1.기본 **table, col** 이름 만들기

```
CREATE TABLE        ex1_1테이블이름(
  컬럼    데이터타입    제약조건,
  컬럼    데이터타입    제약조건
);
```

시노님

```
CREATE OR REPLACE PUBLIC SYNONYM channels_synonym
FOR channels;

SELECT COUNT (*)
FROM channels_synonym;

GRANT SELECT ON channels_synonym TO PUBLIC; -- PUBLIC 권한 설정
GRANT SELECT ON channels_synonym ora_user;  -- PRIVATE 권한 설정
ora_user사용자에게 channels_synonym 시노님에 대한 조회 권한을 부여함

SELECT *
FROM channels_synonym;

DROP SYNONYM channels_synonym;
DROP PUBLIC SYNONYM channels_synonym;
```

***시퀀스

```
DROP SEQUENCE seq_1;

CREATE SEQUENCE seq_1
INCREMENT BY    2
START WITH      3
MINVALUE    2 -- 사이클이 다시 시작할 땐, minivalue부터 시작
MAXVALUE    10
CYCLE
```

```
NOCACHE
;

DELETE ex2_8;

DESC ex2_8;

seq_1.CURRVAL
seq_1.NEXTVAL

--의사컬럼과 같이 사용
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B1', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B2', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B3', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B4', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B5', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B6', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B7', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B8', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B9', 'C2');
INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B10', 'C2');

INSERT INTO ex2_8 VALUES (seq_1.NEXTVAL, 'B11', 'C2');

SELECT seq_1.CURRVAL FROM DUAL;
SELECT seq_1.NEXTVAL FROM DUAL; --10이상 안된다 그래서 오류

SELECT * FROM ex2_8;

SELECT *
FROM SALES PARTITION (SALES_Q1_1998);
```

2장 문제풀이

```
1번문제 -- OK
DROP TABLE ORDERS;
```

```
CREATE TABLE ORDERS (
      ORDER_ID     NUMBER(12, 0) PRIMARY KEY,
      ORDER_DATE   DATE,
      ORDER_MODE   VARCHAR2(8 byte) CONSTRAINTS check1 CHECK(ORDER_MODE IN ('direct',
'online')),
      CUSTOMER_ID  NUMBER(6, 0),
      ORDERS_STATUS    NUMBER(2, 0),
      ORDER_TOTAL  NUMBER(8, 2) DEFAULT 0,
      SALES_REP_ID NUMBER(6,0),
      PROMOTION_ID NUMBER(6,0)
      );
-- CONSTRAINTS check1를 컬럼명 옆에 쓸 경우 생략가능 but 맨 밑에 쓸 경우 써야한다
예시)

DROP TABLE ORDERS;
CREATE TABLE ORDERS (
      ORDER_ID     NUMBER(12, 0),
      ORDER_DATE   DATE,
      ORDER_MODE   VARCHAR2(8 byte),
      CUSTOMER_ID  NUMBER(6, 0),
      ORDERS_STATUS    NUMBER(2, 0),
      ORDER_TOTAL  NUMBER(8, 2) DEFAULT 0,
      SALES_REP_ID NUMBER(6,0),
      PROMOTION_ID NUMBER(6,0),
      CONSTRAINTS pk_order_id PRIMARY KEY (ORDER_ID),
      CONSTRAINTS ck_order_mode CHECK(ORDER_MODE IN ('direct', 'online'))
);
<풀이>

CREATE TABLE 테이블명 (
   컬럼명 데이터타입 (제약조건),   --쉼표 꼭 적어주기
   CONSTRAINTS 명칭 제약조건 (컬럼)
);                    -- 잊지말고 테이블명 옆에 바로 써주기

SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE
FROM USER_CONSTRAINTS -- 제약조건 포함한 시스템 테이블
WHERE table_name = 'ORDERS'
```

;

INSERT INTO ORDERS(ORDER_ID, ORDER_MODE, PROMOTION_ID)
VALUES (1, 'offline', 11); -- 체크 제약조건(ORA_USER.CHECK1)이 위배되었습니다

INSERT INTO ORDERS(ORDER_ID, ORDER_MODE, PROMOTION_ID)
VALUES (4, 'online', 11);

INSERT INTO ORDERS(ORDER_ID, ORDER_MODE, PROMOTION_ID)
VALUES (1, 'direct', 11); -- 무결성 제약 조건(ORA_USER.SYS_C007471)에 위배됩니다

2번문제 -- OK
- 제약사항: 기본키는 3개, 디폴트 0

DROP TABLE ORDER_ITEMS;

CREATE TABLE ORDER_ITEMS (
	ORDER_ID	NUMBER(12, 0),
	LINE_ITEM_ID	NUMBER(3, 0),
	PRODUCT_ID	NUMBER(3, 0),
	UNIT_PRICE	NUMBER(8, 2) DEFAULT 0,
	QUANTITY	NUMBER(8, 0) DEFAULT 0,
	CONSTRAINTS P_ID PRIMARY KEY (ORDER_ID, LINE_ITEM_ID)
); ---어떻게 써야 하지? 이게 맞나? -- 정답

INSERT INTO ORDER_ITEMS(ORDER_ID, LINE_ITEM_ID, PRODUCT_ID)
VALUES(1, 1, 11);

INSERT INTO ORDER_ITEMS(ORDER_ID, LINE_ITEM_ID, PRODUCT_ID)
VALUES(1, 22, 11);

INSERT INTO ORDER_ITEMS(ORDER_ID, LINE_ITEM_ID, PRODUCT_ID)
VALUES(1, 33, 11);

3번문제

```
CREATE TABLE PROMOTIONS(
   PROMO_ID    NUMBER(6, 0) PRIMARY KEY,
   PROMO_NAME  VARCHAR2(20)
);
```

5번 문제
시퀀스 의사컬럼을 사용해 테이블 ORDERS1의 ORDER_ID 컬럼에 적절한 데이터
--(3행)을 삽입하시오

```
DROP SEQUENCE ORDERS1_SEQ;

CREATE SEQUENCE ORDERS1_SEQ
INCREMENT BY 1
START WITH 1
MINVALUE 1
MAXVALUE 10000
CYCLE
NOCACHE;

DROP TABLE ORDERS1;

DELETE FROM ORDERS1;

CREATE TABLE ORDERS1(
      ORDER_ID    NUMBER,
      COL2        VARCHAR(20));

INSERT INTO ORDERS1(order_id) VALUES(ORDERS1_SEQ.NEXTVAL);
INSERT INTO ORDERS1(order_id) VALUES(ORDERS1_SEQ.NEXTVAL);
INSERT INTO ORDERS1(order_id) VALUES(ORDERS1_SEQ.NEXTVAL);

DESC ORDERS1;

SELECT * FROM ORDERS1;
```

SELECT ORDERS1_SEQ.CURRVAL FROM DUAL;

1.SELECT 문 **-** 조회하다

SELECT * 혹은 컬럼
FROM 테이블
WHERE 조건
ORDER BY 컬럼

2. INSERT 문 **- 3**가지

1) 기본형태: 특정 컬럼의 데이터를 지정해서 넣고 싶을 때, 반드시 컬럼과 값의 수와 순서, 데이터 타입이 일치해야한다
INSERT INTO 테이블**(**컬럼1, 컬럼2,....**)**
VALUES**(**값1, 값2,…….**);**

2) 컬럼명 기술 생략 형태: VALUES절에 테이블의 컬럼 순서대로 값을 삽입
INSERT INTO 테이블
VALUES**(**값1, 값2,…….**);**

3) INSERT ~ SELECT 형태: values 값을 일일이 명시하는 대신 쓰는 구절, select 다음 순서와 타입이 같아야한다, 컬럼을 생략할 경우 이 테이블의 모든 컬럼에 값을 넣겠다는 의미
INSERT INTO 테이블**(**컬럼1, 컬럼2,....**)**
SELECT 문**;**

2. DELETE 문

DELETE
FROM
;

4.MERGE 문

목표: 2000년 10월부터 12월 사이에 월 매출을 기준으로
적정 매출을 달성한 사원에게 더 많은 보너스를 지급함
사용 테이블: 사원 employees, 판매 sales

Q) 해당 월에 매출을 달성한 사원은 누구인가?
매출을 달성한 사원 목록을 ex3_3 테이블에 삽입
사원번호가 사원테이블과 판매 테이블에 둘 다 있어야 함
2000년도 10월~ 12월 사이에 월 매출을 기준으로 함

```
CREATE TABLE ex3_3
employee_id     NUMBER,
bonus_amt       DEFAULT 0
;

INSERT INTO   ex3_3
SELECT        employee_id
FROM          employees e, sales s
WHERE e.employee_id = s.employee_id
  AND    s.sales_month = BETWEEN 200010 AND 200012
GROUP BY e.employee_id
```

* 전체를 의미

Q) 사원 테이블을 검색해 select
1. 관리자사번이 146번인 사원을 찾아 where조건
2. ex3_3테이블에 있는 사원의 사번과 일치하면where조건, 보너스 금액bonus_amt에 자신의 급여salary의 1%를 보너스로 갱신하고, 3. ex3_3 테이블에 있는 사원의 사번과 일치하지 않으면 1 의 결과의 사원을

신규로 입력 (이 때 보너스 금액은 급여의 0.1%로 한다) 하는데, 이때 급여가 8000 미만인 사원만 처리해 보자.

```
SELECT      employee_id
FROM        sales
WHERE       manager_id=146

SELECT      employee_id, manager_id, salary, salary * 0.01%
FROM        employees
WHERE e.employee_id = e.employee_id

– 보너스 1% 받는 사람 명단 및 보너스 금액

SELECT      employee_id, manager_id, salary, salary*0.01%
FROM        employees
WHERE employee_id IN (SELECT employee_id FROM ex3_3);

– 보너스 0.1% 받는 사람 명단 및 보너스 금액

SELECT      employee_id, manager_id, salary, salary*0.001%
FROM        employees
WHERE employee_id NOT IN (SELECT employee_id FROM ex3_3);



MERGE INTO          ex3_3
USING
ON
WHEN MATCHED THEN
WHERE
DELETE WHERE
WHEN NOT MATCHED THEN
INSERT      VALUES
WHERE
```

<3장 문제 풀이 방법>

1.문제해석 > 정보찾기 (동사) , 정보분리
2.쿼리문 형태(문법)작성

1번
ex3_6 테이블을 만들고, 사원테이블에서
관리자사번이 124번이고 급여가 2천에서 3천 사이에 있는
사원의 사번, 사원명, 급여, 관리자사번을 입력하는
INSERT문을 작성해보자

```
DROP TABLE ex3_6;

CREATE TABLE ex3_6(
      employee_id  NUMBER,
      emp_name     VARCHAR2(30),
      salary       NUMBER,
      manager_id   NUMBER
      );

INSERT INTO ex3_6
   SELECT  employee_id, emp_name, salary, manager_id
   FROM    employees
   WHERE   manager_id = 124
      AND salary BETWEEN 2000 AND 3000
;
```

2번문제

다음을 입력

```
   DELETE ex3_3;

   INSERT INTO ex3_3 (employee_id)
   SELECT e.employee_id
   FROM    employees e, sales s
   WHERE   e.employee_id = s.employee_id
      AND s.SALES_MONTH BETWEEN'200010' AND '200012'
```

```
   GROUP BY e.employee_id
   ;
```

관리자 사번이 145번이 사원을 찾아
위 테이블에 있는 사원 사번과 일치하면
보너스 금액bonus_amt에 자신의 급여의 1%를 보너스로 갱신하고
ex3_3테이블에 있는 사원의 사원과 일치하지 않은 사원을
신규입력 (이때 보너스는 급여의 0.5% 한다)

> 사원찾기 - SELECT
> 갱신하고 - UPDATE
> 신규입력 - INSERT
>MERGE입력 - MERGE

테이블 ex3_3, employees
컬럼 manager_id, employee_id, bonus_amt, salary
manager_id = 145
bonus_amt = bonus_amt + salary *0.01
bonus_amt = bonus_amt + salary *0.005

수업에서 푼거,

```
SELECT employee_id, salary
FROM employees
WHERE manager_id = 145;
MERGE INTO
   USING (사용할 데이터)
   ON (조건)
WHEN MATCHED THEN
   UPDATE SET 칼럼1=값1
WHEN NOT MATCHED THEN
   INSERT (칼럼1,칼럼2….)
   VALUES  (값1, 값2…)
;)

MERGE INTO ex3_3 x
```

```
   USING (SELECT employee_id, salary
          FROM employees
          WHERE manager_id = 145) e
   ON (x.employee_id = e.employee_id) → 왜 써주는거지? 있으면 업데이트 없으면 인서트
WHEN MATCHED THEN
   UPDATE SET  x.bonus_amt = x.bonus_amt + e.salary *0.01 → 보너스가 0인데 x.bonus_am를 왜
써주는거지? 기존 데이터 + @
WHEN NOT MATCHED THEN
   INSERT (x.employee_id, x.bonus_amt)
   VALUES (e.employee_id, e.salary *0.005)
;
```

