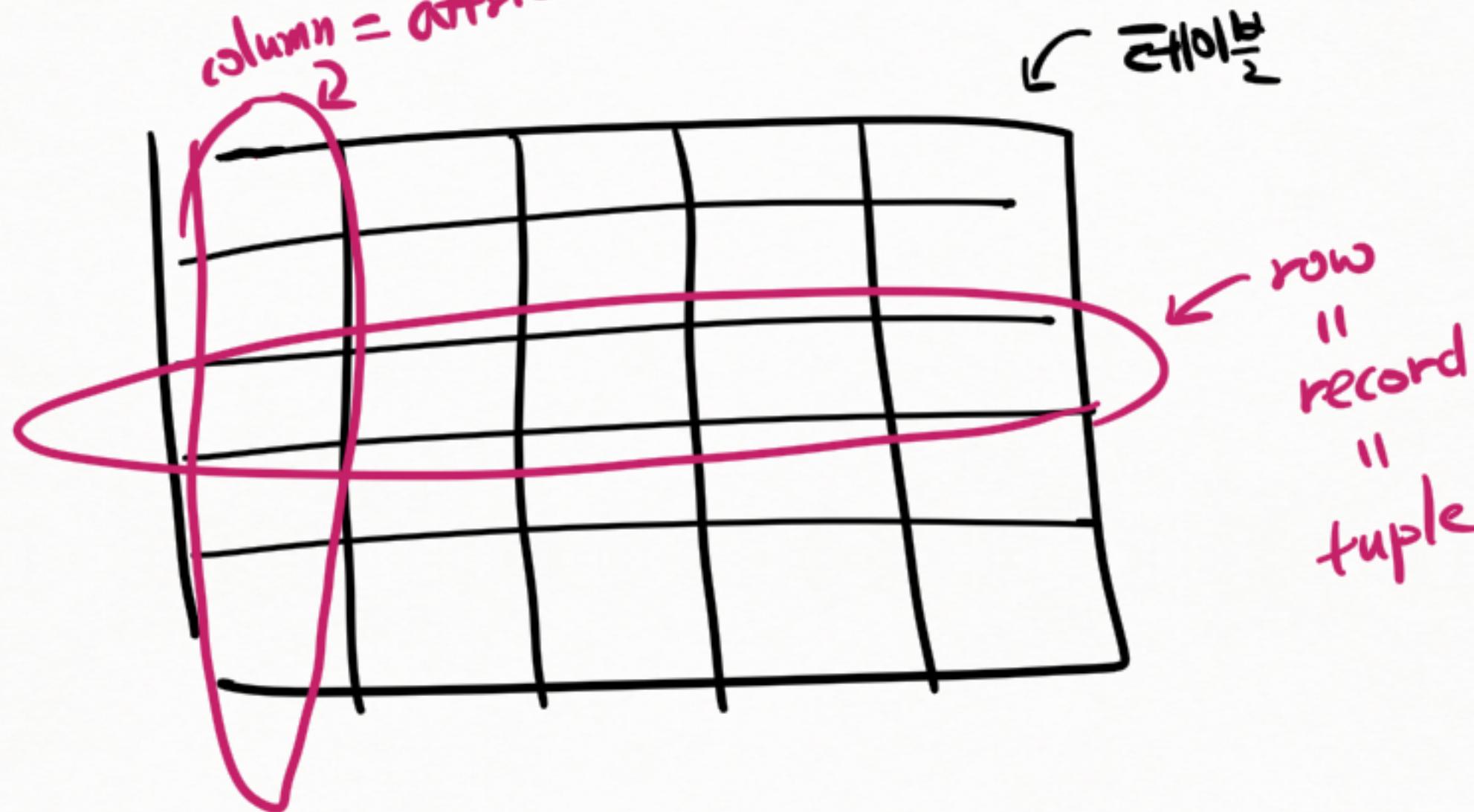


액센트 비즈



\* 레이블과 row, column

column = attribute = field



\* 테이블

✓ create table 테이블명 ( 컬럼, ...) ← 테이블 정의

✓ drop table 테이블명 ← 테이블 삭제

✓ alter table 테이블명 변경사항 ← 테이블 변경

✓ describe 테이블명 ← 테이블 정보 조회  
    "desc"

## \* insert

insert into test1(no, name) values(1, 'aaa')

↑  
레코드명

↑  
컬럼명

↑  
컬럼에 들어갈 값

컬럼을 나열한 순서다 일치해야 한다.

\* select

select no, name from test1

          ↑↑  
          컬럼명

          \*  
          ↑  
          모든컬럼

\* key, candidate key, primary key / alternate key

키/C→I

artificial key

데이터를 구현할 때 사용한 건전진\*

key

[이메일]

[아이디]

[주민번호]

[이름, 전화번호]

[아이디, 전화번호]

[이메일, 주민번호]

[이메일, 이름]

[주민번호, 이름]

[이름, 아이디, 전화번호]

[이름, 주민번호]

\* key, candidate key, primary key / alternate key

키/C→I

artificial key

데이터를 구현할 때 사용한 키는 \*

key

candidate key

↳ 키/C

\* [이디]

\* [아이디]

\* [주민번호]

\* [이름, 전화번호]

[이디, 이름]

[아이디, 전화번호]

[주민번호, 이름]

[이름, 주민번호]

[이디, 주민번호]

[이름, 아이디, 전화번호]

\* key, candidate key, primary key / alternate key

키/C

candidate key

데이터를 구분할 때 사용한 키

Alternate  
key

→ 키로 설정되지 않은  
내부 키로 다른 키를  
'대안키'가 된다

key

candidate key  
↳ 키/C

[아이디]

\* [주민번호]

\* [이름, 전화번호]

↑  
DB 관리자가  
주 키로  
설정한 키

"Primary key"

## \* Artificial key

제시글 : 제목, 내용, 작성일, 작성자, 조회수, 좋아요수

제시글 번호

Primary key?

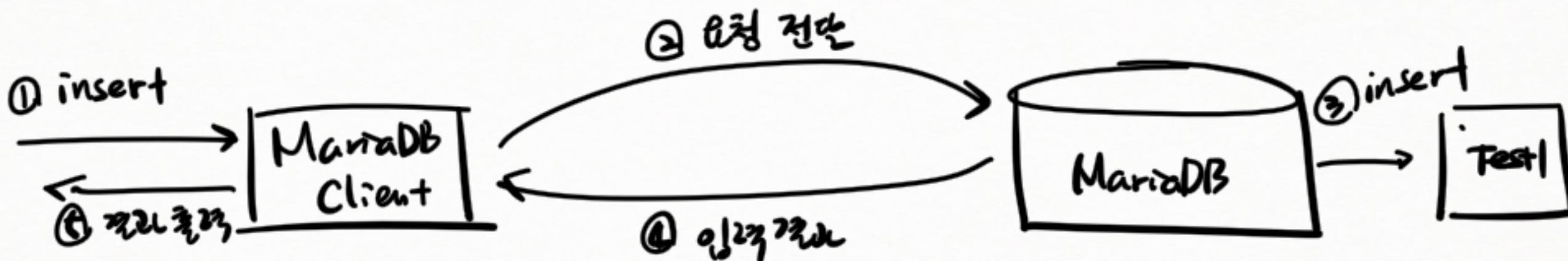
보통히 사용할 key를 없을 경우

Artificial (인공적)  
key

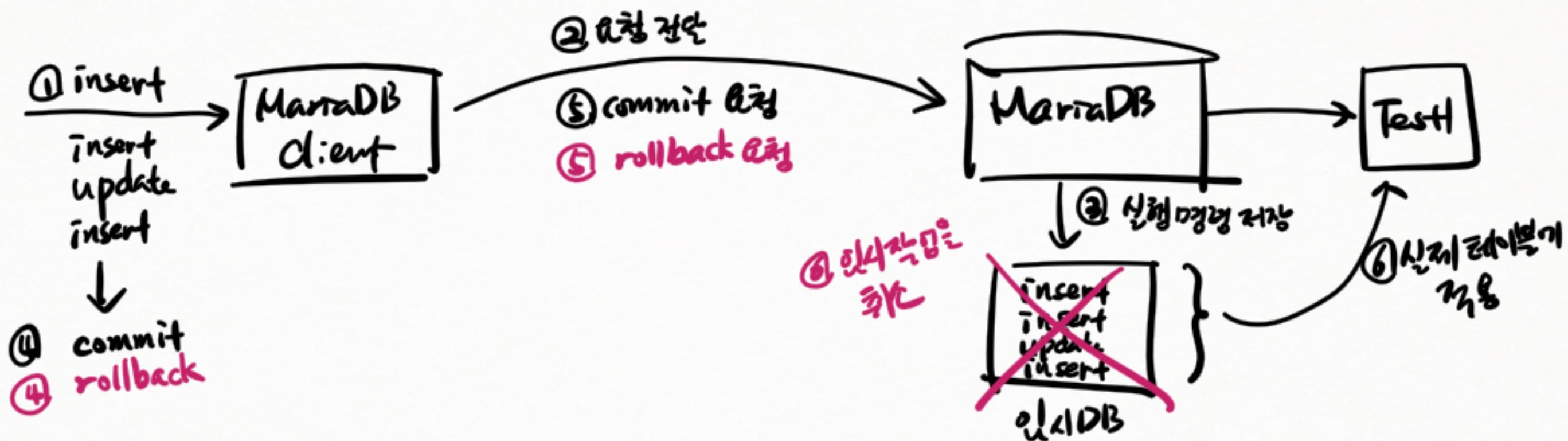
= 인공적 번호를 만들어  
PK로 사용한다  
↳ ex) 일련번호

## \* Commit

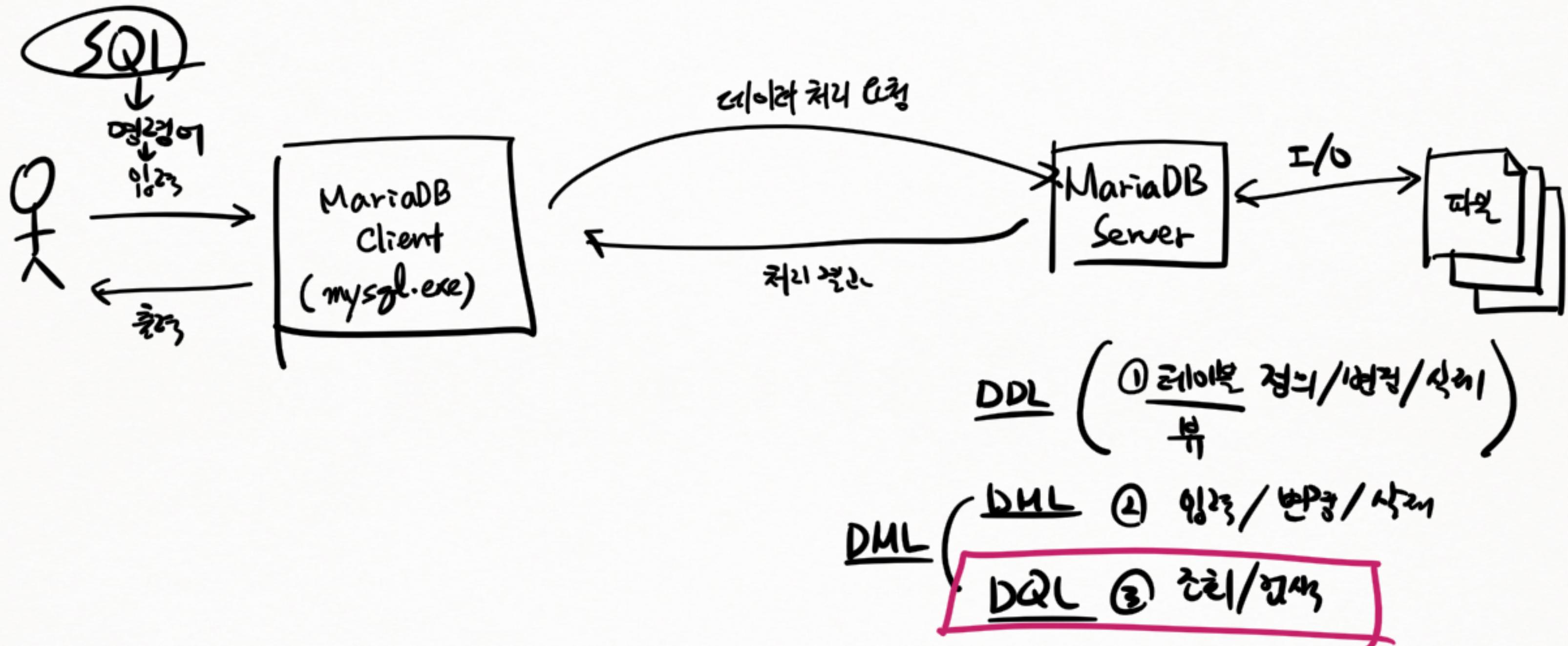
① autocommit = true



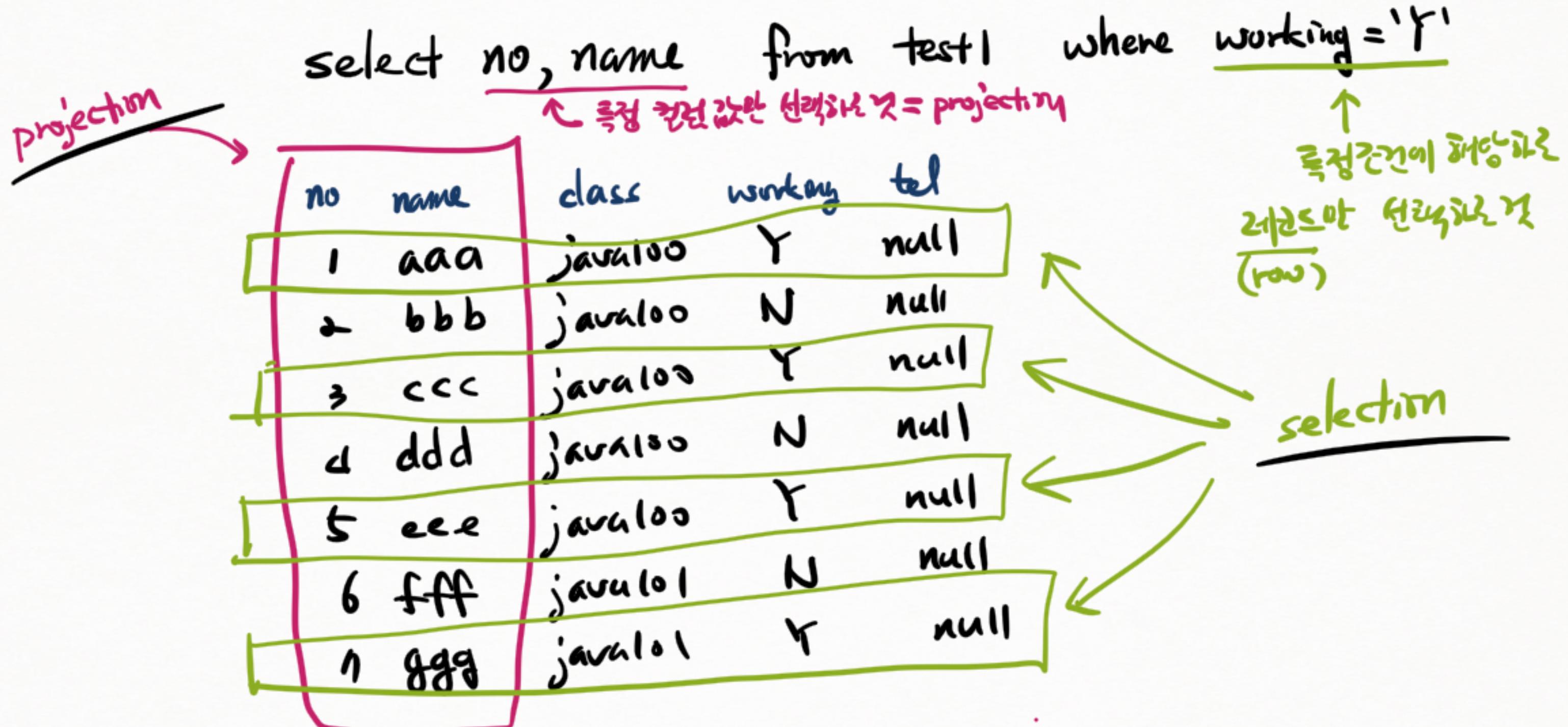
② autocommit = false



\* select

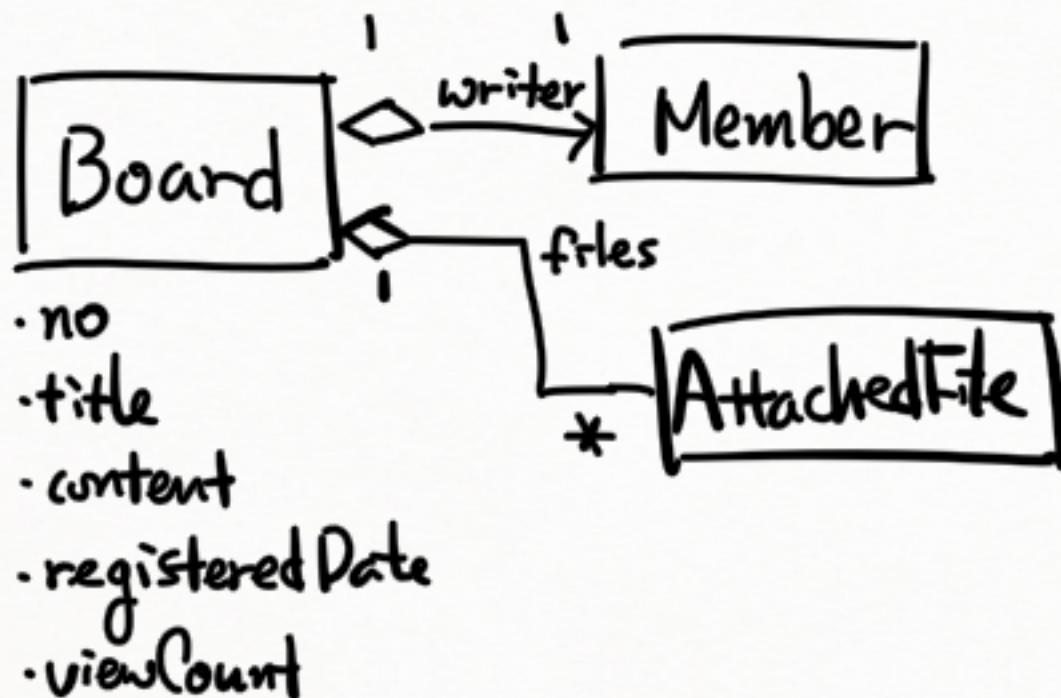


## \* selectin / projection



## \* Foreign Key

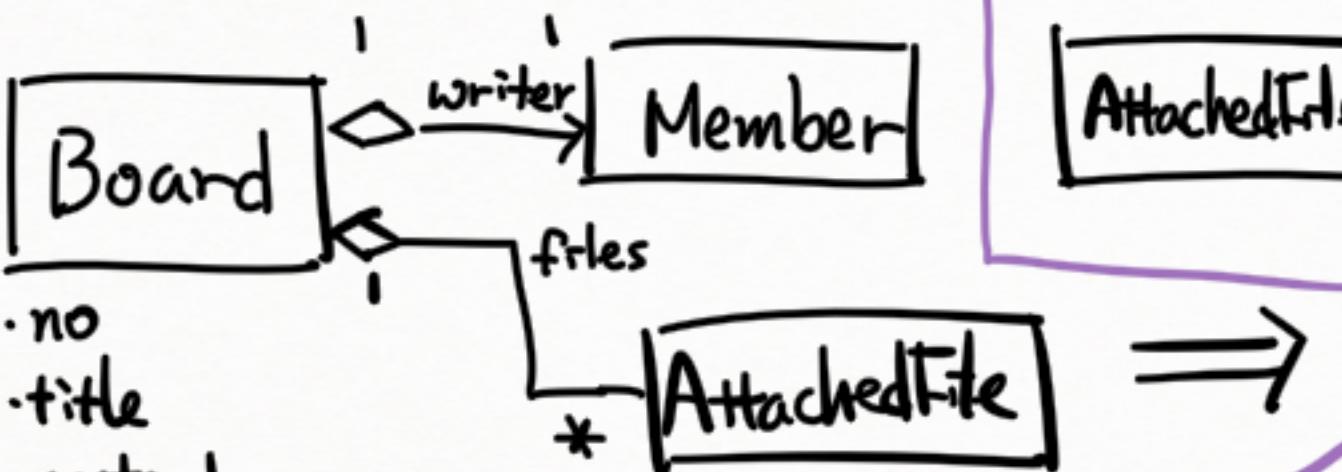
① 자식 가족



```
class Board {  
    int no;  
    String title;  
    String Content;  
    Date registeredDate;  
    int viewCount;  
    Member writer;  
    List<AttachedFile> files;  
    ...  
}
```

