

01. 마당서점의 고객이 요구하는 다음 질문에 대해 SQL 문을 작성하시오.

```
-- (1) 도서번호가 1인 도서의 이름
select bookname
from book
where bookid = 1;
```

```
-- (2) 가격이 20,00원 이상인 도서의 이름
select bookname
from book
where price >= 20000;
```

```
-- (3) 박지성의 총 구매액
select sum(saleprice)
from orders
where custid = (select custid
from customer
where name = "박지성");
```

```
-- (4) 박지성이 구매한 도서의 수
select count(*)
from orders
where custid = 1;
```

```
-- (5) 박지성이 구매한 도서의 출판사 수
select count(distinct publisher) as "출판사 수"
from book
where bookid in (select bookid
from orders
where custid = (select custid
from customer
where name = "박지성"));
```

```
-- (6) 박지성이 구매한 도서의 이름, 가격, 정가와 판매가격의 차이
select bookname as 책이름, price as 가격, price - saleprice as "정가 차이"
from orders join book
on orders.bookid = book.bookid
where custid = (select custid
from customer
where name = "박지성");
```

```
-- (7) 박지성이 구매하지 않은 도서의 이름
select *
from book
where not exists(
    select bookname
```

```

        from orders
        where orders.bookid = book.bookid and orders.custid =
            (select custid
             from customer
             where name = "박지성")
    );

```

02. 마당서점의 운영자와 경영자가 요구하는 다음 질문에 대해 SQL 문을 작성하시오.

```

-- (1) 마당서점 도서의 총 개수
select count(*)
from book;

```

```

-- (2) 마당서점에 도서를 출고하는 출판사의 총 개수
select count(distinct publisher)
from book;

```

```

-- (3) 모든 고객의 이름, 주소
select name, address
from customer;

```

```

-- (4) 2014년 7월 4일~7월7일 사이에 주문받은 도서의 주문번호
select bookid
from orders
where orderdate between date_format("2014-07-04", "%Y-%m-%d") and
date_format("2014-07-07", "%Y-%m-%d");

```

```

-- (5) 2014년 7월 4일~7월7일 사이에 주문받은 도서를 제외한 도서의 주문번호
select bookid
from orders
where orderdate not between date_format("2014-07-04", "%Y-%m-%d") and
date_format("2014-07-07", "%Y-%m-%d");

```

```

-- (6) 성이 '김' 씨인 고객의 이름과 주소
select name, address
from customer
where name like '김%';

```

```

-- (7) 성이 '김' 씨이고 이름이 '아'로 끝나는 고객의 이름과 주소
select name, address
from customer
where name like '김%아';

```

```

-- (8) 주문하지 않은 고객의 이름 (부속질의 사용)
select name

```

```

from customer
where not exists(
    select custid
    from orders
    where customer.custid = orders.custid
);

-- (9) 주문 금액의 총액과 주문의 평균 금액
select sum(orders.saleprice) as "총액", round(avg(orders.saleprice)) as
"평균 금액"
from orders;

-- (10) 고객의 이름과 고객별 구매액
select customer.name, sum(orders.saleprice)
from orders join customer
on orders.custid = customer.custid
group by orders.custid;

-- (11) 고객의 이름과 고객이 구매한 도서 목록
select customer.name, book.bookname
from orders left join customer on orders.custid = customer.custid
left join book on orders.bookid = book.bookid;

-- (12) 도서의 가격 (Book 테이블)과 판매가격 (Orders 테이블)의 차이가 가장 많은
주문
select *
from book join orders on book.bookid = orders.bookid
where price - saleprice like (
    select MAX(price - saleprice)
    from book join orders on book.bookid = orders.bookid
);

-- (13) 도서의 판매액 평균보다 자신의 구매액 평균이 더 높은 고객의 이름
select round(avg(saleprice))
from orders join customer
on orders.custid = customer.custid
group by orders.custid

having avg(saleprice) > (select avg(saleprice) from orders);

```

03. 마당서점에서 다음의 심화된 질문에 대해 SQL 문을 작성하시오.

```

-- 1. 박지성이 구매한 도서의 출판사와 같은 출판사에서 도서를 구매한 고객의 이름
select distinct name
from customer join orders
on customer.custid = orders.custid

```

```

where bookid in (select bookid
from book
where customer.name not like "박지성" and
publisher in (
    select publisher
    from orders join book
    on orders.bookid = book.bookid
    where orders.custid = (
        select custid
        from customer
        where name like "박지성"
    )
));

```

-- 2. 두 개 이상의 서로 다른 출판사에서 도서를 구매한 고객의 이름

```

select name
from customer c1
where 2 >= (
    select count(distinct publisher)
    from orders join book on orders.bookid = book.bookid
    join customer on orders.custid = customer.custid
    where name like c1.name
);

```

-- 3. 전체 고객의 30% 이상이 구매한 도서

```

select bookname
from book b1
where (
    select count(book.bookid)
    from book join orders
    on book.bookid = orders.bookid
    where book.bookid = b1.bookid
) >= 0.3 * (select count(*) from customer);

```

04. 다음 질의에 대해 DDL 문과 DML 문을 작성하시오.

-- (1) 새로운 도서 ('스포츠 세계', '대한미디어', 10000원)이 마당서점에 입고되었다. 삽입이 안 될 경우 필요한 데이터가 더 있는지 찾아보시오.