내가 푼 거

```
   SELECT employee_id, manager_id, salary, salary*0.01
   FROM employees
   WHERE manager_id = 145
      AND employee_id IN (SELECT employee_id
                          FROM ex3_3) ;

   SELECT employee_id, manager_id, salary, salary*0.005
   FROM  employees
   WHERE manager_id = 145
    AND employee_id NOT IN(SELECT employee_id
                           FROM ex3_3) ;



   MERGE INTO ex3_3 b
      USING (SELECT employee_id, manager_id, salary
             FROM  employees
             WHERE manager_id = 145) a
      ON ( a.employee_id = b.employee_id)
   WHEN MATCHED THEN
     UPDATE SET b.bonus_amt = a.salary*0.01+ a.salary
   WHEN NOT MATCHED THEN
```

```
    INSERT (b.employee_id, b.bonus_amt)
    VALUES (a.employee_id, a.salary*0.005)
    WHERE manager_id = 145
  AND employee_id NOT IN(SELECT employee_id
                  FROM ex3_3) ;
```

3장3번문제 -OK

-추출하는 쿼리를 작성 > SELECT

```
SELECT employee_id, emp_name, commission_pct
FROM    employees
WHERE commission_pct IS NULL
;
```

3장 4번문제 - OK
논리연산자: AND OR NOT
비교연산자: > < >= <= ...

```
SELECT employee_id, salary
FROM    employees
WHERE   salary >= 2000
   AND salary <= 2500
;
```

2번문제 다시

관리자 사번이 145번이 사원을 찾아
위 테이블에 있는 사원 사번과 일치하면
보너스 금액bonus_amt에 자신의 급여의 1%를 보너스로 갱신하고
ex3_3테이블에 있는 사원의 사원과 일치하지 않은 사원을
신규입력 (이때 보너스는 급여의 0.5% 한다)

```
MERGE INTO ex3_3
USING (SELECT employee_id,
FROM    employees
```

```
WHERE manager_id = 145)
ON manager_id =  employee_id
WHEN MATCHED THEN
UPDATE SET bonus_amt =  bonus_amt + salary*0.01
WHEN NOT MATCHED THEN
INSERT INTO bonus_amt
VALUES bonus_amt =  bonus_amt + salary*0.005
;
```

10. 조건식 **(7**개**)**

1. 비교 : ANY (값1, 값2) SOME(값1, 값2) ALL (값1, 값2)
2. 논리 : AND OR NOT
3. NULL : IS NULL / IS NOT NULL
4. BETWEEN AND : BETWEEN 시작값 AND 끝값
5. IN 조건식: IN (값1, 값2)
6. EXISTS : EXISTS (서브쿼리 + 조인조건)

조인조건: 테이블a컬럼 = 테이블b컬럼

7. LIKE: LIKE 문자열 패턴 검색

3장3번문제 OK

-추출하는 쿼리를 작성 > SELECT

```
SELECT employee_id, emp_name, commission_pct
FROM    employees
WHERE commission_pct IS NULL
;
```

3장 4번문제 OK

논리연산자: AND OR NOT

비교연산자: > < >= <= ...

```
SELECT employee_id, salary
FROM    employees
```

```
WHERE   salary >= 2000
   AND salary <= 2500
;
```

3장 5번문제
아래 문장을 ANY와 ALL을 사용해서 동일한 결과 추출

```
SELECT employee_id, salary --사원번호, 급여 조회
FROM    employees         -- 사원 테이블로부터
WHERE salary IN (2000, 3000, 4000) -- 급여가 2,3,4천 값이 포함되면
ORDER BY employee_id;     -- 사원 번호로 정렬

SELECT employee_id, salary --사원번호, 급여 조회
FROM    employees         -- 사원 테이블로부터
WHERE salary NOT IN (2000, 3000, 4000) -- 급여가 2,3,4천 값이 포함 안되면
ORDER BY employee_id;     -- 사원 번호로 정렬
```

추출 => SELECT
ANY, ALL을 사용해서 => 비교조건식

테이블: employees
컬럼: employee_id, salary
조건: 컬럼명 (비교연산자) ANY(값1, 값2, 값3)
    컬럼명 (비교연산자) ALL(값1, 값2, 값3)

```
--내가 푼거
SELECT employee_id, salary
FROM    employees
WHERE salary = ANY (2000, 3000, 4000)
ORDER BY employee_id;

--급여가 2,3,4천 모두ALL 같으면 (X) 같지 않으면 (O)
SELECT employee_id, salary
FROM    employees
WHERE salary <> ALL (2000, 3000, 4000) -- 안되는 이유? <> 써줘야함
ORDER BY employee_id;
```

```
SELECT employee_id, salary
FROM    employees
WHERE salary = SOME(2000, 3000, 4000)
ORDER BY employee_id;

SELECT employee_id, salary
FROM    employees
WHERE   salary = 2000
   AND salary = 3000
   AND salary = 4000 ---왜 안되지? -- OR로 써줘야한다
ORDER BY employee_id;

SELECT employee_id, salary
FROM    employees
WHERE   salary = 3000
   OR salary = 3000
   OR salary = 4000
ORDER BY employee_id;
```

--NOT IN 풀이

```
SELECT employee_id, salary
FROM    employees
WHERE salary > 2000
   AND salary > 3000
   AND salary > 4000
ORDER BY employee_id;

SELECT employee_id, salary
FROM    employees
WHERE   salary != 2000
   AND salary != 3000
   AND salary != 4000
ORDER BY employee_id;
```

```
SELECT employee_id, salary
FROM    employees
WHERE salary <> ALL(2000, 3000, 4000)
ORDER BY employee_id;
```

함수**:** 특정 작업을 수행하기 위한 프로그램 코드 집합 어떤 입력값을 넣어서 반환해 주는 **ex** 커피머신 **0**과 **1**

1. 숫자 함수

ABS (숫자) 절대값
CEIL (숫자) 올림
FLOOR(숫자) 버림
ROUND(숫자) 반올림 (5이상일때만)
**TRUNC(숫자, 숫자) 버림
POWER(숫자, 숫자) 제곱
**MOD(숫자,숫자) 나머지
REMINDER(숫자, 숫자) 나머지

2. 문자 함수

INITCAP(문자) 첫자대문자
LOWER(문자) 소문자
UPPER(문자) 대문자
CONCAT(문자) 두문자 연결
' '안에 쓰인 공백도 문자로 처리됨

SUBSTR (문자, 위치, 길이) 부분문자열
SUBSTRB (문자, 위치, 바이트) 부분문자열
*LTRIM (문자, 잘라낼문) 왼쪽 문자열 제거
*RTRIM (문자, 잘라낼문) 오른쪽 문자열 제거

LPAD (문자열1, 길이, 문자열2)
RPAD (문자열1, 길이, 문자열2)
-> 문자열1에 총 문자열의 길이가 n이 될때까지 반복적으로 문자열2를 붙임

**REPLACE (문자열, 대상문자열, 바꿀문자열) -- > 문자열 대체, 공백사용가능 문자삭제할때도

TRANSLATE (문자열, 대상문자, 바꿀문자)--> 각각의 문자 대체, 한 글자씩 찾아 바꾼다

INSTRA (전체문자열, 검색할문자열, 시작위치, 발생횟수)

3. 날짜함수***

날짜 차이: 종료일자 (YYYY-MM-DD) - 시작일자 (YYYY-MM-DD)
SELECT TO_DATE('2021-05-08') - TO_DATE('2021-05-01')
FROM dual;
입력해서 나오는 날짜 단위가 "일"

*SYSDATE - 현재 날짜/시간(시분초)반환
ADD_MONTHS (날짜, 정수) - 해당 날짜 기준 월(정수)를 더한 날짜 변환
*MONTHS_BETWEEN (종료날짜, 시작날짜) - 날짜간 개월 수 반환
*LAST_DAY(날짜) - 해당 월 마지막 일자 반환

*ROUND(날짜, 포맷) 반올림한 날짜
TRUNC(날짜, 포맷) 잘라낸 날짜
NEXT_DAY(날짜, 찾을요일) 다음에 올 요일의 일자

*****함수의 중첩 사용 (함수 내부 먼저 실행)
함수(입력값, 입력값2)- a라면
입력값2 <= 다른함수()가 a에 들어갈 수 있다

함수(입력값1, 다른함수())
단, 반환되는 결과의 데이터타입이 일치하는 경우에만 가능

중첩ex)
ROUND(날짜, 포맷) 반올림한 날짜
TRUNC(ROUND(날짜, 포맷), 포맷) 잘라낸 날짜

예시)

```
SELECT ADD_MONTHS (SYSDATE,6)
FROM DUAL;

SELECT MONTHS_BETWEEN('2023/11/10', '2024/4/10')
FROM DUAL;
```

*****함수(입력값, 입력값2)
입력값2 <= 다른함수()
함수(입력값1, 다른함수())
단, 반환되는 결과의 데이터타입이 일치하는 경우에만 가능

4. 변환함수** (3가지 모두 알아야함 중요!) TO_**

```
--문자변환
--TO CHAR(숫자 or 날짜, 포맷)
1. 소수점 자릿수 맞춰서
2. 음수 <>
3. 양수 음수 표시 + / -

SELECT TO_CHAR(123456789, '999,999,999') FROM DUAL;
--> 자리수 맞춰서 썼을 때와 아닐때 차이가 있다.
SELECT TO_CHAR(1234567, '999,999') FROM DUAL; –오류
SELECT TO_CHAR(1234567, '9,999,999') FROM DUAL; -- 자릿수 맞춰서
SELECT TO_CHAR(1234567.67, '9999999.9') FROM DUAL; -- 소수점 자릿수 맞춰서(반올림)
SELECT TO_CHAR(-123, '999PR') FROM DUAL; -- <음수표시>
```

```
SELECT TO_CHAR(123, 'RN') FROM DUAL; -- 로마숫자 표시
SELECT TO_CHAR(-123, 'S999') FROM DUAL; -- 양수+ 음수- 표시

SELECT TO_CHAR(123, '00000') FROM DUAL; -- 지정한 길이 만큼 0으로 채우기
SELECT TO_CHAR(123456, 'L999,999') FROM DUAL; --원화 표시 넣기L

-- 날짜 포맷에 맞춰서
SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD') FROM DUAL;
SELECT TO_CHAR(SYSDATE, 'YY/MM/DD') FROM DUAL;
SELECT TO_CHAR(SYSDATE, 'YY/MM/DD HH24:MI:SS') FROM DUAL;
SELECT TO_CHAR(SYSDATE, 'YY/MM/DD DAY') FROM DUAL;

SELECT TO_CHAR(SYSDATE-5, 'FMYY/MM/DD DAY') FROM DUAL; -- 날짜 0 없애기
SELECT TO_CHAR(SYSDATE, '""YYYY"년 "MM"월 "DD"일"') FROM DUAL;


--숫자변환
TO NUMBER(문자, 포맷)
-- 문자나 날짜가 포함된 내용을 숫자로 변환하면 오류발생
SELECT TO_NUMBER('123') FROM DUAL;
SELECT TO_NUMBER('ABC') FROM DUAL;  --오류
SELECT TO_NUMBER(SYSDATE) FROM DUAL; --오류

--날짜 변환
TO DATE(문자, 포맷)

SELECT TO_DATE('20140101', 'YY-MM-DD') FROM DUAL;
SELECT TO_DATE('20140101', 'YY MM DD') FROM DUAL;
SELECT TO_DATE('20140101', 'YYYY-MM-DD') FROM DUAL;
SELECT TO_DATE('20140101 13:44:50', 'YYYY-MM-DD HH24:MI:SS') FROM DUAL;
SELECT TO_DATE('2014/01/01 13/44/45', 'YYYY-MM-DD HH24:MI:SS') FROM DUAL;
SELECT TO_DATE('2014-01-01 13/44/45', 'YYYY-MM-DD HH24:MI:SS') FROM DUAL;
SELECT TO_DATE('2014 01 01 13/44/45', 'YYYY-MM-DD HH24:MI:SS') FROM DUAL;


*NLS (National Language Support, 국가별 언어 지원) 설정 확인
SELECT * FROM NLS_SESSION_PARAMETERS;
```