## \* Foreign Key

① 자식 키체



- no
- title
- content
- registeredDate
- viewCount

② 테이블 간의 관계 (ER-Diagram; ERD)



- 1번 게시글
  - 2번 게시글
  - 3번 게시글
- 100번 조회수
  - 101번 댓글수
  - 102번 유저수

ERD 특징:

① Information Engineering Notation

공식

↓  
정준으로 일정 사실을  
학문으로 체계화·나누어서  
연구하는 것

↓  
해당 분야에 종사하고 사는들이  
사용하는 언어로 드러낸다

\* 객체 간의 관계와

테이블 간의 관계는 다르다!

↓  
아마히 일부는 관계화시키는 데로  
관계를 설정.

프로그래밍 언어  
위한 목적으로 객체간의  
관계를 설정.

\* Foreign Key (외부키) - 다른 테이블의 pk 참조



(PK) 번호	파일명	제작자(FK)	(PK) 번호	제작자	제작자(FK)	(PK) 번호	이름
11	aaa.gif	101	101	aaaa	1	1	홍길동
12	bbb.gif	102	102	bbbb	1	2	김꺽정
13	ccc.gif	102	103	cccc	3	3	유재석
14	ddd.gif	103	104	dddd	4	4	안우진
15	eee.gif	103					

## \* 커먼 쿵복

Board							
no	title	content	rdt	f1	f2	f3	f4
1	aaa	—	—	a.gif	b.gif		
2	bbb	—	—	x.gif	y.gif	z.gif	t.gif
3	ccc	—	—	o	o	o	o
4	ddd	—	—	m1.gif	m2.gif	m3.gif	m4.gif m5.gif

이렇게 같은 종류의 파일은 여러 개 저장하기 위해 커먼 쿵복으로 선언된 경우

첨부파일이 없음에도 5개의 커먼이 존재하기 때문에 메모리 낭비

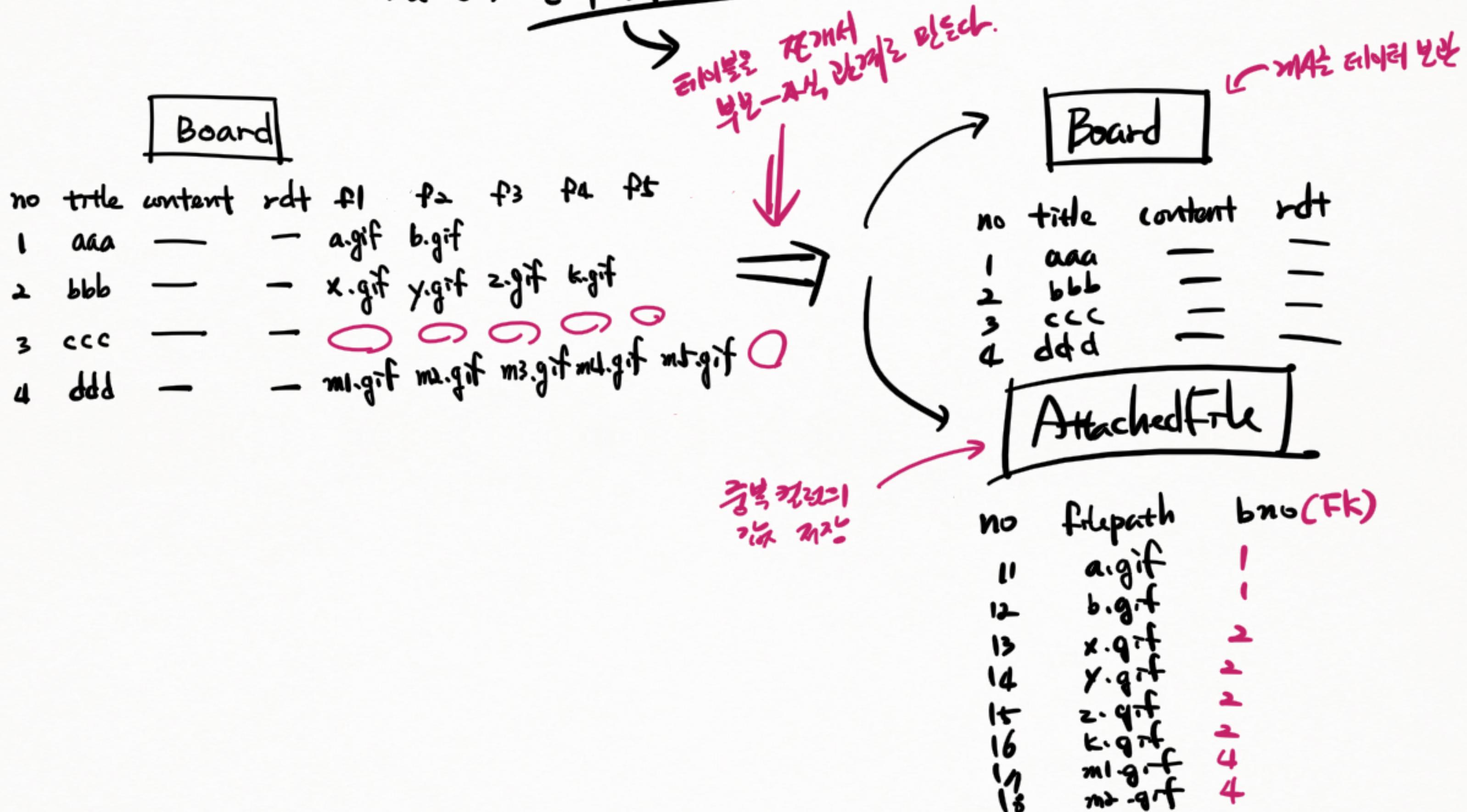
커먼이 부족하여 첨부파일을 더 저장할 수 있다

다음의 위치 발생

이전 커먼 쿵복이나 레이어 쿵복 등에 훔쳐

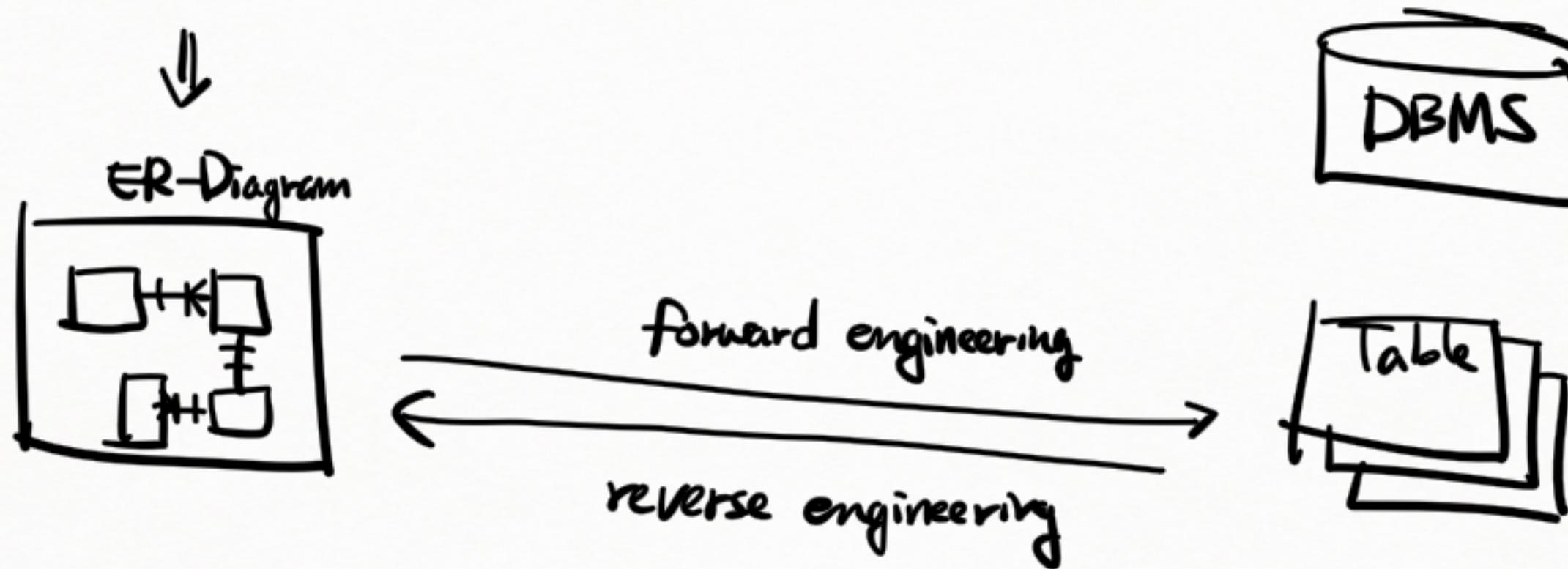
장기화 수행!

## \* 첨부 중복 문제 해결



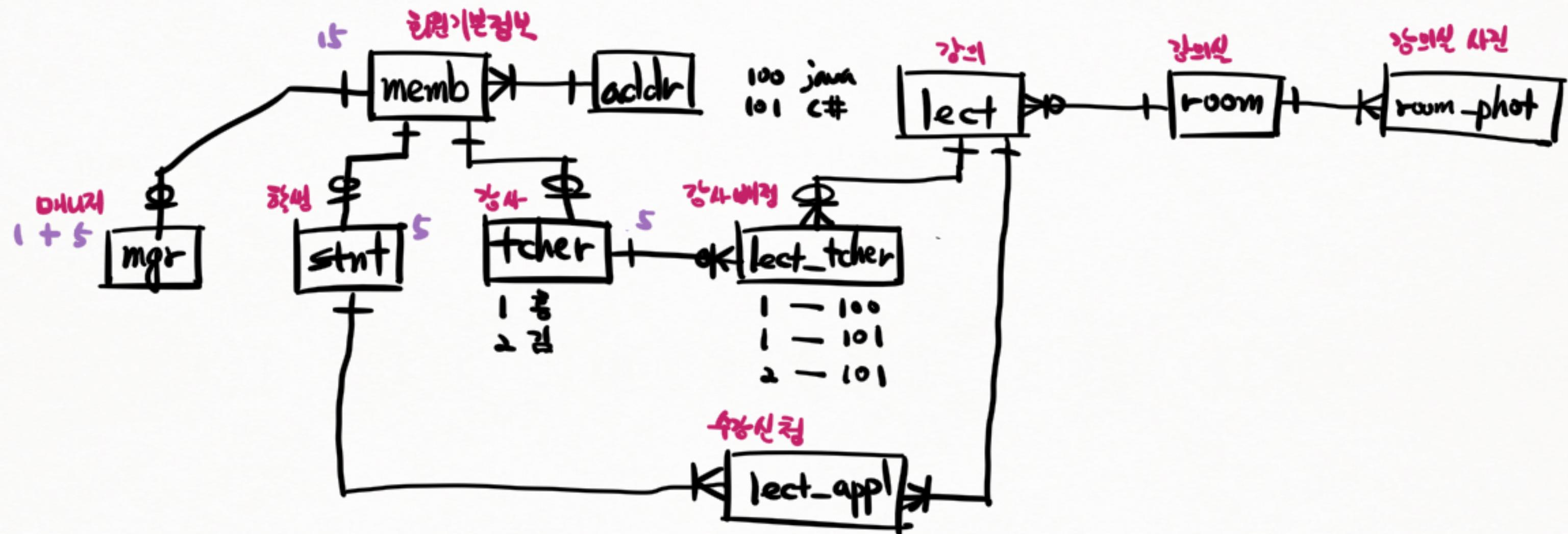
\* ERD : forward / reverse engineering

제작ツ : ER-Win, eXerd, ...



\* SQL 조인 테스트를 위한 데이터의 ERI

예) 교육센터 관리 시스템



\* 조인

↙ 무관 1:1 조인

① cross 조인 (카티션 조인)

A	B
---	---

aaa	111
bbb	222
ccc	



aaa 111

aaa 222

bbb 111

bbb 222

ccc 111

ccc 222

② natural 조인

↑ 같은 이름의 키를 같은  
기준으로 조인

Board
-------

bno title

1	aaa
2	bbb
3	ccc
4	ddd

AttachedFile
--------------

fno	path	bno
11	a.gif	1
12	b.gif	1
13	c.gif	3
14	d.gif	4

Board
-------

no	title
1	aaa
2	bbb
3	ccc
4	ddd

AttachedFile
--------------

no	path	bno
11	a.gif	1
12	b.gif	1
13	c.gif	3
14	d.gif	4

1	aaa	11	a.gif	1
1	aaa	12	b.gif	1
3	ccc	13	c.gif	3
4	ddd	14	d.gif	4



{ ✓ 조인의 기준이 될 키가 암시하지 않은 경우  
{ ✓ 서로 상반되는 키의 이름이 있는 경우

FK 키의 이름

FK + 가리키는  
PK 키의 이름

↳ 올바른 조인이 실현되지 못한다

\* 왜 데이터를 가져올 때 여러 테이블의 데이터를  
엮어서 가져온가?

↓  
데이터 중복을 피하기 위해  
데이터가 여러 테이블에  
포함되어 있기 때문이다.

Project		
번호	제목	책장번호
1	aaa	kim 010-1111-1112
2	bbb	kim 010-1111-1112
3	ccc	kim 010-1111-1111

번호 제목 책장번호

1 aaa kim 010-1111-1112

2 bbb kim 010-1111-1112

3 ccc kim 010-1111-1111

↑  
데이터가 중복

- 변경이 번거롭다
- 번경항목을 누락할 수 있다

결과 = 데이터  
중복성이  
생긴다

데이터 중복 문제

해결

데이터  
복제

Project		
번호	제목	책장번호
1	aaa	aaa

번호 제목 책장번호

1 aaa |

2 bbb |

User		
번호	이름	전화
1	kim	010-1111-1112
2	park	010-1111-2222
3	lee	010-1111-3333

번호 이름 전화

1 kim 010-1111-1112

2 park 010-1111-2222

3 lee 010-1111-3333

같은 책장인데  
전화번호  
아르다?

\* Natural 조인을 수행할 기본 조건의 이론이  
일치하지 않거나 엉뚱한 조건과 일치할 경우

③ join on ~ 문법 사용

Board join AttachedFile  
on Board.no = AttachedFile.bno  
조인 조건

\* Natural 조인을 수행할 때

기준컬럼이 이름이 같은 것뿐 아니라 여러 개 있을 경우

Board	
bno	title
1	aaa
2	bbb

AttachedFile	
no	name
11	x.a.gif
12	x.b.gif
13	c.gif

no	name	bno
11	x.a.gif	1
12	x.b.gif	1
13	c.gif	2



Board join AttachedFile  
using (bno)

↑  
조건을能满足하는  
기준컬럼은  
명시적으로 지정한다.