-- 답 : 북 아이디가 필요하다.

```

insert into book(bookid, bookname, publisher, price) values('북 아이디가 필요하다.', '스포츠 세계', '대한미디어', 10000);

```

-- (2) '삼성당'에서 출판한 도서를 삭제하시오. (safe 옵션 해제)

```

delete from book
where publisher = "삼성당";

```

```

-- (3) '이상미디어'에서 출판한 도서를 삭제하시오. 삭제가 안 되면 원인을
-- 생각해보시오.
-- 답 : orders에 공유 된 키가 있기 때문에 지워지지 않는다. orders에 bookid가
7,8인 경우가 없어야 삭제 가능
delete from book
where publisher = "이상미디어";

-- (4) 출판사 '대한미디어'를 '대한출판사'로 이름을 바꾸시오.
update book
set publisher = '대한출판사'
where publisher = '대한미디어';

-- (5) (테이블 생성) 출판사에 대한 정보를 저장하는 테이블 Bookcompany(name,
address, begin)을 생성하고자 한다.
-- name은 기본키며 VARCHAR(20), address는 VARCHAR(20), begin는 DATE
타입으로 선언하여 생성하시오.
create table Bookcompany(
    name varchar(20) primary key,
    address varchar(20),
    begin date
);

-- (6) (테이블 수정) Bookcompany 테이블에 인터넷 주소를 저장하는 webaddress
속성을 varchar(30)으로 추가하시오.
alter table bookcompany add webaddress varchar(30);

-- (7) Bookcompany 테이블에 임의의 튜플 name = 한빛아카데미, address = 서울시
마포구, begin=1993-01-01, webaddress=http://hanbit.co.kr을 삽입하시오.

insert into bookcompany(name, address, begin, webaddress)
values('한빛아카데미', '서울시 마포구', '1993-01-01', 'http://hanbit.co.kr');

```

05. 다음 Exists 질의의 결과를 보이시오.

```

-- (1) 질의의 결과는 무엇인가?
-- 답 : 책 구매를 하지 않은 고객
select *
from customer c1
where not exists(
    select *
    from orders c2
    where c1.custid = c2.custid
);

-- (2) not을 지우면 질의의 결과는 무엇인가?
-- 답 : 책을 구매한 고객
select *
from customer c1

```

```

where exists (
    select *
    from orders c2
    where c1.custid = c2.custid
);

```

06. [극장 데이터베이스] 다음은 네 개의 지점을 둔 극장 데이터베이스이다. 밑줄 친 속성은 기본키이다. 테이블의 구조를 만들고 데이터를 입력한 후 다음 질의에 대한 **SQL** 문을 작성하시오. 테이블의 구조를 만들 때 다음 제약조건을 반영하여 작성한다.

극장번호	상영관번호	영화제목	가격	좌석수
1	1	어려운 영화	15000	48
3	2	재밌는 영화	8000	110
3	3	멋진 영화	7500	120

극장 테이블

극장번호	상영관번호	고객번호	좌석번호	날짜
3	2	3	15	2014-09-01
3	3	4	16	2014-09-01
1	1	9	48	2014-09-01

상영관 테이블

고객번호	이름	주소
3	홍길동	강남
4	김철수	잠실
9	박영희	강남

예약 테이블

극장번호	극장이름	위치
1	롯데	잠실
2	메가	강남
3	대한	잠실

고객 테이블

● 데이터 베이스 생성 SQL

```

-- 데이터 베이스 생성
create database movie;

-- 데이터 베이스 선택
use movie;

```

```
-- 테이블 생성 : 극장 (theater)
```

```
CREATE TABLE theater (  
    theaterNum INTEGER PRIMARY KEY,  
    theaterName VARCHAR(20),  
    theaterLocation VARCHAR(20)  
);
```

```
-- 테이블 생성 : 고객 (Users)
```

```
CREATE TABLE users (  
    userNum INTEGER PRIMARY KEY,  
    userName VARCHAR(20),  
    userAddress VARCHAR(20)  
);
```

```
-- 테이블 생성 : 상영관 (Cinema)
```

```
CREATE TABLE cinema (  
    theaterNum INTEGER NOT NULL,  
    cinemaNum INTEGER PRIMARY KEY,  
    movieName VARCHAR(20),  
    price INTEGER,  
    seat INTEGER,  
    FOREIGN KEY (theaterNum)  
        REFERENCES theater (theaterNum)  
);
```

```
-- 테이블 생성 : 예약 (reservation)
```

```
CREATE TABLE reservation (  
    theaterNum INTEGER NOT NULL,  
    cinemaNum INTEGER NOT NULL,  
    userNum INTEGER NOT NULL,  
    seatNum INTEGER,  
    date DATE,  
    FOREIGN KEY (theaterNum)  
        REFERENCES theater (theaterNum),  
    FOREIGN KEY (cinemaNum)  
        REFERENCES cinema (cinemaNum),  
    FOREIGN KEY (userNum)  
        REFERENCES users (userNum)  
);
```

```
-- 테이블 데이터 생성 : 극장 (theater)
```

```
insert into theater(theaterNum, theaterName, theaterLocation)  
    values(1,"롯데", "잠실");  
insert into theater(theaterNum, theaterName, theaterLocation)  
    values(2,"메가", "강남");  
insert into theater(theaterNum, theaterName, theaterLocation)  
    values(3,"대한", "잠실");
```

```
-- 테이블 데이터 생성 : 상영관 (cinema)
```

```
insert into cinema(theaterNum, cinemaNum, movieName, price, seat)
```

```

        values(1, 1, "어려운 영화", 15000, 48);
insert into cinema(theaterNum, cinemaNum, movieName, price, seat)
        values(3, 3, "멋진 영화", 7500, 120);
insert into cinema(theaterNum, cinemaNum, movieName, price, seat)
        values(3, 2, "재밌는 영화", 8000, 110);

-- 테이블 데이터 생성 : 고객 (users)
insert into users(userNum, userName, userAddress)
        values(3, "홍길동", "강남");
insert into users(userNum, userName, userAddress)
        values(4, "김철수", "잠실");
insert into users(userNum, userName, userAddress)
        values(9, "박영희", "강남");

-- 테이블 데이터 생성 : 예약 (reservation)
insert into reservation(theaterNum, cinemaNum, userNum, seatNum, date)
        values(3, 2, 3, 15, "2014-09-01");
insert into reservation(theaterNum, cinemaNum, userNum, seatNum, date)
        values(3, 3, 4, 16, "2014-09-01");
insert into reservation(theaterNum, cinemaNum, userNum, seatNum, date)
        values(1, 1, 9, 48, "2014-09-01");

```

- 데이터 베이스 질의 SQL

```

-- (1) 단순질의
-- 1. 모든 극장의 이름과 위치를 보이시오.
SELECT
        theaterName, theaterLocation
FROM
        theater;

-- 2. '잠실'에 있는 극장을 보이시오.
SELECT
        *
FROM
        theater
WHERE
        theaterLocation = '잠실';

-- 3. '잠실'에 사는 고객의 이름을 오름차순으로 보이시오.
SELECT
        userName
FROM
        users
WHERE
        userAddress = '잠실';

-- 4. 가격이 8,000원 이하인 영화의 극장번호, 상영관번호, 영화제목을 보이시오.

