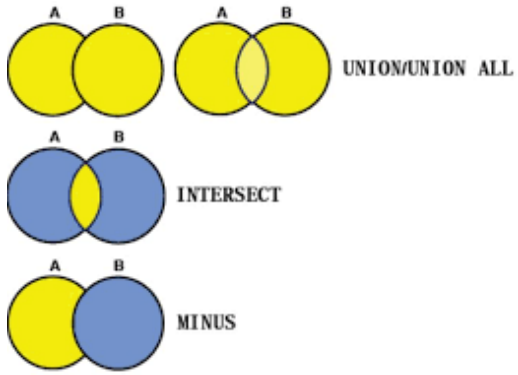
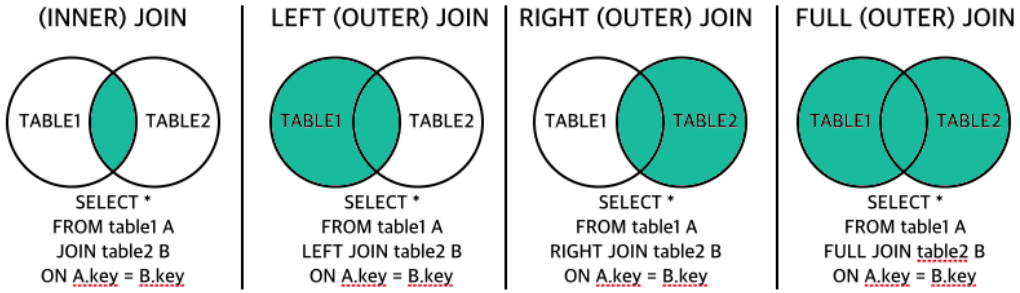
2개 이상의 테이블로부터 데이터를 추출하는 방법은 다양하다.

1. 집합 연산자를 이용하는 방법



2. 서브쿼리를 이용하는 방법

3. 조인을 이용하는 방법



집합 연산자 (SET operator)

집합 연산자를 활용하면, 2개 이상의 SELECT 문을 연결하여 작성 가능하다.

- SELECT 문의 컬럼 개수와 데이터 타입은 일치해야 한다.
- 검색 결과의 헤더는 앞쪽 SELECT 문에 의해 결정된다.
- ORDER BY 절을 사용할 때는 문장의 제일 마지막에 사용한다.

```
SELECT ...
FROM 테이블명 ...
UNION | UNION ALL | INTERSECT | MINUS
SELECT ...
FROM 테이블명 ...
[ORDER BY [열이름] [ASC | DESC]];
```

집합 연산자 (SET operator)

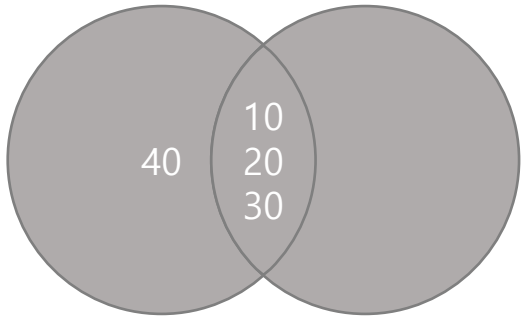
집합 연산자를 활용하면, 2개 이상의 SELECT 문을 연결하여 작성 가능하다.

- UNION : 합집합 (중복되는 값은 한번 출력)
- UNION ALL : 합집합 (중복되는 값 모두 출력)
- INTERSECT : 교집합
- EXCEPT : 차집합 (다른 DBMS에서 MINUS)

집합 연산자 (SET operator)

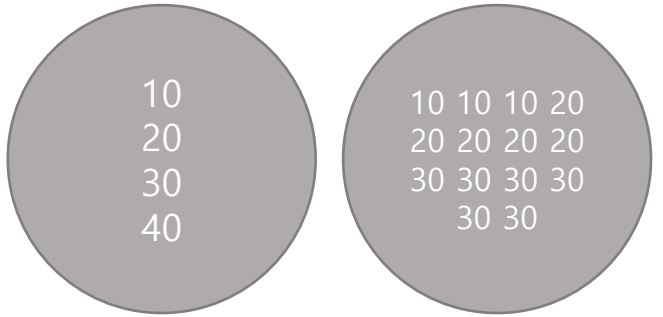
UNION, UNION ALL

```
SELECT deptno FROM dept
 UNION
SELECT deptno FROM emp;
```



UNION

```
SELECT deptno FROM dept
 UNION ALL
SELECT deptno FROM emp;
```

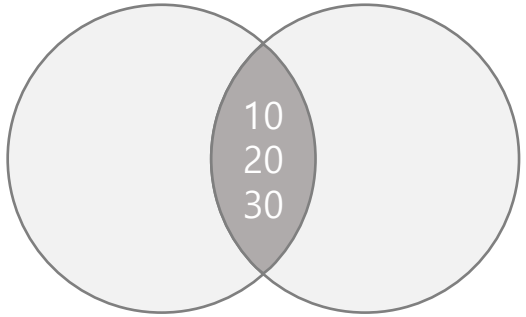


UNION ALL

집합 연산자 (SET operator)

INTERSECT

```
SELECT deptno FROM dept
 INTERSECT
SELECT deptno FROM emp;
```

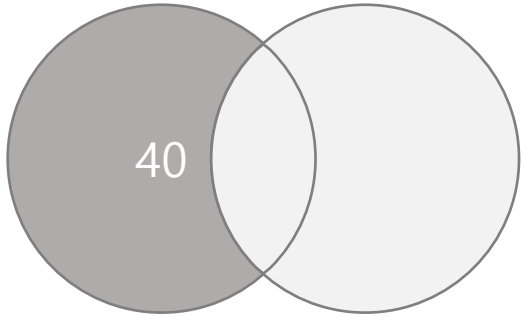


INSERSECT

집합 연산자 (SET operator)

EXCEPT

```
SELECT deptno FROM dept
 EXCEPT
SELECT deptno FROM emp;
```



INSERSECT

서브 쿼리 (Sub Query)

하나의 SQL 문 안에 포함되어 있는 SQL 문을 의미하며,
메인 쿼리가 서브 쿼리를 포함하는 종속적인 관계이다.

- 항상 Main Query의 기준으로 결과를 나타낸다.
- Sub Query는 반드시 괄호로 감싸서 사용한다.
- Sub Query는 Main Query 실행 전에 한 번 실행된다.
- Sub Query에서 ORDER BY 절은 사용 불가하다.
- FROM 절에 Sub Query를 사용하는 것을 '인라인 뷰'라 한다.

서브 쿼리 (Sub Query)

```
SELECT dname
FROM dept
WHERE deptno = (
        SELECT deptno
        FROM emp
        WHERE ename = 'MILLER’
);
```

서브 쿼리 (Sub Query)

단일 행(Single Row) 서브 쿼리

- 수행 결과가 오직 하나의 데이터(행, row)만을 반환하는 Sub Query

인구의 최대값을 먼저 조회한 뒤, 인구 수가 최대값과 동일한 데이터조회

```
SELECT name
FROM tCity
WHERE popu = (
        SELECT MAX(popu) FROM tCity
);
```

서브 쿼리 (Sub Query)

- EMP 테이블을 이용해 평균 급여보다 더 많은 급여를 받는 사원을 검색
- EMP 테이블에서 MILLER와 같은 부서(deptno)에서 근무하는 사원을 검색
- EMP 테이블에서 MILLER와 동일한 job을 가진 사원을 검색
- EMP 테이블에서 MILLER와 급여(SAL)와 동일하거나 더 많이 받는 사원을 검색
- EMP 테이블에서 deptno을 이용해 LOC가 DALLAS인 사원 검색 (DEPT 테이블 활용)
- EMP 테이블에서 직속상관(MGR)의 이름이 KING인 사원 검색

서브 쿼리 (Sub Query)

다중 열(Multi Column) 서브 쿼리

- Sub Query의 수행 결과가 여러 개의 열로 된 경우

안중근 과 같은 부서에 근무하고 성별이 같은 직원의 모든 정보를 조회

```
SELECT *
FROM tStaff
WHERE (depart, gender) = (
        SELECT depart, gender FROM tStaff WHERE name = '안중근'
);
```

서브 쿼리 (Sub Query)

다중 행(Multi Row) 서브 쿼리

- Sub Query에서 반환되는 결과가 하나 이상의 행일 때 사용
- 다중 행 서브 쿼리는 다중 행 연산자와 함께 사용

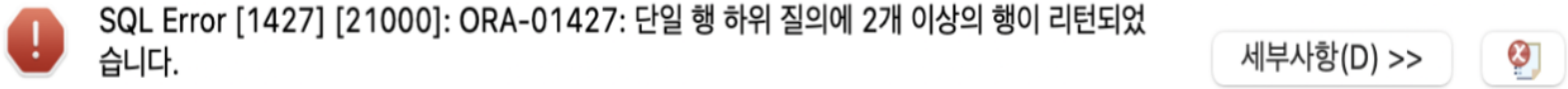
종류	의미
IN	Sub Query의 결과 중에서 하나라도 일치하면 참
ANY, SOME	Sub Query의 검색 결과와 하나 이상이 일치하면 참
ALL	Sub Query의 검색 결과와 모든 값이 일치하면 참
EXISTS	Sub Query의 결과 중에서 만족하는 값이 하나라도 존재하면 참