\* 조인을 수행할 때 테이터가 누락되는 상황

Lecture				Teacher			
no	name	tno(fk)	(pk)tno		name		
11	aaa	1		1	kim		
12	bbb	2		2	park		
13	ccc	1		3	lee		
14	ddd			4	eom		
15	eee						

Lecture right outer join Teacher  
on Lecture.tno = Teacher.tno

11	aaa	1	1	kim
12	bbb	2	2	park
13	ccc	1	1	kim
null	null	null	3	lee
			4	eom

## ① Natural Join

Lecture join Teacher using (tno)

11	aaa	1	1	kim
12	bbb	2	2	park
13	ccc	1	1	kim

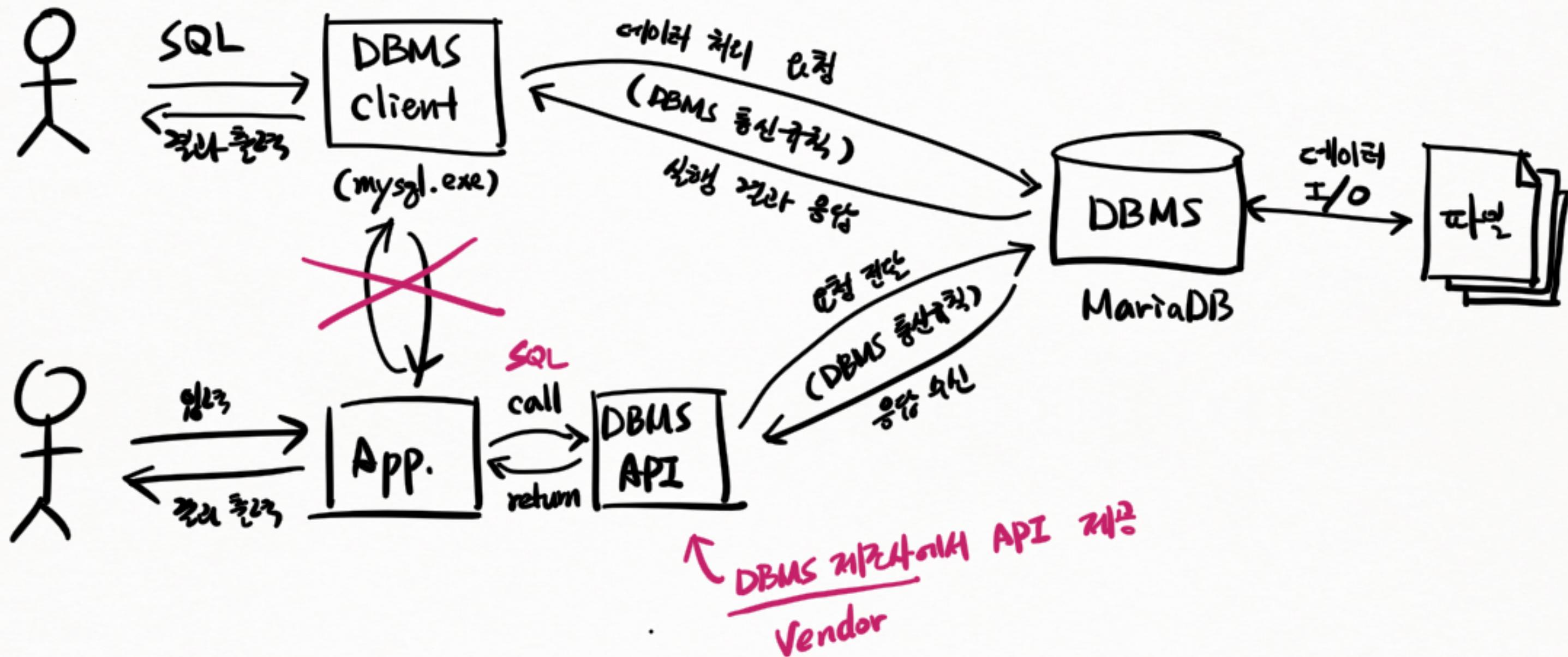
→ 조인할 때 데이터의 티아리가 없으면  
결과에서 누락되는 문제 발생

## ② Outer Join ↴

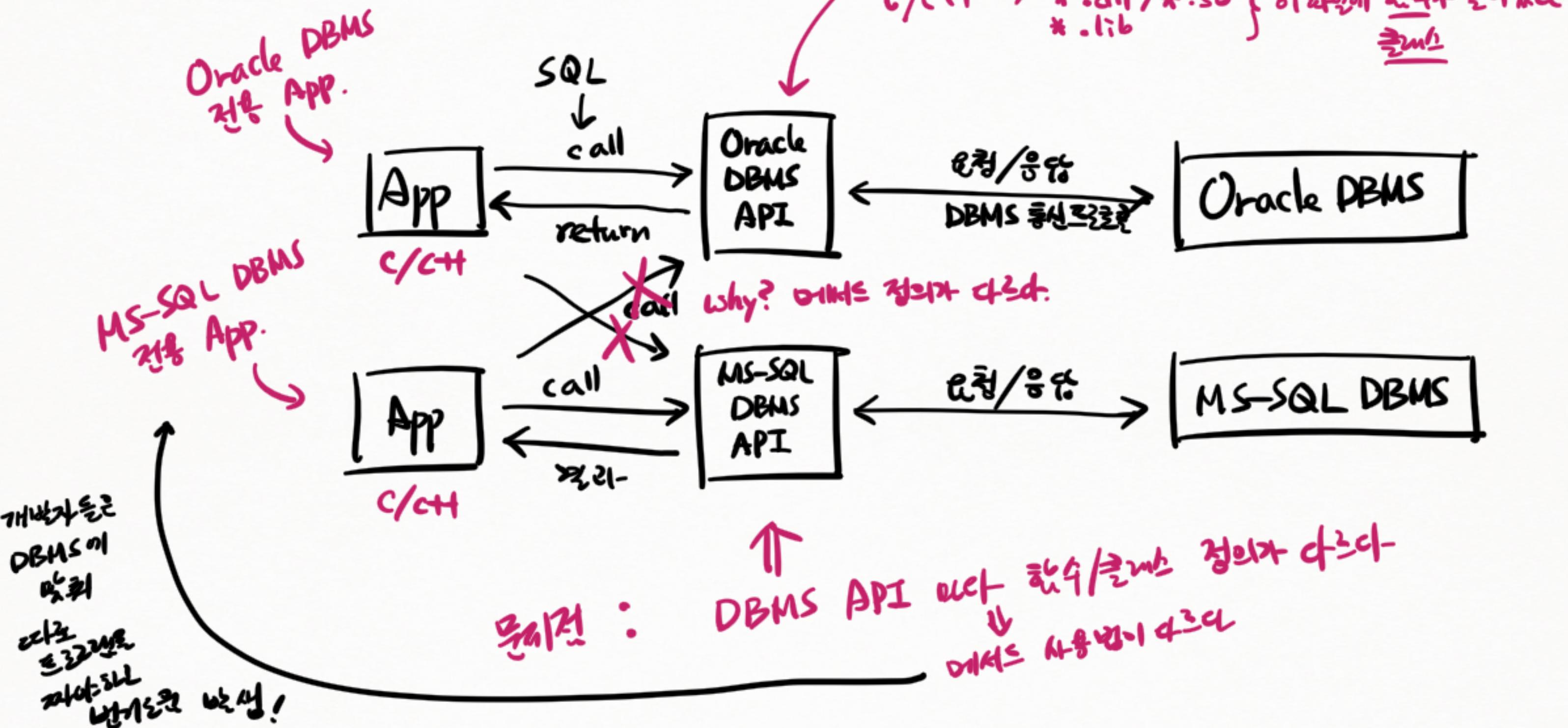
\* Lecture left outer join Teacher  
on Lecture.tno = Teacher.tno

11	aaa	1	1	kim
12	bbb	2	2	park
13	ccc	1	1	kim
14	ddd		null	null
15	eee		null	null