```



```

SELECT
    theaterNum, cinemaNum, movieName
FROM
    cinema
WHERE
    price <= 8000;

```

-- 5. 극장 위치와 고객의 주소가 같은 고객을 보이시오.

```

SELECT DISTINCT
    theater.theaterLocation, users.userName
FROM
    theater,
    users
WHERE
    theater.theaterLocation = users.userAddress;

```

-- (2) 집계질의

-- 1. 극장의 수는 몇 개인가?

```

SELECT
    COUNT(*) AS '극장 수'
FROM
    theater;

```

-- 2. 상영되는 영화의 평균 가격은 얼마인가?

```

SELECT
    ROUND(AVG(price))
FROM
    cinema;

```

-- 3. 2014년 9월 1일에 영화를 관람한 고객의 수는 얼마인가?

```

SELECT
    COUNT(*) AS '고객 수'
FROM
    reservation,
    users
WHERE
    reservation.userNum = users.userNum
    AND date = '2014-09-01';

```

-- (3) 부속질의와 조인

-- 1. '대한' 극장에서 상영된 영화제목을 보이시오.

```

SELECT
    movieName
FROM
    theater,
    cinema
WHERE
    theater.theaterName = '대한'
    AND theater.theaterNum = cinema.theaterNum;

```

-- 2. '대한' 극장에서 영화를 본 고객의 이름을 보이시오.

```

SELECT
    userName
FROM
    users
WHERE
    userNum IN (SELECT
        userNum
        FROM
            reservation
        WHERE
            theaterNum = (SELECT
                theaterNum
                FROM
                    theater
                WHERE
                    theaterName = '대한'));

```

-- 3. 대한 극장의 전체 수입을 보이시오.

```

SELECT
    SUM(price)
FROM
    cinema
WHERE
    cinemaNum IN (SELECT
        cinemaNum
        FROM
            reservation
        WHERE
            theaterNum = (SELECT
                theaterNum
                FROM
                    theater
                WHERE
                    theaterName = '대한'));

```

-- (4) 그룹질의

-- 1. 극장별 상영관 수를 보이시오.

```

SELECT
    theater.theaterName, COUNT(*)
FROM
    cinema
    JOIN
        theater ON cinema.theaterNum = theater.theaterNum
GROUP BY cinema.theaterNum;

```

-- 2. '잠실'에 있는 극장의 상영관을 보이시오.

```

SELECT
    *
FROM
    cinema
WHERE

```

```

theaterNum IN (SELECT
                theaterNum
            FROM
                theater
            WHERE
                theaterLocation = '잠실');

-- 3. 2014년09월01일의 극장별 평균 관람 고객 수를 보이시오.
SELECT
    theater.theaterName, COUNT(*)
FROM
    reservation
    JOIN
        theater ON reservation.theaterNum = theater.theaterNum
WHERE
    reservation.date = '2014-09-01'
GROUP BY reservation.theaterNum;

-- 4. 2014년09월01일에 가장 많은 고객이 관람한 영화를 보이시오.
SELECT
    movieName
FROM
    cinema,
    reservation
WHERE
    cinema.theaterNum = reservation.theaterNum
    AND cinema.cinemaNum = reservation.cinemaNum
    AND date LIKE '2014-09-01'
GROUP BY reservation.theaterNum , reservation.cinemaNum;


-- DML
-- 1. 각 테이블에 데이터를 삽입하는 INSERT 문을 하나씩 실행시켜 보시오.
-- (위 insert문 참조)

-- 2. 영화의 가격을 10%씩 인상하시오.
UPDATE cinema
SET
    price = price + (price * 0.1);

```

07. [판매원 데이터베이스] 다음 릴레이션을 보고 물음에 답하시오.

Salesperson은 판매원, Order는 주문, Customer는 고객을 나타낸다. 밑줄 친 속성은 기본키이고 custname과 salesperson은 각각 Customer.name과 Salesperson.name을 참조하는 외래키이다.



```
Salesperson(name, age, salary)
Order(number, custname, salesperson, amount)
Customer(name, city, industrytype)
```

```
-- (1) 테이블을 생성하는 CREATE 문과 데이터를 삽입하는 Insert 문을 작성하시오.
-- 테이블의 데이터 타입은 임의로 정하고, 데이터는 아래 질의의 결과가 나오도록
삽입한다.
```

```
create database 판매원;
use 판매원;
```

```
-- 테이블 생성 : 판매원
create table Salesperson(
    name varchar(20) primary key,
    age integer,
    salary integer
);
```

```
-- 테이블 생성 : 고객
create table customer (
    name varchar(20) primary key,
    city varchar(20),
    industrytype varchar(20)
);
```

-- 테이블 생성 : 주문

```
create table orders (  
    number integer,  
    custname varchar(20),  
    saesperson varchar(20),  
    amount integer,  
    foreign key (custname) references customer(name),  
    foreign key (saesperson) references Salesperson(name)  
);
```

-- 데이터 삽입 : 판매원

```
insert into salesperson(name, age, salary) values('Tom', '26', 10000);  
insert into salesperson(name, age, salary) values('Roy', '32', 15000);  
insert into salesperson(name, age, salary) values('Sally', '24', 9000);  
insert into salesperson(name, age, salary) values('Nancy', '29', 7000);  
insert into salesperson(name, age, salary) values('Clara', '40', 8500);
```

-- 데이터 삽입 : 고객

```
insert into customer(name, city, industrytype) values('Mary', 'LA',  
'개발자');  
insert into customer(name, city, industrytype) values('Carrie', 'LA',  
'요리사');  
insert into customer(name, city, industrytype) values('IU', 'KR',  
'가수');  
insert into customer(name, city, industrytype) values('Diane', 'LA',  
'개발자');  
insert into customer(name, city, industrytype) values('Grace', 'DE',  
'요리사');  
insert into customer(name, city, industrytype) values('Nancy', 'MX',  
'교육자');  
insert into customer(name, city, industrytype) values('Frank', 'US',  
'교육자');
```