--SESSION 설정변경 명령어
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';
ALTER SESSION SET NLS_DATE_FORMAT = 'RR-MM-DD HH24:MI:SS';

**NULL 함수 > 오라클 함수 NULL 처리 구글 검색
-- NULL 값의 처리: NULL 값의 여부에 따라 관련 값이 달라짐
NVL(표현식1, 표현식2): 표현식**1**이 **null** 일때 표현식**2** 반환

SELECT NVL(표현식1, 표현식2)
FROM 테이블
WHERE 조건;

SELECT NVL(manager_id, employee_id)
FROM employees
WHERE manager_id **IS NULL**;

SELECT manager_id, NVL(manager_id, employee_id)
FROM employees;

NVL2(표현식**1,** 표현식**2,** 표현식**3) :** 표현식**1 <> NULL**이 아닐때 표현식**2** 반환**,** 표현식**1 = NULL**이면 표현식**3**을 반환

사원번호, 급여, 급여+보너스, 보너스 조회

--NULL과 수식연산자를 사용해 연산을 하면 결과값은 NULL
SELECT employee_id,
NVL2(commission_pct, salary+(salary*commission_pct), salary) AS salary2,
(NVL2(commission_pct, salary+(salary*commission_pct), salary) - salary) AS bonus
FROM employees;

SELECT employee_id,
NVL2(commission_pct, salary+(salary*commission_pct), salary) AS salary2

```
FROM employees;

SELECT employee_id,
    NVL2(commission_pct, salary+(salary*commission_pct), salary) AS salary2,
    NVL2(commission_pct, salary+(salary*commission_pct), salary) - salary as bonus
FROM employees;
```

COALESCE(표현식**1,** 표현식**2,...) :** 입력값 중 **NULL**이 아닌 첫번째 표현식 반환

```
SELECT (표현식1, 표현식2,...)
FROM 테이블;

SELECT employee_id,
    COALESCE(salary*commission_pct, salary)
FROM employees;

--사원번호, 커미션 비율, 수령급여

SELECT employee_id, commission_pct,
    COALESCE(salary+salary*commission_pct, salary)
FROM employees;



SELECT employee_id, commission_pct,
    COALESCE(salary*commission_pct, salary)
FROM employees
ORDER BY employee_id;
```

NLNVL(조건식**)**

목표: 커미션이 0.2미만인 사원을 조회 (0% 인 경우)

```
SELECT employee_id, commission_pct
FROM employees
WHERE commission_pct < 0.2;

SELECT COUNT(*)
```

```
FROM employees
WHERE NVL(commission_pct,0) < 0.2;

SELECT count(*)
FROM employees
WHERE LNNVL(commission_pct >= 0.2);  --????????
```

NULLIF(표현식**1,** 표현식**2) :** 표현식**1,2**가 같으면 **NULL** 같지 않으면 표현식**1** 반환

```
SELECT 컬럼
FROM job_history;
```

[문제] 직무의 시작날짜와 종료날짜의 연도가 같으면 NULL,
같지 않으면 종료연도를 출력하는 쿼리를 작성하시오.

날짜 비교 = > 문자데이터 변경 및 연도 추출

```
SELECT employee_id,
      TO_CHAR (start_date, 'YYYY') AS start_date,
      TO_CHAR (end_date,'YYYY')AS end_date
   FROM job_history;
```

---->>> NULLIF(종료년도, 시작년도)

```
SELECT employee_id,
      TO_CHAR (start_date, 'YYYY') AS start_year,
      TO_CHAR (end_date,'YYYY')AS end_year,
      NULLIF(TO_CHAR (end_date,'YYYY'), TO_CHAR (start_date, 'YYYY')) nullif_year
FROM job_history;
```

DESC job_history; –테이블의 열 정보를 알 수있다

******기타함수

GREATEST(표현식1,expr2...): 가장 큰 값
LEAST(expr1, expr2....): 가장 작은 값

```
SELECT GREATEST(1, 2, 3, 4, 5),
      LEAST(1, 2, 3, 4, 5)
```

```
FROM DUAL;

SELECT GREATEST(1+8, 2, 3, 4, 5),
       LEAST(1, 2, 3, 4, 5)
FROM DUAL;
-- 알파벳 순서대로 뒤에 알파벳이 greatest, 한글도 가능
SELECT GREATEST('apple', 'banana', 'orange', 'mango'),
       LEAST('apple', 'banana', 'orange', 'mango')
FROM DUAL;

SELECT GREATEST('홍길동', '유재석', '이특', '조세호'),
       LEAST('홍길동', '유재석', '이특', '조세호')
FROM DUAL;
```

DECODE(표현식, 값1, 결과1, 값2, 결과2, ..., 기본값)

```
SELECT prod_id, channel_id,
      DECODE(channel_id, 3, '이마트', 9, '롯데마트', 5, '트레이디스', 4,
      '티몬', '행사매대')
FROM   sales
WHERE rownum < 10;
```

****DUAL**

- 오라클 자체에서 제공되는 더미 테이블
- 간단한 계산 / 함수 결과 확인용도
- 한 행, 한 컬럼을 담고 있다

** 오라클 날짜 / 시간 차이 계산 방법
종료일자 - 시작일자??
반환 값을 확인하고 사용해야하는데
반환 값: 차이값을 일 기준 수치값으로 변환
예) SYSDATE-5 > 23/11/9

```
SELECT TO_CHAR(SYSDATE-5, 'FMYY/MM/DD DAY') FROM DUAL; -- 날짜 0 없애기
```

FMYY - 날짜의 월, 일 앞의 0을 제거하기 위해

–시간 차이 계산

시간 차이 : (종료일시 (YYYY-MM-DD HH:MI:SS) - 시작일시 (YYYY-MM-DD HH:MI:SS) * 24

분 차이 : (종료일시 (YYYY-MM-DD HH:MI:SS) - 시작일시 (YYYY-MM-DD HH:MI:SS) * 24 * 60

초 차이: : (종료일시 (YYYY-MM-DD HH:MI:SS) - 시작일시 (YYYY-MM-DD HH:MI:SS) * 24 * 60 * 60

- 13:00부터 15:00 시간 차이는 2시간으로 계산 되어서 반환
- 시간과 초가 함께 존재하면 소수점이 발생하므로 꼭 **ROUND** 함수로 소수점을 처리해 줘야 함

```
SELECT (TO_DATE('15:00', 'HH24:MI') - TO_DATE('13:00', 'HH24:MI')) * 24, 2)
FROM dual;
– 16일 이상 (반)올림 처리
SELECT TRUNC(TO_DATE('2023/11/16'), 'MONTH')
FROM DUAL;

SELECT NEXT_DAY(SYSDATE, '토')
FROM DUAL;
```

4장 **SQL** 함수

1. 숫자함수

1) ABS(n입력값): 무조건 양수로 나온다

```
SELECT ABS(10), ABS(-10), ABS(-10.123)
FROM DUAL;
```

2) CEIL(n) 과 FLOOR(n):

- CEIL(n): 천장, 입력값보다 큰 정수를 반환

```
SELECT CEIL(10.123), CEIL(11.011)
FROM DUAL;
```

- FLOOR(n): 바닥, 입력값보다 작거나 정수를 반환

```
SELECT FLOOR(10.123), FLOOR(11.001)
FROM DUAL;
```

3) ROUND(n, i)와 TRUNC(n1, n2)

- ROUND(n, i): n을 소수점 기준(i+1)번 째에서 반올림함 결과

```
SELECT ROUND(10.154, 1), ROUND(10.154, 2), ROUND(10.154, 3)
FROM DUAL;

SELECT ROUND(0, 3), ROUND(115.155, -1), ROUND(115.155, -2) - 반대로 소숫점 앞으로
FROM DUAL;
```

- TRUNC(n1, n2): 반올림을 하지않고 소숫점을 잘라낸 결과를 반환하는데
  n2 소숫점 기준으로 양수일 땐 오른쪽, 음수일 땐 왼쪽으로 잘라낸다

```
SELECT TRUNC(155.155), TRUNC(155.155, 1), TRUNC(155.155, 2), TRUNC(155.155, -2)
FROM DUAL;
```

4) POWER(n2, n1)과 SQRT(n)

- POWER(n2, n1)은 n2를 n1 제곱한 결과
- SQRT(n): n의 제곱근

5) MOD(n2, n1) 와 REMAINDER(n2, n1)

- MOD(n2, n1): n2를 n1으로 나눈 나머지 값을 반환
- REMAINDER(n2, n1): MOD와 같지만 다르다

2. 문자함수

1) INITCAP(char), LOWER(char), UPPER(char)

- INITCAP(char): 표현식안의 첫 문자는 대문자
- LOWER(char): 소문자
- UPPER(char): 대문자

2) CONCAT(char1, char2), SUBSTR(char, pos, len), SUBSTRB(char, pos, len)

- CONCAT(char1, char2): '||' 연산자처럼 두 문자를 붙인다
- SUBSTR(char, pos, len): char의 pos번째 문자부터 len길이만큼 잘라낸 결과
- SUBSTRB(char, pos, len): 문자 len 개수만큼이 아니라 byte만큼 잘라낸 결과

3) LTRIM(char, set), RTRIM(char, set)

- LTRIM(char, set): char 문자열에서 set으로 지정된 문자열을 왼쪽 끝에서 제거한 후 나머지 문자열을 반환

- RTRIM(char, set): 위와 반대로 오른쪽 끝에서 제거한 뒤 나머지 문자열을 반환

4) LPAD(expr1, n, expr2), RPAD(expr1, n, expr2)

- LPAD(expr1, n, expr2): expr2를 문자열을 n자리만큼 왼쪽으로 부터 채워 expr1을 반환하는 함수
- RPAD는 오른쪽으로

5)

REPLACE(char, search_str, replace_str): char 문자열에서 search문자열을 찾아 이를 replace문자열로 대체

TRANSLATE(expr, from_str, to_str): 위와 같지만, 문자열 자체가아닌 문자 한 글 자씩 바꿔준다

6)

INSTR(str, substr, pos, occur)

: str 전체 문자열, substr 검색할 단어, pos 시작위치, occur n 번째

LENGTH(chr): 문자열 개수 반환

LENGTHB(chr): 문자열 바이트 수 반환

3. 날짜함수

1) SYSDATE, SYSTIMESTAMP

2) ADD_MONTHS(date, integer): date에 integer만큼의 월을 더한 날짜 반환

3) MONTHS_BETWEEN(date1, date2): 두 날짜 사이의 개월 수를 반환

4) LAST_DAY(date): date를 기준으로 해당 월의 마지막 일자를 반환

5) ROUND(date, format): format에 반올림한 날짜
TRUNC(date, format): 잘라낸 날짜

4. 변환함수

1) TO_CHAR(숫자/문자, format): 문자로 변환
2) TO_NUMBER(문자/숫자, format): 숫자로
3) TO_DATE(문자, format): 날짜형으로 / TO_TIMESTAMP

5. NULL 관련 함수: IS NULL, IS NOT NULL

1) NVL(expr1, expr2): expr1이 null일 때 expr2 반환
NVL2(expr1, expr2, expr3): NVL을 확장한 함수로, expr1이 null이 아니면 expr2을
null이면 expr3를 반환하는 함수

2) COALESCE(expr1, expr2): expr1이 null이 아니면 expr1을 null 이면 expr2를 반환

3) LNNVL 조건식: NVL의 반대

4) NULLIF(expr1, expr2): expr1 과 2이 같으면 null 같지 않으면 expr1을 반환

6. 기타 함수

1) GREATEST(expr1, expr2,..): 표현식 중 가장 큰 값만 반환
LEAST(expr1, expr2,..):표현식 중 가장 작은 값만 반환

2) DECODE(expr, search1, result1, serch2, reseul2,.. default)
: expr과 search1을 비교해 두 값이 같으면 result1을, 같지 않으면 다시 search2와 비교 값이 같으면 result2를 반환...최종적으로 같은 값이 없으면 default값을 반환

SELF CHECK

4장 **1**번

CONCAT연산자, LPAD - Ok
문자연산자 || - NO

SELECT phone_number, '(02)'|| **'(**SUBSTR(phone_number, 5, 12)**)'** – 필요없는 거
FROM employees;

>> '(02)' || SUBSTR(phone_number, 5, 12)

4장 2번 - X
근속년수를 어떻게 SELECT에 넣어줄까? – 함수로 넣어줄 수 있다
어떻게 SYSDATE - hire_date를 해주지?? – 날짜함수로

```
SELECT  employee_id 사원번호, emp_name 사원명,
       hire_date 입사일자,
       TRUNC(ROUND(SYSDATE - hire_date)/365) 근속년수
FROM    employees
WHERE   TRUNC(ROUND(SYSDATE - hire_date)/365) >= 10
ORDER BY TRUNC(ROUND(SYSDATE - hire_date)/365) DESC;
```