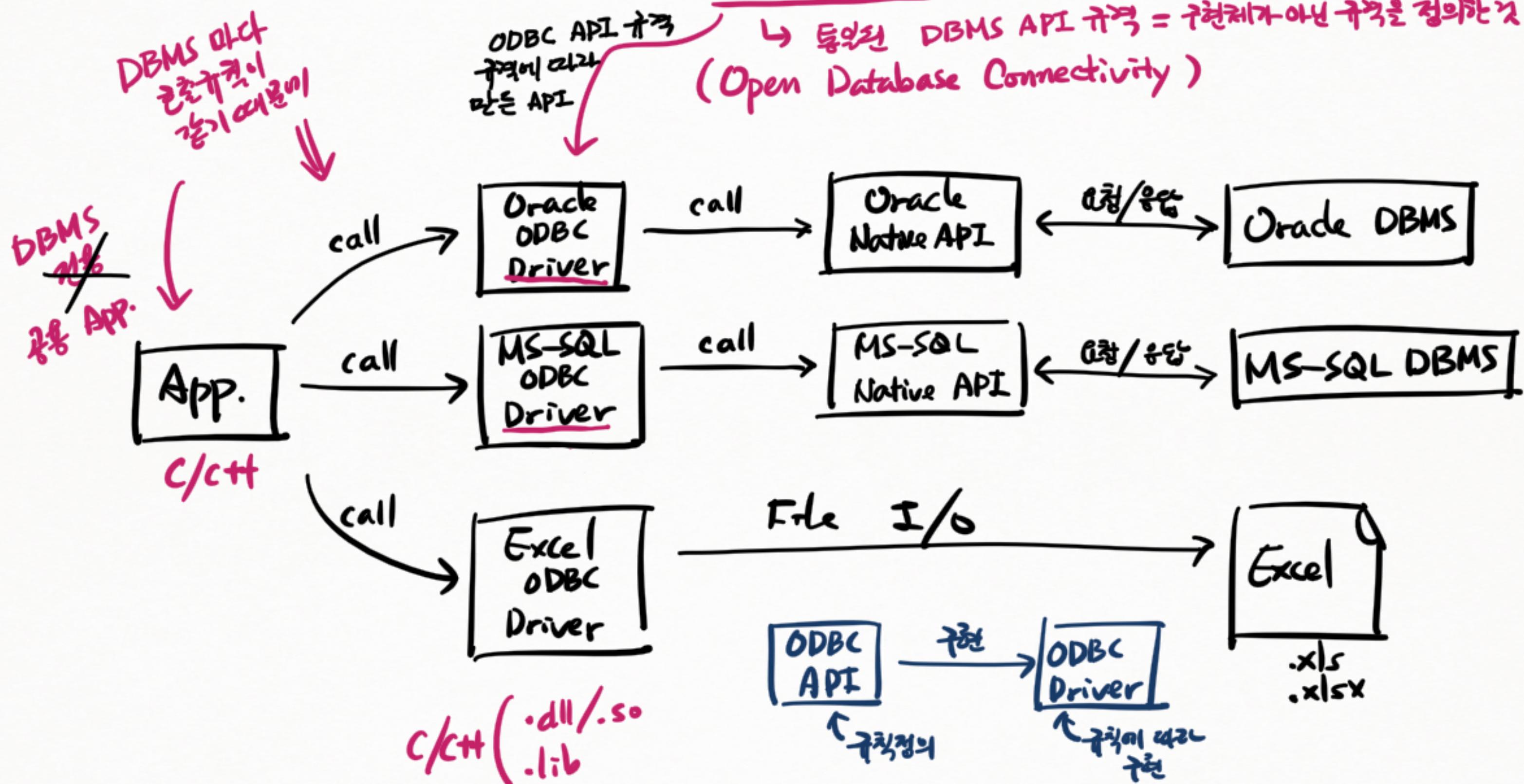
## \* Application & DBMS

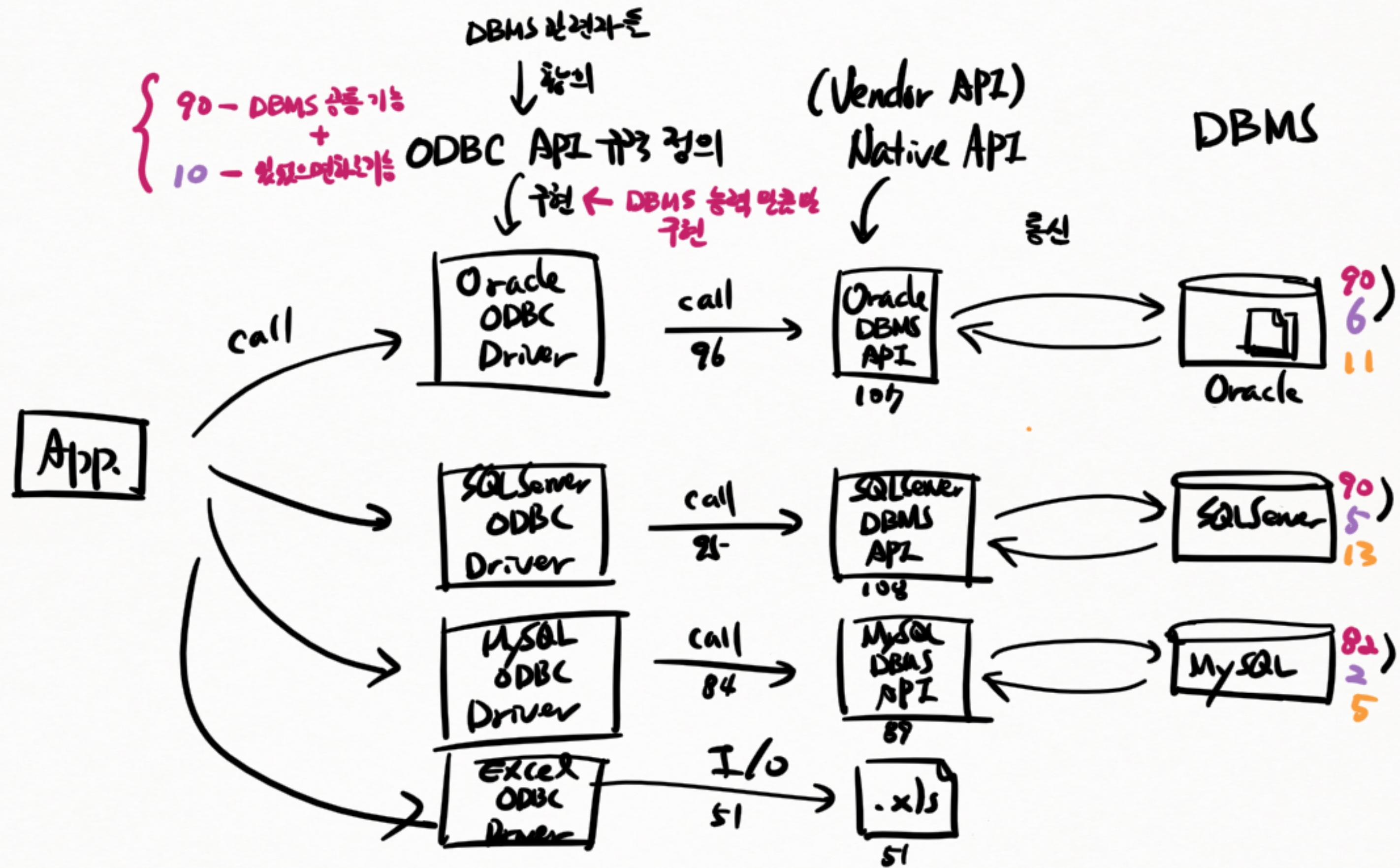


## \* DBMS API - Native API (= Vendor API)

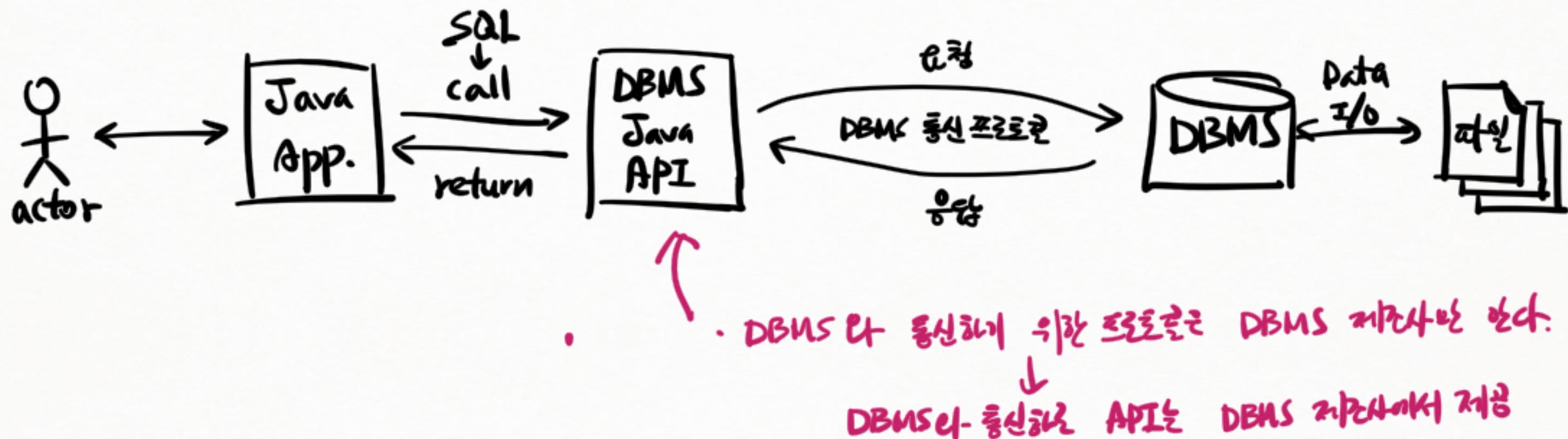


## \* DBMS API - ODBC API 구조





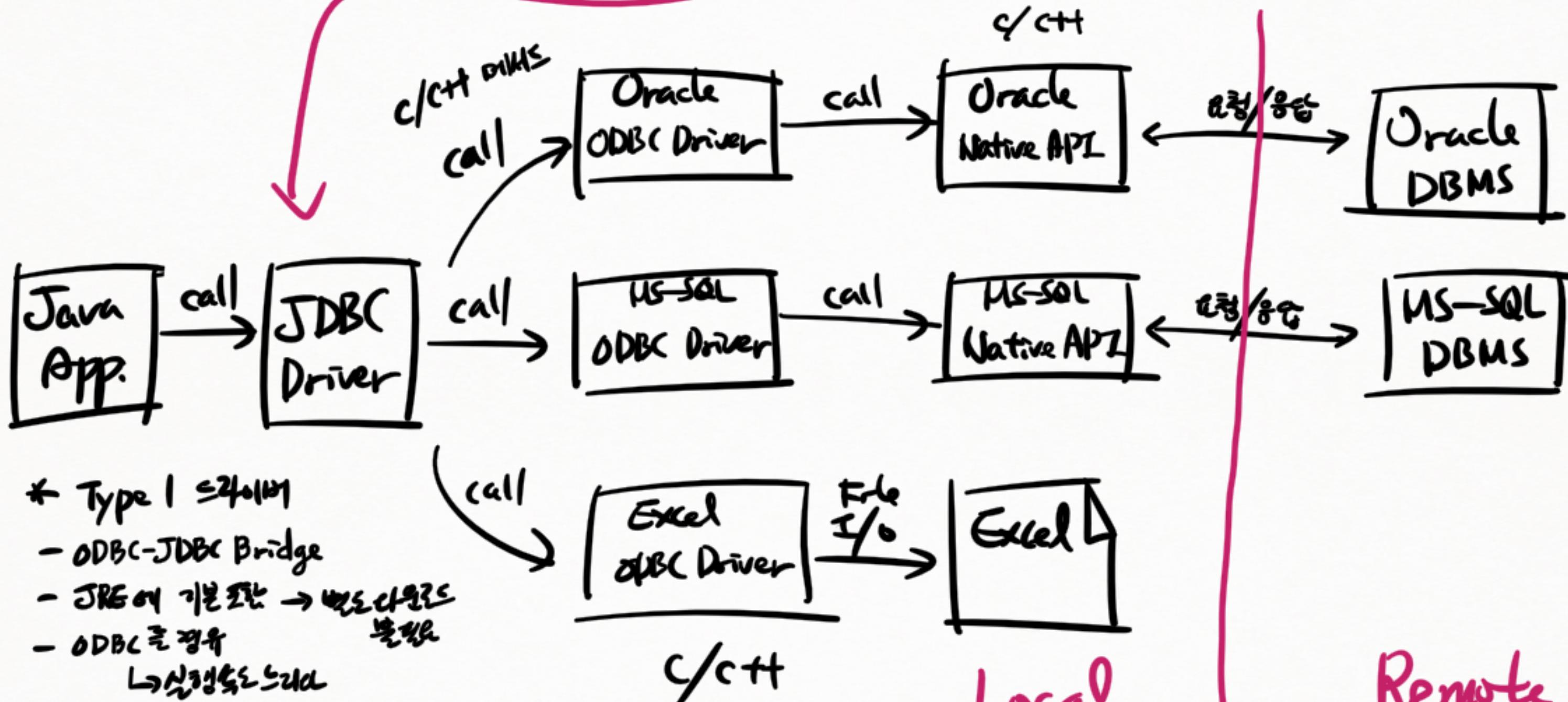
## \* Java 와 DBMS API



## \* DBMS API - JDBC API ↗

JDBC API 구조가 아니라 구현한  
클래스 라이브러리

↳ Java Database Connectivity API 구조



### \* Type I 드라이버

- ODBC-JDBC Bridge
- JRE에 기본 포함 → 별도 다운로드 불필요
- ODBC를 정유

↳ 실행되는 노드

- 다른 JDBC 전용 드라이버 - 예외
  - ODBC 드라이버는 알고 있으니 유익한 예) 엑셀, CSV 등

Local

Remote

## \* DBMS API - JDBC API 之間

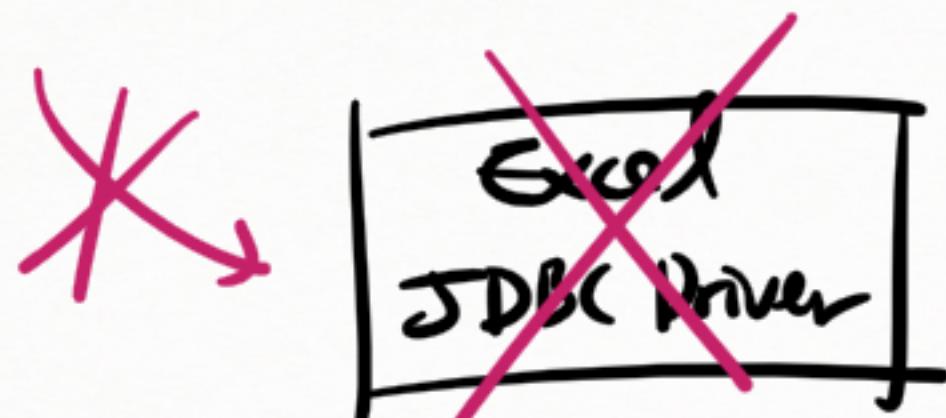
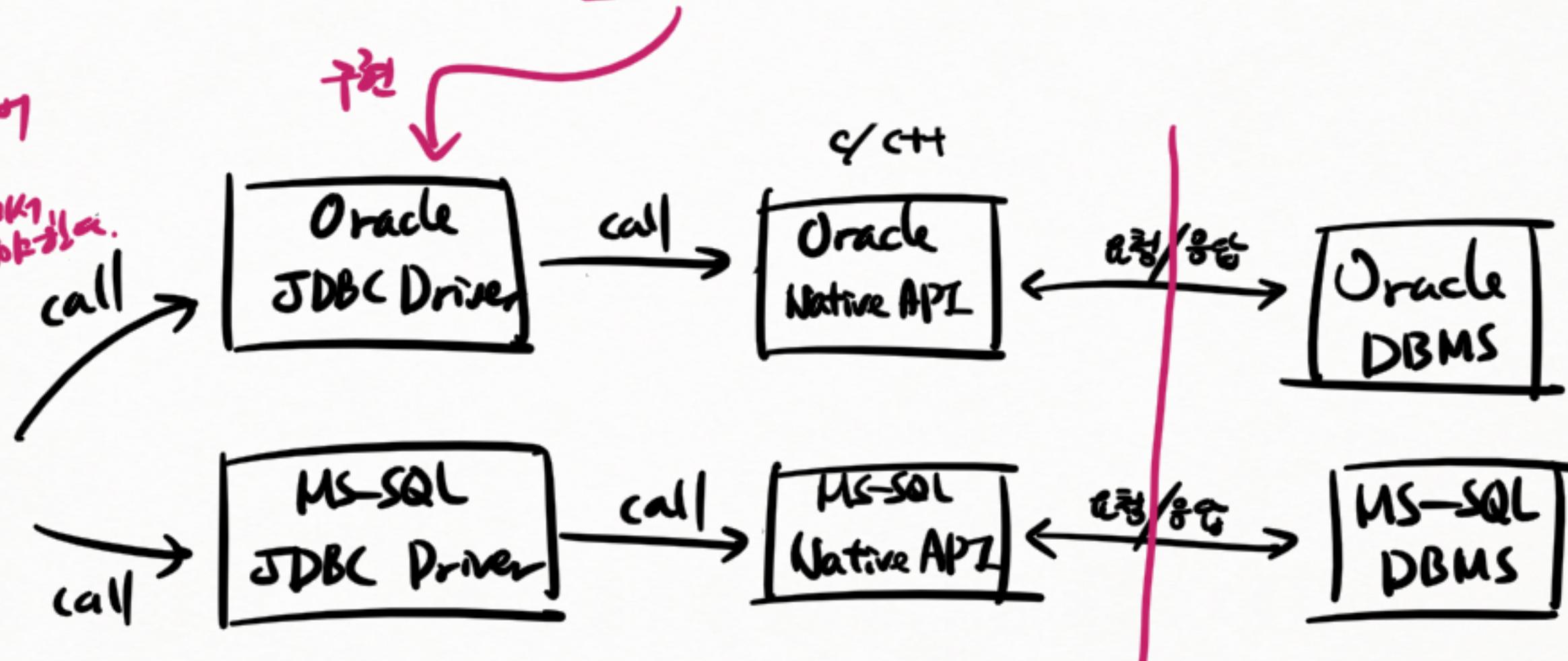
\* Type 2 ≤ 2016

- Native API ≤ 2016

- JDBC API 亂用 안됨

↳ DBMS Vendor API  
별도로 구현되는 드라이버

- 3rd party Native API  
제3자에게서 제공



Type 1 ≤ 2016  
Againts 3rd party

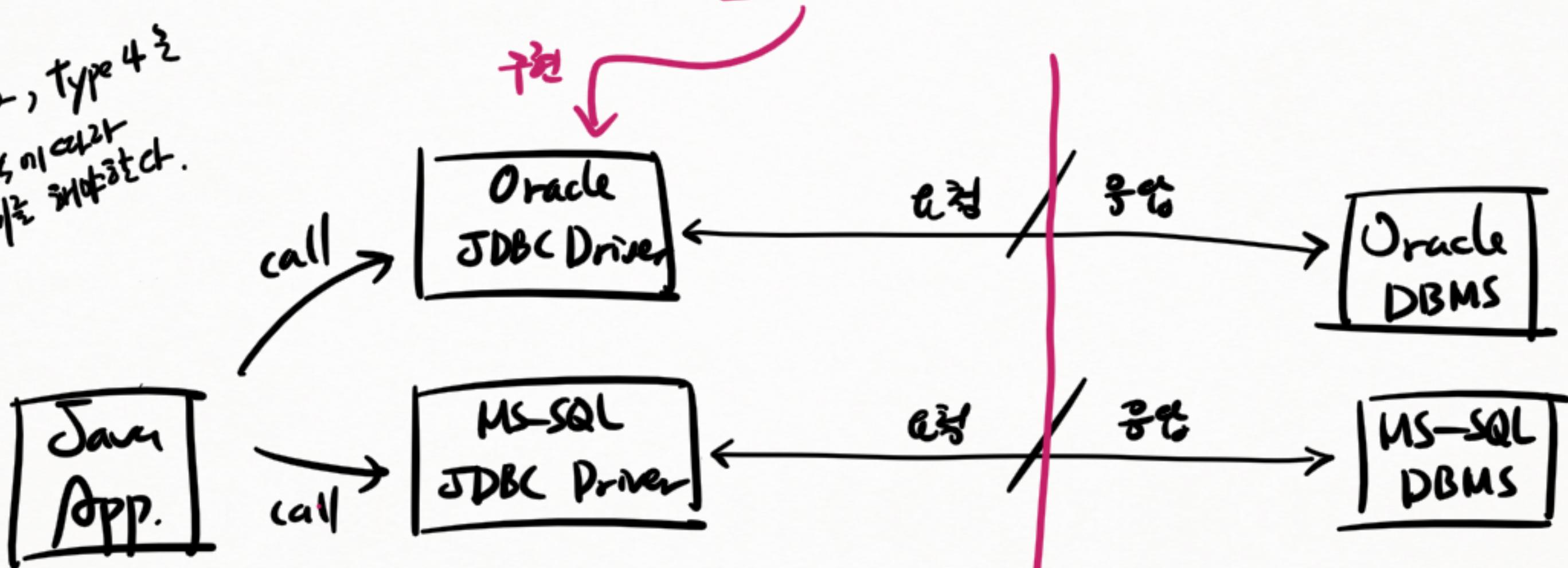


Local

Remote

## \* DBMS API - JDBC API 之間

\* 문제점  
- Type 2, Type 4  
DBMS의 API를  
고려해 해야 한다.



### \* Type 4 드라이버

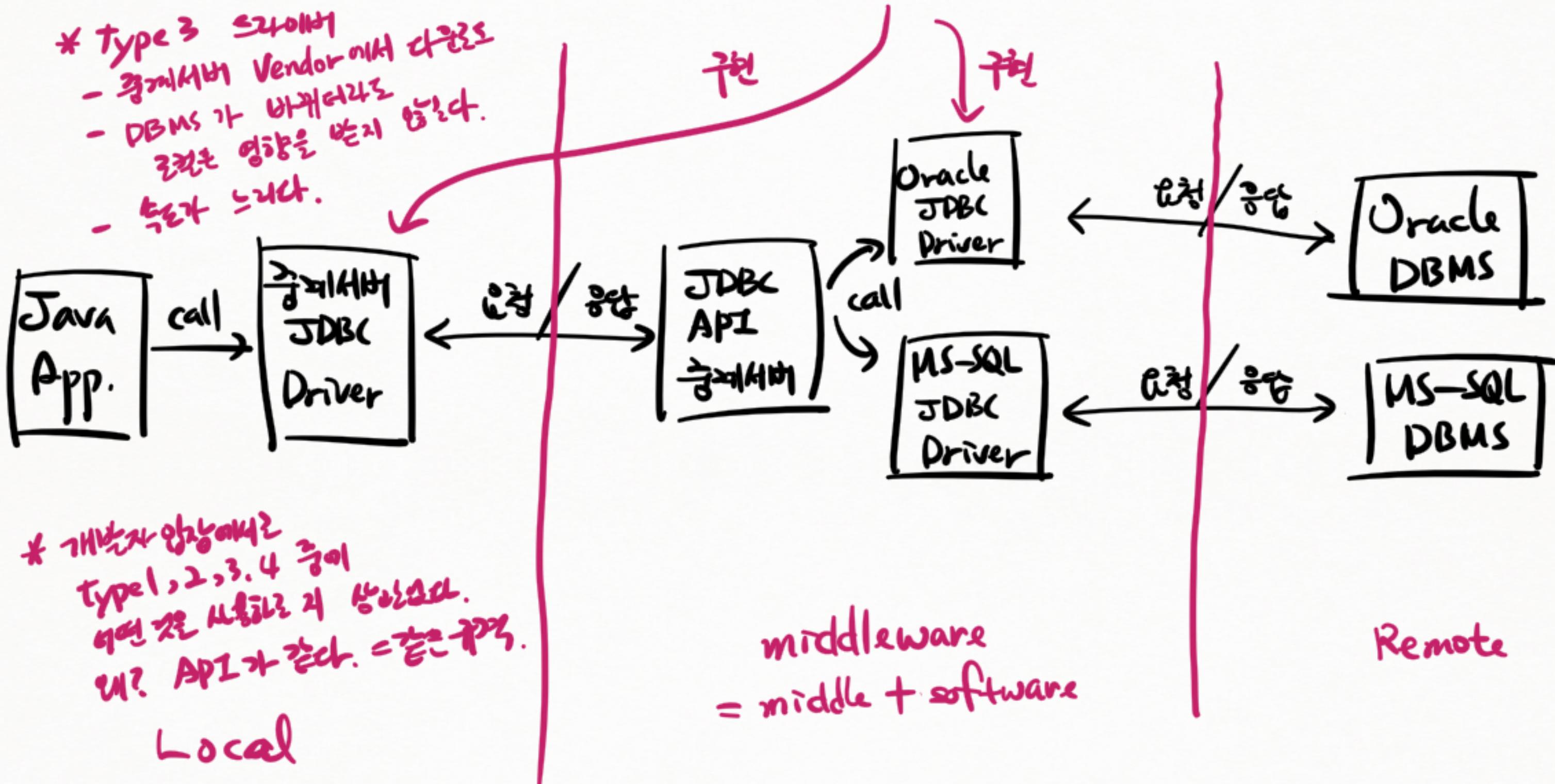
- Network protocol 드라이버
- DBMS Vendor에서 별도 드라이버 필요없다.
- C/C++ 혹은 X → pure Java 드라이버
- 조건지 C/C++ 라이브러리 설치 불필요!

