# DataBase

www.jslhrd.com

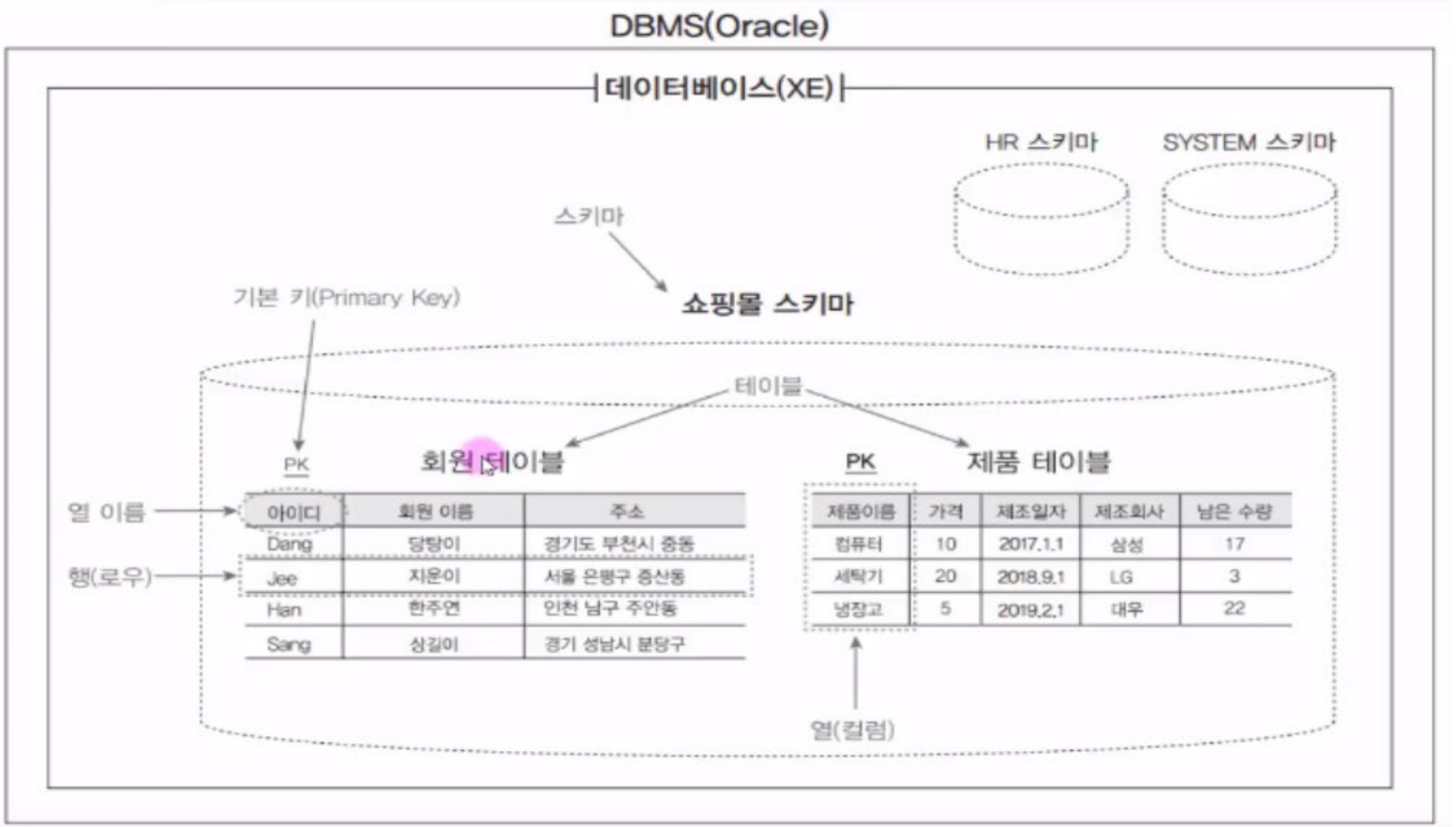
2022. .

JSL 인재개발원

# Part 01.

# DataBase 기본

# ❖ 1. 관계형 DB


DBMS(Oracle)
데이터베이스(XE)
HR 스키마
SYSTEM 스키마
스키마
쇼핑몰 스키마
기본 키(Primary Key)
테이블
PK
회원 테이블
PK
제품 테이블
열 이름
행(로우)
아이디
회원 이름
주소
Dang
당탕이
경기도 부천시 중동
Jee
지은이
서울 은평구 증산동
Han
한주연
인천 남구 주안동
Sang
상길이
경기 성남시 분당구
제품이름
가격
제조일자
제조회사
남은 수량
컴퓨터
10
2017.1.1
삼성
17
세탁기
20
2018.9.1
LG
3
냉장고
5
2019.2.1
대우
22
열(컬럼)

# ❖ 1. 관계형 DB

EMPLOYEE 테이블

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 81/04/02 | 2975 | (null) | 20 |
| 5 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 6 | 7698 | BLAKE | MANAGER | 7839 | 81/05/01 | 2850 | (null) | 30 |
| 7 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 |
| 8 | 7788 | SCOTT | ANALYST | 7566 | 87/07/13 | 3000 | (null) | 20 |
| 9 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 |
| 10 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 11 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 12 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 13 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |
| 14 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |

DEPARTMENT 테이블

| | DNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |
| 5 | 50 | aaaa | bbbb |

# ❖ 1. DDL (데이터 정의어)

1) CREATE : 테이블, 인덱스, 뷰 등 생성

| | |
|---|---|
| 형 식 | CREATE TABLE [schema.] table<br>( column datatype [DEFAULT expression]<br>[column_constraint clause] [,…]) ; |
| 예 제 | CREATE TABLE dept ( dno number(2),<br>dname varchar2(14),<br>loc varchar2(13)) ; |

| | |
|---|---|
| 형 식 | CREATE [OR REPLACE] [FORCE \| NOFORCE] VIEW *view_name*<br>[(*alias, alias, alias, …*)] AS *subquery*<br>[WITH CHECK OPTION] [WITH READ ONLY]; |
| 형 식 | create view v_emp_sample as<br>select eno, ename, job, manager, dno from emp_second; |

| | |
|---|---|
| 예 제 | CREATE INDEX *index_name*<br>ON *table_name (column_name)*; |
| 예 제 | create index idx_employee_ename<br>on employee(ename); |

# ❖ 1. DDL (데이터 정의어)

2) ALTER : 테이블 구조 수정(컬럼 추가, 변경, 삭제)

| | |
|---|---|
| 형 식 | ALTER TABLE table_name ADD ([column_name data_type<br>DEFAULT expr] [, column_name data_type] … ) ; |
| 예 제 | ALTER TABLE dept ADD ( birth DATE ) ; |

| | |
|---|---|
| 형 식 | ALTER TABLE table_name MODIFY [column_name data_type<br>DEFAULT expr] [, column_name data_type] … ) ; |
| 예 제 | ALTER TABLE dept MODIFY ename varchar2(30) ; |

| | |
|---|---|
| 형 식 | ALTER TABLE table_name DROP COLUMN column_ name ; |
| 예 제 | ALTER TABLE dept DROP COLUMN ename ; |

# ❖ 1. DDL (데이터 정의어)

2) DROP : 테이블, 인덱스, 뷰 등 삭제

| | |
|---|---|
| 형 식 | DROP TABLE table_name; |
| 예 제 | drop table emp20 ; |

| | |
|---|---|
| 형 식 | DROP VIEW view_table_name; |
| 예 제 | drop view v_emp_job; |

| | |
|---|---|
| 형 식 | DROP INDEX index_name; |
| 예 제 | drop index emp_index ; |

# ❖ 1. DDL (데이터 정의어)

아래 보기의 <학생> 테이블에 '주소' 컬럼을 추가하는 SQL문을 완성하는 빈칸 ① ~ ②에 알맞은 용어를 기입하시오.(단, 추가 카럼의 이름은 '주소'이고, 데이터 타입은 가변문자형 20자리로 VARCHAR2(20)이다.)