4장 3번 - ok
4장 4번 - ok

4장 5번 - ok

```
SELECT cust_year_of_birth,
      DECODE (TRUNC(ROUND(2023 - cust_year_of_birth)/10), 1, '10대',
                                                2, '20대',
                                                3, '30대',
                                                4, '40대',
                                                5, '50대',
                                                '기타')DECODES
FROM customers;
```

4장 6번 - X

```
SELECT cust_year_of_birth,
    CASE WHEN TRUNC(ROUND(2023 - cust_year_of_birth)/10)>= 1 THEN '10대'
        WHEN TRUNC(ROUND(2023 - cust_year_of_birth)/10)>= 2 THEN '20대'
        WHEN TRUNC(ROUND(2023 - cust_year_of_birth)/10)>= 3 THEN '30대'
   ELSE '기타'
   END
FROM customers;
```

—--->> 1보다 크면 10대? 말이 안됨

5장. 함수

1)기본집계함수

1)-1 COUNT(expr): 전체 로우 수 를 반환하는 집계함수

중복/null값 제거 COUNT(DISTINCT 컬럼명)

```
SELECT  COUNT(*)
FROM    employees;

SELECT COUNT(emp_name)
FROM    employees;

SELECT COUNT(DISTINCT department_id)
FROM    employees;

--NULL 값도 같이나오는 구조
SELECT DISTINCT department_id
FROM    employees;
```

1)-2 SUM(expr): expr의 전체합계를 반환하는 함수로
expr에는 숫자형만 올 수 있다
DISTINCT도 같이 쓸 수 있다

```
SELECT SUM(salary)
FROM    employees
;

SELECT SUM (DISTINCT salary) -- 중복제거
FROM    employees
;
```

1)-3 AVG(expr): 평균값을 반환한다

```
SELECT  AVG(salary), AVG(DISTINCT salary)
FROM    employees
;

SELECT  TRUNC (AVG(salary)) --TRUNC를 써서 소수점 자르기
FROM    employees
;
```

1)-4 MIN(expr), MAX(expr): 최솟값과 최댓값
DISTINCT를 사용할 수 있지만, 굳이 사용할 필요 없음

```
SELECT MIN(salary), MAX(salary)
FROM    employees
;
```

1)-5 VARIANCE(expr), STDDEV(expr): 분산과 표준편차

```
SELECT VARIANCE(salary), STDDEV(salary)
FROM    employees;
```

2) GROUP BY 절과 **HAVING** 절**:** 집계함수와 같이 쓸 수 있다

- 특정 그룹으로 묶어 데이터 집계 시 사용
- GROUP BY는 WHERE절과 다음에 위치, 중복을 묶어준다(DISTINCT는 중복삭제)
- HAVING절은 group by다음에 위치, group by한 결과를 대상으로 다시 필터를 거는 역할
  즉, having 필터조건
- SELECT 절에서 집계함수를 제외한 모든 컬럼과 표현식은 GROUP by 절에 명시해야 함

```
SELECT  컬럼1, 컬럼2, 집계함수(표현식)
FROM    테이블
WHERE   조건
GROUP BY 컬럼1, 컬럼2
ORDER BY  컬럼
;
```

****SQL** 구문순서

```
SELECT 컬럼명 5
FROM 테이블명 1
WHERE 조건    2
GROUP BY 그룹조건 3
HAVING 그룹조건 4
ORDER BY 6 ;
```

```
SELECT  department_id
FROM    employees
GROUP BY    department_id
ORDER BY    1
;

SELECT  department_id, SUM(salary)
FROM    employees
GROUP BY    department_id
ORDER BY    1
;
--2013년 지역별 가계대출 총 잔액
SELECT  period, region, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY    period, region
ORDER BY    1, 2
;
--2013년 11월 총 잔액
```

```
SELECT  period, region, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period = '201311'
GROUP BY    period, region;
```

- HAVING 필터조건
- GROUP BY절 바로 다음에 위치
- SELECT 절에 사용했던 집계함수를 이용해 조건을 명시

```
SELECT  period, region, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period = '201311'
GROUP BY    period, region
HAVING  SUM(loan_jan_amt) > 100000
ORDER BY 2
;
```

3) ROLLUP절과 CUBE절

- 둘다 GROUP BY절에 사용되어 그룹별 소계를 추가로 보여주는 역할
- expr로 명시한 표현식을 기준으로 집계한 결과, 정보를 보여 준다
- 명시한 표현식expr 수와 순서(오른쪽에서 왼쪽으로) 레벨별로 집계한 결과 반환
- N 표현식 개수 +1, 하위레벨에서 상위레벨 순으로 데이터 집계
- ROLLUP은 레벨별, CUBE는 가능한 조합별 집계

3)-1 ROLLUP(expr1, expr2)

```
SELECT 컬럼1, 컬럼2, 집계함수(표현식)
FROM  테이블
WHERE   조건
GROUP BY 컬럼1, 컬럼2
ORDER BY 컬럼
;
```

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY  ROLLUP(period, gubun);
```

-순서중요, period가 중요/ Rollup절에 period와 gubun을 명시해 총 3레벨

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	기타대출	676078
2	201310	주택담보대출	411415.9
3	201310	(null)	1087493.9
4	201311	기타대출	681121.3
5	201311	주택담보대출	414236.9
6	201311	(null)	1095358.2
7	(null)	(null)	2182852.1

3레벨: period와 gubun
2레벨: period
1레벨: 전체합계

- 분할 **ROLLUP**: 다른유형, GROUP BY expr1, ROLLUP(expr2, expr3)

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY  period, ROLLUP(gubun); -- 분할 ROLLUP
```

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	기타대출	676078
2	201310	주택담보대출	411415.9
3	201310	(null)	1087493.9
4	201311	기타대출	681121.3
5	201311	주택담보대출	414236.9
6	201311	(null)	1095358.2

> GROUP BY절에 period, ROLLUP(gubun)이라 1+1=2레벨이 되는데 (period, gubun)이 2레벨, period - 1레벨로 집계되며 전체합계는 집계되지 않는다

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY  ROLLUP(period), gubun;
```

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	기타대출	676078
2	201311	기타대출	681121.3
3	(null)	기타대출	1357199.3
4	201310	주택담보대출	411415.9
5	201311	주택담보대출	414236.9
6	(null)	주택담보대출	825652.8

ROLLUP(period), gubun 으로 (period, gubun) - 2레벨 / gubun - 1레벨

ROLLUP이 "말아 올린다"라는 의미로, ROLLUP에 명시한 표현식 개수에 따라 레벨이 정해지고 하위 레벨부터 상위 레벨로 "말아 올려진" 집계 결과가 계산된다고 생각하자

3)-2 **CUBE(expr1, expr2)**

- ROLLUP이 레벨별로 순차적 집계를 했다면, cube는 명시한 표현식 개수에 따라
- 가능한 모든 조합별로 집계한 결과를 반환하다
- cube는 2의(expr수) 제곱 만큼 종류별로 집계 된다
- expr이 3개일 때, 2^3= 8 로 집계결과 유형은 8개

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY  CUBE(period, gubun);  > () 순서를 다르게 해도 데이터 내용에 차이가 없고 같다
```

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	(null)	(null)	2182852.1
2	(null)	기타대출	1357199.3
3	(null)	주택담보대출	825652.8
4	201310	(null)	1087493.9
5	201310	기타대출	676078
6	201310	주택담보대출	411415.9
7	201311	(null)	1095358.2

Cube안의 expr의 개수가 2개이므로 2^2=4, 총 4가지 유형으로 집계되는데

1 전체, 2 대출별, 3 월별,
4 월별 대출별으로 집계됨

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY  period, CUBE(gubun); --분할cube
```

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	(null)	1087493.9
2	201310	기타대출	676078
3	201310	주택담보대출	411415.9
4	201311	(null)	1095358.2
5	201311	기타대출	681121.3
6	201311	주택담보대출	414236.9

4) 집합 연산자

SELECT 칼럼1, 컬럼2 → select 절의 컬럼의 개수와 타입이 일치 1제한사항
FROM 테이블
WHERE 조건
GROUP BY 그룹컬럼
HAVING 그룹조건
집합연산자
SELECT 칼럼1, 컬럼2
FROM 테이블
ORDER BY 그룹컬럼
;

4)-1 UNION: 합집합
- 두개 이상의 개별 select 쿼리를 연결
- 중복될 경우 한개만 반환됨

```
SELECT  goods
FROM    exp_good_asia
```

```
WHERE   country = '한국'
UNION
SELECT goods
FROM    exp_good_asia
WHERE   country = '일본';
```

4)-2 UNION ALL: 중복된 항목도 모두 조회

4)-3 INTERSECT: 교집합

4)-1 MIUS: 차집합

- A MINUS B: A에서 B를 뺀 값

- 집합연산자의 제한사항
- 각 select문의 리스트 개수와 데이터 값은 일치해야 한다
- Order by절은 맨 마지막 문장에서만 사용할 수 있다
- Blob, Clob, Bfile 타입의 컬럼에 대해 집합연산자를 사용X
- Long형 컬럼에 사용 X

SELF CHECK

5번 **4**번

```
SELECT period, 0 주택담보대출액, SUM(loan_jan_amt) 기타대출액
FROM    kor_loan_status
WHERE   period LIKE '201311'
   and SUM(loan_jan_amt) = '기타대출' – loan_jan_amt 컬럼에서 기타대출로 구분해?
GROUP BY period
;
```

5번 **5**번

```
SELECT  region 지역,
 SUM (CASE WHEN period = '201111' THEN loan_jan_amt ELSE 0 END "201111") – 별칭이 가로 안에 들어가나?? 숫자로 별칭을 할 땐, 큰 따옴표로 표시한다 “ ”
FROM    kor_loan_status;
;
```

6.조인의 종류

1)동등조인: 두 테이블에서 공통된 값을 가진 컬럼을 WHERE절에 조인조건

```
SELECT  e.emp_name, d.department_id
```

```
FROM    employees e, departments d
WHERE  e.department_id = d.department_id
;

DESC departments;
```

2) 세미조인: WHERE절에 IN또는 EXISTS를 넣어 서브쿼리 사용

```
SELECT  e.department_id
FROM    employees e
WHERE   EXISTS(SELECT   *
               FROM    departments d
               WHERE   e.department_id = d.department_id)
;


SELECT  e.department_id
FROM    employees e
WHERE   e.department_id IN (SELECT d.department_id
                        FROM    departments d
                        WHERE   e.department_id = d.department_id)
;
```

3) 안티조인: NOT IN이나 EXISTS연산자 사용

```
SELECT  e.department_id, d.manager_id
FROM    employees e, departments d
WHERE   e.department_id = d.department_id
   AND e.department_id NOT IN (SELECT  d.department_id
                           FROM    departments d
                           WHERE   d.manager_id is null)
--GROUP BY e.department_id, d.manager_id
;

SELECT  COUNT(*)
FROM    employees e
WHERE   NOT EXISTS (SELECT  1
                  FROM    departments d
                  WHERE   e.department_id = d.department_id
                    AND   manager_id is null);
```

4) 셀프조인: 같은 한 테이블을 사용해 조인하는 방법

```
SELECT  e.employee_id, e.emp_name, d.employee_id, d.emp_name, e.department_id
FROM    employees e, departments d
WHERE   e.department_id = d.department_id
  AND e.departent_id = 20;
```

5) 외부조인: OUTER JOIN 컬럼값이 null이라도 모든 데이터 추출
WHERE절 조인조건에서 데이터가 없는(null)인 테이블에 (+)를 붙인다

```
SELECT  d.department_id, j.department_id
FROM    departments d, job_history j
WHERE   d.department_id = j.department_id(+);
```

6) 카타시안 조인: WHERE절이 없다

```
SELECT  COUNT(e.emp_name), COUNT(d.department_name)
FROM    employees e, departments d;
```

7) ANSI 조인: ANSI SQL문법 조인, FROM절에 들어감, WHERE은 X

7)-1 ANSI 내부조인: from절에 inner join 테이블 on()

```
SELECT  e.emp_name, d.department_id
FROM    employees e , departments d
WHERE   e.department_id = d.department_id;
```

```
SELECT  e.emp_name, d.department_id, e.hire_date
FROM    employees e
INNER JOIN departments d
   ON(e.department_id = d.department_id)
WHERE e.hire_date >= '2003-01-01'
ORDER BY 3
;
```

7)-2 ANSI 외부조인: left(right) [outer] join 테이블B ON(조인조건)

기존문법에서 where절의 조인조건에 (+)를 붙였다면 ansi 외부조인은
from절에 left(right) [outer] join 테이블B ON을 쓰고 where절은 X

```
SELECT  e.employee_id, e.emp_name, j.job_id, j.department_id
FROM    employees e
LEFT JOIN   job_history j
ON  e.employee_id = j.employee_id
   and e.department_id = j.department_id
;
```

7)-3 ANSI CROSS 조인: 카타시안 조인과 같다 CROSS JOIN

```
SELECT  e.emp_name, d.department_name
FROM    employees e
CROSS JOIN  departments d;
```

7)-4 FULL OUTER 조인: 모든 컬럼에 null 값이 있다

```
SELECT a.emp_id, b.emp_id
FROM   hong_a a , hong_b b
WHERE  a.emp_id(+) = b.emp_id(+);

SELECT a.emp_id, b.emp_id
FROM    hong_a a
FULL OUTER JOIN hong_b b
   ON (a.emp_id = b.emp_id);
```
두 테이블 모두 외부조인대상

4장 실전문제 **1**번 **X**

사원테이블(employees)에는 phone_number라는 컬럼에 사원의 전화번호가 ###.###.#### 형태로 저장되어 있다. 처음 세 자리 숫자 대신 서울 지역번호인 (02)를 붙여 전화번호를 출력해 쿼리를 작성해라.

내가 푼 것 --1 –

```
SELECT LPAD(phone_number, 16, '(02)')
FROM employees;
```

내가 푼 것 --2 다른 가능성??