Local

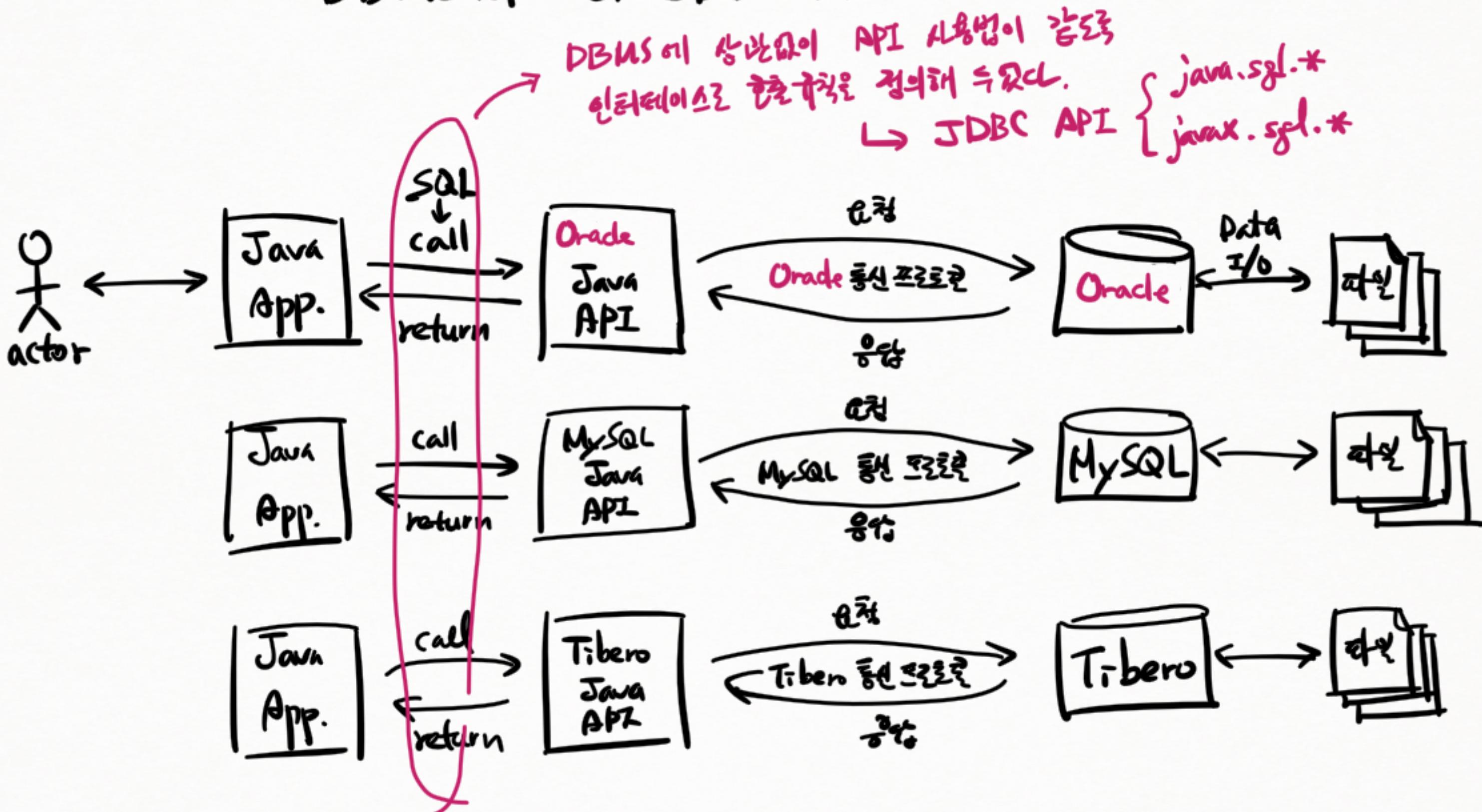
Remote

## \* DBMS API - JDBC API ↗

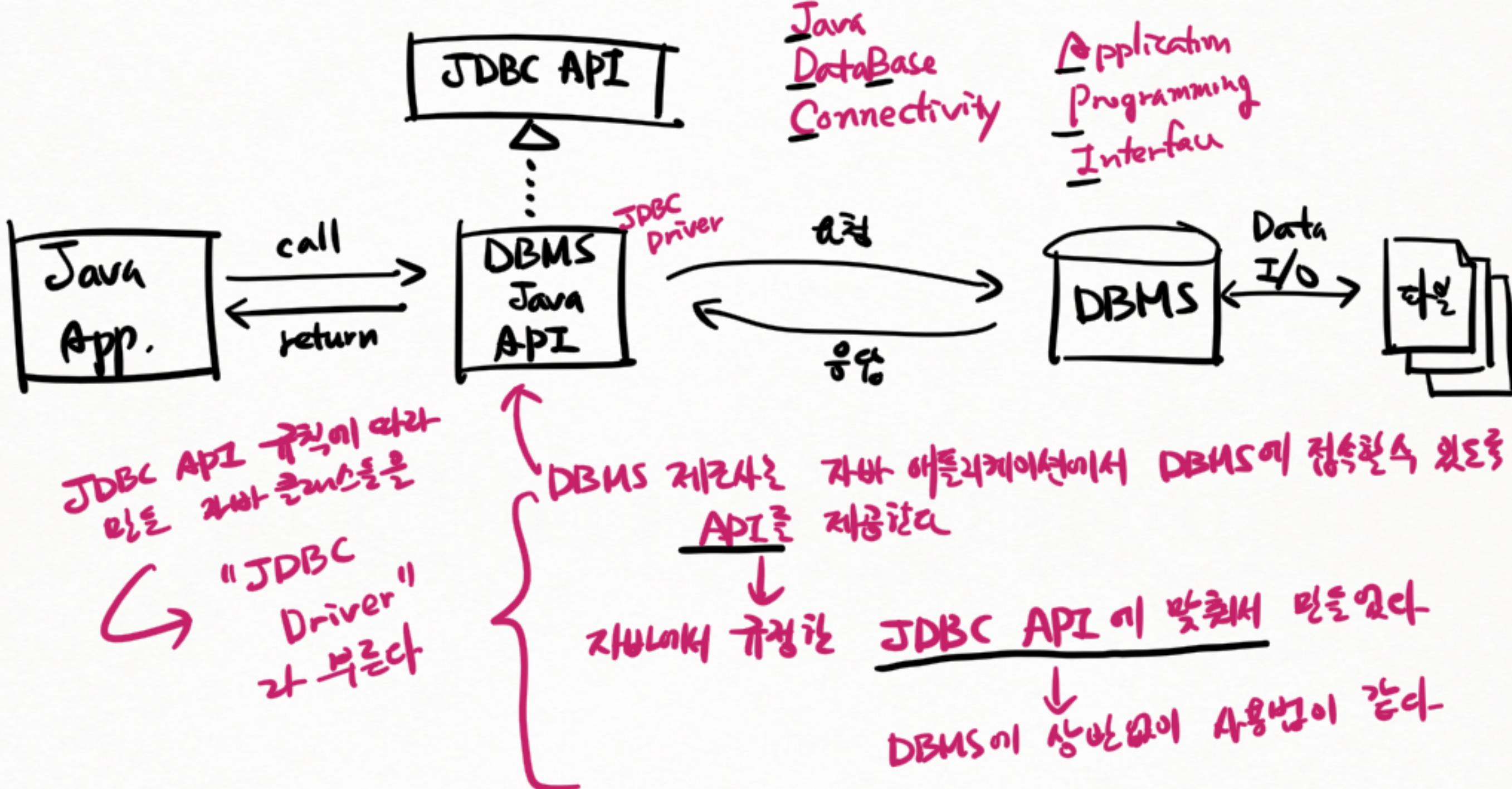
- \* Type 3 드라이버
  - 중재서버 Vendor에서 제공됨
  - DBMS가 바뀌더라도  
조직은 영향을 받지 않는다.
  - 속도가 느리다.



## \* DBMS API 와 JDBC API



## \* JDBC API 와 JDBC Driver의 관계



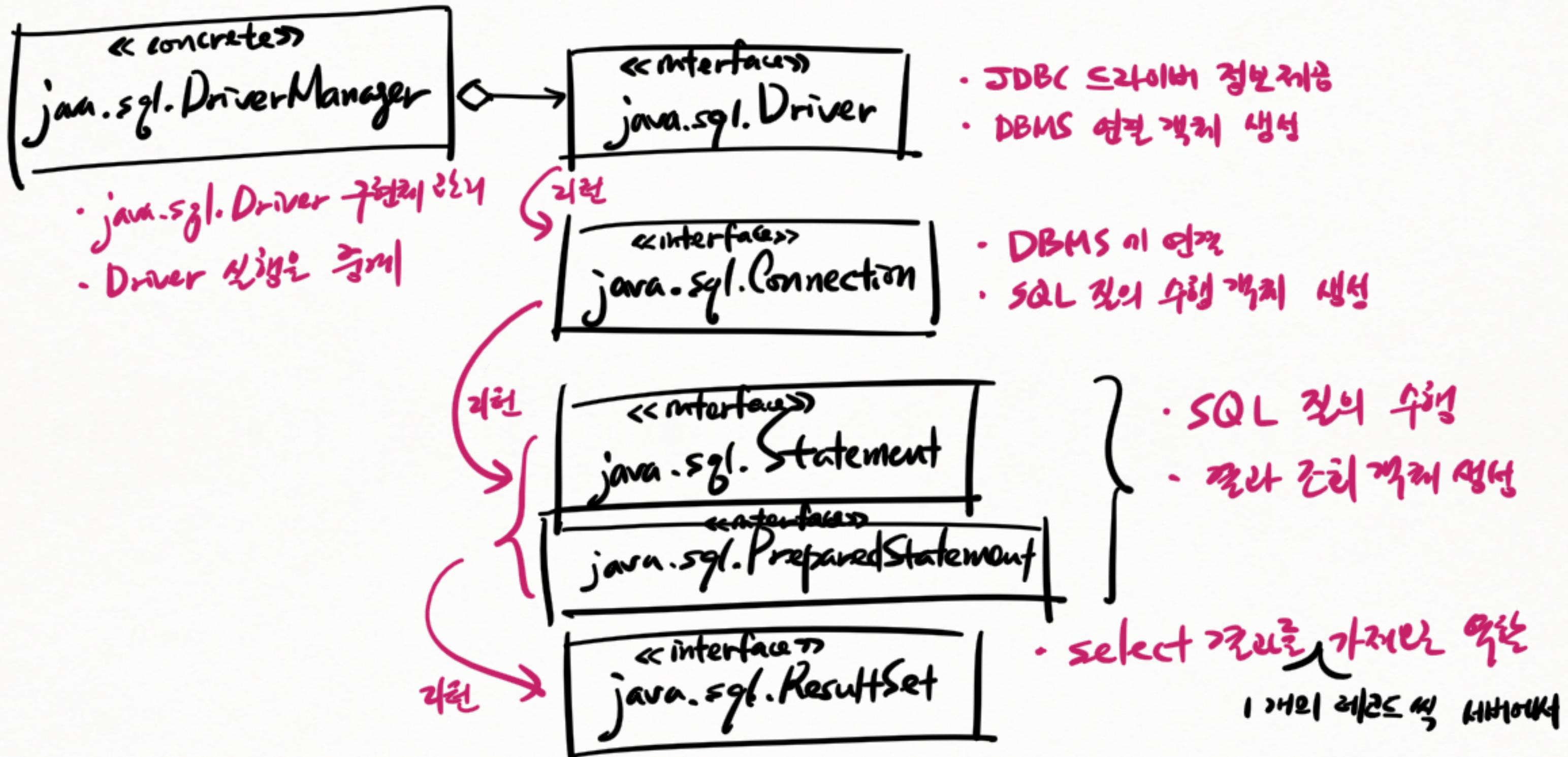
## \* JDBC 쓰는법

- ✓ API 구조에 따라 JDBC API 사용법이 따라  
     JDBC 드라이버로  
     DBMS 서버로 접속하여  
     데이터를 처리하는 것!
- ✓ 그 JDBC 드라이버  
     DBMS와 같은  
     SQL을 전송하여  
     처리해 처리하여
- ✓ JDBC 드라이버  
     DBMS와 같은  
     SQL을 전송하여  
     처리해 처리하여

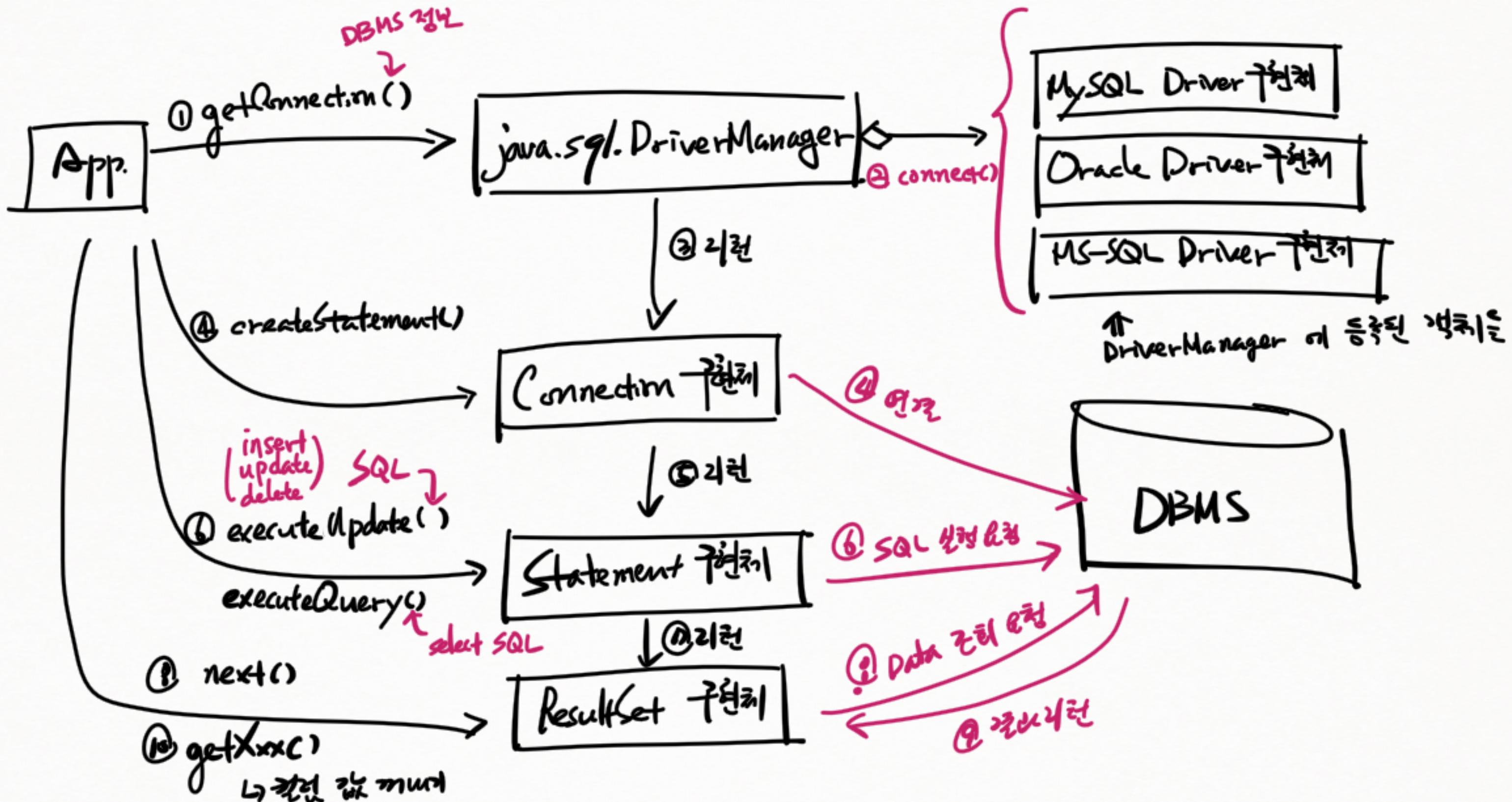
\* JDBC 쓰는법 준비

- ① MariaDB JDBC 드라이버 준비  
(Type4)
- ② JDBC API 규칙이 따라 머서드 훈련

## \* JDBC API 구조 개요



## \* JDBC 제작 과정 기반 구조



## \* DAO 및 VO



## \* Statement vs PreparedStatement

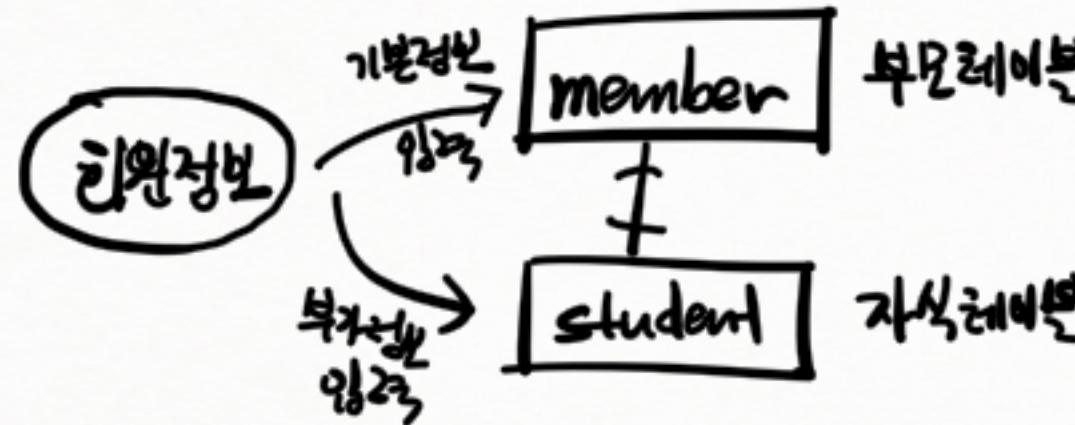
	Statement	PreparedStatement
간접적인 SQL	X	O
SQL 허용하지 않는 DB	O	X
유저에게 데이터를 보여주기	X	O
반복 실행할 때	slow	fast

## \* Transaction

↳ 여러 개의 데이터 변경 작업을 한 단위로 묶은 것

(insert, update, delete)

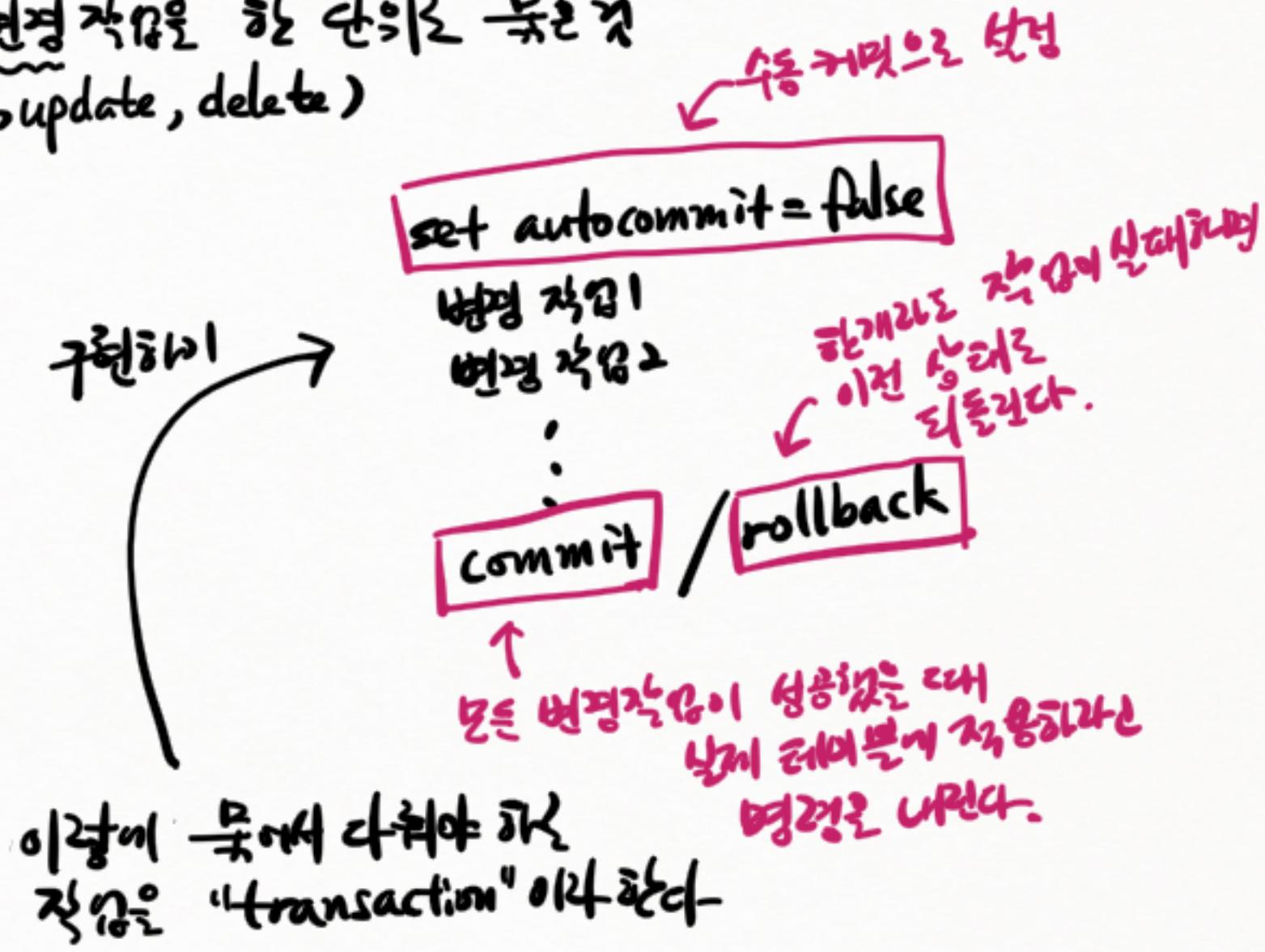
예1) 회원 정보 입력



회원정보를 2개의 테이블에  
분산해서 저장할 때,

2개 테이블에 모두  
정상적으로 데이터가 저장되어야만  
유 효하다.

둘 중 한 개라도 일치하지 않으면  
이전에 일어난 모든 데이터가 취소-



## 예2) 쓰루 구매하기

Set autocommit = false

한 작업으로  
마친다 { 모든 테이블에 주소정보 넣기  
각각 테이블에 결제정보 넣기  
배송 요청 테이블 배송 요청 정보 넣기 }

commit

transaction

→ 다른 테이블

한 개의  
작업

→ 테이블

✓ 자동차 번호가입 (학번등록 → 번호등록 → 결제등록)

✓ 수강신청 (수강신청 정보등록 → 결제등록)

↓  
실제 기업의 업무는 마지막의 결과와 같이  
여러 개의 작업을 한 단위로 다수로 묶어 많다.