<학생>

| 학번 | 이름 | 학과 | 전화번호 |
|---|---|---|---|
| 2020021 | 철수 | 컴퓨터 | 010-1111-1111 |
| 2020001 | 민수 | 수학 | 010-2222-2222 |
| 2021022 | 영희 | 컴퓨터 | 010-3333-3333 |
| 2022013 | 민호 | 통계 | 010-4444-4444 |

<SQL문>

( ① ) TABLE 학생 ( ② ) 주소 VARCHAR2(20);

답 : ① ALTER ② ADD

# ❖ 1. DDL (데이터 정의어)

다음 주어진 <STUDENT> 테이블의 name 속성을 오름차순하여 idx_name를 인덱스명으로 하는 인덱스를 생성하는 SQL문을 작성하시오

<student>

| sid | name | grade | major | Address |
|---|---|---|---|---|
| 1000 | 홍길동 | 1 | 컴퓨터공학 | 서울 |
| 2000 | 김철수 | 1 | 전기공학 | 경기 |
| 3000 | 이순신 | 2 | 전자공학 | 경기 |
| 4000 | 강희영 | 2 | 컴퓨터공학 | 경기 |
| 5000 | 임꺽정 | 3 | 전자공학 | 서울 |

답 : CREATE INDEX idx_name ON STUDENT(name);

# ❖ 2. DML [ 데이터 조작어 ]

아래 보기의 <학생> 릴레이션의 카디널리티(Cardinality)와 디그리(Degree)를 쓰시오

<학생> 릴레이션

| 학번 | 이름 | 학년 | 전화번호 |
|---|---|---|---|
| 2020021 | 김철수 | 1 | 010-1111-1111 |
| 2020001 | 이민수 | 2 | 010-2222-2222 |
| 2021022 | 강영희 | 2 | 010-3333-3333 |
| 2022023 | 오민호 | 3 | 010-4444-4444 |
| 2020007 | 박광철 | 4 | 010-5555-5555 |

답 : 카디널리티 : 5
디그리 : 4

# ❖ 2. DML [ 데이터 조작어 ]

1) INSERT : 테이블에 데이터를 입력하기 위한 명령어

| 형 식 | INSERT INTO table_name (column_name, … )<br>VALUES (column_value, … ) ; |
|---|---|

2) UPDATE : 테이블, 인덱스, 뷰 등 삭제

| 형 식 | UPDATE table_name SET column_name1 = value1,<br>column_name2 = value2, … WHERE conditions ; |
|---|---|

3) DELETE : 테이블, 인덱스, 뷰 등 삭제

| 형 식 | DELETE [FROM] table_name WHERE conditions ; |
|---|---|

# ❖ 2. DML [ 데이터 조작어 ]

아래 보기의 <학생> 테이블에서 이름이 '민수'인 학생 튜플을 삭제하는 SQL문을 작성하시오(단, 다음의 요구사항을 참고하여 작성하시오)

<요구사항>

1. 이름 속성의 데이터는 문자형이다. 문자형 데이터는 작은따옴표(' ')로 표시하시오
2. SQL 명령문은 대/소문자를 구분하지 않는다.
3. SQL 명령문의 종결 문자인 세미콜론(;) 생략가능합니다.
4. 실행결과가 일치하더라도 <요구사항>을 모두 적용하지 않은 SQL문을 작성하면 오답으로 간주합니다.

<학생>

| 학번 | 이름 | 학과 | 전화번호 |
|---|---|---|---|
| 2020021 | 철수 | 컴퓨터 | 010-1111-1111 |
| 2020001 | 민수 | 수학 | 010-2222-2222 |
| 2021022 | 영희 | 컴퓨터 | 010-3333-3333 |
| 2022013 | 민호 | 통계 | 010-4444-4444 |

답 : DELETE FROM 학생 WHERE 이름 = '민수';

# ❖ 2. DML [ 데이터 조작어 ]

아래 보기의 <학생> 테이블을 대상으로 하는 <지시사항>을 수행하는 SQL 문의 빈칸 ① ~ ②에 알맞은 명령을 쓰시오

<학생>

| 학번 | 이름 | 학년 | 점수 |
|---|---|---|---|
| 211101 | 이영진 | 1 | 70 |
| 191107 | 감희영 | 3 | 80 |
| 191403 | 김철수 | 3 | 90 |
| 181511 | 이영희 | 4 | 100 |

<지시사항>

<학생> 테이블에서 성명이 '강희영'인 학생의 WJATNFMF 92점으로 수정하시오

<SQL 문>

(　　①　　) 학생 (　　②　　) 점수 = 92 WHERE 성명 = '강희영';

답 : (1) : UPDATE　　(2) SET

# ❖ 2. DML ( 데이터 조작어 )

## 1. SELECT ~ FROM 기본 형식

```
SELECT [DISTINCT] {*, column[Alias], . . .}
        FROM table_name     [WHERE condition]
                [GROUP BY group_by_expression]
                        [HAVING group_condition]
                                [ORDER BY column];
```

| 절 | 기능 |
|---|---|
| SELECT 절 | 조회하고자 하는 칼럼명의 리스트를 나열합니다. |
| DISTINCT 절 | 동일한 내용을 한 번씩만 출력하여 중복을 제거합니다. |
| FROM 절 | 조회하고자 하는 테이블명의 리스트를 나열합니다. |
| WHERE 절 | 조회하고자 하는 로우의 조건을 나열합니다. |
| GROUP BY 절 | 동일한 값을 갖는 로우들을 한 그룹으로 묶습니다. |
| HAVING 절 | 로우들의 그룹이 만족해야 하는 조건을 제시합니다. |
| ORDER BY 절 | 로우들의 정렬 순서를 제시합니다, |

# ❖ 2. DML [ 데이터 조작어 ]

테이블 R1에 대한 아래 SQL 문의 실행결과 ?

[ R1 ]

| 학번 | 이름 | 학년 | 학과 | 주소 |
|---|---|---|---|---|
| 1000 | 홍길동 | 1 | 컴퓨터공학 | 서울 |
| 2000 | 김철수 | 1 | 전기공학 | 경기 |
| 3000 | 강남길 | 2 | 전자공학 | 경기 |
| 4000 | 오말자 | 2 | 컴퓨터공학 | 경기 |
| 5000 | 장미화 | 3 | 전자공학 | 서울 |

[ SQL 문 ]

```
SELECT DISTINCT 학년 FROM R1;
```

| 학년 |
|---|
| 1 |
| 2 |
| 3 |

다음 SQL문의 실행결과로 생성되는 튜플 수는?

```
SELECT 급여 FROM 사원;
```

<사원> 테이블

| 사원ID | 사원명 | 급여 | 부서ID |
|---|---|---|---|
| 101 | 박철수 | 30000 | 1 |
| 102 | 한나라 | 35000 | 2 |
| 103 | 김감동 | 40000 | 3 |
| 104 | 이구수 | 35000 | 2 |
| 105 | 최초록 | 40000 | 3 |