서브 쿼리 (Sub Query)

IN 연산자 - 반환되는 여러 개의 행 중에서 하나만 참이 되어도 참

EMP 테이블에서 부서(deptno)별로 가장 급여를 많이 받는 사원들과 동일한 급여를 받는 사원 검색

```
SELECT empno, ename, sal, deptno
FROM emp
WHERE sal = (
        SELECT MAX(sal) FROM emp GROUP BY deptno
);
```



서브 쿼리 (Sub Query)

IN 연산자 - 반환되는 여러 개의 행 중에서 하나만 참이 되어도 참

EMP 테이블에서 부서(deptno)별로 가장 급여를 많이 받는 사원들과 동일한 급여를 받는 사원 검색

```
SELECT empno, ename, sal, deptno
FROM emp
WHERE sal IN (
        SELECT MAX(sal) FROM emp GROUP BY deptno
);
```




서브 쿼리 (Sub Query)

ALL 연산자 - Sub Query의 검색 결과와 모든 값이 일치하면 참

deptno가 30번인 소속 직원들 중에서 급여를 가장 많이 받는 사원보다 많은 급여를 받는 직원 검색

```
SELECT ename, sal
FROM emp
WHERE sal > ALL(
        SELECT sal FROM emp WHERE deptno = 30
);
```

```
SELECT ename, sal
FROM emp
WHERE sal > (
        SELECT MAX(sal) FROM emp WHERE deptno = 30
);
```

서브 쿼리 (Sub Query)

ANY 연산자 - Sub Query의 검색 결과에서 하나만 일치하면 참

deptno가 30번인 소속 직원들 중에서 급여를 가장 적게 받는 사원보다 많은 급여를 받는 직원 검색

```
SELECT ename, sal
FROM emp
WHERE sal > ANY (
        SELECT sal FROM emp WHERE deptno = 30
);
```

```
SELECT enaem, sal
FROM emp
WHERE sal > (
        SELECT MIN(sal) FROM emp WHERE deptno = 30
);
```

서브 쿼리 (Sub Query)

EXISTS 연산자 - Sub Query로 어떤 데이터 존재 여부를 확인 (참 또는 거짓 반환)

EMP 테이블에서 SAL이 2,000을 넘는 사원이 있으면 모든 직원을 조회

```
SELECT ENAME, SAL
FROM EMP
WHERE EXISTS (
        SELECT 1 FROM EMP WHERE SAL > 2000
);
```

서브 쿼리 (Sub Query)

- EMP 테이블에서 부서별로 가장 급여를 많이 받는 사원들과 동일한 급여를 받는 사원 검색
- EMP 테이블에서 SAL를 3,000 이상 받는 사원이 소속된 부서와 동일한 부서에서 근무하는 사원 검색
- EMP 테이블에서 JOB이 MANAGER인 사람이 속한 부서 정보 검색
- EMP 테이블에서 BLAKE와 동일한 부서에 있는 모든 사원 검색
- EMP 테이블에서 평균 급여(SAL) 이상을 받는 모든 사원 검색. 급여가 많은 순으로 출력
- EMP 테이블에서 이름에 “T”가 있는 사원이 근무하는 부서에서 있는 모든 사원 검색. 사원번호 순으로 출력
- EMP 테이블에서 근무 지역이 DALLAS인 사원 정보 검색
- EMP 테이블에서 MGR의 이름이 KING인 사원 검색

조인 (JOIN)

두 개의 테이블을 엮어서 원하는 결과를 가져오고자 할 때 사용

	car	capacity	price	maker
1	소나타	2,000	2,500	현대
2	티볼리	1,600	2,300	쌍용
3	A8	3,000	4,800	Audi
4	SM5	2,000	2,600	삼성

capacity: int

	maker	factory	domestic
1	현대	부산	y
2	쌍용	청주	y
3	Audi	독일	n
4	기아	서울	y

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7,369	SMITH	CLERK	7,902	1980-12-17	800	[NULL]	20
7,499	ALLEN	SALESMAN	7,698	1981-02-20	1,600	300	30
7,521	WARD	SALESMAN	7,698	1981-02-22	1,250	500	30
7,566	JONES	MANAGER	7,839	1981-04-02	2,975	[NULL]	20
7,654	MARTIN	SALESMAN	7,698	1981-09-28	1,250	1,400	30
7,698	BLAKE	MANAGER	7,839	1981-05-01	2,850	[NULL]	30
7,782	CLARK	MANAGER	7,839	1981-06-09	2,450	[NULL]	10
7,788	SCOTT	ANALYST	7,566	1987-07-13	3,000	[NULL]	20

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

조인 (JOIN)

CROSS JOIN - 2개 테이블의 모든 집합의 JOIN

조인 결과의 전체 행 개수는 두 테이블의 각 행의 개수를 곱한 수이고,
이를 카티션 곱(CARTESIAN PRODUCT)이라 한다.

```
SELECT * FROM tcar, tmaker;

SELECT * FROM tcar CROSS JOIN tmaker;
```

조인 (JOIN)

INNER JOIN - 두 테이블의 공통된 컬럼의 데이터를 활용한 조인

두 테이블을 연결할 때 가장 많이 사용

```
SELECT *
FROM tcar, tmaker
WHERE tcar.maker = tmaker.maker;
```

```
SELECT *
FROM tcar c, tmaker m
WHERE c.maker = m.maker;
```

```
SELECT *
FROM tcar
JOIN tmaker ON tcar.maker = tmaker.maker;
```

```
SELECT *
FROM tcar c
JOIN tmaker m ON c.maker = m.maker;
```

조인 (JOIN)

두 테이블에 각각 조인을 정의한 컬럼명이 동일하다면

USING 절에서 조인할 컬럼을 지정하여 구문을 사용하거나,

```
SELECT *
FROM tcar
JOIN tmaker USING (maker);
```

NATURAL JOIN을 이용해 더욱 간단하게 표현할 수 있다.

```
SELECT *
FROM tcar
NATURAL JOIN tmaker;
```

조인 (JOIN)

SELF JOIN – 테이블이 자기 자신의 테이블과 조인

하나의 테이블 내에서 조인을 해야만 원하는 자료를 얻는 경우가 사용

```
SELECT CONCAT(e.ename, '의 매니저는 ', m.ename)
FROM emp e, emp m
WHERE e.mgr = m.empno;
```

```
SELECT CONCAT(e.ename, '의 매니저는 ', m.ename)
FROM emp e
JOIN emp m ON e.mgr = m.empno;
```

조인 (JOIN)

- DEPT 테이블의 LOC가 ‘NEW YORK’ 인 사원의 EMP 테이블의 이름과 급여를 조회
- DEPT 테이블의 DNAME 컬럼의 값이 ‘ACCOUNTING’ 인 사원의 EMP 테이블의 이름과 입사일을 조회
- EMP 테이블의 JOB이 ‘MANAGER’인 사원의 EMP 테이블의 이름, DEPT 테이블의 부서명을 조회
- EMP 테이블와 SALGRADE 테이블을 이용해 각 급여에 해당하는 등급을 매핑하여, 이름, 급여, 등급을 조회
- EMP 테이블에서 MANAGER 가 ‘KING’인 사원들의 이름, 직급을 조회

조인 (JOIN)

OUTER JOIN - JOIN 조건에서 한쪽 값이 없더라도 행을 반환

만약 한쪽 테이블에는 데이터가 존재하는데,
다른 쪽 테이블에는 해당 데이터가 존재하지 않을 경우 그 데이터는 출력되지 않는다.

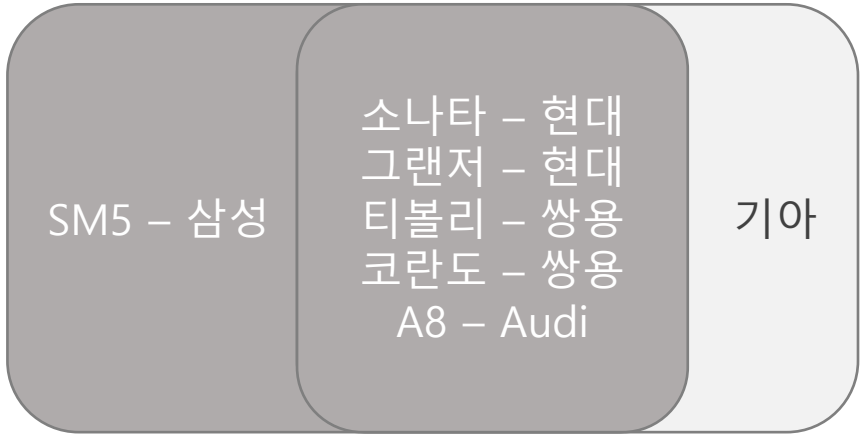
해당 문제점을 해결하기 위해 사용하는 조인 기법이 OUTER JOIN이다.