### 예3) 멤버 할당

set autocommit=false ← 트랜잭션 시작

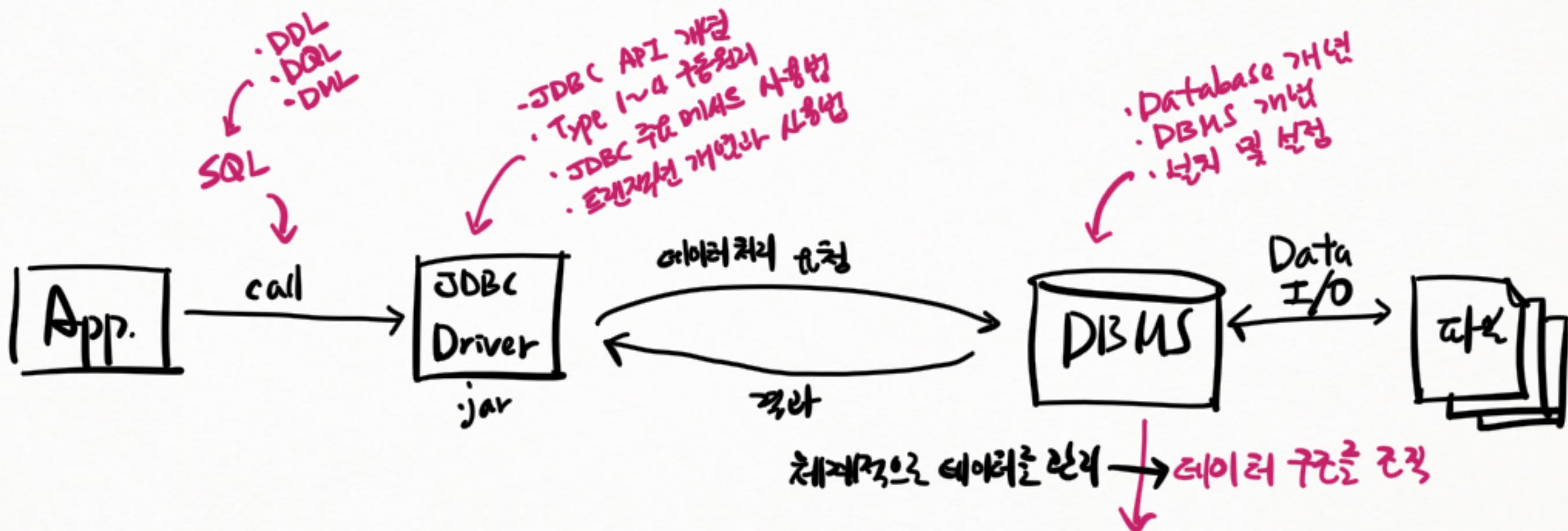
} update → 회원 정보 테이블에서 할당회원의 기본정보를 인의의 값으로 변경하고 비밀번호로 보정한다

트랜잭션 } delete → 회원에 할당된 책임정보 삭제

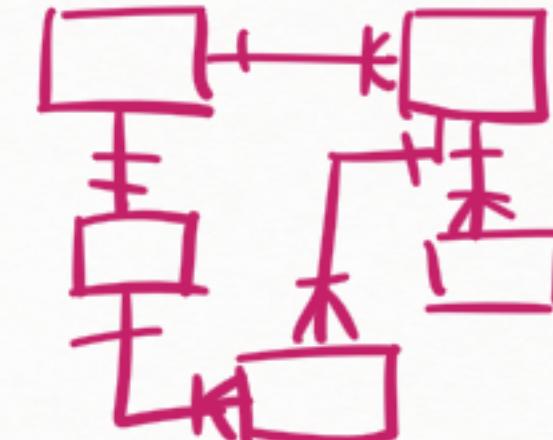
delete → 프로젝트 멤버 목록에서 제거

commit ← 트랜잭션 종료

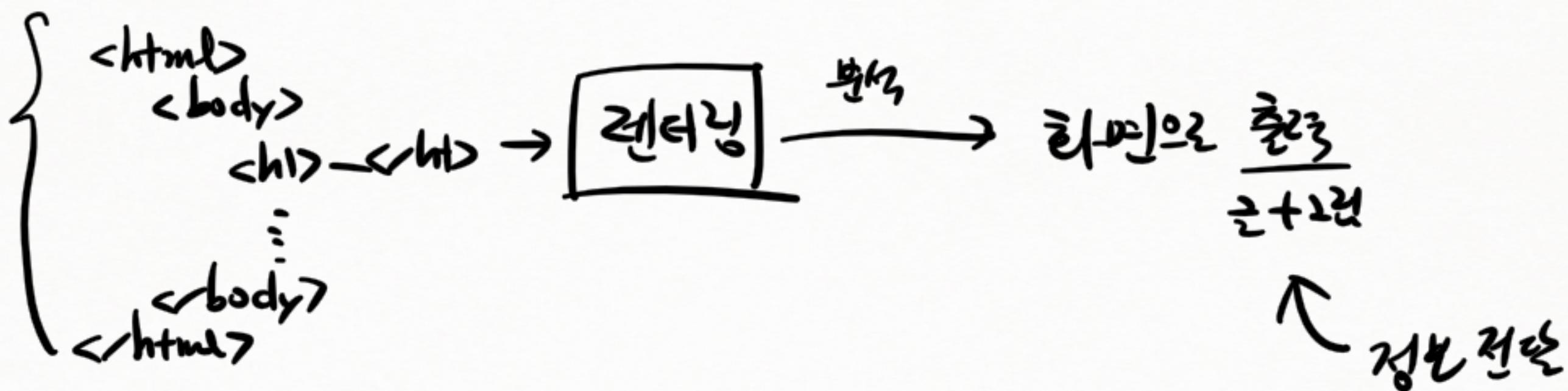
## \* DBMS 학습 기초



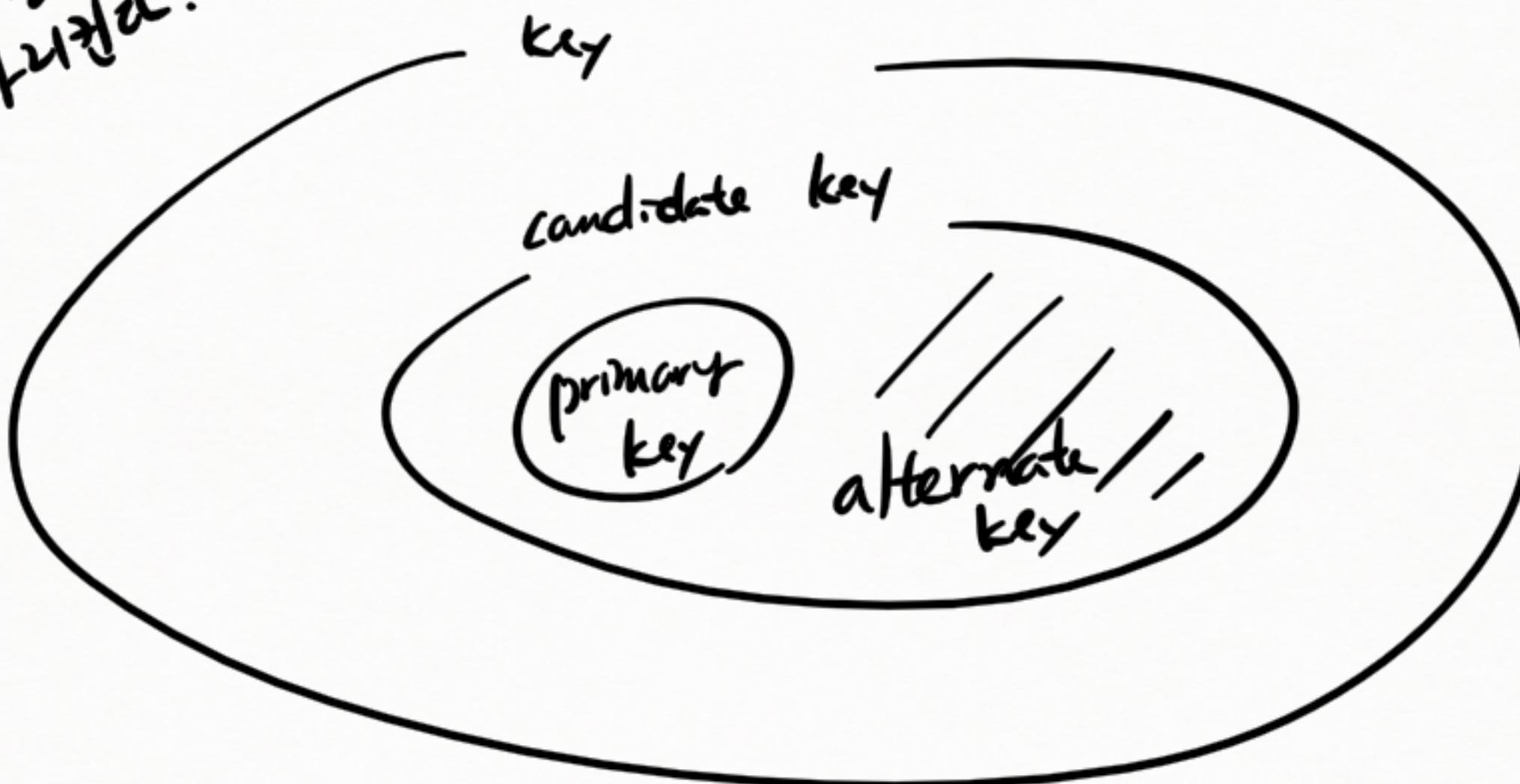
\* 데이터가 충분 저장되지 않기  
 데이터 구조를 조직하는 방법 ⇒  
 ↑  
"DB 모델링"



## \* DB 온톨로지



\* key



+ Artificial key  
"surrogate key"