실행결과
5

# ❖ 2. DML [ 데이터 조작어 ]

학생(STUDENT) 테이블에 전자과 학생 50명, 정보통신과 학생 100명, 건축과 학생 50명의 정보가 저장되어 있을 때, 다음 ① ~ ③ SQL문의 실행 결과 튜플 수를 쓰시오.(단, DEPT컬럼은 학과명이다)

① SELECT DEPT FROM STUDENT;
② SELECT DISTINCT DEPT FROM STUDENT;
③ SELECT COUNT(DISTINCT DEPT) FROM STUDENT WHERE DEPT='정보통신'

실행결과
(1) 200
(2) 3
(3) 1

# ❖ 2. DML [ 데이터 조작어 ]

## 2. SELECT ~ FROM ~ WHERE

```
SELECT * [column1, column2, .. ,columnn]
        FROM table_name
                        WHERE condition;
```

비교연산자

| 연산자 | 의 미 | 예 제 |
|---|---|---|
| = | 같다 | select eno, ename, salary from employee where salary=1500; |
| > | 크다 | select eno, ename, salary from employee where salary>1500; |
| < | 작다 | select eno, ename, salary from employee where salary<1500; |
| >= | 크거나 같다 | select eno, ename, salary from employee where salary>=1500; |
| <= | 작거나 같다 | select eno, ename, salary from employee where salary<=1500; |
| <>, !=, ^= | 같지 않다 | select eno, ename, salary from employee where salary<>1500; |

# ❖ 2. DML [ 데이터 조작어 ]

다음 [조건]에 부합하는 SQL문을 작성하고자 할 때, [SQL문]의 빈칸에 들어갈 내용으로 옳은 것은? (단, '팀코드' 및 '이름'은 속성이며, '직원'은 테이블이다.)

```
[조건]
이름이 '정도일'인 팀원이 소속된 팀코드를 이용
하여 해당 팀에 소속된 팀원들의 이름을 출력하는
SQL문 작성
```

```
[SQL문]
SELECT    이름
FROM      직원
WHERE     팀코드 = (          );
```

SELECT 팀코드 FROM 직원WHERE 이름 = '정도일'

# ❖ 2. DML [ 데이터 조작어 ]

논리 연산자

| 연산자 | 의미 |
| --- | --- |
| AND | 두 가지 조건을 모두 만족해야만 검색할 수 있습니다.<br>select * from employee where dno=10 and job='MANAGER'; |
| OR | 두 가지 조건 중에서 한 가지만 만족하더라도 검색할 수 있습니다.<br>select * from employee where dno=10 or job='MANAGER'; |
| NOT | 조건에 만족하지 못하는 것만 검색합니다.<br>select * from employee where not dno=10; |

# ❖ 2. DML [ 데이터 조작어 ]

## BETWEEN AND 연산자

BETWEEN AND 연산자는 특정 칼럼의 데이터 값이 하한값(A)와 상한값(B) 사이에 포함되는 로우를 검색하기 위한 연산자

*[ 형식 ]*

*column_name* BETWEEN *A* AND *B*

다음은 BETWEEN AND 연산자를 사용하여 급여가 1000에서 1500 사이인 사원을 출력한 예입니다.

```
select * from employee
        where salary between 1000 and 1500;
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 2 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 3 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 4 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 5 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |

# ❖ 2. DML [ 데이터 조작어 ]

## IN 연산자

IN 연산자는 특정 칼럼의 값이 A, B, C 중에 하나라도 일치하면 참이 되는 연산자입니다.

*[ 형식 ]*
*column_name* IN (*A, B, C*)

다음은 커미션이 300이거나 500이거나 1400인 사원을 검색하기 위해서 IN 연산자를 사용한 예입니다.

```
select * from employee
                where commission in(300, 500, 1400);
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 2 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 3 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |

# ❖ 2. DML [ 데이터 조작어 ]

## LIKE 연산자와 와일드카드

LIKE 연산자는 칼럼에 저장된 문자상수 중에서 LIKE 연산자에서 지정한 문자패턴과 부분적으로 일치하면 참이 되는 연산자로 이 씨 성을 갖는 사람을 찾거나 거주지가 서울인 사람을 찾는 것과 같이 칼럼에 저장된 데이터의 일부만 일치하더라도 조회가 가능하도록 하기 위해서 사용합니다.

*[ 형식 ]*
*column_name* LIKE *pattern*

| 와일드카드 | 의미 |
|---|---|
| % | 문자가 없거나, 하나 이상의 문자가 어떤 값이 와도 상관없다. |
| _ | 하나의 문자가 어떤 값이 와도 상관없다. |

select * from employee where **ename like 'F%';**

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |

# ❖ 2. DML [ 데이터 조작어 ]

IS NULL , NOT NULL 연산자

어떤 컬럼을 NULL 즉, 모르는 값과 같다(=)라는 것은 의미상으로 말이 되지 않기 때문에 = 대신 IS NULL 연산자를 사용해야 합니다.

*[ 형식 ]*
대상컬럼 IS (연산자) NULL(비교값)

예) 다음은 커미션이 NULL인 사원 출력

```
select * from employee  where commission is null;
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 |
| 2 | 7566 | JONES | MANAGER | 7839 | 81/04/02 | 2975 | (null) | 20 |
| 3 | 7698 | BLAKE | MANAGER | 7839 | 81/05/01 | 2850 | (null) | 30 |
| 4 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 |
| 5 | 7788 | SCOTT | ANALYST | 7566 | 87/07/13 | 3000 | (null) | 20 |
| 6 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 |
| 7 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 8 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 9 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |
| 10 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |

# ❖ 2. DML ( 데이터 조작어 )

예) 다음은 커미션이 NULL 아닌 사원 출력

```
select * from employee  where commission is  not  null;
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 2 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 3 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 4 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |

# ❖ 2. DML [ 데이터 조작어 ]

결과 값이 아래와 같을 때 SQL 질의로 옳은 것은?

[공급자] Table

| 공급자번호 | 공급자명 | 위치 |
|---|---|---|
| 16 | 대신공업사 | 수원 |
| 27 | 삼진사 | 서울 |
| 39 | 삼양사 | 인천 |
| 62 | 진아공업사 | 대전 |
| 70 | 신촌상사 | 서울 |

[결과]

| 공급자번호 | 공급자명 | 위치 |
|---|---|---|
| 16 | 대신공업사 | 수원 |
| 70 | 신촌상사 | 서울 |

SELECT * FROM 공급자 WHERE공급자명 LIKE '%신%';

# ❖ 2. DML [ 데이터 조작어 ]

아래 보기의 <학생)> 테이블을 대상으로 <요구사항>을 적용하여 출력하는 SQL문을 작성하시오.(단, 이름 속성의 데이터는 문자형이고, 학번과 학년 속성의 데이터는 숫자형(int)이다.

<학생>

| 학번 | 이름 | 학년 |
|---|---|---|
| 181101 | KKK | 1 |
| 171201 | HHH | 2 |
| 171302 | XXX | 3 |
| 161107 | YYY | 3 |
| 151403 | QQQ | 4 |

<요구사항>

1. <학생> 테이블에서 학년이 3학년이거나 4학년 학생의 학번과 이름을 검색하시오.
2. 단, 조건절 작성시 in(value1, value2) 문법을 사용하여 작성하시오
2. 실행 결과가 일치하더라도 <요구사항>을 적용하지 않은 SQL문을 작성하면 오답으로 간주합니다.