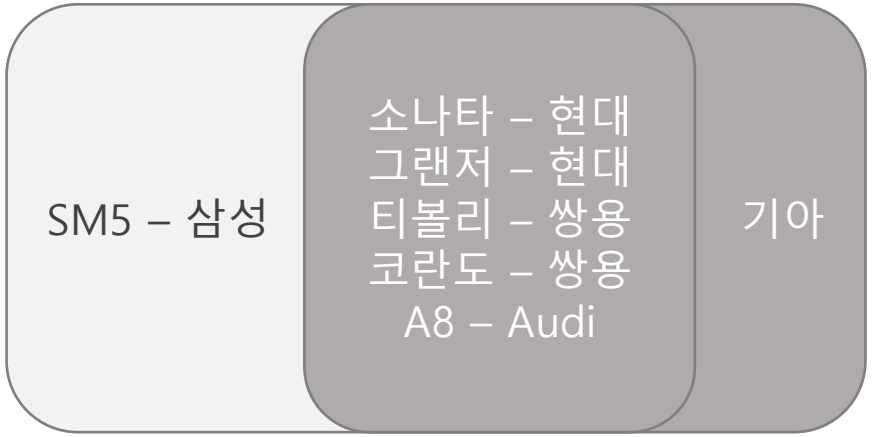
소나타 – 현대
그랜저 – 현대
티볼리 – 쌍용
코란도 – 쌍용
A8 – Audi
SM5 – 삼성

현대
쌍용
Audi
기아

조인 (JOIN)

OUTER JOIN - JOIN 조건에서 한쪽 값이 없더라도 행을 반환






```
SELECT * FROM tcar c
LEFT JOIN tmaker m ON c.maker = m.maker;

SELECT * FROM tcar
NATURAL LEFT JOIN tmaker;

SELECT * FROM tcar
RIGHT JOIN tmaker USING (maker);
```

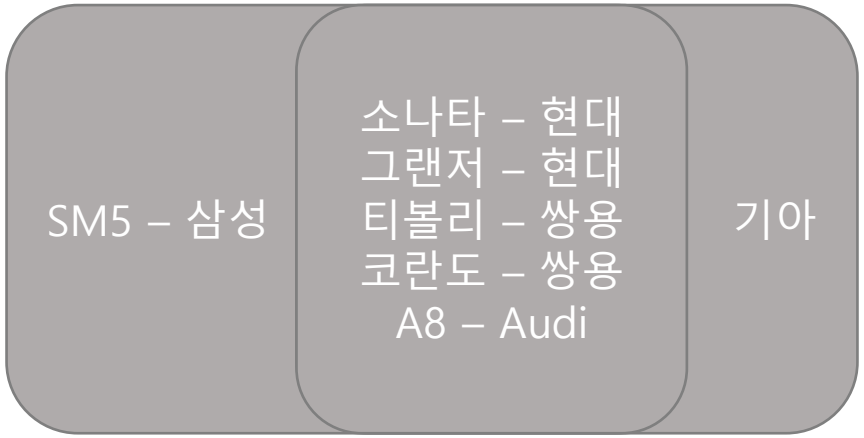
```
SELECT * FROM tcar c
RIGHT JOIN tmaker m ON c.maker = m.maker;

SELECT * FROM tcar
NATURAL RIGHT JOIN tmaker;

SELECT * FROM tcar
LEFT JOIN tmaker USING (maker);
```

조인 (JOIN)

MySQL에서는 FULL OUTER JOIN을 지원하지 않기 때문에,
LEFT OUTER JOIN, RIGHT OUTER JOIN 을 합쳐서 FULL OUTER JOIN 구현




```
SELECT * FROM tcar LEFT JOIN tmaker ON tcar.maker = tmaker.maker
UNION
SELECT * FROM tcar RIGHT JOIN tmaker ON tcar.maker = tmaker.maker;
```




조인 (JOIN)

다중 조인 - 조인을 중첩하면 여러 개의 테이블을 조인할 수 있다.

조인한 결과도 하나의 테이블이므로 그 결과와 다른 테이블을 조인하는 것이 가능

```
SELECT * FROM tcar c
JOIN tmaker m ON c.maker = m.maker
JOIN tcity ct ON m.factory = ct.name;
```

조인 (JOIN)

- 사원(EMP) 테이블과 부서(DEPT) 테이블을 조인하여, 사원명, 부서번호,부서명을 출력
  사원 테이블에는 부서번호 40번 데이터가 없지만, 40번 부서의 부서명도 함께 출력

- NEW YORK에서 근무하고 있는 사원에 대하여 사원명, 업무, 급여, 부서명을 출력

- 보너스(comm)가 null 이 아닌 사원에 대하여 사원명, 부서명, 위치를 출력

- 사원명 중 L자가 있는 사원에 대하여 사원명, 업무, 부서명, 위치를 출력

- 자신의 관리자보다 먼저 입사한 사원에 대하여 이름, 입사일, 관리자 이름, 관리자 입사일을 출력

데이터 추가 - INSERT

INSERT :

INSERT INTO 뒤에 추가할 테이블을 입력하고,
행의 데이터는 VALUES 구를 사용해 작성한다.

값을 지정할 때는 해당 열의 데이터 형식에 맞도록 지정해야 한다.

```
INSERT INTO 테이블명 VALUES(값1, 값2, ...);
```

INSERT 명령을 실행

데이터가 추가되는 테이블을 지정

추가되는 데이터의 값을 형식에 맞게 지정

데이터 추가 - INSERT

INSERT :

INSERT문을 실행하면 처리상태만 표시되기 때문에
SELECT문을 이용해야 입력된 데이터를 직접 확인할 수 있다.

INSERT문은 저장할 열을 지정할 수 있다.
지정되지 않은 열은 기본값 또는 null이 저장됩니다.

```
INSERT
    INTO 테이블명(열이름1, 열이름2, ...)
    VALUES(값1, 값2, ...);
```

데이터 추가 - INSERT

INSERT :

```
INSERT INTO tCity
VALUES ('강릉', 1040, 21, 'N', '강원');

INSERT INTO tCity (name, area, popu, metro, region)
VALUES ('원주', 867, 35,  ' y ' , '강원');
```

데이터 추가 - INSERT

INSERT :

```
DELETE FROM tCity;

INSERT INTO tCity (name, area, popu, metro, region) VALUES
('서울',605,974,'y','경기'),
('부산',765,342,'y','경상'),
('오산',42,21,'n','경기'),
('전주',205,65,'n','전라'),
('순천',910,27,'n','전라'),
('춘천',1116,27,'n','강원'),
('홍천',1819,7,'n','강원');
```




데이터 추가 - INSERT

도시 목록에 이천(461, 21) 과 대구(883, 248) 그리고 영월(1127, 4)을 삽입




데이터 추가 - INSERT

INSERT SELECT :

```
INSERT INTO tCity (name, area, popu, metro, region)
SELECT factory, 940, 83, 'n', '충청'
FROM tmaker
WHERE maker = '쌍용';
```

데이터 삭제 - DELETE

DELETE :

만약 WHERE구를 생략하게 되면 해당 테이블의 모든 데이터가 삭제된다.

따라서 DELETE 명령을 실행할 때는 주의를 기울여야 한다.

```
DELETE FROM 테이블명 WHERE 조건식;
```

DELETE 명령을 실행

데이터가 삭제되는 테이블을 지정

삭제되는 데이터 지정
(생략 시 모든 데이터 삭제)

데이터 삭제 - DELETE

DELETE :

```
DELETE FROM tCity WHERE name = '부산';

DELETE FROM tCity WHERE region = '경기';

DELETE * FROM tCity WHERE popu > 50;
```




데이터 삭제 - DELETE

영업부 직원을 전부 삭제

데이터 갱신 - UPDATE

UPDATE :

데이터 갱신 작업은 시스템을 다루는 과정에서 빈번히 발생합니다.

UPDATE 는 셀 단위로 데이터가 갱신됩니다.