-- 데이터 삽입 : 주문

```
insert into orders(number, custname, saesperson, amount) values('1',  
'Mary', 'Tom', '1000');  
insert into orders(number, custname, saesperson, amount) values('2',  
'IU', 'Sally', '2000');  
insert into orders(number, custname, saesperson, amount) values('3',  
'Diane', 'Nancy', '3000');  
insert into orders(number, custname, saesperson, amount) values('4',  
'Grace', 'Roy', '4000');  
insert into orders(number, custname, saesperson, amount) values('5',  
'Nancy', 'Tom', '5000');  
insert into orders(number, custname, saesperson, amount) values('6',  
'Diane', 'Sally', '6000');  
insert into orders(number, custname, saesperson, amount) values('7',  
'Grace', 'Roy', '7000');  
insert into orders(number, custname, saesperson, amount) values('8',  
'Nancy', 'Nancy', '8000');
```

-- (2) 모든 판매원의 이름과 급여를 보이시오. 단, 중복 행은 제거한다.

```
select name, salary
from salesperson;
```

-- (3) 나이가 30세 미만인 판매원의 이름을 보이시오.

```
select name
from salesperson
where age < 30;
```

-- (4) 'S'로 끝나는 도시에 사는 고객의 이름을 보이시오.

```
select name
from customer
where city like "%S";
```

-- (5) 주문을 한 고객의 수 (서로 다른 고객만)를 구하시오.

```
select count(distinct custname)
from orders;
```

-- (6) 판매원 각각에 대하여 주문의 수를 계산하시오.

```
select count(salesperson)
from orders
group by salesperson;
```

-- (7) 'LA'에 사는 고객으로부터 주문을 받은 판매원의 이름과 나이를 보이시오. (부속질의를 사용)

```
select name, age
from salesperson
where name in (
    select salesperson
    from orders
    where custname in (
        select name
        from customer
        where city like 'LA')
);
```

-- (8) 'LA'에 사는 고객으로부터 주문을 받은 판매원의 이름과 나이를 보이시오. (조인 사용)

```
select salesperson.name, salesperson.age
from orders join customer on orders.custname = customer.name
        join salesperson on orders.salesperson =
salesperson.name
where customer.city = 'LA';
```

-- (9) 두 번 이상 주문을 받은 판매원의 이름을 보이시오.

```
select name
from salesperson s1
where 2 <= (
    select count(salesperson)
    from orders
```

```
where orders.salesperson = s1.name  
);
```

-- (10) 판매원 'TOM'의 봉급을 45,000원으로 변경하는 SQL 문을 작성하시오.

```
update salesperson  
set salary = 45000
```

```
where name = 'TOM';
```

08. [기업 프로젝트 데이터베이스]

다음 릴레이션을 보고 물음에
답하시오.

```
Employee(empno, name, phoneno, address, sex, position, deptno)  
Department(deptno, deptname, manager)  
Project(projno, projname, deptno)  
Works(empno, projno, hours-worked)
```

-- (1) 테이블 생성, 데이터 저장

-- 초기화 설정

```
drop database if exists ch03;  
create database ch03;  
use ch03;
```

-- 테이블 생성

```
create table Employee  
(  
    empno int auto_increment primary key,  
    name varchar(10),  
    phoneno varchar(13),  
    address varchar(20),  
    sex varchar(20),  
    position varchar(20),  
    deptno int not null  
);
```

```
create table Department  
(
```

```
deptno int primary key,  
deptname varchar(20),  
manager varchar(20)  
);
```

```
create table Project  
(  
    projno int primary key,  
    projname varchar(20),  
    deptno int  
);
```

```
create table Works  
(  
    empno int not null,  
    projno int not null,  
    hours_worked int  
);
```

-- 외래키 설정

```
alter table Employee add constraint foreign key(deptno) references  
Department(deptno) on delete cascade on update cascade;  
alter table Project add constraint foreign key(deptno) references  
Department(deptno) on delete cascade on update cascade;  
alter table Works add constraint foreign key(empno) references  
Employee(empno) on delete cascade on update cascade;  
alter table Works add constraint foreign key(projno) references  
Project(projno) on delete cascade on update cascade;
```

-- 데이터 저장

```
insert into Department values(1, '개발1팀', '재호');  
insert into Department values(2, '개발2팀', '하윤');  
insert into Department values(3, '서비스1팀', '진우');  
insert into Department values(4, '서비스2팀', '하윤');
```

```
insert into Project values(1, 'Admin Page Project', 1);  
insert into Project values(2, 'Client Page Project', 2);  
insert into Project values(3, 'Ad TV', 3);  
insert into Project values(4, 'Ad Web', 4);
```

```
insert into Employee values(null, '민준', '010-1234-1230', '서울',  
'남자', 'IT', 1);  
insert into Employee values(null, '서준', '010-1234-1231', '부산',  
'남자', 'IT', 2);  
insert into Employee values(null, '예준', '010-1234-1232', '울산',  
'여자', 'service', 3);  
insert into Employee values(null, '도윤', '010-1234-1233', '김포',  
'남자', 'IT', 1);  
insert into Employee values(null, '시우', '010-1234-1234', '서울',  
'여자', 'service', 4);
```



```

insert into Employee values(null, '주원', '010-1234-1235', '부산',
'여자', 'IT', 2);
insert into Employee values(null, '하준', '010-1234-1236', '김포',
'남자', 'service', 4);
insert into Employee values(null, '지호', '010-1234-1237', '울산',
'여자', 'service', 3);
insert into Employee values(null, '지후', '010-1234-1238', '서울',
'남자', 'IT', 1);
insert into Employee values(null, '준서', '010-1234-1239', '서울',
'여자', 'IT', 2);

```

```

insert into Works values(1, 1, 10);
insert into Works values(2, 2, 2);
insert into Works values(3, 3, 40);
insert into Works values(4, 1, 20);
insert into Works values(5, 4, 11);
insert into Works values(6, 2, 5);
insert into Works values(7, 4, 9);
insert into Works values(8, 3, 4);
insert into Works values(9, 1, 4);
insert into Works values(10, 2, 12);

```

```

select * from department d ;
select * from employee e ;
select * from project p ;
select * from works w ;

```

-- (2) 모든 사원의 이름을 보이시오.

```
select name from employee;
```

-- (3) 여자 사원의 이름을 보이시오.

```
select name from employee where sex = '여자';
```

-- (4) 팀장(manager)의 이름을 보이시오.

```
select manager from department;
```

-- (5) 'IT' 부서에서 일하는 사원의 이름과 주소를 보이시오.

```
select name, address
from employee
where `position` = 'IT';
```