답 : SELECT 학번, 이름 FROM 학생 WHERE 학번 IN(3, 4);
SELECT 학번, 이름 FROM 학생 WHERE 학번 = 3 OR 학번 = 4;

# ❖ 2. DML [ 데이터 조작어 ]

<EMP> 테이블을 대상으로 다음 SQL문을 적용한 출력 결과를 쓰시오

<EMP>

| EMPNO | ENAME | AGE | SAL | EDPT_ID |
|---|---|---|---|---|
| 100 | 홍길동 | 25 | 1000 | 20 |
| 200 | 강감찬 | 40 | 3000 | 30 |
| 300 | 이순신 | 42 | 2000 | 40 |
| 400 | 강희영 | 25 | 2500 | 40 |

<SQL문>

```
SELECT  COUNT(*)  FROM EMP
        WHERE EMPNO > 100 AND SAL >= 3000 OR  EMPNO = 200;
```

답 : 1

# ❖ 2. DML [ 데이터 조작어 ]

## 3. 정렬을 위한 ORDER BY 절

```
SELECT select_list
    [ FROM table_source ] [ WHERE search_condition ]
        ORDER BY order_expression [ ASC | DESC ]
```

| | ASC(오름차순) | DESC(내림차순) |
|---|---|---|
| 숫자 | 작은 값부터 정렬 | 큰 값부터 정렬 |
| 문자 | 사전 순서로 정렬 | 사전 반대 순서로 정렬 |
| 날짜 | 빠른 날짜 순서로 정렬 | 늦은 날짜 순서로 정렬 |
| NULL | 가장 마지막에 나온다. | 가장 먼저 나온다. |

# ❖ 2. DML ( 데이터 조작어 )

예) 급여 컬럼을 기준으로 오름차순으로 정렬

```
select * from employee  order by salary asc;
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 |
| 2 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 3 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 4 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 5 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 6 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |
| 7 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 8 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 9 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 |
| 10 | 7698 | BLAKE | MANAGER | 7839 | 81/05/01 | 2850 | (null) | 30 |
| 11 | 7566 | JONES | MANAGER | 7839 | 81/04/02 | 2975 | (null) | 20 |
| 12 | 7788 | SCOTT | ANALYST | 7566 | 87/07/13 | 3000 | (null) | 20 |
| 13 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |
| 14 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 |

# ❖ 2. DML ( 데이터 조작어 )

예) 급여가 같은 사람이 존재할 경우 이름의 철자가 빠른 사람부터 출력되도록 하려면 정렬 방식을 여러 가지로 지정

```
select * from employee order by salary desc, ename asc;
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 |
| 2 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |
| 3 | 7788 | SCOTT | ANALYST | 7566 | 87/07/13 | 3000 | (null) | 20 |
| 4 | 7566 | JONES | MANAGER | 7839 | 81/04/02 | 2975 | (null) | 20 |
| 5 | 7698 | BLAKE | MANAGER | 7839 | 81/05/01 | 2850 | (null) | 30 |
| 6 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 |
| 7 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 8 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 9 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |
| 10 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 11 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 12 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 13 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 14 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 |

# ❖ 2. DML [ 데이터 조작어 ]

예) 급여가 같은 사람이 존재할 경우 이름의 철자가 빠른 사람부터 출력되도록 하려면 정렬 방식을 여러 가지로 지정(단 급여가 2000미만인 사원만 출력)

```
select * from employee where salary < 2000
                        order by salary desc, ename asc ;
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 2 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 3 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |
| 4 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 5 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 6 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 7 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 8 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 |

# ❖ 2. DML [ 데이터 조작어 ]

아래 보기의 <학생> 테이블에 이름 속성이 '이'로 시작하는 학생들의 학번을 검색하여 학년이 높은 학생 순으로(내림차순) 출력하도록 SQL 문의 빈칸 ① ~ ②에 알맞은 명령을 쓰시오

<학생>

| 학번 | 이름 | 학년 |
|---|---|---|
| 1000 | 이영진 | 1 |
| 2000 | 홍순신 | 2 |
| 3000 | 김감찬 | 3 |
| 4000 | 김희영 | 3 |
| 5000 | 이철수 | 3 |
| 6000 | 이영희 | 4 |

<SQL 문>

```
SELECT 학번 FROM 학생 WHERE 이름 LIKE (  ①  )
                ORDER BY 학번 (  ②  );
```

답 : (1) : '%이'  (2) DESC

# ❖ 3. 그룹 함수

## 1. 그룹함수

- 테이블의 전체 데이터에서 통계적인 결과를 구하기 위해서 행 집합에 적용하여 하나의 결과를 생산
- 하나 이상의 행을 그룹으로 묶어 연산하여 하나의 결과를 구한다.

| 구 분 | 설 명 |
|---|---|
| SUM | 그룹의 누적 합계를 반환합니다. |
| AVG | 그룹의 평균을 반환합니다. |
| COUNT | 그룹의 총 개수를 반환합니다. |
| MAX | 그룹의 최대값을 반환합니다. |
| MIN | 그룹의 최소값을 반환합니다. |

예 1) 테이블의 행 개수를 반환

```
select count(*) as "사원의 수“  from employee;
```

예 2)

```
select sum(salary) as "급여총액",  avg(salary) as "급여평균",
            max(salary) as "최대급여",  min(salary) as "최소급여“  from employee;
```

예 3) 중복되지 않은 직업 수

```
select count(distinct job) as "직업 종류의 개수“  from employee;
```

# ❖ 3. 그룹 함수

## 2. 데이터 그룹 : GROUP BY

- 특정 칼럼을 기준으로 그룹별로 나눌 때 사용

```
SELECT 칼럼명, 그룹함수  FROM 테이블명
      WHERE 조건 (연산자)   GROUP BY 칼럼명;
```

- 부서번호 별 급여 평균

```
select dno as "부서 번호", avg(salary) as "급여 평균“  from employee
                                                    group by dno;
```

- 다중 칼럼을 이용한 그룹별로 검색하기

```
select dno, job, count(*), sum(salary)  from employee
                          group by dno, job   order by dno, job;
```

- 표시할 그룹을 지정하여 집계 정보를 기준으로 그룹 결과를 제한