WHERE 구를 생략한 경우에는 테이블의 모든 행이 갱신됩니다.

```
UPDATE 테이블명 SET 열이름 = 값 WHERE 조건식;
```

UPDATE 명령을 실행

갱신할 열과 값 지정

갱신되는 데이터 지정
(생략 시 모든 데이터 갱신)

데이터 갱신 - UPDATE

UPDATE :

갱신해야 할 열이 복수인 경우에는 콤마(,)로 구분하여 지정할 수 있습니다.

```
UPDATE 테이블명
    SET 열이름1 = 값1, 열이름2 = 값2, ...
    WHERE 조건식;
```




데이터 갱신 - UPDATE

name 이 서울인 데이터의 popu 는 1000 으로 region 은 충청으로 수정

name 이 오산인 데이터의 popu 을 2배로 갱신

여자 사원 모두를 차장으로 갱신

총무부 직원의 월급을 10% 인상




테이블 관련 명령

테이블 생성

CREATE TABLE 테이블명 (
컬럼명1 타입 [**CONSTRAINT** 제약조건이름] 컬럼제약조건,
컬럼명2 타입,
컬럼명3 타입,
...
[**CONSTRAINT** 제약조건이름] 테이블 제약 조건) **옵션**=옵션값;

테이블 관련 명령

데이터 타입 - 숫자

데이터 형식	바이트 수	숫자 범위	설명
BIT(N)	N/8		1~64bit를 표현. b'0000' 형식으로 표현
TINYINT	1	-128~127	정수
★SMALLINT	2	-32,768~32,767	정수
MEDIUMINT	3	-8,388,608~8,388,607	정수
★INT INTEGER	4	약 -21억~+21억	정수
★BIGINT	8	약 -900경~+900경	정수
★FLOAT	4	-3.40E+38~-1.17E-38	소수점 아래 7자리까지 표현
★DOUBLE REAL	8	-1.22E-308~1.79E+308	소수점 아래 15자리까지 표현
★DECIMAL(m, [d]) NUMERIC(m, [d])	5~17	$-10^{38}+1$~$+10^{38}-1$	전체 자릿수(m)와 소수점 이하 자릿수(d)를 가진 숫자형 예) decimal(5, 2)은 전체 자릿수를 5자리로 하되, 그 중 소수점 이하를 2자리로 하겠다는 의미

테이블 관련 명령

데이터 타입 – 문자

데이터 형식		바이트 수	설명
★CHAR(n)		1~255	고정길이 문자형. n을 1부터 255까지 지정. character의 약자 그냥 CHAR만 쓰면 CHAR(1)과 동일
★VARCHAR(n)		1~65535	가변길이 문자형. n을 사용하면 1부터 65535까지 지정. Variable character의 약자
BINARY(n)		1~255	고정길이의 이진 데이터 값
VARBINARY(n)		1~255	가변길이의 이진 데이터 값
TEXT 형식	TINYTEXT	1~255	255 크기의 TEXT 데이터 값
	TEXT	1~65535	N 크기의 TEXT 데이터 값
	MEDIUMTEXT	1~16777215	16777215 크기의 TEXT 데이터 값
	★LONGTEXT	1~4294967295	최대 4GB 크기의 TEXT 데이터 값
BLOB 형식	TINYBLOB	1~255	255 크기의 BLOB 데이터 값
	BLOB	1~65535	N 크기의 BLOB 데이터 값
	MEDIUMBLOB	1~16777215	16777215 크기의 BLOB 데이터 값
	★LONGBLOB	1~4294967295	최대 4GB 크기의 BLOB 데이터 값
ENUM(값들…)		1 또는 2	최대 65535개의 열거형 데이터 값
SET(값들…)		1, 2, 3, 4, 8	최대 64개의 서로 다른 데이터 값

테이블 관련 명령

데이터 타입 – 날짜

✓ 날짜 데이터 형식

DATE	1000-01-01 ~ 9999-12-31	YYYY-MM-DD
DATETIME	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS
TIMESTAMP	1970-01-01 ~ 2037년 임의 시간(1970-01-01 00:00:00 를 0으로 해서 1초단위로 표기)	
TIME	-838:59:59 ~ 838:59:59	
YEAR	901~2155	

✓ BOOL: true 또는 false

✓ JSON: 5.7.8 버전 이후에서 제공

✓ GEOMETRY: 공간 데이터 형식

테이블 관련 명령

테이블 옵션

ENGINE - 데이터를 디스크에 '쓰기'하거나 저장된 데이터를 '읽기'하는 역할을 한다

InnoDB :

기본값으로 설정되는 스토리지 엔진
Transaction-safe, Commit, Rollback, 데이터 복구 기능, Row-Level Locking 제공
데이터를 Clustered index에 저장하여 PK 기반의 query의 I/O 비용을 줄임
FK 제약을 제공하여 데이터 무결성을 보장

MyISAM

트랜잭션을 지원하지 않고 Table-Level Locking을 제공
Multi-thread 환경에서 성능이 저하 될 수 있음

Archive

'로그 수집'에 적합한 엔진
데이터가 메모리상에서 압축되고 압축된 상태로 디스크에 저장

테이블 관련 명령

테이블 옵션

ENGINE - 데이터를 디스크에 '쓰기'하거나 저장된 데이터를 '읽기'하는 역할을 한다

InnoDB :

기본값으로 설정되는 스토리지 엔진
Transaction-safe, Commit, Rollback, 데이터 복구 기능, Row-Level Locking 제공
데이터를 Clustered index에 저장하여 PK 기반의 query의 I/O 비용을 줄임
FK 제약을 제공하여 데이터 무결성을 보장

MyISAM

트랜잭션을 지원하지 않고 Table-Level Locking을 제공
Multi-thread 환경에서 성능이 저하 될 수 있음

Archive

'로그 수집'에 적합한 엔진
데이터가 메모리상에서 압축되고 압축된 상태로 디스크에 저장

테이블 관련 명령

테이블 옵션

auto_increment - 시퀀스의 초기 값을 설정

* **ALTER TABLE** [테이블명] **auto_increment**=[시작하려는 값]; 으로 초기값 재설정 가능

DEFAULT CHARSET - 인코딩 방식 설정

* UTF8로 설정하지 않으면 한글 사용 불가능




테이블 관련 명령

테이블 변경 – 테이블 이름 수정

```
ALTER TABLE 이전테이블명 RENAME 새로운테이블명;
```

테이블 관련 명령

테이블 변경 – 테이블 컬럼 추가, 삭제

테이블 컬럼 추가

```
ALTER TABLE 테이블명 ADD COLUMN 컬럼명1 데이터타입, 컬럼명2 데이터타입;
```

테이블 변경 – 테이블 컬럼 삭제 (제약 조건이 설정된 경우, 제약조건 먼저 삭제)

```
ALTER TABLE 테이블명 DROP 컬럼명1, 컬럼명2;
```

테이블 관련 명령

```
CREATE TABLE contact (
        num INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        address VARCHAR(255),
        tel VARCHAR(20),
        email VARCHAR(255),
        birthday DATE
) ENGINE=InnoDB auto_increment=1 DEFAULT CHARSET=utf8;


ALTER TABLE contact ADD COLUMN age INT;

DESC contact;


ALTER TABLE contact DROP age INT;

DESC contact;
```

테이블 관련 명령

테이블 변경 – 테이블 컬럼 변경

기존 컬럼의 자료형과 이름 변경

```
ALTER TABLE 테이블명 CHANGE 이전컬럼명 새로운컬럼명 컬럼타입;
```

기존 컬럼의 자료형 변경

```
ALTER TABLE 테이블명 MODIFY 컬럼명 컬럼타입;
```

```
ALTER TABLE contact CHANGE tel phone int;
DESC contact;
```




테이블 관련 명령

테이블 변경 – 테이블 컬럼 순서 조정

첫번째 순서로 이동

```
ALTER TABLE 테이블명 MODIFY COLUMN 컬럼명 자료형 FIRST;
```

다른 컬럼의 컬럼 뒤로 이동

```
ALTER TABLE 테이블명 MODIFY COLUMN 컬럼명 자료형 AFTER 다른컬럼;
```




테이블 관련 명령

테이블 삭제

```
DROP TABLE 테이블명;
```

```
DROP TABLE contact;
SHOW TABLES;
```




테이블 관련 명령

제약 조건

- NULL 허용 여부
  기본은 NULL을 허용하는 것이며, 컬럼 제약조건에 NOT NULL을 설정하면, NULL 입력 불가

```
CREATE TABLE tNullable (
        name CHAR(10) NOT NULL,
        age INT
);

INSERT INTO tNullable (name, age) VALUES ('흥부', 36);
INSERT INTO tNullable (name) VALUES ('놀부');
INSERT INTO tNullable (age) VALUES (44); -- 에러
```

테이블 관련 명령

제약 조건

- 기본값

  필드 값을 입력하지 않았을 때, 자동으로 입력할 값을 설정할 수 있다
  숫자형은 0, 문자열은 비워두거나 N/A 등을 많이 사용한다.
  DEFAULT 키워드와 함께 기본값을 지정

```
CREATE TABLE tCheckTest(
        gender CHAR(3) CHECK(gender = '남' OR gender = '여'),
        grade INT CHECK (grade >= 1 AND grade <= 3),
        origin CHAR(3) CHECK(origin IN ('동','서','남','북')),
        name CHAR(10) CHECK(name LIKE '김%')
);
```