-- (6) '홍길동' 팀장(manager) 부서에서 일하는 사원의 수를 보이시오.

```
select count(*)
from employee e,
(
    select deptno
    from department
    where manager = '홍길동'
) dn
where e.deptno = dn.deptno;
```

```

-- (7) 직원들이 일한 시간 수를 부서별, 직원 이름별 오름차순으로 보이시오.
select d.deptname, e.name, w.hours_worked
from works w, employee e, department d
where w.empno = e.empno and e.deptno = d.deptno
order by hours_worked;

-- (8) 두 명 이상의 직원이 참여한 프로젝트의 번호, 이름, 직원의 수를 보이시오.

select re.projno '프로젝트 번호', re.projname '프로젝트 이름', re.count
'staff 수'
from employee e join
(
    select w.empno, w.projno, p.projname, if(count(*) >= 2, 'y', 'n')
'check', count(*) count
    from works w join project p
    on w.projno = p.projno
    group by projno
) re
on e.empno = re.empno
where re.check = 'y';

-- (9) 세 명 이상의 직원이 있는 부서의 직원 이름을 보이시오.
select e1.name as '이름'
from employee e1,
(
    select if(count(*) >= 3, deptno, null) 'check'
    from employee
    group by deptno
) e2

where e1.deptno = e2.check;

```

09. [직원 데이터베이스] 다음은 demo_scott.sql 스크립트에 저장된 ...(중략)

```

SELECT * FROM DEPT D ; -- 부서
SELECT * FROM EMP E ; -- 직원

-- (1) 직원의 이름과 직위를 출력하시오. 단,
-- 직원이 이름은 '직원이름', 직위는 '직원직위' 머릿글이 나오도록 출력한다.
SELECT E.ENAME '직원이름', D.DNAME '직원직위'

```

```
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

-- (2) 30번 부서에 근무하는 모든 사원의 이름과 급여를 출력하시오.

```
SELECT E.ENAME '사원이름', E.SAL
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND D.DEPTNO = 30;
```

-- (3) 사원번호와 이름, 현재 급여, 증가된 급여분 (열 이름은 '증가액'), 10% 인상된 급여 (열 이름은 '인상된 급여')를 사원번호순으로 출력하시오.

```
SELECT ENAME , SAL , IFNULL(SAL+COMM, SAL) '증가액', (SAL+(SAL * 0.1))
'인상된 급여'
FROM EMP
ORDER BY EMPNO ;
```

-- (4) 'S'로 시작하는 모든 사원과 부서번호를 출력하시오.

```
SELECT ENAME , DEPTNO
FROM EMP
WHERE ENAME LIKE 'S%'
```

-- (5) 모든 사원의 최대 및 최소 급여, 합계 및 평균 급여를 출력하시오
-- 열 이름은 각각 MAX, MIN, SUM, AVG로 한다. 단, 소수점 이하는 반올림하여 정수로 출력한다.

```
SELECT MAX(SAL), MIN(SAL), SUM(SAL), ROUND(AVG(SAL))
FROM EMP;
```

-- (6) 업무 이름과 업무별로 동일한 업무를 하는 사원의 수를 출력하시오.
-- 열 이름은 각각 '업무'와 '업무별 사원수'로 한다.

```
SELECT D.DNAME '업무', COUNT(*) '업무별 사원수'
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
GROUP BY D.DNAME;
```

-- (7) 사원의 최대 급여와 최소 급여의 차액을 출력하시오.

```
SELECT (MAX(SAL) - MIN(SAL)) '차액'
FROM EMP;
```

-- (8) 30번 부서의 구성원 수와 사원들 급여의 합계와 평균을 출력하시오.

```
SELECT COUNT(*) '구성원 수', SUM(SAL) '합계', ROUND(AVG(SAL)) '평균'
FROM EMP
WHERE DEPTNO = 30
ORDER BY DEPTNO;
```

-- (9) 평균급여가 가장 높은 부서의 번호를 출력하시오.

```
SELECT DEPTNO DNO, ROUND(AVG(SAL)) AVGSAL
FROM EMP
GROUP BY DEPTNO
ORDER BY AVGSAL DESC
LIMIT 1;
```

-- (10) 세일즈맨 (SALESMAN)을 제외하고, 각 업무별 사원의 총 급여가 3,000 이상인
각 업무에 대해서,

-- 업무명과 각 업무별 평균급여를 출력하시오. 단 평균급여의 내림차순으로 출력한다.

```
SELECT E.JOB '업무명', ROUND(AVG(SAL)) 평균급여
FROM EMP E
WHERE E.JOB != 'SALESMAN'
GROUP BY E.JOB
HAVING SUM(E.SAL) >= 3000
ORDER BY 평균급여 DESC;
```

-- (11) 전체 직원 가운데 직속상관이 있는 사원의 수를 출력하시오.

```
SELECT COUNT(*)
FROM EMP
WHERE NOT MGR IS NULL;
```

-- (12) EMP 테이블에서 이름, 급여, 커미션 (COMM) 금액, 총액 (SAL + COMM)을
구하여 총액이 많은 순서대로 출력하시오.

-- 단, 커미션이 NULL인 사람은 제외한다.

```
SELECT ENAME '이름', SAL '금액', COMM '커미션', (SAL + COMM) 총액
FROM EMP E
WHERE COMM IS NOT NULL
ORDER BY 총액 DESC;
```

-- (13) 각 부서별로 같은 업무를 하는 사람의 인원 수를 구하여 부서번호, 업무 이름,
인원 수를 출력하시오.

```
SELECT DEPTNO '부서번호', JOB '업무 이름', COUNT(*) '인원 수'
FROM EMP
GROUP BY JOB
```

-- (14) 사원이 한 명도 없는 부서의 이름을 출력하시오.

```
SELECT D.DNAME
FROM EMP E RIGHT JOIN DEPT D
ON E.DEPTNO = D.DEPTNO
WHERE E.EMPNO IS NULL;
```

-- (15) 같은 업무를 하는 사람의 수가 네 명 이상인 업무와 인원 수를 출력하시오.