```
SELECT REPLACE(phone_number, '011', '(02)'),
       REPLACE(phone_number, '650', '(02)')
FROM employees;
```

풀이

```
--LPAD(문자열1, 길이, 문자열2)
SELECT phone_number,
      LPAD(SUBSTR(phone_number, 5, 12), 12, '(02)')
FROM employees;

--문자연산자
SELECT phone_number,
      '(02)' || SUBSTR(phone_number, 5, 12)
FROM employees;

--CONCAT (문자, 문자)
SELECT phone_number,
       CONCAT('(02)', SUBSTR(phone_number, 5, 12))
FROM employees;
```

4장 실전문제 **2** 번 **– X**

현재일자를 기준으로 사원 테이블의 입사일자를 참조해,
근속년수가 10년 이상인 사원을 출력 쿼리를 작성,
근속년수가 높은 사워 순서대로
>>사원번호 사원명 입사일자 근속년수

오름차순**(ASC, ascending order) 1 > 2 > 3 > 4**
내림차순**(DESC, descending order) 4 > 3 >2 > 1**
예**)**
오름차순 **: ORDER BY** 기준컬럼 **ASC;**
내림차순 **: ORDER BY** 기준컬럼 **DESC;**

```
SELECT employee_id, emp_name, hire_date, 함수
FROM employees
WHERE employee_id IN (SELECT employee_id FROM job_history)
;

SELECT employee_id,
      TO_CHAR (start_date, 'YYYY') AS start_year,
      TO_CHAR (end_date,'YYYY')AS end_year
FROM job_history;
```

오답노트**)**
현재일자는 SYSDATE, 사원테이블 참조
함수를 써서 Select에 넣을 수 있다

풀이방법

1.문제해석
> 정보찾기(동사): 사용할 쿼리문/ 함수확인,
> 정보분리, 사용할 테이블/컬럼/조건확인

현재일자를 기준으로 SYSDATE
사원 테이블employees 의 입사일자hire_date를 참조해,
근속년수가 10년 이상인 사원을 출력 쿼리를 작성,
 근속일수: 현재일자(SYSDATE)-시작일자hire_date 365일 = 1년
 365일/365=1년
 500일/365=1.xx년 > 소수점 잘라내기 (FLOOR/TRUNC) / 올림CEIL / 반올림 ROUND
근속년수가 높은 사원 순서대로

테이블: employees
컬럼: 사원번호 사원명 입사일자 근속년수
조건:
함수: ROUND

SELECT 컬럼
FROM 테이블
WHERE 근속년수(SYSDATE - hire_date) >= 10
ORDER BY 기준컬럼 DESC;

```
SELECT employee_id, emp_name, hire_date,
       ROUND((SYSDATE - hire_date) / 365)
FROM employees
WHERE ROUND((SYSDATE - hire_date) / 365) >= 10
ORDER BY ROUND((SYSDATE - hire_date) / 365) DESC
;
```

```
SELECT employee_id 사원번호, emp_name 사원명, hire_date 입사일자,
       ROUND((SYSDATE - hire_date) / 365) 근속년수
FROM employees
WHERE ROUND((SYSDATE - hire_date) / 365) >= 10
ORDER BY ROUND((SYSDATE - hire_date) / 365) DESC
;
```

4장 3번문제 - OK

고객테이블에 고객 전화번호 컬럼. 이 컬럼값은 '###-###-####' 인데

- 대신 / 로 바꾼다 출력쿼리

```
SELECT cust_main_phone_number,
      REPLACE(cust_main_phone_number, '-', '/') new_phone_number
FROM customers;
```

풀이방법

바꿔 > REPLACE (문자열, 대상문자, 바꿀문자)
출력 > SELECT

테이블: customer
컬럼: cust_main_phone_number
함수: REPLACE (문자열, 대상문자, 바꿀문자)

4장 4번문제 - OK

고객 테이블(customer)의 고객 전화번호 컬럼(ust_main_phone_number)을
다른 문자로 대체 일종의 암호화를 해서 쿼리 작성

내가푼 것,

```
SELECT cust_main_phone_number,
      TRANSLATE(cust_main_phone_number, '01234568-', 'iloveyou*') new_phone_number
FROM customers;
```

풀이방법

대체 암호화 > TRANSLATE
쿼리 > SELECT? UPDATE???

UPDATE 테이블
SET 컬럼**1 =** 변경값**1,**
  컬럼**2 =** 변경값**2**
;

```
UPDATE customers
```

```
SET cust_main_phone_number = TRANSLATE(cust_main_phone_number, '01234568-', 'iloveyou*')
;
```

되돌리기

```
UPDATE customers
SET cust_main_phone_number = TRANSLATE(cust_main_phone_number, 'iloveyou*', '01234568-')
;
```

4장 5번문제 - X

고객테이블 customers에는 고객의 출생년도 cust_year_of_birth 컬럼이 있다
현재일자sysdate 기준으로 이 컬럼을 활용해
30, 40, 50를 구분해 출력 > 조건 분기 DECODE 함수 CASE 표현식
나머지는 기타로 출력

내가 푼 것,

내가 놓친 것,
SYSDATE 를 연도로 바꿔줘야해서 TO_CHAR(SYSDATE, 'YYYY') 를 사용
함수를 써서 나머지 없애주기 - TRUNC

```
SELECT cust_year_of_birth
CASE WHEN SYSDATE - cust_year_of_birth >= 30 THEN '30대',
    WHEN SYSDATE - cust_year_of_birth >= 40 THEN '40대',
    WHEN SYSDATE - cust_year_of_birth >= 50 THEN '50대',
    ELSE '기타'
END
FROM customers;
```

풀이방법

```
SELECT  TO_CHAR(SYSDATE, 'YYYY') 현재년도, cust_year_of_birth 출생년도,
      TO_CHAR(SYSDATE, 'YYYY')-cust_year_of_birth 나이
```

```
FROM customers;

--TRUNC (n, i) 버림

SELECT  TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth), -1) 세대,
       TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth)/10) 세대
FROM customers;

SELECT DECODE(TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth)/10), 30, '30대', 40, '40대',
5, '50대', '기타') generation
FROM customers;

SELECT cust_year_of_birth 출생년도,
   DECODE(TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth)/10), 30, '30대', 40, '40대', 5,
'50대', '기타') generation
FROM customers;
```

4장 6번문제 - X

5문제는 30-50대 까지만 표현했는데, 모든 연령대를 표시하도록 쿼리를 DECODE 대신 CASE 표현식을 쓰자

**CASE 표현식
CASE WHEN 조건1 then 값1
    WHEN 조건2 then 값2
   ELSE   값3

내가푼것,
```
SELECT cust_year_of_birth,
   CASE WHEN TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth >= 30 THEN '30대',
             WHEN TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth >= 40 THEN '40대',
           WHEN TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth >= 50 THEN '50대',
    ELSE '기타'
END
FROM customers;
```

```
SELECT cust_year_of_birth, TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth)/10) 세대,
   CASE WHEN (TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth)/10) >= 3 THEN '30대',
       WHEN (TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth)/10) >= 4 THEN '40대',
       WHEN (TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth)/10) >= 5 THEN '50대',
    ELSE as '기타'
END
FROM customers;
```

풀이방법

CASE WHEN 나이의 앞자리가 1이면 then '10대'
    WHEN 나이의 앞자리가 2이면 then '20대'
   ELSE   '50대이상'

나이의 앞자리 => TRUNC(현재년도 - 출생년도/10)

현재년도
```
SELECT SYSDATE FROM DUAL;
SELECT TO_CHAR (SYSDATE, 'YYYY')FROM DUAL;
```

출생년도

```
SELECT cust_year_of_birth FROM customers;
SELECT (TO_CHAR (SYSDATE, 'YYYY')- cust_year_of_birth) FROM customers;
SELECT TRUNC ((TO_CHAR(SYSDATE, 'YYYY')- cust_year_of_birth) / 10) FROM customers;


  SELECT CASE WHEN TRUNC ((TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth)/10)=1         THEN
'10대'
        WHEN TRUNC ((TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth)/10)=2 THEN '20대'
        WHEN TRUNC ((TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth)/10)=3 THEN '30대'
        WHEN TRUNC ((TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth)/10)=4 THEN '40대'
 ELSE '50대이상'
END
FROM customers;

SELECT CASE WHEN TRUNC (TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth) BETWEEN 10 AND 19
THEN '10대'
        WHEN TRUNC (TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth) BETWEEN 20 AND 29 THEN
'20대'
        WHEN TRUNC (TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth) BETWEEN 30 AND 39 THEN
'30대'
        WHEN TRUNC (TO_CHAR(SYSDATE, 'YYYY') - cust_year_of_birth) BETWEEN 40 AND 49 THEN
'40대'
 ELSE '50대이상'
END
FROM customers;
```

5장

1. 기본집계함수

COUNT(expr)

```
SELECT COUNT(department_id)
FROM    employees;
```

결과: null 값을 포함해 106

SELECT COUNT(**DISTINCT** department_id)
FROM employees;
결과: null 값을 제거해 11

—----------이 둘의 차이는??

SELECT **DISTINCT** department_id
FROM employees;
결과: null 값을 포함해 12

VARIANCE(expr) - 분산

STDDEV(expr) - 표준편차

5 장 복습

```
SELECT AVG (ABS (cust_year_of_birth)),
       AVG (FLOOR (cust_year_of_birth)), -- 왜 소수점들이 나오나?? floor 구문을 AVG로 나눈값이라서
       TRUNC (AVG (cust_year_of_birth)),
       SUBSTR(AVG(cust_year_of_birth), 1, 4)
FROM customers;
SELECT TRUNC (AVG (cust_year_of_birth))
FROM customers;
```

****GROUB BY** 와 **HAVING** 절

*** **GROUP BY**

-특정 그룹으로 묶어 데이터 집계 시 사용
-WHERE과 ORDER BY절 사이에 위치
-집계함수와 함께 사용
-SELECT 절에서 집계함수를 제외한 모든 컬럼과 표현식은 GROUP by 절에 명시해야 함

```
SELECT  컬럼1, 컬럼2, 집계함수(표현식)
FROM    테이블
WHERE   조건
GROUP BY 컬럼1, 컬럼2
ORDER BY  컬럼
;
```

```
SELECT  department_id, SUM(salary)
FROM    employees
WHERE   department_id = 70
GROUP BY department_id       --GROUP BY 를 써서 중복되는 데이터를 하나로 묶는다
ORDER BY  department_id                                    DISTINCT 는 중복제거
;
```

<응용>

```
SELECT  department_id, SUM(salary), COUNT(employee_id)
FROM    employees
GROUP BY department_id
ORDER BY  department_id
;
```

```
SELECT  department_id, SUM(salary), COUNT(employee_id), AVG(salary)
FROM    employees
GROUP BY department_id
ORDER BY  department_id
;
```

```
SELECT  department_id, SUM(salary), COUNT(employee_id), TRUNC(AVG(salary))
FROM    employees
--WHERE   salary > 5000
GROUP BY department_id
ORDER BY  TRUNC(AVG(salary)) DESC
;
```

**SQL 구문순서
--SELECT 컬럼명 5
--FROM 테이블명 1
--WHERE 조건 2
--GROUP BY 그룹조건 3
--HAVING 그룹조건 4
--ORDER BY 6 ;