테이블 관련 명령

제약 조건

- 체크

  필드의 값 종류 제한
  컬럼 선언문에 CHECK 키워드와 함께 컬럼값으로 가능한 값을 지정

```
CREATE TABLE tCityDefault(
        name CHAR(10) PRIMARY KEY,
        area INT NULL ,
        popu INT NULL ,
        metro CHAR(1) DEFAULT 'n' NOT NULL,
        region CHAR(6) NOT NULL
);

INSERT INTO tCityDefault (name, area, popu, region) VALUES ('진주', 712, 34, '경상');
INSERT INTO tCityDefault (name, area, popu, metro, region) VALUES ('인천', 1063, 295, 'y', '경기');
INSERT INTO tCityDefault VALUES ('강릉', 131, 24, '강원');
INSERT INTO tCityDefault VALUES ('강릉', 131, 24, DEFAULT, '강원');

UPDATE tCityDefault SET metro = DEFAULT WHERE name = '인천';
```

테이블 관련 명령

각 필드에 기본값을 적용하여 tStaffDefault 테이블을 생성
(부서는 영업부, 직급은 수습, 초봉은 280, 성취도는 1.0 으로 기본값을 적용)

직원 테이블의 각 필드에 제약 조건을 설정하여 tStaffDefault2 테이블을 생성
(부서는 영업부, 총무부, 인사과 중 하나만, 성별은 남 또는 여, 월급은 0 보다 크다는 조건 설정)

테이블 관련 명령

제약 조건

- 슈퍼키(Super Key)
  각 행을 유일하게 식별할 수 있는 하나 또는 그 이상의 속성들의 집합
  슈퍼키는 유일성만 만족하면 슈퍼키가 될 수 있다.

- 후보키(Candidate Key)
  각 행을 유일하게 식별할 수 있는 최소한의 속성들의 집합

- 기본키(Primary Key)
  후보키 중에서 선정한 키로 테이블에서 오직 1개만 지정 가능
  레코드를 상징하는 값으로 자주 참조하는 속성이어야 한다.

테이블 관련 명령

기본키(Primary Key) 설정 방법

- 기본키는 NOT NULL 속성을 겸하기 때문에 NOT NULL을 붙일 필요 없음
- 제약조건이름은 나중에 편집이나 삭제 등을 편리하게 하기 위해서 사용
- 복합키로 기본키를 설정하는 경우에는 테이블 제약조건 형태로 설정

* 2개 이상 컬럼의 조합으로 기본키를 설정하는 것을 복합키라고 한다.

```
CREATE TABLE 테이블명 (
컬럼명1 타입 [CONSTRAINT 제약조건이름] PRIMARY KEY,
...
);
```

```
CREATE TABLE 테이블명 (
컬럼명1 타입 [CONSTRAINT 제약조건이름],
...
CONSTRAINT 제약조건이름 PRIMARY KEY(컬럼명1)
);
```

테이블 관련 명령

이름과 부서와 성별을 복합키로 지정하여 새로운 직원 테이블 tStaffCompoKey를 생성
(구성은 tstaff 테이블과 동일)

테이블 관련 명령

제약 조건

- UNIQUE

필드의 중복값을 방지하여 모든 필드가 고유한 값을 가지도록 강제

기본키와 차이점

- UNIQUE는 NULL을 허용
- 즉, UNIQUE와 NOT NULL을 설정하면 기본키와 유사해진다.




테이블 관련 명령

제약 조건의 추가, 수정, 삭제

제약조건의 추가

```
ALTER TABLE 테이블이름 ADD 제약조건(컬럼이름);
```

제약조건의 수정

```
ALTER TABLE 테이블이름 MODIFY 컬럼이름 자료형 제약조건;
```

제약조건의 삭제

```
ALTER TABLE 테이블이름 DROP CONSTRAINT 제약조건이름;
```

테이블 관련 명령

AUTO_INCREMENT

필드 선언문에 AUTO_INCREMENT라고 선언하면 자동 증가하는 일련번호가 매겨짐

```
CREATE TABLE tSale (
        saleno INT AUTO_INCREMENT PRIMARY KEY,
        customer NCHAR(10),
        product NCHAR(30)
);

INSERT INTO tSale (customer, product) VALUES ('단군', '지팡이');
INSERT INTO tSale (customer, product) VALUES ('고주몽', '고등어');

SELECT * FROM tSale;
```




테이블 관련 명령

AUTO_INCREMENT

```
DELETE FROM tSale WHERE saleno = 2;
```

```
INSERT INTO tSale (customer, product) VALUES ('박혁거세', '계란');
SELECT * FROM tSale;
```

```
INSERT INTO tSale (saleno, customer, product) VALUES (2, '고주몽', '고등어');
SELECT * FROM tSale;
```

테이블 관련 명령

AUTO_INCREMENT

```
ALTER TABLE tSale AUTO_INCREMENT = 100;

INSERT INTO tSale (customer, product) VALUES ('왕건', '너구리');

UPDATE tSale SET product = '짜파게티' WHERE saleno = LAST_INSERT_ID();

SELECT * FROM tSale;
```

테이블 관련 명령

참조 무결성 - 2개의 테이블 생성 (외래키 미설정 상태)

```
CREATE TABLE tEmployee(
        name CHAR(10) PRIMARY KEY,
        salary INT NOT NULL,
        addr VARCHAR(30) NOT NULL
);
INSERT INTO tEmployee VALUES ('아이린', 650, '대구시');
INSERT INTO tEmployee VALUES ('슬기', 480, '안산시');
INSERT INTO tEmployee VALUES ('웬디', 625, '서울시');

CREATE TABLE tProject
(
        projectID INT PRIMARY KEY,
        employee CHAR(10) NOT NULL,
        project VARCHAR(30) NOT NULL,
        cost INT
);
INSERT INTO tProject VALUES (1, '아이린', '홍콩 수출건', 800);
INSERT INTO tProject VALUES (2, '아이린', 'TV 광고건', 3400);
INSERT INTO tProject VALUES (3, '아이린', '매출분석건', 200);
INSERT INTO tProject VALUES (4, '슬기', '경영 혁신안 작성', 120);
INSERT INTO tProject VALUES (5, '슬기', '대리점 계획', 85);
INSERT INTO tProject VALUES (6, '웬디', '노조 협상건', 24);
```

테이블 관련 명령

참조 무결성 - 2개의 테이블 생성 (외래키 미설정 상태)

기본 키와 참조 키 간의 관계가 항상 유지됨을 보장할 수 없다.

```
INSERT INTO tProject VALUES (7, '조이', '원자재 매입', 900);

DELETE FROM tEmployee WHERE name = '아이린';
```

테이블 관련 명령

참조 무결성 - 2개의 테이블 생성 (외래키 설정)

```
DROP TABLE tProject;
DROP TABLE tEmployee;
CREATE TABLE tEmployee
(
	name CHAR(10) PRIMARY KEY,
	salary INT NOT NULL,
	addr VARCHAR(30) NOT NULL
);

INSERT INTO tEmployee VALUES ('아이린', 650, '대구시');
INSERT INTO tEmployee VALUES ('슬기', 480, '안산시');
INSERT INTO tEmployee VALUES ('웬디', 625, '서울시');
```

테이블 관련 명령

참조 무결성 - 2개의 테이블 생성 (외래키 설정)

```
CREATE TABLE tProject(
        projectID INT PRIMARY KEY,
        employee CHAR(10) NOT NULL,
        project VARCHAR(30) NOT NULL,
        cost INT,
        CONSTRAINT FK_emp FOREIGN KEY(employee) REFERENCES tEmployee(name)
);
```

CONSTRAINT 제약조건명 **FOREIGN KEY**(외래키) **REFERENCES** 참조테이블(참조키)

테이블 관련 명령

참조 무결성 - 2개의 테이블 생성 (외래키 설정)

```
INSERT INTO tProject VALUES (1, '아이린', '홍콩 수출건', 800);
INSERT INTO tProject VALUES (2, '아이린', 'TV 광고건', 3400);
INSERT INTO tProject VALUES (3, '아이린', '매출분석건', 200);
INSERT INTO tProject VALUES (4, '슬기', '경영 혁신안 작성', 120);
INSERT INTO tProject VALUES (5, '슬기', '대리점 계획', 85);
INSERT INTO tProject VALUES (6, '웬디', '노조 협상건', 24);
```

테이블 관련 명령

참조 무결성 - 2개의 테이블 생성 (외래키 설정 상태)

기본 키와 참조 키 간의 관계가 항상 유지됨을 보장한다.

```
INSERT INTO tProject VALUES (7, '조이', '원자재 매입', 900);

DELETE FROM tEmployee WHERE name = '아이린';

DROP TABLE tEmployee;
```