## \* DB 모델링 목차

### ① 논리 모델 정의

↳ 일상적인 개념으로 표현

1) 엔티티 속성 및 관련 속성

2) PK 선정 및 인증이 되는

중복데이터  
제거하고  
유지법  
3) 제1 정규화 → 데이터 중복 및 관련 중복제거

4) 제2 정규화 → 여러 PK가 중복되거나 없는 데이터 속성

5) 제3 정규화 → 일부 결합된 중복된 데이터 속성

6) 다 대다 관계 헤스 / 포함관계, 세그먼트화 속성

7) 관계 차수 차정

8) 대안기 → 유니크로 차정

9) 결합의 유연성을 설정

### ② 물리 모델 정의 → ③ SQL 생성

↳ DBMS가 맞춰서 표현

1) DBMS 구조에 따라  
데이터 및 접근방법 설정

로우드  
엔진어닝

2) 트리거 정의 및 적용

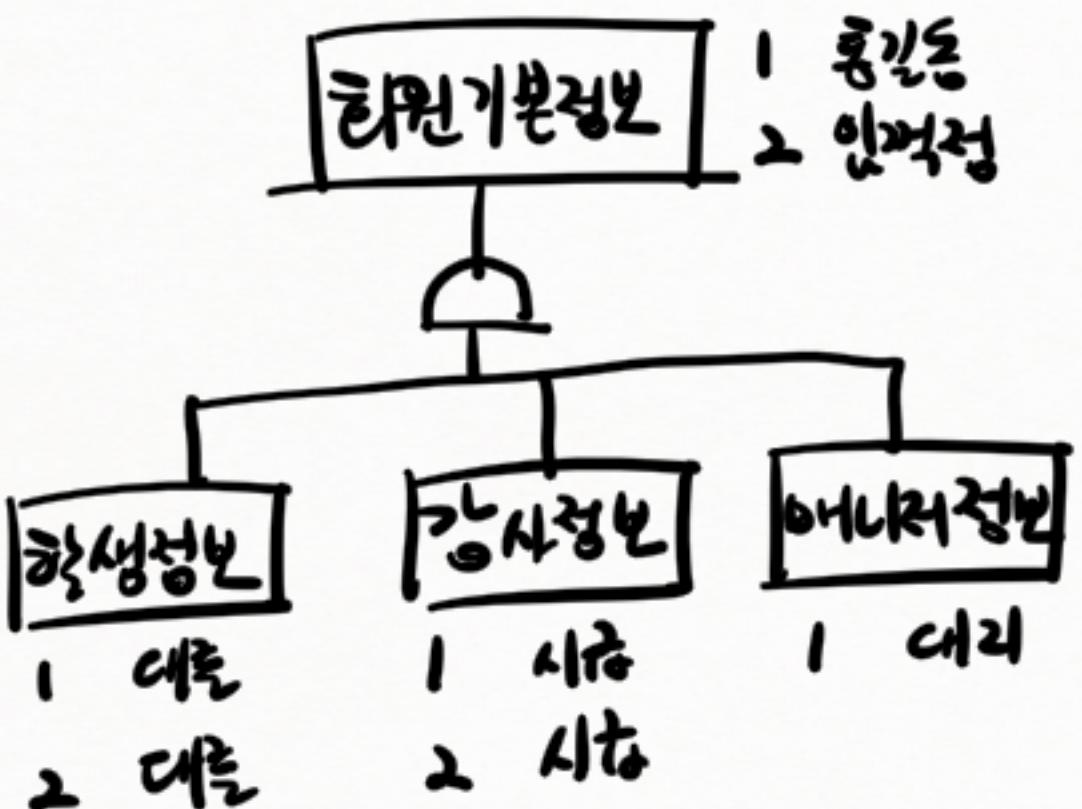
3) 자동 증가 관련 지정

4) 기본값 및 제약조건

## \* 포함 관계와 배제적 관계

### ① 포함 관계 (include)

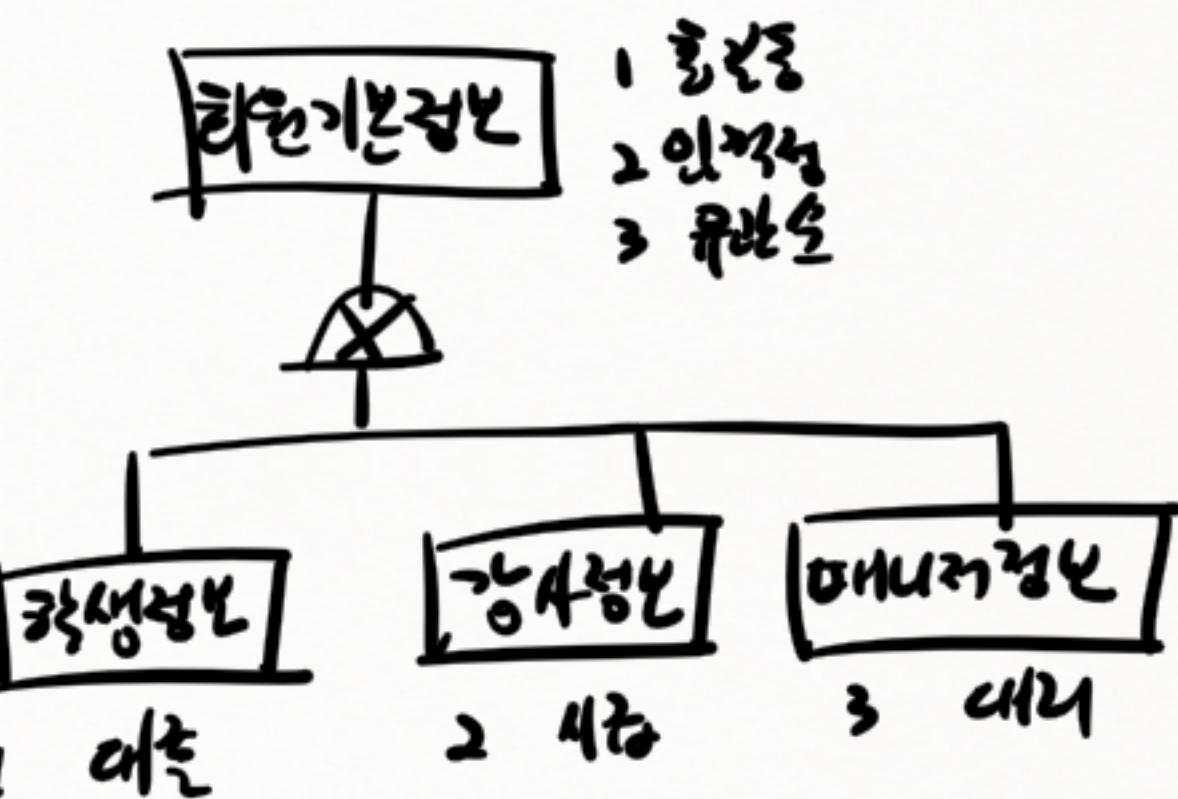
↳ 여러 개를 결합할 수 있다



\* 학원정보가 여러 레이블이 포함될 수 있다

### ② 배제적 관계(exclude)

↳ 선종기 단계, 라디오채널  
콘텐츠와 일정

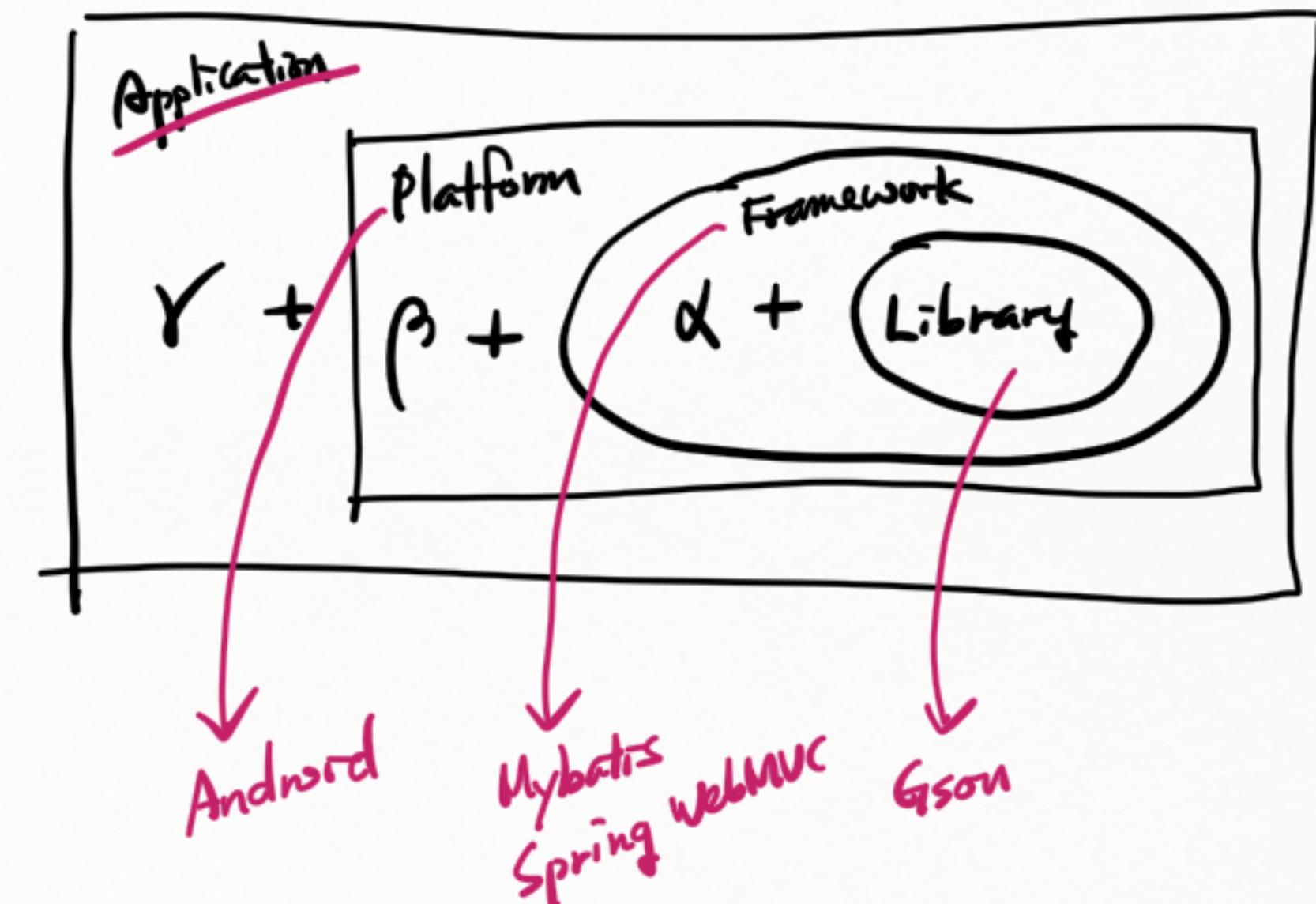
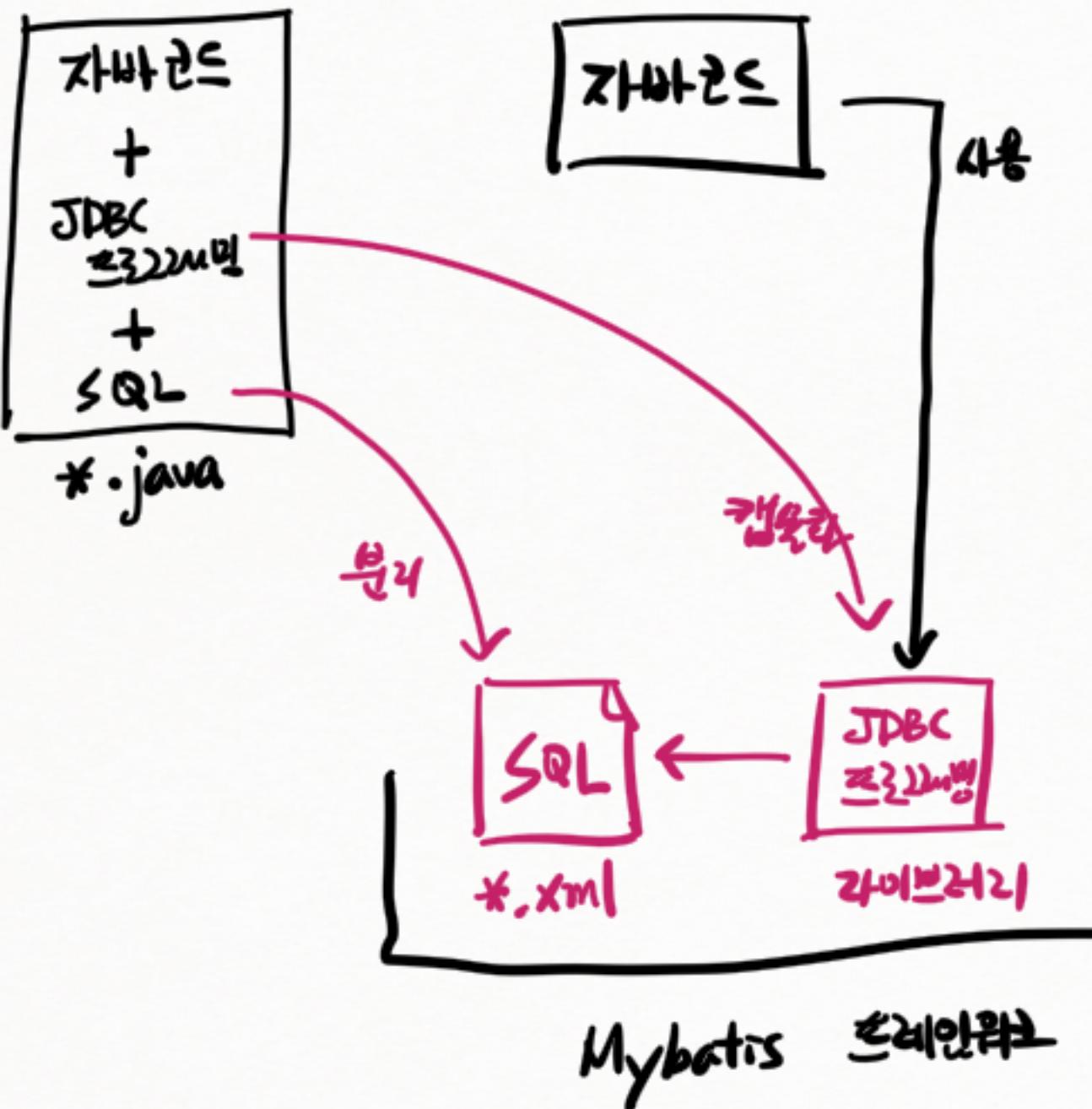


\* 학원정보가 3 개의 레이블 중  
언제 2 개의 레이블이  
선택되는 것을 수 있다.

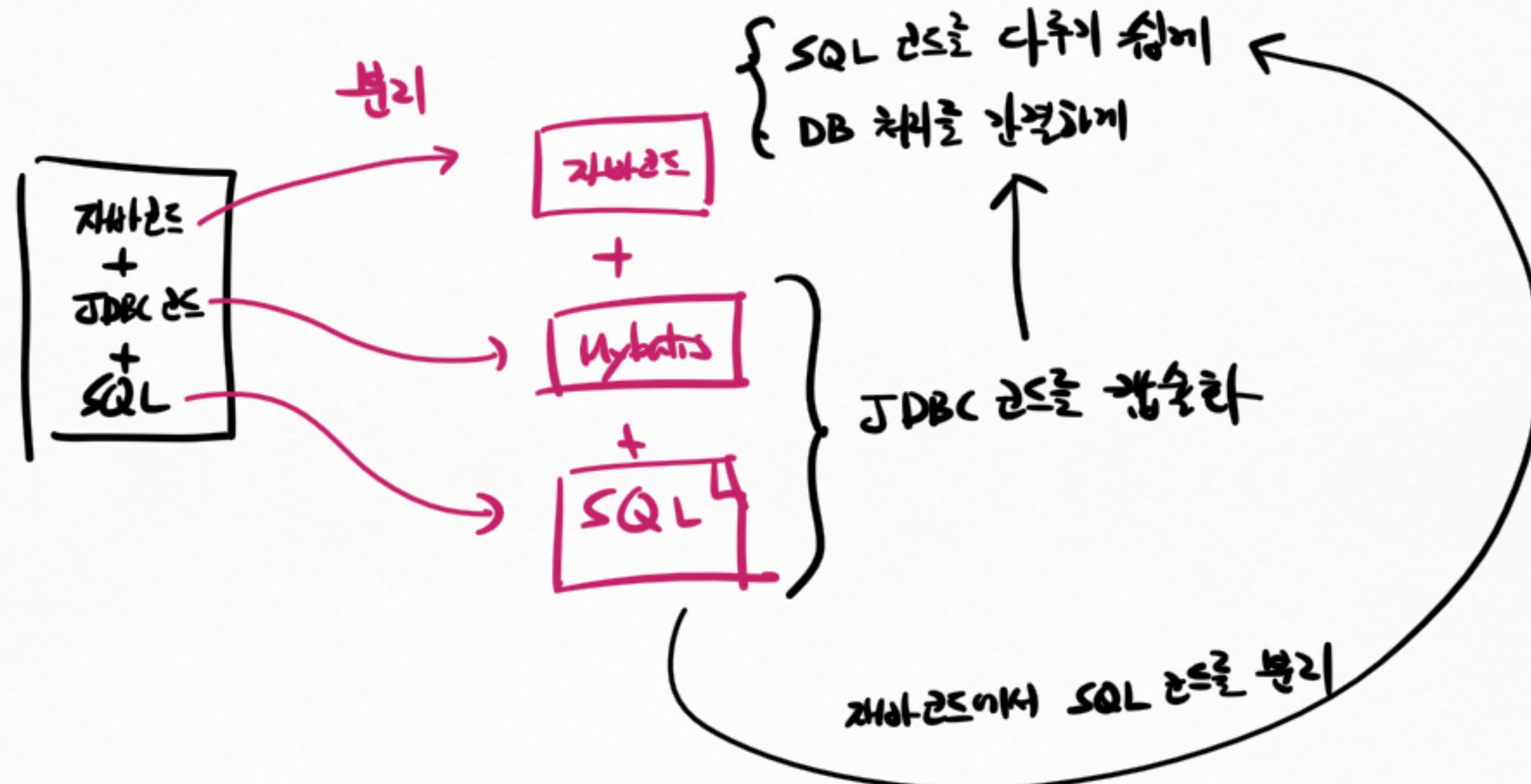
# \* Mybatis Persistence Framework

지속성

기존방식 → 별도(유지보수↑)



## \* Mybatis 드라이버 와 그

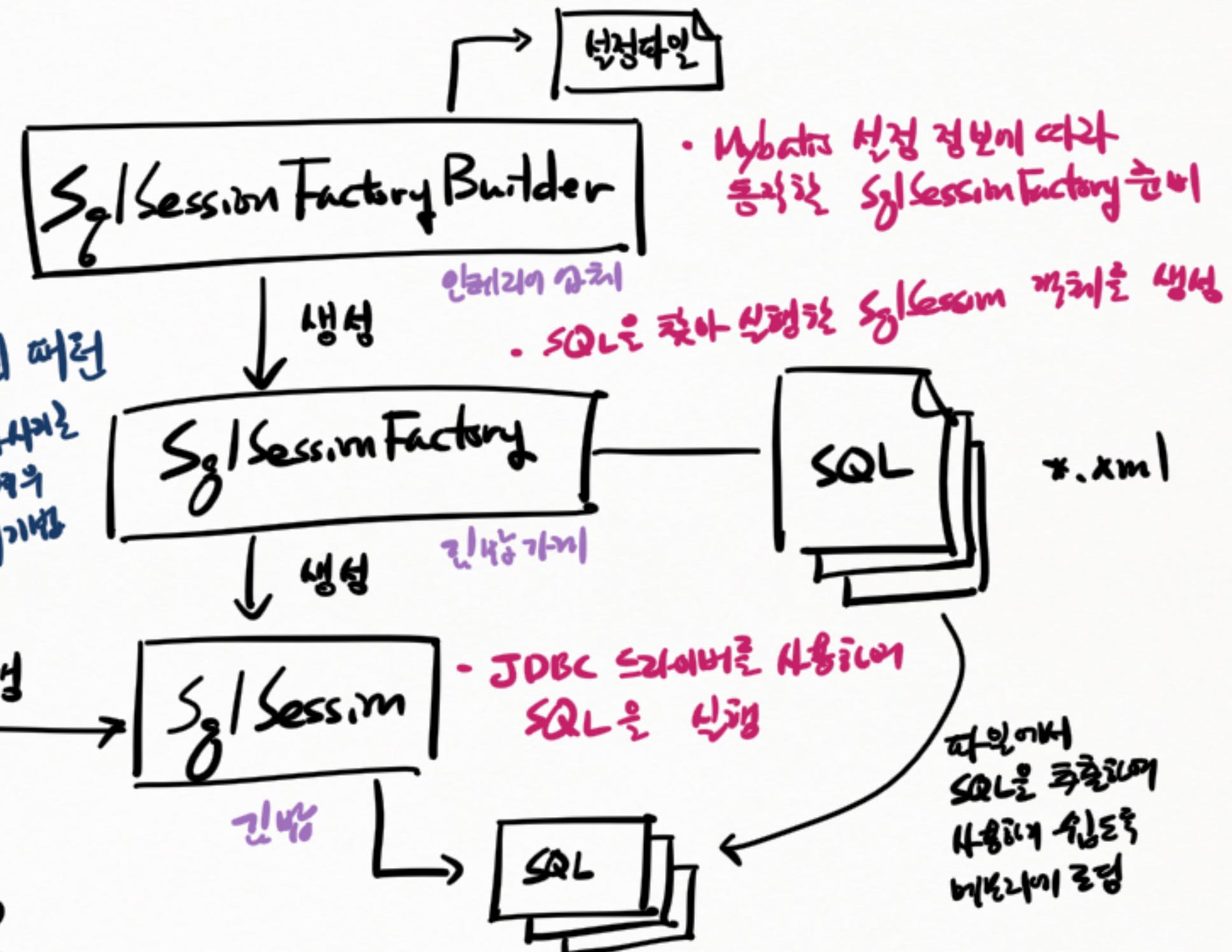


## \* Mybatis 프레임워크의 핵심 클래스

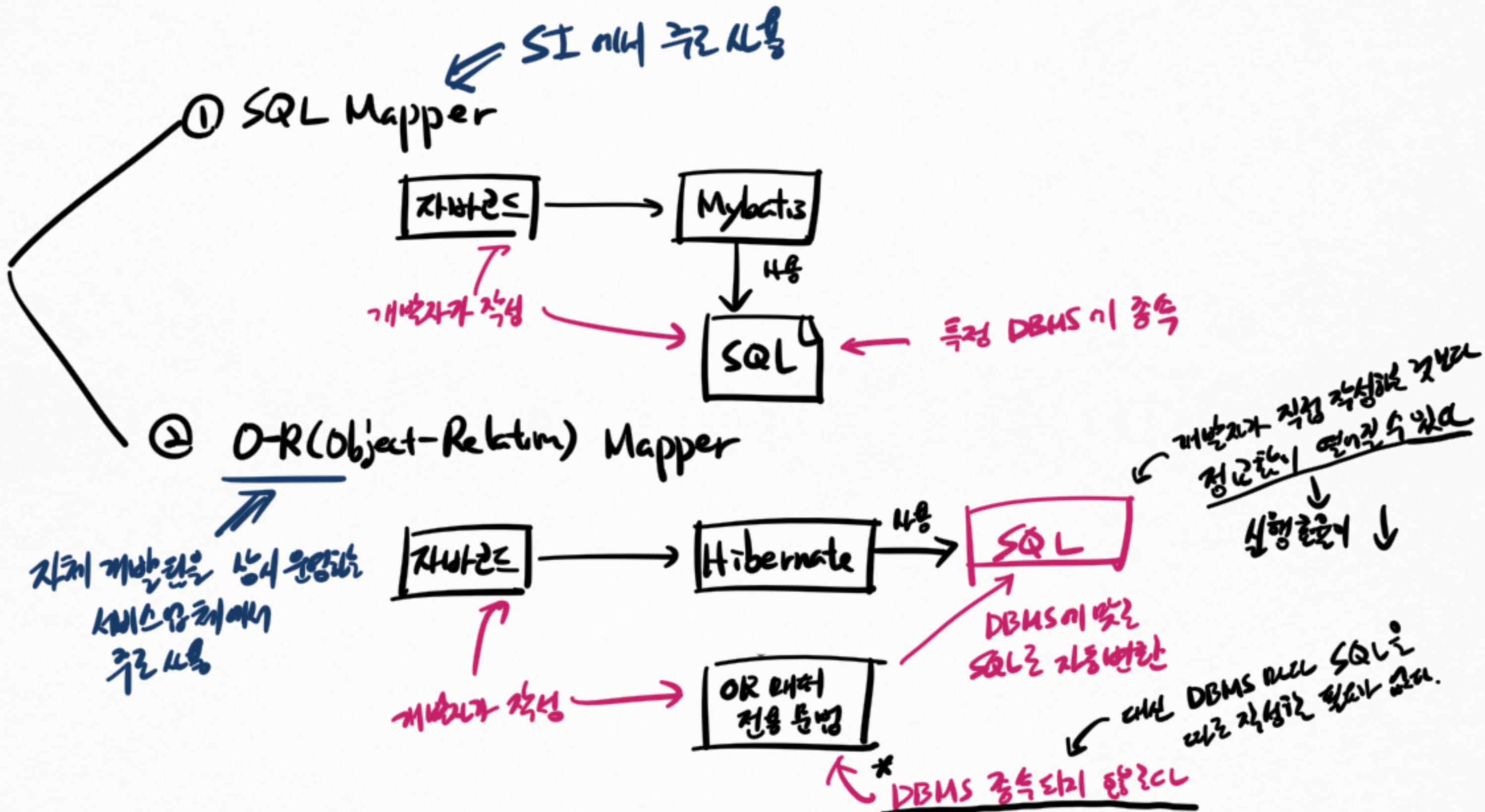
\* Builder 패턴  
 ↓  
 여러 가지로 구성된  
 복잡한 구조의 객체를  
 단계로 대 A용으로 분기기법

\* Factory Method 패턴  
 ↓  
 개체 생성 후 초기화시키고  
 리셋이 복잡한 경우  
 활용하는 단계기법

SQL 실행  
 APP →  
 insert()  
 update()  
 delete()  
 selectList()  
 selectOne()