```
select dno, max(salary) from employee  group by dno
                                  having max(salary) >= 3000;
```

# ❖ 3. 그룹 함수

<학생> 테이블을 대상으로 <요구사항>을 적용하여 아래 <결과>와 같이 출력하는 SQL문을 작성하시오.

<학생>

| 학번 | 이름 | 학과 | 성적 | 전화번호 |
|---|---|---|---|---|
| 2020021 | 철수 | 전기 | 90 | 010-1111-1111 |
| 2020001 | 민수 | 컴퓨터 | 70 | 010-2222-2222 |
| 2021022 | 영희 | 건축 | 85 | 010-3333-3333 |
| 2022023 | 민호 | 건축 | 95 | 010-4444-4444 |
| 2020007 | 광철 | 컴퓨터 | 100 | 010-5555-5555 |

<결과>

| 학과 | 학과별튜플수 |
|---|---|
| 전기 | 1 |
| 컴퓨터 | 2 |
| 건축 | 2 |

<요구사항>

1. WHERE 구문은 사용하지 않고 SQL문을 작성히시오
2. GROUP BY 구문과 집계함수를 반드시 사용하여 SQL문을 작성하시오.
3. 인용 문구를 사용 시 작은따옴표(' ') 사용가능합니다.
4. AS 구문은 반드시 사용하여 작성하시오.
5. SQL 명령문은 대/소문자를 구분하지 않으며, SQL 명령문의 종결 문자인 세미콜론(;) 생략가능합니다.
6. 실행결과가 일치하더라도 <요구사항>을 모두 적용하지 않은 SQL문을 작성하면 오답으로 간주합니다.

답 : SELECT 학과 , COUNT(*) AS 학과별튜플수 FROM 학생 GROUP BY 학과;

# ❖ 3. 그룹 함수

<성적> 테이블을 대상으로 <요구사항>을 적용하여 아래 <결과>와 같이 출력하는 SQL문을 작성하시오.

<성적>

| 학번 | 과목번호 | 과목이름 | 학점 | 점수 |
|---|---|---|---|---|
| 100 | 2000 | 데이터베이스 | A | 95 |
| 101 | 1000 | 자료구조 | B | 80 |
| 102 | 2000 | 데이터베이스 | A | 99 |
| 103 | 2000 | 데이터베이스 | B | 88 |
| 104 | 1000 | 자료구조 | C | 79 |

<결과>

| 과목이름 | 최소점수 | 최대점수 |
|---|---|---|
| 데이터베이스 | 88 | 99 |

<요구사항>

1. <성적> 테이블에서 과목별 평균 점수가 90점 이상인 과목이름, 최소점수, 최대점수를 출력하시오
2. 단, WHERE 구문은 사용 불가능하며, GROUP BY, HAVING, AS 구문을 반드시 포함하여 작성하시오.
3. SQL 명령문은 대/소문자를 구분하지 않는다.
4. SQL 명령문의 종결 문자인 세미콜론(;) 생략가능합니다.
5. 실행결과가 일치하더라도 <요구사항>을 모두 적용하지 않은 SQL문을 작성하면 오답으로 간주합니다.

답 : SELECT 과목이름, MIN(점수) AS 최소점수, MAX(점수) AS 최대점수
FROM 성적 GROUP BY 과목이름 HAVING AGV(점수) >= 90 ;

# ❖ 4. 조인

- 여러 테이블에 저장된 데이터를 한 번에 조회해야 할 필요가 있을 때 사용

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 81/04/02 | 2975 | (null) | 20 |
| 5 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 6 | 7698 | BLAKE | MANAGER | 7839 | 81/05/01 | 2850 | (null) | 30 |
| 7 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 |
| 8 | 7788 | SCOTT | ANALYST | 7566 | 87/07/13 | 3000 | (null) | 20 |
| 9 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 |
| 10 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 11 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 12 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 13 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |
| 14 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |

| | DNO | DNAME | LOC |
|---|---|---|---|
| 1 | 10 | ACCOUNTING | NEW YORK |
| 2 | 20 | RESEARCH | DALLAS |
| 3 | 30 | SALES | CHICAGO |
| 4 | 40 | OPERATIONS | BOSTON |
| 5 | 50 | aaaa | bbbb |

## 1. 카디시안 곱(Cartesian Product)

- 특별한 키워드 없이 SELECT 문의 FROM 절에 사원(employee) 테이블과 부서(department) 테이블을 콤마로 연결하여 연속하여 기술(컬럼 수 : 두 테이블 컬럼수 + , 로우 수 : 두 테이블의 로우스 * )

```
SELECT *  from department, employee;
```

# ❖ 4. 조인

## 2. EQUI JOIN

- 조인 대상 테이블에서 공통 칼럼을 '='(equal) 비교를 통해 같은 값을 가지는 행을 연결

```
SELECT table1.column, table2.column    FROM table1, table2
                                  WHERE table1.column1 = table2.column2;
```

- 각 사원들이 소속된 부서정보 얻기

```
select *   from employee, department
                          where employee.dno = department.dno;
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO | DNO_1 | DNAME | LOC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 | 20 | RESEARCH | DALLAS |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 | 30 | SALES | CHICAGO |
| 3 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 | 30 | SALES | CHICAGO |
| 4 | 7566 | JONES | MANAGER | 7839 | 81/04/02 | 2975 | (null) | 20 | 20 | RESEARCH | DALLAS |
| 5 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 | 30 | SALES | CHICAGO |
| 6 | 7698 | BLAKE | MANAGER | 7839 | 81/05/01 | 2850 | (null) | 30 | 30 | SALES | CHICAGO |
| 7 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 | 10 | ACCOUNTING | NEW YORK |
| 8 | 7788 | SCOTT | ANALYST | 7566 | 87/07/13 | 3000 | (null) | 20 | 20 | RESEARCH | DALLAS |
| 9 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 | 10 | ACCOUNTING | NEW YORK |
| 10 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 | 30 | SALES | CHICAGO |
| 11 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 | 20 | RESEARCH | DALLAS |
| 12 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 | 30 | SALES | CHICAGO |
| 13 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 | 20 | RESEARCH | DALLAS |
| 14 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 | 10 | ACCOUNTING | NEW YORK |

# ❖ 4. 조인

## 3. EQUI JOIN ~ USING

- USING 절에 조인 대상이 되는 칼럼을 지정
- 컬럼명은 조인 대상 테이블에서 동일한 이름으로 정의되어 있어야 한다.

```
SELECT table1.column, table2.column  FROM table1 JOIN table2
                                                    USING(column);
```

- 사원번호가 7788인 사원의 사원번호, 이름, 부서이름, 부서번호 출력

```
select e.eno, e.ename, d.dname, dno  from employee e join department d
        using(dno)    where e.eno=7788;
```

| | ENO | ENAME | DNAME | DNO |
|---|---|---|---|---|
| 1 | 7788 | SCOTT | RESEARCH | 20 |

# ❖ 4. 조인

## 4. EQUI JOIN ~ ON

- 임의의 조건을 지정하거나 조인할 칼럼을 지정하려면 ON 절을 사용

- 각 사원들이 소속된 부서정보 얻기

```
select e.eno, e.ename, d.dname, e.dno  from employee e join department d
                on e.dno = d.dno    where e.eno=7788;
```

| | ENO | ENAME | DNAME | DNO |
|---|---|---|---|---|
| 1 | 7788 | SCOTT | RESEARCH | 20 |

## 5. CLOSS JOIN

- **상호 조인**이라고도 불리며, **한 쪽 테이블의 모든 행들과 다른 테이블의 모든 행을 조인**시키는 기능을 한다.
  그래서, CROSS JOIN의 결과 개수는 두 테이블의 행의 개수를 곱한 개수가 된다.
- 카티션 곱 (Cartesian Product)라고도 한다.

# ❖ 4. 조인

다음 R1과 R2의 테이블에서 아래의 실행 결과를 얻기 위한 SQL문은

[R1] 테이블

| 학번 | 이름 | 학년 | 학과 | 주소 |
|---|---|---|---|---|
| 1000 | 홍길동 | 1 | 컴퓨터공학 | 서울 |
| 2000 | 김철수 | 1 | 전기공학 | 경기 |
| 3000 | 강남길 | 2 | 전자공학 | 경기 |
| 4000 | 오말자 | 2 | 컴퓨터공학 | 경기 |
| 5000 | 장미화 | 3 | 전자공학 | 서울 |

[R2] 테이블

| 학번 | 과목번호 | 과목이름 | 학점 | 점수 |
|---|---|---|---|---|
| 1000 | C100 | 컴퓨터구조 | A | 91 |
| 2000 | C200 | 데이터베이스 | A+ | 99 |
| 3000 | C100 | 컴퓨터구조 | B+ | 89 |
| 3000 | C200 | 데이터베이스 | B | 85 |
| 4000 | C200 | 데이터베이스 | A | 93 |
| 4000 | C300 | 운영체제 | B+ | 88 |
| 5000 | C300 | 운영체제 | B | 82 |