테이블 관련 명령

참조 무결성 - 2개의 테이블 생성 (외래키 설정 상태)

기본 키와 참조 키 간의 관계가 항상 유지됨을 보장한다.

```
INSERT INTO tEmployee VALUES ('조이', 330, '제주');
INSERT INTO tProject VALUES (7, '조이', '원자재 매입', 900);

DELETE FROM tProject WHERE employee = '아이린';
DELETE FROM tEmployee WHERE name = '아이린';
```

테이블 관련 명령

참조 무결성 옵션 - 참조된 행이 삭제되거나 업데이트 될 때 수행할 작업을 지정

- ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

- ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

- NO ACTION : 참조된 행이 수정, 삭제될 때 아무런 작업을 하지 않음
- CASCADE : 참조된 행이 수정, 삭제될 때 해당 행을 참조하는 모든 행도 함께 수정, 삭제
- SET NULL : 참조된 행이 수정, 삭제될 때 해당 외래 키를 포함하는 열의 값을 NULL로 설정
- SET DEFAULT : 참조된 행이 수정, 삭제될 때 해당 외래 키를 포함하는 열의 값을 기본값으로 설정

데이터 모델링

관계형 데이터베이스(RDBMS – Relational DataBase Management System)

- 데이터베이스를 테이블의 집합으로 설명하는 데이터베이스 관리 시스템
- 1970년 Codd에 의해 개발
- 상용 RDBMS : Oracle, IBM DB2, MS-SQL Server 등
- 오픈 소스 RDBMS : MySQL, PostgreSQL, SQLite, MariaDB 등

데이터 모델링

관계형 데이터베이스를 구성하는 용어

- 릴레이션(Relation): 정보 저장의 기본 형태가 2차원 구조인 테이블
- 속성(attribute): 테이블의 각 열(Column, Field)
- 도메인(Domain): 속성이 가질 수 있는 값들의 집합
- 튜플(Tuple): 테이블이 한 행을 구성하는 속성들의 집합(Record, Row)
- 카디널리티(Cardinality): 서로 다른 테이블 사이에 대응되는 수 (고유 값의 수)

데이터 모델링

관계형 데이터베이스를 구성하는 용어

- 키(key) : 각 튜플들을 유일하게 식별할 수 있는 속성 또는 속성의 집합
  [예시 : 학번, {학번, 이름}, {학번, 학년}, {학번, 이름, 학과}]

- 후보키(Candidate Key) : 유일성과 최소성을 만족하는 속성 또는 속성의 집합
  [예시 : 학번]

- 기본키(Primary Key) : 후보키 중에서 선정한 키로, 튜플들을 유일하게 식별할 수 있는 키
  * 기본키는 null이거나 중복될 수 없음

- 대체키(Alternate Key) : 기본키를 제외한 나머지 후보키

- 외래키(Foreign Key) : 다른 테이블의 행을 식별할 수 있는 속성

데이터 모델링

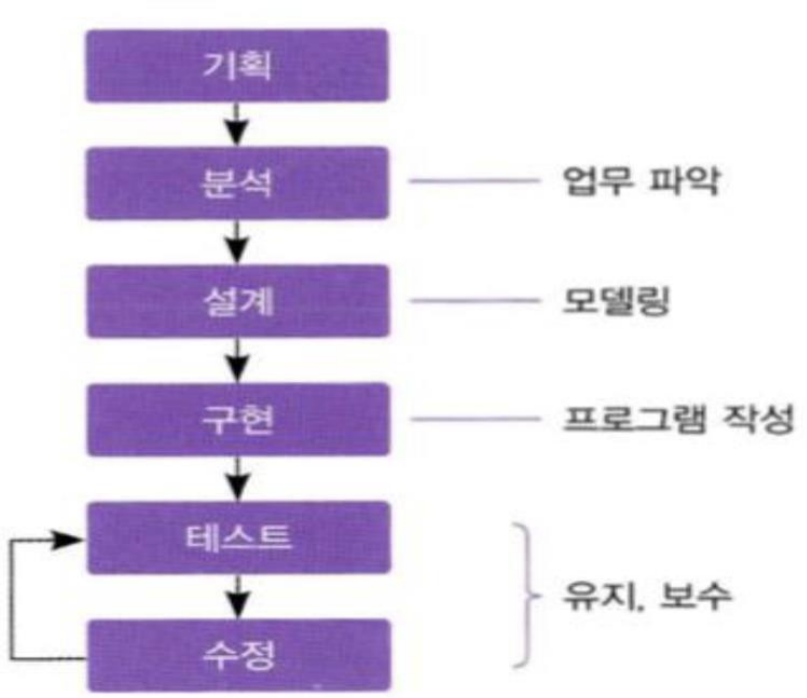
외래키 관계 설정 종류

1 : 1 관계 – 양쪽 테이블의 기본키를 서로 다른 테이블의 외래키로 추가
1 : M 관계 – 1 쪽 테이블의 기본키를, M 쪽 테이블의 외래키로 추가
N : M 관계 – 양쪽 테이블의 기본키를 가지는 별도 테이블을 생성해서 외래키로 설정

* 테이블에서 참조할 수 없는 외래키를 가져서는 안된다.
* 즉, 참조하고 있는 테이블에 존재하는 값이거나 null이어야 한다,

데이터 모델링

모델링 : 복잡한 현실세계를 일정한 표기법에 의해 표현하는 것



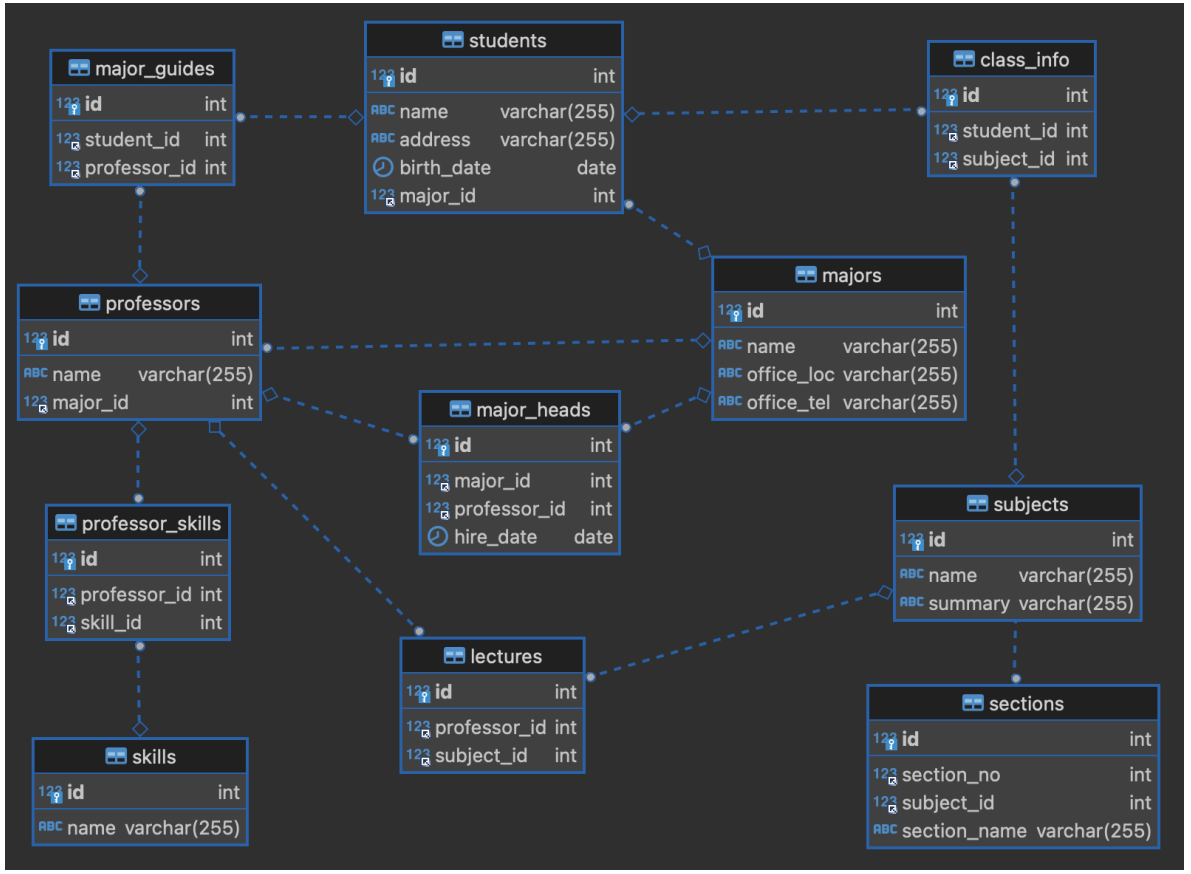
데이터 모델링

ERD (Entity-Relationship Diagram)