```
SELECT SUM(loan_jan_amt)
FROM kor_loan_status;

SELECT period, region, SUM(loan_jan_amt)
FROM kor_loan_status
--WHERE
GROUP BY period, region;
HAVING
ORDER BY

-- 기간별 대출액 합계
SELECT period, SUM(loan_jan_amt)
FROM kor_loan_status
GROUP BY period
ORDER BY period ASC  --DESC
;

-- 한국대출상태 테이블에서 2013년도 기간동안 대출액 합계를 기간순서(오름차순)으로 조회
SELECT period 기간, TRUNC (SUM(loan_jan_amt)) 대출합계
FROM kor_loan_status
WHERE period LIKE '2013%'
GROUP BY period
ORDER BY period ASC  --DESC
;

-- 기간별 지역별 대출액 합계 조회
SELECT period, region, (SUM(loan_jan_amt))
FROM kor_loan_status
WHERE period LIKE '2013%'
GROUP BY period, region
ORDER BY period ASC  --DESC
;

-- 지역별 기간별 대출액 합계 조회
SELECT region, period, (SUM(loan_jan_amt))
FROM kor_loan_status
WHERE period LIKE '2013%'
```

```
GROUP BY period, region --순서상관없다
ORDER BY region
;


SELECT region, TRUNC (SUM(loan_jan_amt))
FROM kor_loan_status
GROUP BY region
ORDER BY SUM(loan_jan_amt)
;
```

****HAVING** 절

1.GROUP BY절 바로 다음에 위치

2.SELECT 절에 사용했던 집계함수를 이용해 조건을 명시

```
SELECT 컬럼1, 컬럼2, 집계함수(표현식)
FROM    테이블
WHERE   조건
GROUP BY    컬럼1, 컬럼2
HAVING 집계함수 그룹조건
ORDER BY    컬럼;

-- 2013년 11월 기간,지역별 총 잔액 조회
SELECT period, region, SUM(loan_jan_amt) -- 오류: 단일 그룹의 그룹 함수가 아닙니다
FROM kor_loan_status
WHERE period = '201311'
;

SELECT period, region, SUM(loan_jan_amt)
FROM kor_loan_status
WHERE period = '201311' AND SUM(loan_jan_amt) > 10000 --오류: 그룹 함수는 허가되지 않습니다
GROUP BY period, region
ORDER BY region
;

-- 2013년 11월 기간별/지역별 총 잔액이 (100억 초과) 내용 조회
SELECT period, region, SUM(loan_jan_amt)
```

```
FROM kor_loan_status
WHERE period = '201311'
GROUP BY period, region
HAVING SUM(loan_jan_amt) > 10000
ORDER BY region
;
```

ROLLUP 절

```
SELECT 컬럼1, 컬럼2, 집계함수(표현식)
FROM  테이블
WHERE  조건
GROUP BY 컬럼1, 컬럼2
ORDER BY 컬럼
;
```

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY period, gubun
;
```

결과

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	기타대출	676078
2	201310	주택담보대출	411415.9
3	201311	기타대출	681121.3
4	201311	주택담보대출	414236.9

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY ROLLUP(period, gubun) → 3개의 종류 레벨로 집계한 결과가 나옴
;                                중요O, 중요X > 앞에 있는 period 무조건 포함시킨다
```

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	기타대출	676078
2	201310	주택담보대출	411415.9
3	201310	(null)	1087493.9
4	201311	기타대출	681121.3
5	201311	주택담보대출	414236.9
6	201311	(null)	1095358.2
7	(null)	(null)	2182852.1

3번의 null 값은 1+2

6번의 null 값은 4+5

7번의 null 값 = 3번의 null 값 + 6번의 null 값

→ ROLLUP(n1, n2) > n +1 레벨으로 3레벨

– gubun의 순서를 바꿔주었을때

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY ROLLUP(gubun, period)
;
```

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	기타대출	676078
2	201311	기타대출	681121.3
3	(null)	기타대출	1357199.3
4	201310	주택담보대출	411415.9
5	201311	주택담보대출	414236.9
6	(null)	주택담보대출	825652.8
7	(null)	(null)	2182852.1

```
--표현식 2 개 - 2+ 1 레밸
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY ROLLUP(gubun, period)
;

-- 표현식 1개 > 1+1 레밸
SELECT  period, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
```

```
GROUP BY ROLLUP(period)
;

-- 표현식 3개 > 3+1 레벨
SELECT  period, gubun, region, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY ROLLUP(gubun, period, region)
;



SELECT  컬럼1, 컬럼2, 집계함수(표현식)
FROM    테이블
WHERE   조건
GROUP BY ROLLUP(표현식1, 표현식2)
;

--분할(partial) ROLLUP
ROLLIP만 사용한 형태: 2+1 레벨
(표현식2, 표현식3)
(표현식2)
(전체)
SELECT  컬럼1, 컬럼2, 집계함수(표현식)
FROM    테이블
WHERE   조건
GROUP BY 표현식1, ROLLUP(표현식1, 표현식2)
;
-----------------------------------------
ROLLIP만 사용한 형태: 1+1 레벨
(표현식1, 표현식2)
(표현식1)
;

SELECT  gubun, period, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY gubun, ROLLUP(period)
```

;

-CUBE

```
SELECT  컬럼
FROM    테이블
WHERE   조건
GROUP BY 그룹컬럼
;
```

표현식1개일때 but cube가 안쓰였을때,

```
SELECT  gubun
FROM    kor_loan_status
WHERE   gubun = '기타대출'
GROUP BY gubun
;

SELECT  gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   gubun = '기타대출'
GROUP BY gubun
;
```

표현식1개일때 with CUBE

```
SELECT  gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   gubun = '기타대출'
GROUP BY CUBE(gubun)
;

--표현식2개일때 wth CUBE - cube에 period를 먼저쓸때,
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   gubun = '기타대출'
GROUP BY CUBE(period, gubun)
;
```

```
--- cube에 gubun을 먼저쓸때,
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   gubun = '기타대출'
GROUP BY CUBE(gubun, period)
;
-->>> 순서를 다르게 해도 데이터 내용에 차이가 없고 같다

SELECT  period, gubun, region, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   gubun = '기타대출'
GROUP BY CUBE(gubun, period, region)
;

**집합연산자
--수출 품목에 대한 테이블 생성
CREATE TABLE exp_goods(
        Country VARCHAR2(10),
        seq     NUMBER,
        Goods   VARCHAR2(80)
    );

INSERT INTO exp_goods VALUES ('일본', 1, '자동차');
INSERT INTO exp_goods VALUES ('일본', 2, '자동차부품');
INSERT INTO exp_goods VALUES ('일본', 3, '전자직접회로');
INSERT INTO exp_goods VALUES ('일본', 4, '선박');
INSERT INTO exp_goods VALUES ('일본', 5, '반도체웨이퍼');
INSERT INTO exp_goods VALUES ('일본', 6, '화물차');
INSERT INTO exp_goods VALUES ('일본', 7, '원유제외 석유류');
INSERT INTO exp_goods VALUES ('일본', 8, '건설기계');
INSERT INTO exp_goods VALUES ('일본', 9, '다이오드, 트래지스터');
INSERT INTO exp_goods VALUES ('일본', 10, '기계류');

COMMIT;

--수출품 테이블에서 국가가 한국인 상품을 모두 조회
```

--(단, 품목 번호 순서대로 조회할 것, 오름차순)

```
SELECT *
FROM exp_goods
WHERE country = '한국'
ORDER BY seq ASC;

SELECT goods
FROM exp_goods
WHERE country = '한국'
UNION
SELECT goods
FROM exp_goods
WHERE country = '일본'
ORDER BY 1;

SELECT goods
FROM exp_goods
WHERE country = '한국'
UNION ALL
SELECT goods
FROM exp_goods
WHERE country = '일본'
ORDER BY 1;
```

--한국만 수출하는 품목

```
SELECT goods
FROM exp_goods
WHERE country = '한국'
MINUS
SELECT goods
FROM exp_goods
WHERE country = '일본'
ORDER BY 1;
```

--일본만 수출하는 품목

```
SELECT goods
FROM exp_goods
WHERE country = '일본'
MINUS
SELECT goods
FROM exp_goods
WHERE country = '한국'
ORDER BY 1
;
```

<주의사항>
1 SELECT 리스트의 개수 및 데이터타입일치
2 ORDER BY 절은 마지막 SELECT 문에서만 사용 가능

```
SELECT  칼럼1, 컬럼2
FROM    테이블
WHERE   조건

UNION

SELECT  칼럼1, 컬럼2
FROM    테이블
ORDER BY 컬럼
;
--UNION
--UNION ALL

--INTERSECT
```

교집합 개념
두 개 이상의 개별 select 쿼리를 연결
개별 select 쿼리 반환 결과 중 중복된 것을 제외한 선행 쿼리 결과 추출

MIUNS
차집합
두 개 이상의 개별 select 쿼리를 연결
개별 select 쿼리 반환 결과 중 중복된 것을 제외한 선행 쿼리 결과 추출

--집합 연산자 제한사항
--개별 select 쿼리의 select리스트 개수와 데이터 타입이 일치해야함
order by절은 맨 마지막 개별 select 쿼리에
BLOB, CLOB, BFILE 같은 LOB 타입 컬럼은 집합
union, intersect, minus 연산자는 long

SELECT 칼럼1, 컬럼2 --select 절의 컬럼의 개수와 타입이 일치 1제한사항
FROM 테이블
WHERE 조건
GROUP BY 그룹컬럼
HAVING 그룹조건
--ORDER BY 중간에 쓰지 않고 마지막에 쓴다 2제한사항
집합연산자
SELECT 칼럼1, 컬럼2
FROM 테이블
ORDER BY 컬럼
;

--5장 1번문제 - OK

사원테이블에서 입사년도별 사원 수를 구하는 쿼리를 작성

사원테이블 employees
입사년도 hire_date
사원 수 년도마다 총 몇명??
-내가 푼 것
```
SELECT TO_CHAR(hire_date, 'YYYY'), COUNT(employee_id)
FROM employees
GROUP BY TO_CHAR(hire_date, 'YYYY')
ORDER BY 1;
```

--<풀이방법>
1.문제해석
>정보 분리, 사용할 테이블/컬럼/조건확인
>정보 찾기(동사): 사용할 쿼리문/함수확인

2. 쿼리문 (테이블, CRUD) 및 쿼리문 작성
3. 문제 해석 내용을 데이터베이스 문법에 맞게 적용
4. 앞서 작성한 쿼리문 구문에 맞게 대입
5. 결과 테스트
6. 오류 발생시 내용확인(오류발생행렬관련)
7. 결과 재 테스트

SELECT 컬럼
FROM 테이블
WHERE 조건
GROUP BY 그룹 칼럼
HAVING 그룹조건
ORDER BY 컬럼
;

사원테이블에서
입사년도별 > 포맷을 변경 TO_CHAR(값, 포맷)
사원수를 구하는 > COUNT(컬럼)

테이블: employees
컬럼: hire_date
함수(집계): TO_CHAR(값, 포맷), COUNT(칼럼)
--5장 2번문제 --OK
kor_loan_status 테이블에서 2012년도 월별, 지역별 대출 총 잔액을 구하는
쿼리는 작성하자
--나의 풀이
테이블 kor_loan_status
컬럼: 2012년도 월별 MM, 지역, 대출 총 잔액
함수: SUM
조건: 2012

```
SELECT period, region, SUM(loan_jan_amt)
FROM kor_loan_status
WHERE period LIKE '2012%'
GROUP BY period, region
ORDER BY 1
;
```

--5장 3번문제 XX

아래 쿼리를 rollup을 사용하지않고, 집합 연산자로 동일한 결과가
나오도록 쿼리를 작성해라

--시도 3 - 어떻게 각 기간의 총합계를 구할까??