[실행결과]

| 과목번호 | 과목이름 |
|---|---|
| C100 | 컴퓨터구조 |
| C200 | 데이터베이스 |

```
SELECT 과목번호, 과목이름 FROM RI, R2
    WHERE R1.학번 = R2.학번
        AND R1.학과='전자공학'
            AND R1.이름 = '강남길';
```

# ❖ 4. 조인

아래 보기의 두 테이블을 결합하여 학생이름과 학과이름을 출력하도록 SQL문의 빈칸 ① ~ ②에 알맞은 명령을 쓰시오

<SQL 문>

```
SELECT A.이름 "학생이름", B.학과명 "학과이름" FROM  학생정보 A  JOIN 학과정보  B
        (     ①      ) A.학과 = B.(     ②     );
```

<학생정보>

| 학번 | 이름 | 학과 | 학년 | 점수 |
|---|---|---|---|---|
| 1122 | 홍길동 | 컴퓨터 | 4 | 80 |
| 2233 | 김길동 | 기계 | 3 | 90 |
| 3344 | 박길동 | 컴퓨터 | 2 | 60 |
| 4455 | 최길동 | 수학 | 1 | 100 |
| 6677 | 강길동 | 법학 | 1 | 70 |

<학과정보>

| 학과 | 전화번호 |
|---|---|
| 컴퓨터 | 02-333-1111 |
| 기계 | 02-333-2222 |
| 수학 | 02-333-3333 |
| 법학 | 02-333-4444 |
| 체육 | 02-333-5555 |

답 : (1) : ON　　(2) 학과

# ❖ 4. 조인

아래 보기의 두 테이블 <A>, <B>에 대하여 다음 SQL문의 수행결과를 쓰시오

<A>

| SNO | NAME | GRADE |
|---|---|---|
| 1000 | SMITH | 1 |
| 2000 | ALLEN | 2 |
| 3000 | SCOTT | 3 |

<B>

| RULE |
|---|
| S% |
| %T% |

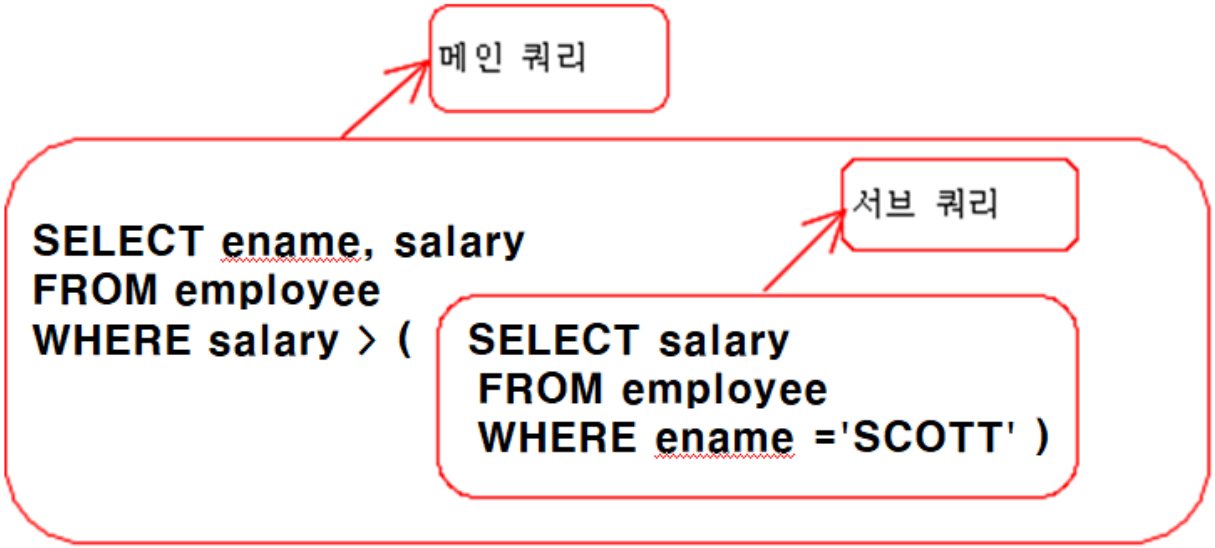
<SQL 문>

```
SELECT  COUNT(*)  CNT FROM A  CROSS JOIN B
                WHERE A.NAME LIKE B.RULE
```

답 :  4

# ❖ 5. 서브 쿼리

## 1. 서브 쿼리 기본 개념




- 단일 행 서브 쿼리
  내부 쿼리문의 결과로 얻어지는 로우가 한 개
  단일 행 비교 연산자(>, =, >=, <, <>, <=)

- 다중 행 서브 쿼리
  내부 쿼리문의 결과로 얻어지는 로우가 여러 개
  다중 행 비교 연산자(IN, ANY, SOM, ALL, EXISTS)

# ❖ 5. 서브 쿼리

## 2. 단일 행 서브 쿼리

```
SELECT select_list    FROM table  WHERE expr operator
                ( SELECT select_list   FROM table    WHERE ……);
```

| | ENO | ENAME | JOB | MANAGER | HIREDATE | SALARY | COMMISSION | DNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 7902 | 80/12/17 | 800 | (null) | 20 |
| 2 | 7499 | ALLEN | SALESMAN | 7698 | 81/02/20 | 1600 | 300 | 30 |
| 3 | 7521 | WARD | SALESMAN | 7698 | 81/02/22 | 1250 | 500 | 30 |
| 4 | 7566 | JONES | MANAGER | 7839 | 81/04/02 | 2975 | (null) | 20 |
| 5 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 6 | 7698 | BLAKE | MANAGER | 7839 | 81/05/01 | 2850 | (null) | 30 |
| 7 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 |
| 8 | 7788 | SCOTT | ANALYST | 7566 | 87/07/13 | 3000 | (null) | 20 |
| 9 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 |
| 10 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 11 | 7876 | ADAMS | CLERK | 7788 | 87/07/13 | 1100 | (null) | 20 |
| 12 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 13 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |
| 14 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |

```
select ename, dno  from employee  where dno =
                (select dno  from employee where ename='SCOTT');
```

# ❖ 5. 서브 쿼리

## 2. 단일 행 서브 쿼리

다음 SQL 문의 실행 결과는?

```
SELECT 가격 FROM 도서가격
  WHERE 책번호 = (SELECT 책번호
    FROM 도서 WHERE 책명='자료구조');
```

[도서]

| 책번호 | 책명 |
|---|---|
| 111 | 운영체제 |
| 222 | 자료구조 |
| 333 | 컴퓨터구조 |

[도서가격]

| 책번호 | 가격 |
|---|---|
| 111 | 20,000 |
| 222 | 25,000 |
| 333 | 10,000 |
| 444 | 15,000 |

가격
25,000

# ❖ 5. 서브 쿼리

## 3. 다중 행 서브쿼리

- 다중 행 서브 쿼리는 서브 쿼리에서 반환되는 결과가 하나 이상의 행일 때 사용하는 서브 쿼리입니다.
  다중 행 서브 쿼리는 반드시 다중 행 연산자(Multiple Row Operator)와 함께 사용해야 합니다.

| 종류 | 의미 |
|---|---|
| IN | 메인 쿼리의 비교 조건('=' 연산자로 비교할 경우)이 서브 쿼리의 결과 중에서 하나라도 일치하면 참입니다. |
| ANY, SOME | 메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 하나 이상이 일치하면 참입니다. |
| ALL | 메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 모든 값이 일치하면 참입니다. |
| EXISTS | 메인 쿼리의 비교 조건이 서브 쿼리의 결과 중에서 만족하는 값이 하나라도 존재하면 참입니다. |