- 모든 **학생**은 고유한 학번을 갖고 특정 **학과**에 소속되며 이름, 주소, 생년월일, 나이를 관리
- **학과**는 학과명, 학과사무실 위치, 전화번호 등을 관리하고 학교 내에서 같은 이름의 학과는 없음
- **학생**은 수강할 **과목**을 등록하는데 과목에는 과목번호, 과목명, 과목개요 등이 있음
- **과목**은 여러 섹션으로 나누어질 수 있는데 섹션에는 고유한 섹션번호가 있으며 모든 과목이 섹션으로 나누어지는 것이 아니므로 섹션은 과목이 없으면 존재할 필요가 없고 또한 다른 과목의 섹션은 같은 섹션번호를 가질 수 있음
- **교수**는 교수번호로 식별할 수 있고 교수이름, 전공분야, 보유기술 등을 관리하며 교수는 여러 개의 보유기술을 가질 수 있음
- 교수는 과목을 강의하고 학생에 대해 전공지도를 하는데 일부는 학과의 학과장이 되고 학과마다 학과장은 한 명씩 있음

데이터 모델링

ERD (Entity-Relationship Diagram)



데이터베이스 설계

데이터베이스 설계는 모델링을 통해서
현실 세계의 업무적인 데이터 항목들과 프로세스를 추상화한 것을
실제적으로 구현하여 물리적으로 데이터베이스화 시켜가는 일련의 과정

설계 과정
1. 요구 사항에 대한 분석
2. 개념적 설계
3. 논리적 설계
4. 물리적 설계

데이터베이스 설계

1. 요구 사항에 대한 분석

DB 구축에 필요한 분석 과정으로 구축 대상이 되는 조직에 대한 분석, 부서 간의 업무 영역, 부서의 업무 내용 등을 파악하고 제약 사항은 무엇인지 알아내는 과정

만약, 기존 시스템(Legacy System)이 있다면 이를 분석해서 문제점이나 개선할 내용을 조사하는 것도 이 단계에서 수행한다.

DB 설계의 각 단계 중에서 가장 많은 시간을 할애해야 하는 단계이다.

- 조직이나 기관에 대한 일반 사항(회사 내규라든지 조직도 등에 대한 자료 조사)
- 기존 시스템에 대한 기초 조사{양식지나 컴퓨터 화면 상 입출력 정보 분석)
- 설문지 작성
- 담당자 면담
- 업무 현장 방문
- 기능에 대한 분석(시스템을 몇 개의 기능 단위로 나누고 다시 기능을 세분화한 단위로 나누며 기능과 기능 사이의 관계를 명확히 정의, 데이터 흐름도(DFD, Data Flow Diagram)를 작성하기도 한다.)

데이터베이스 설계

1. 요구 사항에 대한 분석

DB 구축에 필요한 분석 과정으로 구축 대상이 되는 조직에 대한 분석, 부서 간의 업무 영역, 부서의 업무 내용 등을 파악하고 제약 사항은 무엇인지 알아내는 과정

만약, 기존 시스템(Legacy System)이 있다면 이를 분석해서 문제점이나 개선할 내용을 조사하는 것도 이 단계에서 수행한다.

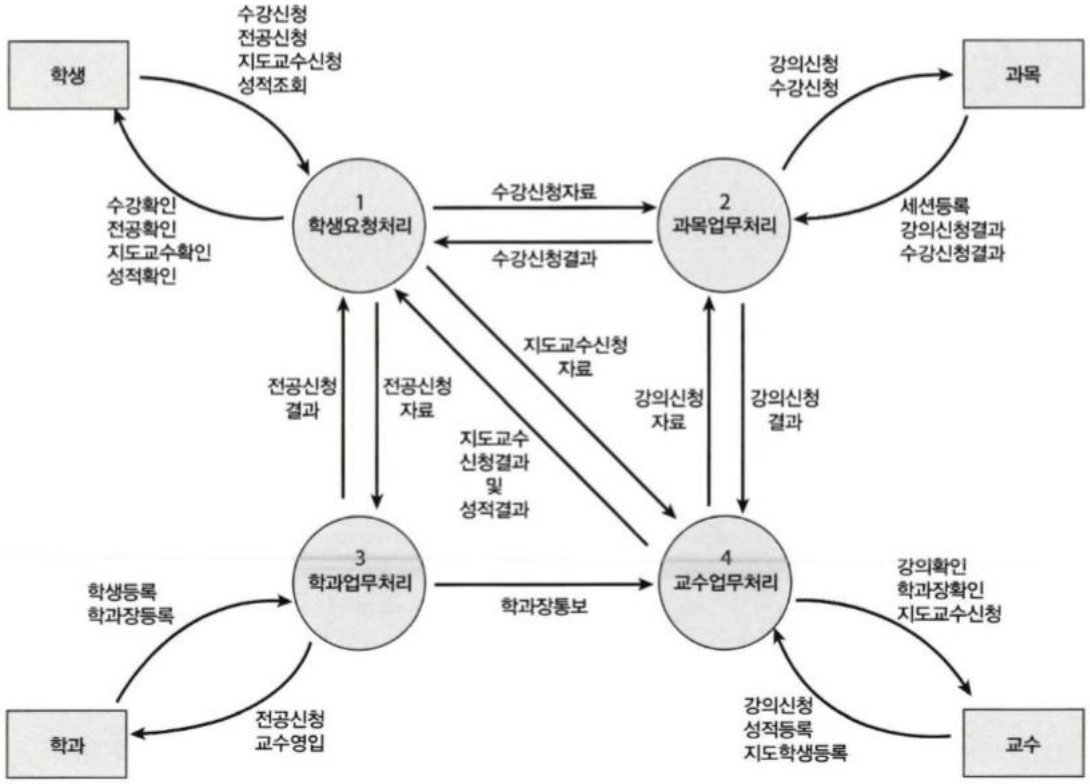
DB 설계의 각 단계 중에서 가장 많은 시간을 할애해야 하는 단계이다.

- 조직이나 기관에 대한 일반 사항(회사 내규라든지 조직도 등에 대한 자료 조사)
- 기존 시스템에 대한 기초 조사{양식지나 컴퓨터 화면 상 입출력 정보 분석)
- 설문지 작성
- 담당자 면담
- 업무 현장 방문
- 기능에 대한 분석(시스템을 몇 개의 기능 단위로 나누고 다시 기능을 세분화한 단위로 나누며 기능과 기능 사이의 관계를 명확히 정의, 데이터 흐름도(DFD, Data Flow Diagram)를 작성하기도 한다.)

데이터베이스 설계

1. 요구 사항에 대한 분석

데이터 흐름도(DFD, Data Flow Diagram)



데이터베이스 설계

2. 개념적 설계

개념적 설계는 상향식(Bottom Up)과 하향식(Top Down)으로 나뉜다.

- 상향식 설계 : DB를 구축하려는 대상에서 데이터 항목을 모두 추출해 낸 후에 비슷한 애트리뷰트를 그룹 지어 엔티티를 만들어서 설계하는 방식

- 하향식 설계 : DB를 구축하려는 대상을 전체적으로 분석한 후에 엔티티, 애트리뷰트, 관계를 추출해 내서 설계하는 방식

보통 하향식으로 엔티티를 추출해 낸 후에 상향식으로 하향식에서 아직 분석하지 못한 내용이나 애트리뷰트 등을 보완한다. 개념적 데이터 모델인 ER 모델을 사용해서 설계한 산출물은 ER 다이어그램을 작성한다.

개념적 설계는 DB 관리 시스템의 종류에 상관없다.
즉, DB 시스템이 갖는 제약조건에 구애받지 않는다.

데이터베이스 설계

3. 논리적 설계

DB 관리 시스템을 선정하고 나서는 개념적 설계에서 얻은 산출물을 DB 관리 시스템의 스키마로 변환하는 작업을 하게 되는데 개념 스키마를 논리적 스키마로 변환하는 작업

DB 관리 시스템을 무엇으로 할지 선정하고, 정해진 DB 관리 시스템의 스키마를 만든다.

DB 관리 시스템을 선정할 때는 정책적인 사항을 비롯해서 기술성과 경제성 등을 고려해야 한다.
- 정책적인 사항 : 시장 점유율이나 기존 시스템과의 호환성, 동일 분야에서 많이 사용되는가 등
- 기술적인 사항 : 데이터 모델에 따라서 사용되는 언어, 제공되는 기능, 구현 도구, 인터페이스 등을 결정

데이터베이스 설계

4. 물리적 설계

주어진 응용 프로그램에 대한 성능을 향상시키기 위해서 DB 파일의 저장 구조와 접근 경로를 결정

가장 효율적인 DB 시스템을 구축하고자 모든 DB 관리 시스템에 대해 가장 적합한 구조를 선택해야 하고 최소한의 응답 시간과 저장 공간의 효율성을 고려해야 한다.