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY period, gubun
UNION
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY period, gubun
;
```

GROUP BY에 나오는 컬럼을 그룹으로 묶어 계산하므로 period, gubun을 넣으면 두 컬럼 내용의 값이 나온다.

period만 넣을경우 gubun 의 값을 제외하기 때문에 총 값을 구할 수 있다.

<풀이방법**>**

--레벨별 데이터 결과

1. (기간, 구분)에 따른 대출총액

2.(기간)에 따른 대출총액

집합연산자를 사용해서 > 4가지: UNION, UNION ALL, INTERSECT, MINUS

동일한 결과가 나오도록 > SELECT

```
SELECT  period, gubun, SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY period, gubun
UNION
SELECT  period, '', SUM(loan_jan_amt)
FROM    kor_loan_status
WHERE   period LIKE '2013%'
GROUP BY period        ---gubun 을 빼고 써준다????
```

;

왜 구분을 위에는 생략하지 않고 밑에만?
조건에 기타대출 또는 주택담보대출을 설정하지 않고 값이 어떻게 나오나?
--> NULL쓰는 방법 **> NULL** 또는 ‘ ‘

	PERIOD	GUBUN	SUM(LOAN_JAN_AMT)
1	201310	기타대출	676078
2	201310	주택담보대출	411415.9
3	201310	(null)	1087493.9
4	201311	기타대출	681121.3
5	201311	주택담보대출	414236.9
6	201311	(null)	1095358.2

--5장 **4**번문제 **XO**

```
SELECT period,
    CASE WHEN gubun = '주택담보대출' THEN SUM(loan_jan_amt)
    ELSE 0 END 주택담보대출액,
    CASE WHEN gubun = '기타대출' THEN SUM(loan_jan_amt)
    ELSE 0 END  기타대출액
FROM kor_loan_status
WHERE period = '201311'
GROUP BY period, gubun;
```

위의 결과를 집합연산자로 동일한 결과를 추출하는 쿼리를 작성하자

--내가 풀이
--2013년 11월 기타대출액 총 합계 + 주택담보대출액

```
SELECT period, 0 주택담보대출액, SUM(loan_jan_amt) 기타대출액
FROM kor_loan_status
WHERE period = '201311'
    AND gubun = '기타대출'
GROUP BY period
```

```
UNION

SELECT period, SUM(loan_jan_amt) 주택담보대출액, 0 기타대출액
FROM kor_loan_status
WHERE period = '201311'
   AND gubun = '주택담보대출'
GROUP BY period
;
```

- 컬럼에 어떻게 널 값을 넣지??? >> 0 주택담보대출액/ 값이 null이 아니라 0
- SELECT 컬럼 값에 테이블에 없는 값을 넣을 수 있다.

--5장 문제 **5**번

```
SELECT region, SUM(loan_jan_amt) "201111",
       0 "201112", 0 "201210", 0 "201212", 0 "201310", 0 "201311"
FROM kor_loan_status
WHERE period = '201111'
GROUP BY region
;

SELECT region 지역,
   CASE WHEN period = '201111' THEN SUM(loan_jan_amt) ELSE 0  END "201111",
   CASE WHEN period = '201112' THEN SUM(loan_jan_amt) ELSE 0  END "201112"
 FROM kor_loan_status
WHERE period = '201111'
GROUP BY region, period
ORDER BY 1
;
```

<문제풀이>

테이블 kor_loan_status
컬럼 region, 컬럼"201111", 201112, 201210,
조건
함수: SUM(loan_jan_amt)

```
--<1방법> SUM 과 CASE 의 중첩
SELECT region 지역, SUM(CASE WHEN period = '201111' THEN loan_jan_amt ELSE 0 END)"201111",
            SUM(CASE WHEN period = '201112' THEN loan_jan_amt ELSE 0 END)"201112",
            SUM(CASE WHEN period = '201210' THEN loan_jan_amt ELSE 0 END)"201210",
            SUM(CASE WHEN period = '201211' THEN loan_jan_amt ELSE 0 END)"201211",
            SUM(CASE WHEN period = '201212' THEN loan_jan_amt ELSE 0 END)"201212",
            SUM(CASE WHEN period = '201310' THEN loan_jan_amt ELSE 0 END)"201310",
            SUM(CASE WHEN period = '201311' THEN loan_jan_amt ELSE 0 END)"201311"
FROM kor_loan_status
GROUP BY region
ORDER BY 1;


--SELECT region 지역, CASE WHEN period = '201111' THEN SUM(loan_jan_amt) ELSE 0 END "201111",
--            CASE WHEN period = '201112' THEN SUM(loan_jan_amt) ELSE 0 END"201112",
--            CASE WHEN period = '201210' THEN SUM(loan_jan_amt) ELSE 0 END"201210",
--            CASE WHEN period = '201211' THEN SUM(loan_jan_amt) ELSE 0 END"201211",
--            CASE WHEN period = '201212' THEN SUM(loan_jan_amt) ELSE 0 END"201212",
--            CASE WHEN period = '201310' THEN SUM(loan_jan_amt) ELSE 0 END"201310",
--            CASE WHEN period = '201311' THEN SUM(loan_jan_amt) ELSE 0 END"201311"
--FROM kor_loan_status
--GROUP BY region, loan_jan_amt
--ORDER BY 1;

--내가 놓친 것, case문을 썼지만 sum을 앞으로 빼주진 못했다

--<2방법> SELECT 중첩/활용
SELECT region, CASE WHEN period = '201111' THEN loan_jan_amt ELSE 0 END y1,
            CASE WHEN period = '201112' THEN loan_jan_amt ELSE 0 END y2,
            CASE WHEN period = '201210' THEN loan_jan_amt ELSE 0 END y3,
            CASE WHEN period = '201211' THEN loan_jan_amt ELSE 0 END y4,
            CASE WHEN period = '201212' THEN loan_jan_amt ELSE 0 END y5,
            CASE WHEN period = '201310' THEN loan_jan_amt ELSE 0 END y6,
            CASE WHEN period = '201311' THEN loan_jan_amt ELSE 0 END y7
FROM kor_loan_status;
```

```
SELECT region,
     SUM(y1) "201111", SUM(y2) "201112", SUM(y3) "201210",
       SUM(y4) "201211", SUM(y5) "201212", SUM(y6) "201310", SUM(y7) "201311"

FROM (SELECT region, CASE WHEN period = '201111' THEN loan_jan_amt ELSE 0 END y1,
            CASE WHEN period = '201112' THEN loan_jan_amt ELSE 0 END y2,
            CASE WHEN period = '201210' THEN loan_jan_amt ELSE 0 END y3,
            CASE WHEN period = '201211' THEN loan_jan_amt ELSE 0 END y4,
            CASE WHEN period = '201212' THEN loan_jan_amt ELSE 0 END y5,
            CASE WHEN period = '201310' THEN loan_jan_amt ELSE 0 END y6,
            CASE WHEN period = '201311' THEN loan_jan_amt ELSE 0 END y7
      FROM kor_loan_status)
GROUP BY region;

COMMIT;
```

--6장

*****SQL(DML)** 매우중요

CRUD: 소프트웨어가 가지는 기본적인 데이터를 처리 기능 **<**프로젝트에서 많이 사용될 예정**...>**

Create(생성**) Read(**읽기**) Update(**수정**) Delete (**삭제**)**

Create(생성) > INSERT

Read(읽기) > SELECT

Update(수정) > UPDATE

Delete (삭제) > DELETE

*****조인: 테이블간의 관계를 맺는 방법

1.내부조인

1)동등조인

- 가장 기본적이고 일반적인 조인방법
- where 절에서 등호= 연산자를 사용해 2개이상의 테이블이나 뷰를 연결한 조인 > 조인조건
- 컬럼 단위(기준으)로 조인조건 기술
- SELECT *

```
 FROM TAB1 a, TAB2 b
 WHERE a.col1 = b.col1

SELECT  T1.컬럼, T2.컬럼
FROM    테이블1 T1, 테이블2 T2
WHERE   T1.컬럼명 = T2.컬럼명 (같은이름을 가지고 있는 컬럼명) --조인조건
;

SELECT  *
FROM    employees T1, departments T2
WHERE   T1.manager_id = T2.manager_id
;

SELECT  T1.employee_id
FROM    employees T1, departments T2
WHERE   T1.manager_id = T2.manager_id
;
--부서번호를 기준으로 2개의 테이블을 동등 조인하고.
--사원테이블, 사원이름, 부서번호를 부서테이블의 부서이름 컬럼을 조회
SELECT  a.employee_id, a.emp_name, a.department_id, b.department_id
FROM    employees a, departments b
WHERE   a.department_id = b.department_id --조인조건
;
```

2)세미조인**:** 서브쿼리에서 존재하는 데이터만 메인쿼리에서 추출

EXISTS 연산자

: 조건에 만족하는 데이터가 하나라도 있으면 결과 반환

```
SELECT  컬럼
FROM    테이블1
WHERE   EXISTS( SELECT  컬럼
            FROM    테이블2
            WHERE   조인조건 테이블1.컬럼 = 테이블2.컬럼
);

--부서번호, 부서이름 조회
```

```
SELECT  d.department_id, department_name
FROM   departments d
WHERE   EXISTS( SELECT  *
            FROM    employees e
            WHERE   d.department_id = e.department_id
);
```

**테이블에서 행 삭제

DELETE dep4_1 테이블

WHERE dep_id 컬럼>= 100 and dep_id <= 106;

IN 연산자**:** 조인조건이 없다**,** 일반적인 쿼리의 형태 문장

: OR 논리연산자를 사용한 형태와 같다.

특징

1. where 절에 조인조건이 없다
2. IN 절에 서브쿼리 컬럼과 메인쿼리 조건절에 명시된 컬럼이 같다/ 메인쿼리 where절과 = 서브쿼리의 select
3. 메인쿼리 select 에 별칭이 없다, 이미 서브쿼리에서 값을 주었기 때문에

```
SELECT
FROM
WHERE   컬럼 IN ( SELECT 칼럼
                        FROM  테이블
                        WHERE 조건
            );

SELECT  *
FROM    employees e
WHERE   d.manager_id = e.manager_id --오류
;
```

사원테이블에서 급여가 3000만원 이상인 사원의 부서번호를 조회해

부서테이블의 부서번호에 내용이 포함되어 있으면 부서번호와 부서이름을 추출해 쿼리를 작성

```
SELECT  department_id, department_name      --별칭필요 없다
FROM    departments d
```

```
WHERE   d.department_id IN ( SELECT e.department_id  -- 같은 컬럼에서 비교해야한다
                    FROM  employees e
                    WHERE e.salary > 13000
              );
--조인하는이유?? 다른테이블에 정보가 있기 때문에~


SELECT  *
FROM    employees e
WHERE   e.salary > 13000
;
```

서브쿼리 : 팀장이 없는 부서 번호 조회

```
SELECT department_id
FROM departments
WHERE manager_id is null;
```

메인쿼리 (동등 조인): 사원테이블과 부서테이블에서 (
팀장이 없는 부서번호를 조회해 메인쿼리의 내용과 일치하면)사원번호,이름,번호 및 부서명을 조회

```
SELECT e.employee_id, e.emp_name, e.department_id, d.department_name
FROM    employees e, departments d
WHERE e.department_id = d.department_id;

SELECT e.employee_id, e.emp_name, e.department_id, d.department_name
FROM    employees e, departments d
WHERE e.department_id = d.department_id
   AND e.department_id IN (SELECT d.department_id
                FROM departments d
                WHERE d.manager_id is null)
;
```

안티조인**:** 세미조인**(EXISTS, IN)**과 반대개념**,** 앞에 **NOT**을 붙여 사용한다

--NOT IN

```
SELECT e.employee_id, e.emp_name, e.department_id, d.department_name
FROM    employees e, departments d
WHERE e.department_id = d.department_id
   AND e.department_id NOT IN (SELECT d.department_id
                                          FROM departments d
                                       WHERE d.manager_id is null)
;
```

--NOT EXISTS

서브쿼리

```
SELECT  *
FROM    departments
WHERE   manager_id is null
;
```

메인쿼리

팀장이 없는 부서에 존재하지 않는 데이터에서 사원 정보를 조회

```
SELECT  employee_id, emp_name, department_id
FROM    employees e
WHERE   NOT EXISTS (SELECT  *
            FROM    departments d
            WHERE   manager_id is null
         AND e.department_id = d.department_id)
;
```

셀프조인

사용예시

```
SELECT  employee_id, emp_name, manager_id
FROM    employees
WHERE   salary > 9000 and salary < 11000
;

SELECT
FROM    employees e1, employees e2
WHERE   el.manager_id = e2.employee_id
```

;

2. 외부조인

-일반조인을 확장한 개념

- 조인조건에 만족하는 데이터 뿐만 아니라, 어느 한 쪽 테이블에 조인 조건에 명시된 컬럼값이 없거나(null) 해당 로우가 아예 없더라도 데이터를 모두 추출1

- 조인조건에서 데이터가 없는 쪽 테이블 컬럼 끝에 (+)를 붙인다

- 조인조건이 여러개일 경우 모든 조인조건에 (+)를 붙여야한다

<일반조인>

동등조인: 기준컬럼의 값이 동일하고 널 값이 미포함