```
SELECT JOB '업무', COUNT(*) '인원 수'
FROM EMP
GROUP BY JOB
HAVING COUNT(*) >= 4;
```

-- (16) 직원번호가 7400 이상 7600 이하인 사원의 이름을 출력하시오.

```
SELECT ENAME
FROM EMP
WHERE EMPNO BETWEEN 7400 AND 7600;
```

-- (17) 사원의 이름과 사원의 부서를 출력하시오.

```
SELECT E.ENAME '이름', D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO ;
```

```
-- (18) 사원의 이름과 팀장(MGR)의 이름을 출력하시오. ? 팀장 이름 테이블이 없다.

-- (19) 사원 SCOTT보다 급여를 많이 받는 사람의 이름을 출력하시오.
SELECT E.ENAME '이름'
FROM EMP E
WHERE E.SAL > (SELECT SAL
FROM EMP E

WHERE ENAME = 'SCOTT');
```

10. [인사부서 데이터베이스] 다음은 어느 기업의 인사부서 데이터베이스... (중략)

10번 문제는 영문 해석 문제가 있습니다.

나름대로 해석하여 풀긴했지만 정확하진 않습니다.

* 11 ~ 20번 문제를 해석해서 댓글에 달아주시면 감사하겠습니다!

```
-- (1) 각 테이블에 대하여 기본키, 외래키를 표시하시오.
USE HR;
```

```
DESC COUNTRIES;
DESC '데이터 베이스 이름';
```

```
-- (2) 테이블의 구조를 살펴보자.
-- 살펴보자
```

```
-- (3) EMPLOYEES와 DEPARTMENTS 테이블에 저장된 튜플의 개수를 출력하시오.
```

```
SELECT COUNT(*)
FROM EMPLOYEES E;
```

```
SELECT COUNT(*)
FROM DEPARTMENTS D ;
```

```
-- (4) EMPLOYEES 테이블에 대한 EMPLOYEE_ID, LAST_NAME, JOB_ID, HIRE_DATE를 출력하시오.
```

```
SELECT EMPLOYEE_ID , LAST_NAME , JOB_ID , HIRE_DATE
FROM EMPLOYEES E ;
```

-- (5) EMPLOYEES 테이블에서 SALARY가 12,000 이상인 직원의 LAST_NAME과 SALARY를 출력하시오.

```
SELECT LAST_NAME , SALARY
FROM EMPLOYEES E
WHERE SALARY >= 12000;
```

-- (6) 부서번호 (DEPARTMENT_ID)가 20 혹은 50인 직원의 LAST_NAME과 DEPARTMENT_ID를

-- LAST_NAME에 대하여 오름차순으로 출력하시오.

```
SELECT LAST_NAME , DEPARTMENT_ID
FROM EMPLOYEES
WHERE DEPARTMENT_ID IN (20, 50)
ORDER BY LAST_NAME;
```

-- (7) LAST_NAME의 세 번째에 A가 들어가는 직원의 LAST_NAME을 출력하시오.

```
SELECT LAST_NAME
FROM EMPLOYEES
WHERE SUBSTR(LAST_NAME, 3,1) = 'A' ;
```

-- (8) 같은 일 (JOB)을 하는 사람의 수를 세어 출력하시오.

```
SELECT JOB_ID ,COUNT(*)
FROM EMPLOYEES E
GROUP BY JOB_ID ;
```

-- (9) 급여 (SALARY)의 최댓값과 최솟값을 차이를 구하시오.

```
SELECT MAX(SALARY) - MIN(SALARY)
FROM EMPLOYEES E ;
```

-- (10) TORONTO에서 일하는 직원의 LAST_NAME, JOB, DEPARTMENT_ID, DEPARTMENT_NAME을 출력하시오.

```
SELECT E.LAST_NAME , J.JOB_TITLE , E.DEPARTMENT_ID , D.DEPARTMENT_NAME
FROM EMPLOYEES E , DEPARTMENTS D ,LOCATIONS L, JOBS J
WHERE E.DEPARTMENT_ID = D.DEPARTMENT_ID
AND D.LOCATION_ID = L.LOCATION_ID
AND E.JOB_ID = J.JOB_ID
AND L.CITY = 'TORONTO';
```

-- (11) CREATE A REPORT TO DISPLAY THE LAST NAME, JOB ID, AND START DATE FOR THE EMPLOYEES

-- WHOSE LAST NAMES ARE MATOS AND TALYORM ORDER THE QUERY IN ASCENDING ORDER BY START DATE.

-- 해석

-- LAST_NAME, JOB_ID, START_DATE를 표시하는 테이블을 생성하는데

-- LAST_NAME이 MATOS 이거나 TALYOR인 데이터를 START DATE를 기준으로 오름차순 정렬후 보여준다.

```
SELECT E.LAST_NAME, E.JOB_ID , JH.START_DATE
FROM EMPLOYEES E JOIN JOB_HISTORY JH
```

```

ON E.JOB_ID = JH.JOB_ID
WHERE LAST_NAME = 'MATOS' OR LAST_NAME = 'TAYLOR'
ORDER BY JH.START_DATE ASC;

```

```

-- (12) THE HR DEPARTMENT NEEDS A REPORT THAT DISPLAYS THE LAST NAME
AND HIRE DATE FOR ALL
-- EMPLOYEES WHO WERE HIRED IN 1994

```

```

-- 해석
-- LAST_NAME, HIRE_DATE를 모두 보여주는 부서의 테이블이 필요한데 HIRED가
1994에 고용된 데이터를 보여줘라
SELECT LAST_NAME , HIRE_DATE
FROM EMPLOYEES
WHERE DATE_FORMAT(HIRE_DATE, '%Y') = '1994'

```

```

-- (13) DISPLAY THE LAST NAME, JOB, AND SALARY FOR ALL EMPLOYEES WHOSE
JOB IS EITHER THAT
-- OF A SALES REPRESENTATIVE OR A STOCK CLERK, AND WHOSE SALARY IS NOT
EQUAL TO $2,500, $3,500, OR $7,000.

```