- 응답 시간 : 질의에서 응답까지의 시간. 보통 평균 응답 시간과 부하가 걸려있을 때의 응답 시간을 모두 고려
- 저장 공간 : 질의 중간 과정에서나 최종 저장 메모리를 모두 고려

물리적 설계 단계에서 수행하는 작업 : 인덱스와 역정규화
- 인덱스 : DB 내의 레코드에 쉽게 접근하기 위해서 원하는 데이터를 좀 더 빨리 찾게 도와주는 DB의 객체 중 하나로 인덱스의 종류에는 기본 인덱스, 클러스터링 인덱스, 다단계 인덱스, B-tree 인덱스 등이 있음
- 역정규화 : 정규화 때문에 분리된 테이블들을 참조할 때 과도한 조인(Join) 연산이 발생하는 것을 방지하기 위해서 정규화에 어긋나는 행위를 하는 것으로 유형에는 칼럼 역정규화, 테이블 분리, 테이블 통합, 요약 테이블 생성 등이 있음

데이터베이스 설계

무결성 제약 조건 (Integrity Constraint)

무결성 : 데이터의 내용이 서로 모순되는 일이 없고 데이터베이스에 걸린 제약을 완전히 만족하게 되는 성질로 데이터베이스의 정합성과 안정성을 준다.

- 개체 무결성. : 기본키를 구성하는 어떠한 속성값이라도 중복값이나 NULL을 가질 수 없음
- 참조 무결성 : 참조할 수 없는 외래키 값은 가질 수 없음
  (외래키는 NULL을 가질 수 없으며 참조하는 릴레이션의 기본키와 동일해야 함)
- 도메인 무결성 : 각 속성 값은 반드시 정의된 도메인만을 가져야 함

데이터베이스 설계

함수적 종속 (Functional Dependency)

- 완전 함수 종속 (Full Functional Dependency)
  기본키를 구성하는 모든 속성이 포함된 기본키의 집합에 종속된 경우,
  기본키가 하나의 속성으로 구성된 경우는 모든 속성이 기본키에 대하여 무조건 완전 함수적 종속

- 완전 함수 종속 (Partial Functional Dependency)
  기본키가 여러 속성으로 구성되어 있을 경우 기본키를 구성하는 속성 중 일부에 종속되는 경우

- 이행적 함수 종속 (Transitive Functional Dependency)
  X, Y, Z라는 3 개의 속성이 있을 때 X→Y, Y→Z 이란 종속 관계가 있을 경우, X→Z가 성립되는 경우
  X를 알면 Y를 알고, 그를 통해 Z를 알 수 있다.

데이터베이스 설계

이상 현상(Anomaly)

정규화를 거치지 않은 DB 내에 데이터들이 불필요하게 중복되어 릴레이션 조작 시 발생하는 예기치 않은 현상

- 삽입 이상(Insertion Anomaly)
  새 데이터를 삽입하기 위해 불필요한 데이터도 함께 삽입해야 하는 문제

- 변경 이상(Modification Anomaly)
  중복된 데이터들 중 일부만 수정하여 데이터가 불일치하게 되는 모순이 발생하는 문제

- 삭제 이상(Deletion Anomaly)
  데이터를 삭제하면 필요한 데이터까지 함께 삭제하여 데이터가 손실되는 연쇄 삭제 문제

데이터베이스 설계

정규화 (Data Nomalization)

논리 데이터 모델을 일관성이 있고 안정성 있는 자료 구조로 만드는 단계

데이터의 일관성, 최소한의 데이터 중복, 최대한의 데이터 유연성을 위한 방법으로 데이터를 분해하는 과정

데이터 모델의 독립성을 확보하기 위한 방법

정규화를 수행하면 비지니스의 변경에 유연하게 대처해서 데이터 모델의 변경을 최소화 할 수 있음

정규화의 장점
1. 중복값이 줄어듬
2. NULL 값이 줄어듬
3. 복잡한 코드로 데이터 모델을 보완할 필요가 없음
4. 새로운 요구 사항의 발견 과정을 도움
5. 업무 규칙의 정밀한 포착을 보증
6. 데이터 구조의 안정성을 최대화

데이터베이스 설계

정규화 (Data Nomalization)

정규화는 1차 정규화부터 BCNF(Boyce-Codd Normal Form)을 포함한 5차 정규화까지로 구성되어 있다.

정규화의 기본 목표 : 테이블 간의 중복 데이터를 허용하지 않는 것.
일반적으로는 중복이 최소로 발생하는 제3정규화까지 진행한다.

데이터베이스 설계

정규화 (Data Nomalization)

제1정규화 : 한 릴레이션을 구성하는 모든 도메인이 원자값으로 구성

이름	취미들
유재석	인터넷
박명수	영화, 음악
정준하	음악, 쇼핑
정형돈	음악
하하	게임

이름	취미
유재석	인터넷
박명수	영화
박명수	음악
정준하	음악
정준하	쇼핑
정형돈	음악
하하	게임

데이터베이스 설계

정규화 (Data Nomalization)

제2정규화 : 제1정규화를 만족하면서 모든 속성이 기본키에 완전 함수 종속이 되도록 테이블을 분해

학생번호	강좌이름	강의실	성적
501	데이터베이스	1	3.5
401	데이터베이스	1	4.0
402	웹 프로그래밍	2	3.5
501	자료구조	3	4.0
502	자료구조	3	3.5

학생번호	강좌이름	성적
501	데이터베이스	3.5
401	데이터베이스	4.0
402	웹 프로그래밍	3.5
501	자료구조	4.0
502	자료구조	3.5

강좌이름	강의실
데이터베이스	1
웹 프로그래밍	2
자료구조	3

데이터베이스 설계

정규화 (Data Nomalization)

제3정규화 : 제2정규화를 만족하면서 이행적 함수 종속관계를 없애고 비이행적 함수 종속관계를 만족하도록 분해

학생번호	강좌이름	수강료
501	데이터베이스	20000
401	데이터베이스	20000
402	웹 프로그래밍	15000
502	자료구조	25000

학생번호	강좌이름
501	데이터베이스
401	데이터베이스
402	웹 프로그래밍
502	자료구조

강좌이름	수강료
데이터베이스	20000
웹 프로그래밍	15000
자료구조	25000

데이터베이스 설계

정규화 (Data Nomalization)

보이스-코드 정규화(BCNF): 제 3 정규형을 만족하면서 모든 결정자가 후보키가 되도록 테이블을 분해

학생번호	특강이름	교수
501	Git / GitHub	김교수
401	Git / GitHub	김교수
402	포트폴리오	이교수
501	UX /UI	박교수
502	UX /UI	최교수

학생번호	교수
501	김교수
401	김교수
402	이교수
501	박교수
502	최교수

특강이름	교수
Git / GitHub	김교수
포트폴리오	이교수
UX /UI	박교수
UX /UI	최교수

데이터베이스 설계

정규화 (Data Nomalization)

제4정규화 : BCNF를 만족하면서 테이블에서 다치 종속 관계를 제거

* 다치 종속(Multi-valued Dependency) : 같은 테이블 내의 독립적인 두 개 이상의 컬럼이 또 다른 컬럼에 종속되는 것

개발자	자격증	취미
유재석	정보처리기사	복싱
유재석	빅데이터 분석기사	음악
정준하	정보보안기사	독서

개발자	자격증
유재석	정보처리기사
유재석	빅데이터 분석기사
정준하	정보보안기사

개발자	취미
유재석	복싱
유재석	음악
정준하	독서

데이터베이스 설계

정규화 (Data Nomalization)

제5정규화 : 제4정규화를 만족하면서 후보키를 통하지 않은 조인종속(Join Dependency)을 제거

* 조인 종속(Joint Dependency) : 여러 개의 테이블로 분해 후, 다시 조인했을 때 데이터 손실이 없고 필요 없는 데이터가 생기는 것

개발자	자격증	취미
유재석	정보처리기사	복싱
유재석	빅데이터 분석기사	음악
유재석	정보처리기사	음악
유재석	빅데이터 분석기사	복싱
정준하	정보보안기사	독서

개발자	자격증
유재석	정보처리기사
유재석	빅데이터 분석기사
정준하	정보보안기사

자격증	취미
정보처리기사	복싱
빅데이터 분석기사	음악
정보보안기사	독서

개발자	취미
유재석	복싱
유재석	음악
정준하	독서

데이터베이스 설계

정규화 (Data Nomalization)

정규화의 장점
- 업무 변경에 유연한 대처가 가능, 높은 확장성
- 인덱스 수의 감소
- 속성 추가의 가능성이 높을 때 사용

정규화의 단점
- 빈번한 Join 연산의 증가
- 부자연스러운 DB
- 조회/검색 위주의 응용시스템에 부적합

기억장치의 성능이 좋아지고, 저렴해지면서 정규화의 중요성이 저하되고, 성능을 우선 시하게 되는 경향이 있다.