# ❖ 5. 서브 쿼리

예) 부서별 최소 급여를 받는 사원의 사원번호와 이름, 급여을 출력하는 쿼리문

- 부서별 최소 급여

```
select dno, min(salary) from employee  group by dno;
```

| | DNO | MIN(SALARY) |
|---|---|---|
| 1 | 30 | 950 |
| 2 | 20 | 800 |
| 3 | 10 | 1300 |

- 부서별 최소급여을 받는 사원번호, 이름, 급여  출력

```
select eno, ename from employee where salary in
                ( select min(salary) from employee  group by dno);
```

| | ENO | ENAME | SALARY |
|---|---|---|---|
| 1 | 7369 | SMITH | 800 |
| 2 | 7900 | JAMES | 950 |
| 3 | 7934 | MILLER | 1300 |

# ❖ 5. 서브 쿼리

예) 부서별 최소 급여를 받는 사원의 사원번호와 이름, 급여을 출력하는 쿼리문

- 부서별 최소 급여

```
select dno, min(salary) from employee  group by dno;
```

| | DNO | MIN(SALARY) |
|---|---|---|
| 1 | 30 | 950 |
| 2 | 20 | 800 |
| 3 | 10 | 1300 |

- > ANY는 찾아진 값에 대해서 하나라도 크면 참이 되는 셈이 됩니다. 그러므로 찾아진 값 중에서 가장 작은 값 즉, 최소값 보다 크면 참이 됩니다. ( < any : 최대값 보다 작으면 참)

```
select eno, ename, salary  from employee  where salary > any
        ( select min(salary)  from employee  group by dno);
```

| | ENO | ENAME | SALARY |
|---|---|---|---|
| 1 | 7839 | KING | 5000 |
| 2 | 7902 | FORD | 3000 |
| 3 | 7788 | SCOTT | 3000 |
| 4 | 7566 | JONES | 2975 |
| 5 | 7698 | BLAKE | 2850 |
| 6 | 7782 | CLARK | 2450 |
| 7 | 7499 | ALLEN | 1600 |
| 8 | 7844 | TURNER | 1500 |
| 9 | 7934 | MILLER | 1300 |
| 10 | 7521 | WARD | 1250 |
| 11 | 7654 | MARTIN | 1250 |
| 12 | 7876 | ADAMS | 1100 |
| 13 | 7900 | JAMES | 950 |

# ❖ 5. 서브 쿼리

예) 부서별 최소 급여를 받는 사원의 사원번호와 이름, 급여을 출력하는 쿼리문

- 부서별 최소 급여

```
select dno, min(salary) from employee  group by dno;
```

| | DNO | MIN(SALARY) |
|---|---|---|
| 1 | 30 | 950 |
| 2 | 20 | 800 |
| 3 | 10 | 1300 |

- 만족하는 값이 하나라도 존재하면 참입니다.

```
select eno, ename, salary  from employee
        where exists ( select min(salary)  from employee group by dno );
```

| | ENO | ENAME | SALARY |
|---|---|---|---|
| 1 | 7369 | SMITH | 800 |
| 2 | 7499 | ALLEN | 1600 |
| 3 | 7521 | WARD | 1250 |
| 4 | 7566 | JONES | 2975 |
| 5 | 7654 | MARTIN | 1250 |
| 6 | 7698 | BLAKE | 2850 |
| 7 | 7782 | CLARK | 2450 |
| 8 | 7788 | SCOTT | 3000 |
| 9 | 7839 | KING | 5000 |
| 10 | 7844 | TURNER | 1500 |
| 11 | 7876 | ADAMS | 1100 |
| 12 | 7900 | JAMES | 950 |
| 13 | 7902 | FORD | 3000 |
| 14 | 7934 | MILLER | 1300 |

# ❖ 5. 서브 쿼리

다음 SQL문의 실행 결과는?

[학생] 테이블

| 학번 | 이름 | 학년 | 학과 | 주소 |
|---|---|---|---|---|
| 1000 | 김철수 | 1 | 전산 | 서울 |
| 2000 | 고영준 | 1 | 전기 | 경기 |
| 3000 | 유진호 | 2 | 전자 | 경기 |
| 4000 | 김영진 | 2 | 전산 | 경기 |
| 5000 | 정현영 | 3 | 전자 | 서울 |

[성적] 테이블

| 학번 | 과목번호 | 과목이름 | 학점 | 점수 |
|---|---|---|---|---|
| 1000 | A100 | 자료구조 | A | 91 |
| 2000 | A200 | DB | A+ | 99 |
| 3000 | A100 | 자료구조 | B+ | 88 |
| 3000 | A200 | DB | B | 85 |
| 4000 | A200 | DB | A | 94 |
| 4000 | A300 | 운영체제 | B+ | 89 |
| 5000 | A300 | 운영체제 | B | 88 |

```
SELECT 과목이름
FROM 성적
WHERE EXISTS (SELECT 학번
FROM 학생 WHERE 학생.학번 = 성적.학번 AND
학생.학과 IN ('전산', '전기') AND
학생.주소 = '경기');
```

실행결과

| 과목이름 |
|---|
| DB |
| DB |
| 운영체제 |

# ❖ 5. 서브 쿼리

아래의 SQL문을 실행한 결과는?

[R1 테이블]

| 학번 | 이름 | 학년 | 학과 | 주소 |
|---|---|---|---|---|
| 1000 | 홍길동 | 4 | 컴퓨터 | 서울 |
| 2000 | 김철수 | 3 | 전기 | 경기 |
| 3000 | 강남길 | 1 | 컴퓨터 | 경기 |
| 4000 | 오말자 | 4 | 컴퓨터 | 경기 |
| 5000 | 장미화 | 2 | 전자 | 서울 |

[R2 테이블]

| 학번 | 과목번호 | 학점 | 점수 |
|---|---|---|---|
| 1000 | C100 | A | 91 |
| 1000 | C200 | A | 94 |
| 2000 | C300 | B | 85 |
| 3000 | C400 | A | 90 |
| 3000 | C500 | C | 75 |
| 3000 | C100 | A | 90 |
| 4000 | C400 | A | 95 |
| 4000 | C500 | A | 91 |
| 4000 | C100 | B | 80 |
| 4000 | C200 | C | 74 |
| 5000 | C400 | B | 85 |

[SQL 문]

```
SELECT 이름
FROM R1
WHERE 학번 IN
      (SELECT 학번
       FROM R2
       WHERE 과목번호 = 'C100');
```

실행결과

| 이름 |
|---|
| 홍길동 |
| 강남길 |
| 오말자 |

# ❖ 6. 관계대수

## 1. 순수 관계 연산자

다음 R과 S 두 릴레이션에 대한 Division 연산의 수행 결과는?