```

-- 해석
-- LAST_NAME, JOB, SALARY를 보여주는 부서 테이블을 만들려고 한다.
-- 급여가 2500, 3500, 7000과 같지 않은 판매원 (SALES REPRESENTATIVE)과
증권사 (STOCK CLERK) 직원을 보여줘라
SELECT E.LAST_NAME , J.JOB_TITLE , E.SALARY
FROM EMPLOYEES E JOIN JOBS J
ON E.JOB_ID = J.JOB_ID
WHERE E.SALARY NOT IN(2500,3500,7000)
AND (J.JOB_TITLE LIKE 'SALES REPRESENTATIVE'
OR J.JOB_TITLE LIKE 'STOCK CLERK');

```

```

-- (14) DETERMINE THE NUMBER OF MANAGERS WITHOUT LISTING THEM. LABEL
THE COLUMN NUMBER OF MANAGERS.

```

```

-- 해석
-- 관리자 수를 나열하지 않고 보여줄려 한다. 컬럼 넘버와 매니저를 라벨로 보여줘라
(이 문제는 해석을 잘 못하겠네요.)

```

```

SELECT IFNULL(MANAGER_ID, 'NOT MANAGER') 'MANAGER NUMBER' ,COUNT(*)
'COUNT'
FROM EMPLOYEES
GROUP BY MANAGER_ID;

```

```

-- (15) create a report to display the manager number and the salary of
the lowestpaid employee for the manager.
-- exclude anyone whose manager is not known. exclude any groups where
the minimum salary is $6,000 or less.
-- sort the output in descending order of salary.

```

```

-- 해석

```

-- 관리자 번호와 관리자에게 가장 낮은 급여를 지급한 직원의 급여를 표시하는 보고서를 작성합니다.

-- 관리자를 알 수 없는 사람은 제외합니다. 최저 급여가 \$6,000 이하인 그룹은 제외합니다

-- 급여의 내림차순으로 분류한다.

```
select e.manager_id, e.salary
from employees e
where e.salary = (
    select min(salary)
    from employees e
    where e.salary > 6000
)
and e.manager_id is not null
order by e.salary desc;
```

-- (16) Create a report to display the last name and employee number of employee along with their manager's

-- last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively.

-- 해석 (풀지 못한다..?)

-- 보고서를 작성하여 직원의 성과 직원 번호와 관리자의 성과 관리자 번호를 표시합니다.

-- 각각 직원, Emp#, Manager 및 Mgr# 열에 레이블을 표시합니다.

-- manager의 last_name과 manager number를 기입하라는데 manager의 last_name 데이터가 없다.

-- (17) Create a report for the HR department that displays employee last names, department numbers,

-- and all the employee who work in the same department as a given employee.

01. 마당서점의 고객이 요구하는 다음 질문에 대해 SQL 문을 작성하시오.

-- (1) 도서번호가 1인 도서의 이름

```
select bookname
from book
where bookid = 1;
```

-- (2) 가격이 20,00원 이상인 도서의 이름

```
select bookname
```



```

from book
where price >= 20000;

-- (3) 박지성의 총 구매액
select sum(saleprice)
from orders
where custid = (select custid
from customer
where name = "박지성");

-- (4) 박지성이 구매한 도서의 수
select count(*)
from orders
where custid = 1;

-- (5) 박지성이 구매한 도서의 출판사 수
select count(distinct publisher) as "출판사 수"
from book
where bookid in (select bookid
from orders
where custid = (select custid
from customer
where name = "박지성"));

-- (6) 박지성이 구매한 도서의 이름, 가격, 정가와 판매가격의 차이
select bookname as 책이름, price as 가격, price - saleprice as "정가 차이"
from orders join book
on orders.bookid = book.bookid
where custid = (select custid
from customer
where name = "박지성");

-- (7) 박지성이 구매하지 않은 도서의 이름
select *
from book
where not exists(
    select bookname
    from orders
    where orders.bookid = book.bookid and orders.custid =
        (select custid
        from customer
        where name = "박지성")
);

```

02. 마당서점의 운영자와 경영자가 요구하는 다음 질문에 대해 SQL 문을 작성하시오.

-- (1) 마당서점 도서의 총 개수

```
select count(*)  
from book;
```

-- (2) 마당서점에 도서를 출고하는 출판사의 총 개수

```
select count(distinct publisher)  
from book;
```

-- (3) 모든 고객의 이름, 주소

```
select name, address  
from customer;
```

-- (4) 2014년 7월 4일~7월 7일 사이에 주문받은 도서의 주문번호

```
select bookid  
from orders  
where orderdate between date_format("2014-07-04", "%Y-%m-%d") and  
date_format("2014-07-07", "%Y-%m-%d");
```

-- (5) 2014년 7월 4일~7월 7일 사이에 주문받은 도서를 제외한 도서의 주문번호

```
select bookid  
from orders  
where orderdate not between date_format("2014-07-04", "%Y-%m-%d") and  
date_format("2014-07-07", "%Y-%m-%d");
```

-- (6) 성이 '김' 씨인 고객의 이름과 주소

```
select name, address  
from customer  
where name like '김%';
```

-- (7) 성이 '김' 씨이고 이름이 '아'로 끝나는 고객의 이름과 주소

```
select name, address  
from customer  
where name like '김%아';
```

-- (8) 주문하지 않은 고객의 이름 (부속질의 사용)

```
select name  
from customer  
where not exists(  
    select custid  
    from orders  
    where customer.custid = orders.custid  
);
```

-- (9) 주문 금액의 총액과 주문의 평균 금액

```
select sum(orders.saleprice) as "총액", round(avg(orders.saleprice)) as  
"평균 금액"  
from orders;
```

-- (10) 고객의 이름과 고객별 구매액

```
select customer.name, sum(orders.saleprice)  
from orders join customer
```

```
on orders.custid = customer.custid
group by orders.custid;
```

-- (11) 고객의 이름과 고객이 구매한 도서 목록

```
select customer.name, book.bookname
from orders left join customer on orders.custid = customer.custid
         left join book      on orders.bookid = book.bookid;
```

-- (12) 도서의 가격 (Book 테이블)과 판매가격 (Orders 테이블)의 차이가 가장 많은 주문

```
select *
from book join orders on book.bookid = orders.bookid
where price - saleprice like (
    select MAX(price - saleprice)
    from book join orders on book.bookid = orders.bookid
);
```

-- (13) 도서의 판매액 평균보다 자신의 구매액 평균이 더 높은 고객의 이름

```
select round(avg(saleprice))
from orders join customer
on orders.custid = customer.custid
group by orders.custid
```