## \* Data Persistence Framework



## \* XML

XML 선언  
반드시 첫 번째 줄에 있어야 한다

가장 빠른 뜻 태그

↓

root element

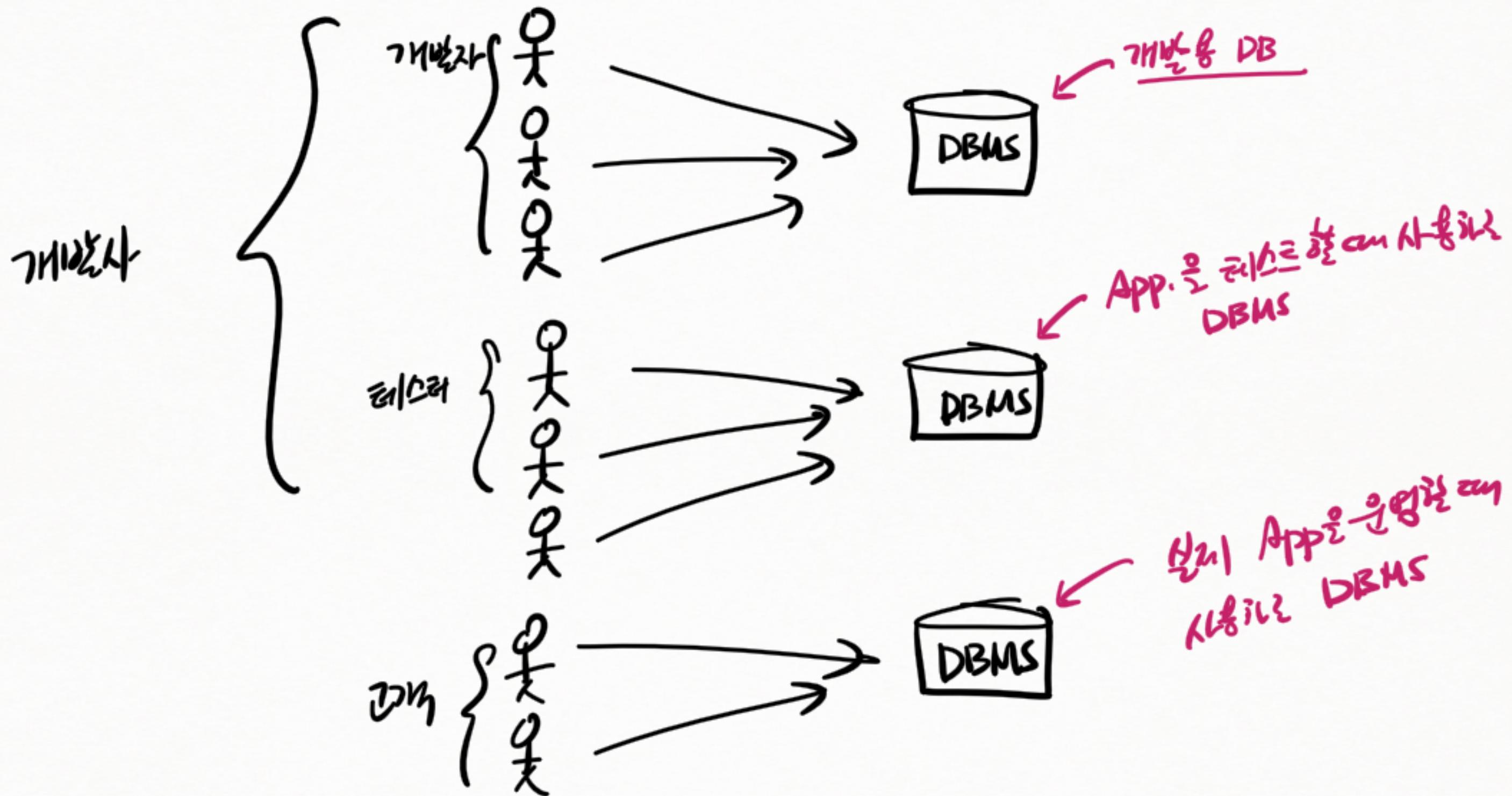
요기, 나중에  
가능

element = {

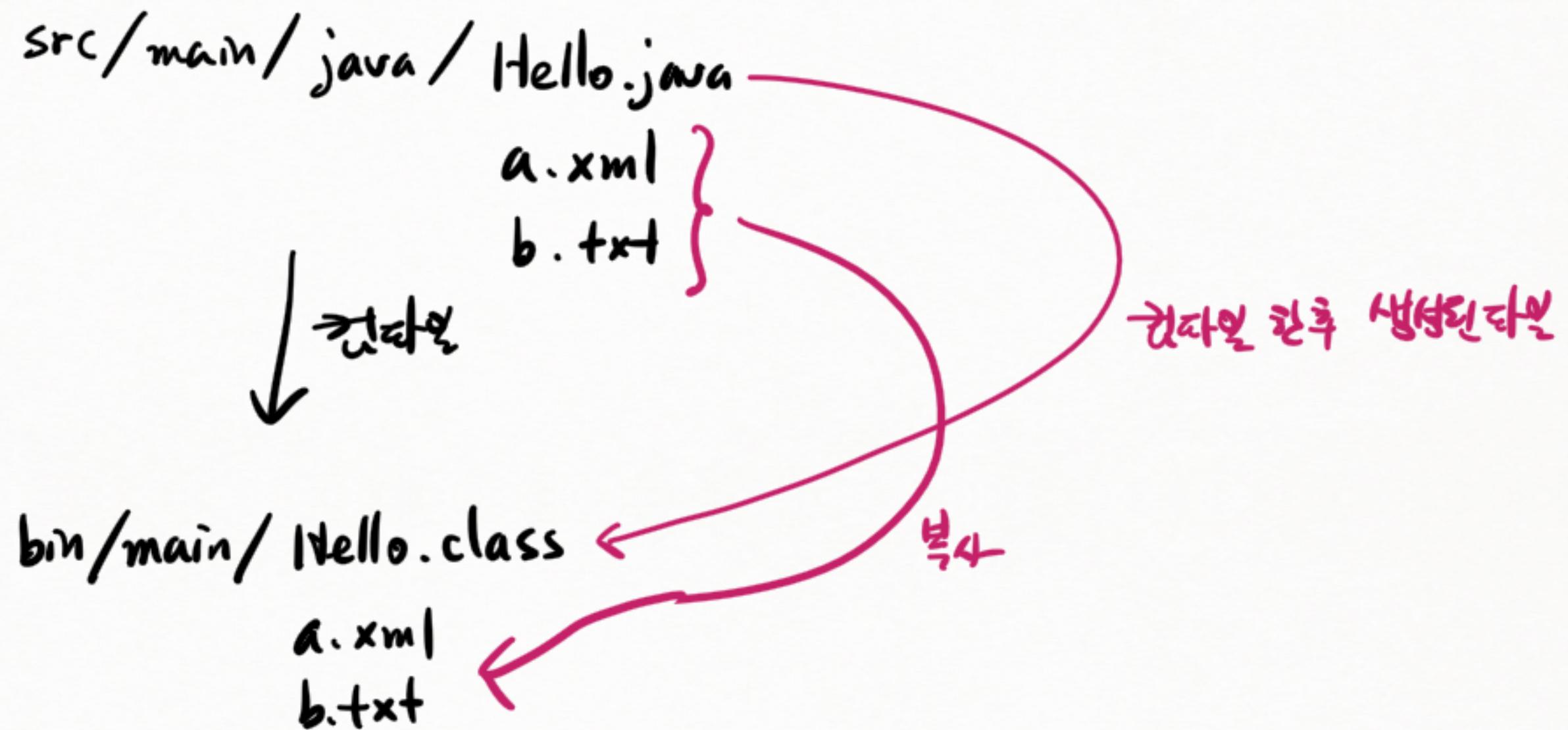
    "tag"      → environments default = "development"  
    "markup"     → <environments>      ← 바로 element  
                ≡ }      ← 컨텐츠      ← 데이터  
                플레이스 → </environments>  
                ↓  
                <mappers>      → 라이브러리  
                <mapper>      resource = " — "/>  
                </mappers>      ↑ 속성(attribute)  
                ↓  
                </configuration>

속성의 값  
플레이스를  
생략할 때

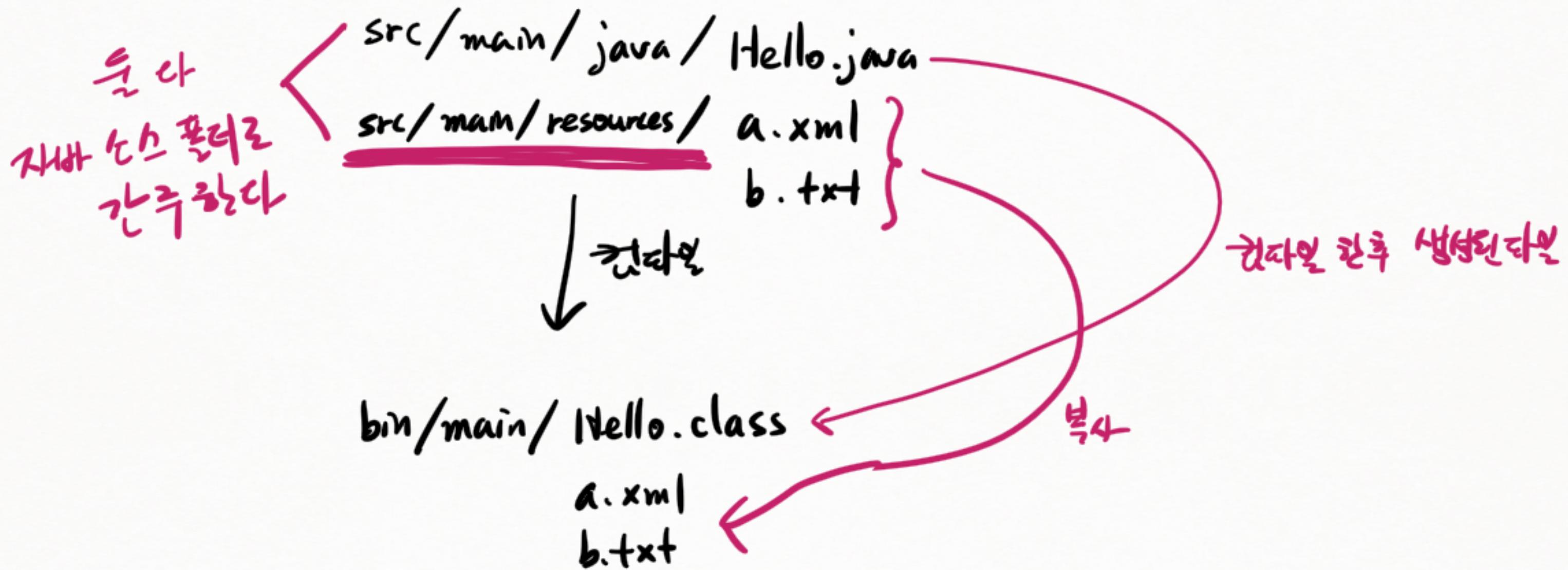
## \* 실무의 DBMS 구조



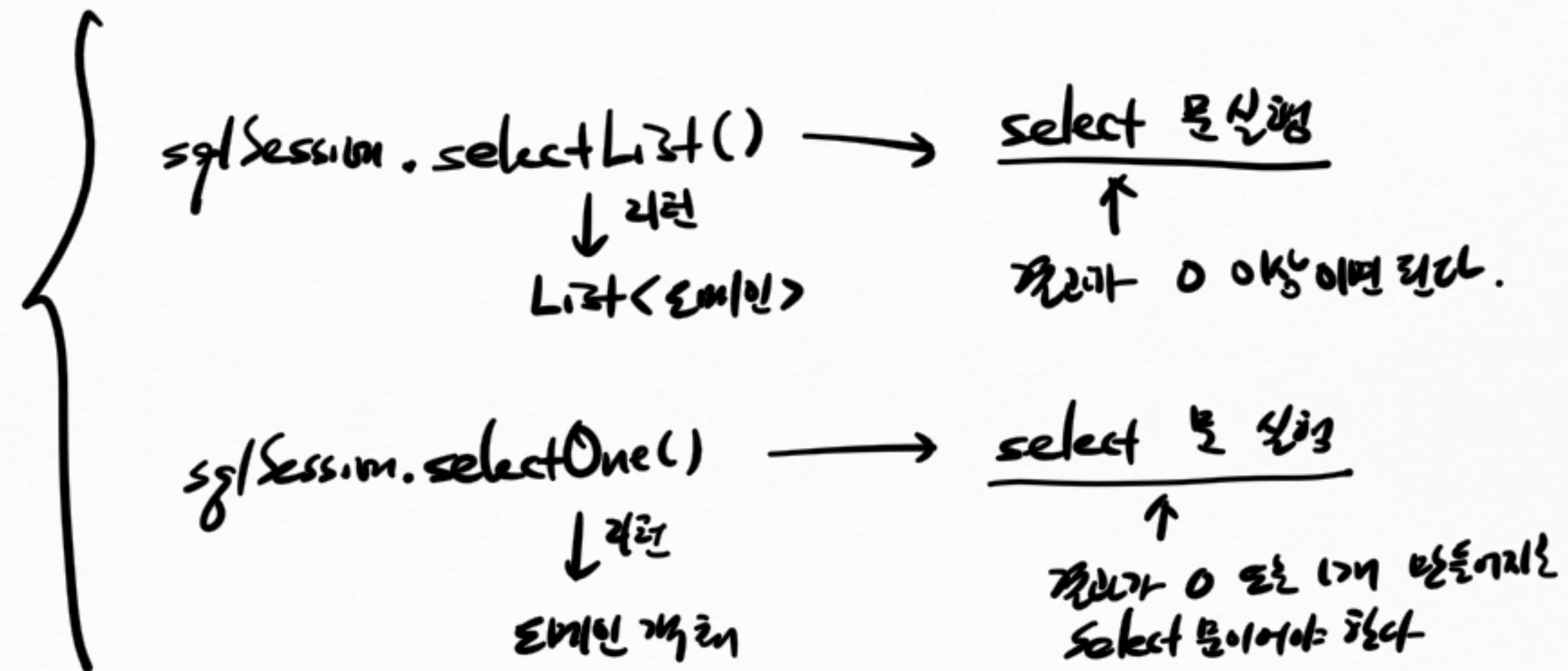
## \* 터스파일 컴파일

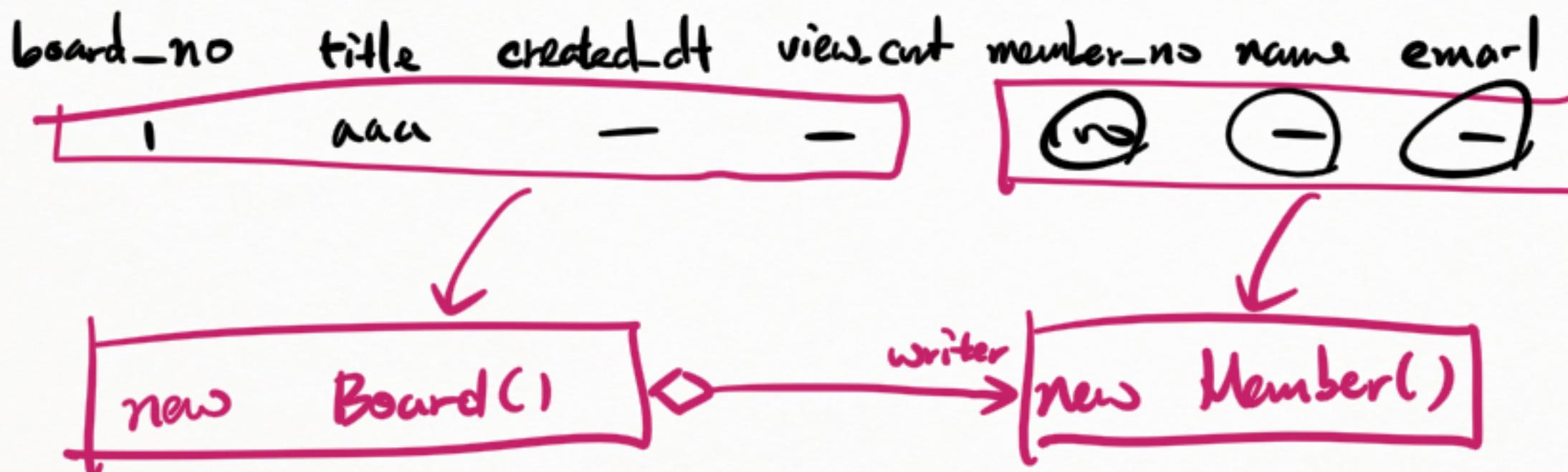


\* 터스파일 컴파일 - 자바코드와 익스포트 파일을  
분리해서 관리하기



\* SqlSession - selectList() 와 selectOne()





```

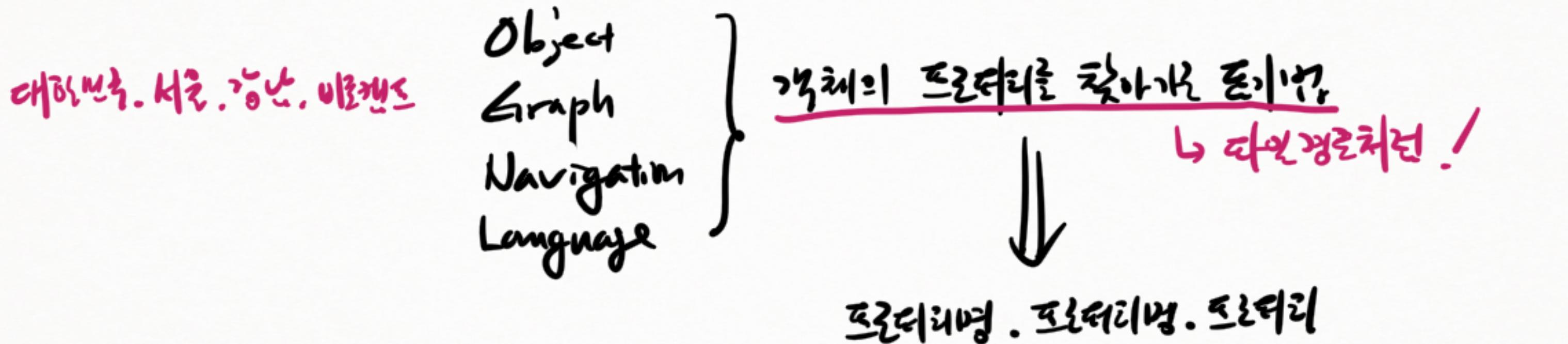
Board b = new Board()
b.setNo(-);
b.setTitle(-);
b.setRegisteredDate(-);
b.setViewCount(-);
    
```

```

Member m = new Member();
m.setNo(-);
    
```

~~m.setName(-);~~  
~~m.setEmail(-);~~  
~~b.setWriter(m);~~

## \* OGNL 동기법



```
class Member {  
    Address address;  
}  
:  
class Address {  
    String basicAddress;  
}  
:=
```

owner.address.basicAddress  
↳ 재귀호출  
getOwner().getAddress().getBasicAddress()  
↓            ↓            ↓  
Member      Address      String