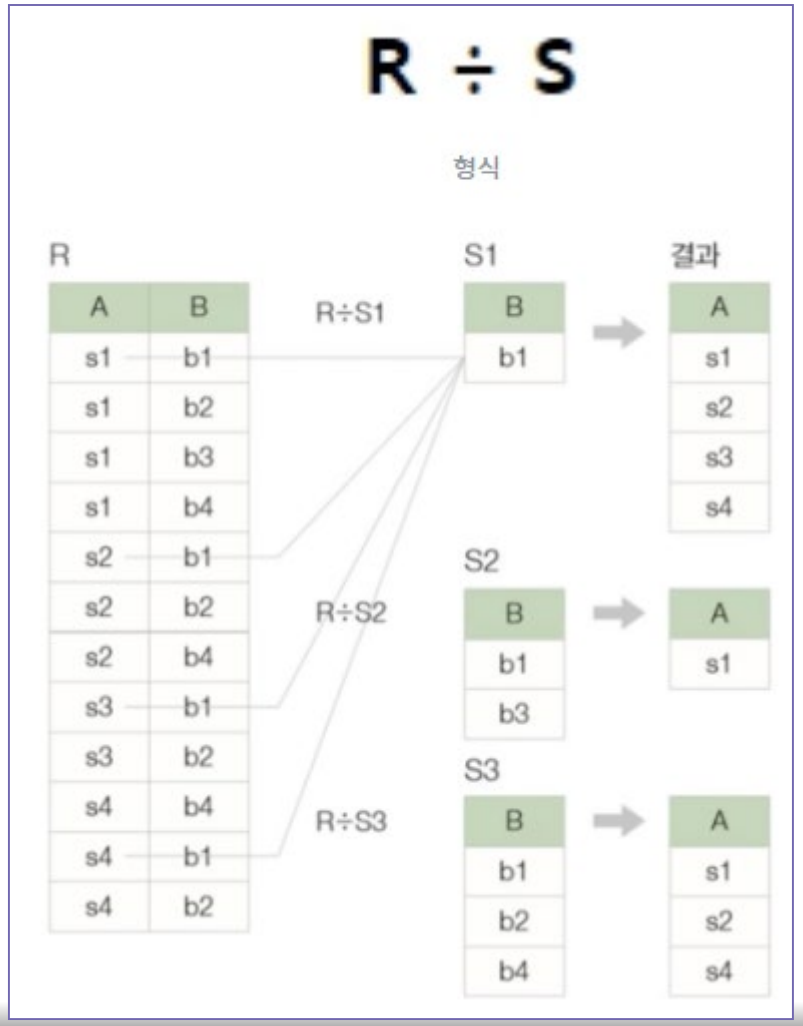
R

| D1 | D2 | D3 |
|---|---|---|
| a | 1 | A |
| b | 1 | A |
| c | 2 | A |
| d | 2 | B |

S

| D2 | D3 |
|---|---|
| 1 | A |

| D1 |
|---|
| a |
| b |



# ❖ 6. 관계대수

## 2. 일반 집합 연산자

집합연산자는 **조회(SELECT) 쿼리의 결과를 대상으로 연산을 수행하는 연산자**이다.
UNION, UNION ALL, INTERSECT, MINUS등이 있다.
조회의 결과를 대상으로 연산을 수행하므로, 여러 개의 SELECT문을 하나의 쿼리로 만드는 연산자라고 할 수 있다

```
순번      과목명      이름
----------------------
1         국어        철수
2         영어        철수
3         수학        철수
1         국어        영희
2         사회        영희
3         과학        영희
4         도덕        영희
5         영어        영희
```

### 1. UNION

UNION은 합집합 개념이다. 철수와 영희가 듣는 과목을 각각 조회한 쿼리를 UNION연산으로 처리하면, 철수와 영희가 같이 듣는 **국어와 영어는 수행 결과 한번만 나오게 된다.** UNION의 사용 예와 결과 예는 아래와 같다.

```
SELECT subject   FROM timetable WHERE name='철수'
UNION
SELECT subject  FROM timetable  WHERE name='영희';
```

결과 예

```
subject
--------
국어
영어
수학
사회
과학
도덕
```

# ❖ 6. 관계대수

```
순번      과목명      이름
----------------------
1         국어        철수
2         영어        철수
3         수학        철수
1         국어        영희
2         사회        영희
3         과학        영희
4         도덕        영희
5         영어        영희
```

2. UNION ALL

UNION ALL은 UNION과 매우 유사하다. UNION과의 차이점은,
중복된 항목도 모두 조회한다는 것이다. 사용 예와 결과 예는 아래와 같다.

```
SELECT subject  FROM timetable  WHERE name='철수'
UNION ALL
SELECT subject  FROM timetable  WHERE name='영희';
```

결과 예

```
subject
--------
국어
영어
수학
국어
사회
과학
도덕
```

# ❖ 6. 관계대수

| 순번 | 과목명 | 이름 |
|---|---|---|
| 1 | 국어 | 철수 |
| 2 | 영어 | 철수 |
| 3 | 수학 | 철수 |
| 1 | 국어 | 영희 |
| 2 | 사회 | 영희 |
| 3 | 과학 | 영희 |
| 4 | 도덕 | 영희 |
| 5 | 영어 | 영희 |

### 3. INTERSECT

INTERSECT는 교집합을 의미한다. 사용 예와 결과 예는 아래와 같다.

```
SELECT subject   FROM timetable  WHERE name='철수'
INTERSECT
SELECT subject  FROM timetable  WHERE name='영희';
```

결과 예

```
subject
--------
국어
영어
```

### 4. MINUS

MINUS는 차집합을 의미한다. **먼저 위치한 SELECT문을 기준**으로, 다른 SELECT문과 공통된 레코드를 제외한 항목만 추출된다. 사용 예와 결과 예는 아래와 같다.

```
SELECT subject  FROM timetable  WHERE name='철수'
MINUS
SELECT subject  FROM timetable  WHERE name='영희';
```

결과 예

```
subject
--------
수학
```

# ❖ 6. 관계대수

테이블 R과 S에 대한 SQL에 대한 SQL문이 실행되었을 때, 실행결과로 옳은 것은?

R

| A | B |
|---|---|
| 1 | A |
| 3 | B |

S

| A | B |
|---|---|
| 1 | A |
| 2 | B |

```
SELECT A FROM R
UNION ALL
SELECT A FROM S;
```

결과

| |
|---|
| 1 |
| 3 |
| 1 |
| 2 |

# ❖ 6. 관계대수

테이블 R1, R2에 대하여 다음 SQL문의결과는? ( 교집합 )

```
(SELECT 학번 FROM R1)
INTERSECT
(SELECT 학번 FROM R2)
```

[R1] 테이블

| 학번 | 학점 수 |
|---|---|
| 20201111 | 15 |
| 20202222 | 20 |

[R2] 테이블

| 학번 | 과목번호 |
|---|---|
| 20202222 | CS200 |
| 20203333 | CS300 |

실행결과

| 학번 |
|---|
| 20202222 |

# ❖ 7. 정규화

다음과 같이 위쪽 릴레이션을 아래쪽 릴레이션으로 정규화를 하였을 때 어떤 정규화 작업을 한 것인가?

| 국가 | 도시 |
|---|---|
| 대한민국 | 서울, 부산 |
| 미국 | 워싱턴, 뉴욕 |
| 중국 | 베이징 |

↓

| 국가 | 도시 |
|---|---|
| 대한민국 | 서울 |
| 대한민국 | 부산 |
| 미국 | 워싱턴 |
| 미국 | 뉴욕 |
| 중국 | 베이징 |

제1정규형

정규화 [ 원 부 이 결 다 조 ]

비정규 릴레이션에서 시작

▼

1정규화 (1NF) - 도메인이 원자값

▼

2정규화 (2NF) - 부분적 함수 종속 제거 (완전 함수 종속 상태)

▼

3정규화 (3NF) - 이행적 함수 종속 제거

▼

BCNF (보이스코드 정규화) - 결정자 (모든 결정자가 후보키인 상태)

▼

4정규화 (4NF) - 다치 종속 제거 (다중 종속 제거)

▼

5정규화 (5NF) - 조인종속 제거