```
having avg(saleprice) > (select avg(saleprice) from orders);
```

03. 마당서점에서 다음의 심화된 질문에 대해 SQL 문을 작성하시오.

-- 1. 박지성이 구매한 도서의 출판사와 같은 출판사에서 도서를 구매한 고객의 이름

```
select distinct name
from customer join orders
on customer.custid = orders.custid
where bookid in (select bookid
from book
where customer.name not like "박지성" and
publisher in (
    select publisher
    from orders join book
    on orders.bookid = book.bookid
    where orders.custid = (
        select custid
        from customer
        where name like "박지성"
    )
));
```

-- 2. 두 개 이상의 서로 다른 출판사에서 도서를 구매한 고객의 이름

```

select name
from customer c1
where 2 >= (
    select count(distinct publisher)
    from orders join book on orders.bookid = book.bookid
    join customer on orders.custid = customer.custid
    where name like c1.name
);

```

-- 3. 전체 고객의 30% 이상이 구매한 도서

```

select bookname
from book b1
where (
    select count(book.bookid)
    from book join orders
    on book.bookid = orders.bookid
    where book.bookid = b1.bookid

```

```

) >= 0.3 * (select count(*) from customer);

```

04. 다음 질의에 대해 DDL 문과 DML 문을 작성하시오.

-- (1) 새로운 도서 ('스포츠 세계', '대한미디어', 10000원)이 마당서점에 입고되었다. 삽입이 안 될 경우 필요한 데이터가 더 있는지 찾아보시오.

-- 답 : 북 아이디가 필요하다.

```

insert into book(bookid, bookname, publisher, price) values('북 아이디가 필요하다.', '스포츠 세계', '대한미디어', 10000);

```

-- (2) '삼성당'에서 출판한 도서를 삭제하시오. (safe 옵션 해제)

```

delete from book
where publisher = "삼성당";

```

-- (3) '이상미디어'에서 출판한 도서를 삭제하시오. 삭제가 안 되면 원인을 생각해보시오.

-- 답 : orders에 공유 된 키가 있기 때문에 지워지지 않는다. orders에 bookid가 7,8인 경우가 없어야 삭제 가능

```

delete from book
where publisher = "이상미디어";

```

-- (4) 출판사 '대한미디어'를 '대한출판사'로 이름을 바꾸시오.

```

update book
set publisher = '대한출판사'
where publisher = '대한미디어';

```

-- (5) (테이블 생성) 출판사에 대한 정보를 저장하는 테이블 Bookcompany(name, address, begin)을 생성하고자 한다.

-- name은 기본키며 VARCHAR(20), address는 VARCHAR(20), begin는 DATE 타입으로 선언하여 생성하시오.

```

create table Bookcompany(
    name varchar(20) primary key,
    address varchar(20),
    begin date
);

-- (6) (테이블 수정) Bookcompany 테이블에 인터넷 주소를 저장하는 webaddress
속성을 varchar(30)으로 추가하시오.
alter table bookcompany add webaddress varchar(30);

-- (7) Bookcompany 테이블에 임의의 튜플 name = 한빛아카데미, address = 서울시
마포구, begin=1993-01-01, webaddress=http://hanbit.co.kr을 삽입하시오.

insert into bookcompany(name, address, begin, webaddress)
values ('한빛아카데미', '서울시 마포구', '1993-01-01', 'http://hanbit.co.kr');

```

05. 다음 Exists 질의의 결과를 보이시오.

```

-- (1) 질의의 결과는 무엇인가?
-- 답 : 책 구매를 하지 않은 고객
select *
from customer c1
where not exists(
    select *
    from orders c2
    where c1.custid = c2.custid
);

-- (2) not을 지우면 질의의 결과는 무엇인가?
-- 답 : 책을 구매한 고객
select *
from customer c1
where exists(
    select *
    from orders c2
    where c1.custid = c2.custid
);

```

06번 - SQL 명령문 분류 (DML, DDL, DCL)

명령어 분류

CREATE DDL

ALTER DDL

DROP DDL

SELECT DML

INSERT DML

DELETE DML

UPDATE DML

GRANT DCL

REVOKE DCL

07번 - 관계대수 → SQL 문으로 변환

관계 $R(A, B, C)$, $S(C, D, E)$ 에 대해:

1. $\sigma_{A=2}(R)$

→ `SELECT * FROM R WHERE A = 2;`

2. $\pi_{A, B}(R)$

→ `SELECT A, B FROM R;`

3. $R \bowtie_{\{R.C = S.C\}} S$

→ `SELECT * FROM R INNER JOIN S ON R.C = S.C;`

08번 - 조인 결과 작성 (테이블 분석)

조인 결과 예상

1. **INNER JOIN**: `C = C` 기준으로 일치하는 것만 출력

- 일치하는 C 값: c1, c2

- 결과: a1/c1, a2/c1, a3/c2 (곱해짐 → 총 5개 행)

2. **LEFT OUTER JOIN**: R의 모든 행 + S에서 매칭되는 값

- R의 각 C 값: c1, c1, c2, c4

- c4는 S에 없음 → NULL 포함

3. **RIGHT OUTER JOIN**: S의 모든 행 + R에서 매칭되는 값

- S의 C: c1, c1, c2, c5

- c5는 R에 없음 → NULL 포함

4. **FULL OUTER JOIN**: R과 S 모두 포함 (MySQL 미지원)

- 모든 조합 포함 → NULL 포함 가능

5. CROSS JOIN: R × S (카티션 곱)

○ 4행 × 4행 = 16행

09번 - SQL 순차 실행 결과

sql

```
INSERT INTO R(col1, col2) VALUES('ABCD', NULL);
```

```
INSERT INTO R(col1, col2) VALUES('BC', NULL);
```

```
ALTER TABLE R MODIFY col2 INT DEFAULT 10;
```

```
INSERT INTO R(col1, col2) VALUES('XY', NULL); -- col2: DEFAULT 10
```

적용됨

```
INSERT INTO R(col1) VALUES('EFG'); -- col2: DEFAULT 10
```

적용됨

```
SELECT SUM(col2) FROM R;
```

- $NULL + NULL + 10 + 10 \rightarrow 20$
- 정답: 20

10번 : 최종 결과 행 수: 8

11번 : 2 | 2