```
SELECT  d.department_id, d.department_name, j.job_id, j.department_id
FROM    departments d, job_history j
WHERE   d.department_id = j.department_id
;
```

<외부조인>

특징**:**

-기준컬럼의 값이 동일하지(해당되는 null값 때문에) 않고 널 값이 포함

- 외부조인할 때 조인조건 2개 이상인 경우, 모든 조인 조건에 (+) 표시를 할 것

null로 처리한 값도 포함시켜 보여주기 때문에~

- (+) 연산자가 붙은 조건과 OR 와 IN 연산자를 같이 사용 불가

- 한번에 한 테이블에만 외부조인 가능

```
SELECT  d.department_id, d.department_name, j.job_id, j.department_id
FROM    departments d, job_history j
WHERE   d.department_id = j.department_id(+)
;
```

기준컬럼이 아닌 컬럼에 (+) 표시, 즉 널 값이 포함됨

```
SELECT  e.department_id, e.emp_name, j.job_id, j.department_id
FROM    employees e, job_history j
WHERE   e.employee_id = j.employee_id(+)
;
```

```
SELECT  e.department_id, e.emp_name, j.job_id, j.department_id
FROM    employees e, job_history j
```

```
WHERE   e.employee_id = j.employee_id(+)  -- 밑의 조건으로 (+)의미가 없다
    and e.department_id = j.department_id
;
```

외부조인할 때 조인조건 2개 이상인 경우, 모든 조인 조건에 (+) 표시를 할 것
null로 처리한 값도 포함시켜 보여주기 때문에~

```
SELECT  e.department_id, e.emp_name, j.job_id, j.department_id
FROM    employees e, job_history j
WHERE   e.employee_id = j.employee_id(+)
    and e.department_id = j.department_id(+)
;
```

(+) 연산자가 붙은 조건과 OR 와 IN 연산자를 같이 사용 불가
--포괄 조인 운영 (+)는 OR 또는 IN의 연산수를 허용하지 않습니다

```
SELECT  e.department_id, e.emp_name, j.job_id, j.department_id
FROM    employees e, job_history j
WHERE   e.employee_id = j.employee_id(+)
    or e.department_id = j.department_id(+)
;
```

3. ANSI 조인

<오라클 조인 : 동등조인 >

```
SELECT  T1.컬럼, T2컬럼
FROM    테이블 T1, 테이블 T2
WHERE   T1컬럼 = T2컬럼
;
```

<ANSI 조인>

```
SELECT  T1.컬럼, T2컬럼
FROM    테이블 T1
INNER JOIN 테이블2 T2
ON  T1컬럼 = T2컬럼
WHERE 조건
;
```

<ANSI 조인>

```
SELECT  e.employee_id, d.department_name, e.hire_date
FROM    employees e
INNER JOIN  departments d
ON  e.department_id = d.department_id
WHERE    e.hire_date >= TO_DATE('2003-01-01') -- 문자열에서 날짜타입으로 변환
ORDER BY 3;
```

<오라클 조인 : 동등조인 >

```
SELECT  e.employee_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id = d.department_id
;
```

ANSI 외부조인

<오라클 조인: 외부조인>

```
SELECT  a.컬럼1, a컬럼2, b.컬럼.1, b.컬럼2
FROM    테이블 a, 테이블 b
WHERE   a.컬럼1 = b.컬럼
;
```

<ASNI 외부조인>

```
SELECT  a.컬럼1, a컬럼2, b.컬럼.1, b.컬럼2
FROM    테이블 a
LEFT (RIGHT) [OUTER] JOIN 테이블 b
ON (a.컬럼1 = b.컬럼)
WHERE...
;
```

```
--(+)와 의미는 같지만, (+) 널 값의 방향과는 반대된다
SELECT  a.emp_name, b.job_id
FROM    employees a LEFT OUTER JOIN job_history b
-- LEFT OUTER JOIN 왼쪽에 있는 테이블 employees을 기준으로 삼는다, RIGHT 일 경우 job_history테이블
--그러므로 왼쪽테이블을 다 쓰면 오른쪽은 null값이 나올 수 있다
ON (a.employee_id = b.employee_id)
;
```

```
SELECT  a.emp_name, b.job_id
FROM    employees a
RIGHT OUTER JOIN job_history b
ON a.employee_id = b.employee_id
 AND a.department_id = b.department_id
 ;
```

ANSI CROSS 조인

<오라클 조인: 카타시안 조인>

```
SELECT  a.employee_id, a.emp_name, b.department_id, b.department_name
FROM    employees a, departments b;
```

<ANSI CROSS 조인>

```
SELECT a.employee_id, a.emp_name, b.department_id, b.department_name
FROM    employees a
CROSS JOIN  departments b;
```

ANSI FULL OUTER 조인

```
DROP TABLE ex4_1;

CREATE TABLE dep4_1(
      dep_id      VARCHAR(10),
      dep_name    VARCHAR(20),
      manager_id VARCHAR(10)
);

INSERT INTO emp4_1 VALUES('100', '유재석', '10');
INSERT INTO emp4_1 VALUES('101', '강호동', '20');
INSERT INTO emp4_1 VALUES('102', '김구라', '50');
INSERT INTO emp4_1 VALUES('103', '김희철', '30');
INSERT INTO emp4_1 VALUES('105', '이경규', '50');
INSERT INTO emp4_1 VALUES('106', '박나래', '60');

DELETE dep4_1
```

```
WHERE dep_id >= 100 and dep_id <= 106;

CREATE TABLE emp4_1(
        emp_id      VARCHAR(10),
        emp_name    VARCHAR(20),
        department_id VARCHAR(10)
);

ALTER TABLE emp4_1 RENAME COLUMN department_id to dep_id;
--기본
SELECT  e.emp_name, d.dep_id
FROM    emp4_1 e, dep4_1 d
WHERE   e.dep_id = d.dep_id;

SELECT  e.emp_name, d.dep_id
FROM    emp4_1 e
FULL OUTER JOIN dep4_1 d
ON  e.dep_id = d.dep_id;
```

4. 서브쿼리**: SQL** 문장 내 보조로 사용되는 다른 **SELECT** 문

-메인쿼리 연관성에 따라: 조인조건 유무

SELECT절 **>** 서브쿼리
FROM절 **>** 인라인뷰
WHERE절 **>** 중첩쿼리 **–** 연관성 있는 없는 서브 쿼리

<서브쿼리>

```
SELECT  AVG(salary)
FROM    employees
--WHERE   조건
;
```

1) 연관성없는 서브쿼리
<메인쿼리>: 내 급여가 평균 아래인지 위인지
총 사원수(107) 평균 더 많이 (51명)

```
SELECT  count(*)
FROM    employees
WHERE   salary >= (SELECT  AVG(salary) FROM    employees)
;
```

<서브쿼리>: 상급부서가 없는 부서번호 조회

```
SELECT  department_id
FROM    departments
WHERE   parent_id is null
;
```

<메인쿼리>: 상급부서가 없는 부서의 개수 조회

```
SELECT COUNT(*)
FROM    departments
```

```
WHERE   department_id NOT IN (SELECT  department_id
                    FROM    departments
                    WHERE   parent_id is null)
;

<서브쿼리>: 경력테이블로부터 사원번호와 직업번호 조회
SELECT  employee_id, job_id
FROM    job_history;


<메인쿼리>: 사원테이블에서 사원이름과 매칭, 사원번호와 직업번호도 조회
--동시에 2개의 컬럼 값이 동일할 경우
SELECT  emp_name, employee_id, job_id
FROM    employees
WHERE   (employee_id, job_id) IN (SELECT  employee_id, job_id
                        FROM    job_history)
;
1개의 컬럼 값이 동일할 경우
SELECT  emp_name, employee_id, job_id
FROM    employees
WHERE   employee_id IN (SELECT  employee_id
                        FROM    job_history)
;

COMMIT;

사원테이블의 급여를 전직원 동일하게 평균급여로 변경
UPDATE  employees
SET     salary = (SELECT AVG(salary) FROM employees) --가로넣기
;

평균급여보다 많이 받는 사람 삭제

SELECT COUNT(*) FROM employees;


DELETE  employees
```

```
WHERE   salary > (SELECT AVG(salary) FROM employees)
;

ROLLBACK;
```

2) 연관성 있는 서브쿼리

<서브쿼리>: 경력테이블에서 부서번호 조회

```
SELECT  department_id
FROM    job_history;
```

<메인쿼리>: 경력테이블에서 조회된 부서번호를 부서명과 함께 조회

```
SELECT  department_id, department_name
FROM    departments d
WHERE   EXISTS (SELECT department_id
                FROM    job_history j
                WHERE d.department_id = j.department_id);
```

<서브쿼리1> 사원테이블에서 사원명 조회 107건

```
SELECT  emp_name
FROM    employees;
WHERE
```

<서브쿼리2> 부서명 조회 27건

```
SELECT  department_name
FROM    departments;
WHERE
```

<메인쿼리> 사원번호, 사원명, 부서번호, 부서명

```
SELECT  employee_id,
       (SELECT  e.emp_name
        FROM employees e
        WHERE  e.employee_id = j.employee_id ) emp_name,
```

```
        department_id,
        (SELECT  d.department_name
         FROM departments d
         WHERE d.department_id = j.department_id)department_id
FROM    job_history j;
```

<서브쿼리1> 사원테이블에서 평균급여조회

```
SELECT AVG(salary)
FROM employees;
```

<서브쿼리2> 평균급여 보다 높은급여를 받는 사원의 부서번호 조회

```
SELECT  department_id
FROM    employees
WHERE   salary > (SELECT AVG(salary) FROM employees)
--GROUP BY department_id
;
```

<메인쿼리> 평균급여 보다 높은급여를 받는 사원의 부서번호와 매칭되는 부서명 조회

```
SELECT  d.department_name, d.department_name
FROM    departments d
WHERE    EXISTS (SELECT  department_id
                 FROM    employees e
                 WHERE   e.department_id = d.department_id
                  AND e.salary > (SELECT AVG(salary) FROM employees))
;
```

<서브쿼리1> 부서테이블에서 상위부서(번호)가 90번인 부서번호 조회

```
SELECT  department_id
FROM    departments
WHERE   parent_id = 90;
```

<서브쿼리2> 사원테이블에서 상위부서(번호)가 90번인 부서번호, 평균급여 조회

```
SELECT  e.department_id, AVG(e.salary)
FROM    employees e,
        departments d
WHERE   d.parent_id = 90
   AND  e.department_id = d.department_id
GROUP BY   e.department_id
;
```

<서브쿼리3> (사원테이블에서 상위부서(번호)가 90번인 부서번호, 평균급여)에서 평균급여만 추출

```
SELECT  avg_sal
FROM    (SELECT  e.department_id, AVG(e.salary) avg_sal -- 별칭
          FROM    employees e, departments d
         WHERE   d.parent_id = 90
         AND  e.department_id = d.department_id
        GROUP BY   e.department_id)
--WHERE
;
```

<메인쿼리> 상위부서가 90번(기획부)인 모든 사원의 급여를 자신의 부서별 평균급여로 갱신

```
 UPDATE employees
 SET    salary = 부서별 평균급여
 WHERE  department_id IN (SELECT  department_id
                    FROM    departments
                    WHERE   parent_id = 90)
 ;

-- 서브쿼리 넣기 198p
 UPDATE employees e1
 SET    e1.salary = (SELECT  avg_sal
                FROM      (SELECT  e.department_id, AVG(e.salary) avg_sal -- 별칭
                          FROM    employees e, departments d
                          WHERE   d.parent_id = 90
                            AND  e.department_id = d.department_id
                         GROUP BY   e.department_id) x1
               WHERE e1.컬럼 = x1.컬럼 조인 조건         )
 WHERE  department_id IN (SELECT  department_id
```

```
                    FROM    departments
                    WHERE   parent_id = 90)
;
COMMIT;

-- WHERE 조인조건 넣기
UPDATE employees e1
SET    e1.salary = (SELECT  avg_sal
                FROM      (SELECT  e.department_id, AVG(e.salary) avg_sal -- 별칭
                        FROM    employees e, departments d
                        WHERE   d.parent_id = 90
                          AND  e.department_id = d.department_id
                       GROUP BY   e.department_id) x1
              WHERE e1.department_id = x1.department_id        )
WHERE  department_id IN (SELECT  department_id
                    FROM    departments
                    WHERE   parent_id = 90)
;

ROLLBACK;

UPDATE  사원테이블 e1
SET     e1.급여 = (부서별 평균급여(서브쿼리))
WHERE   e1.부서번호 IN (상위부서가 기획부인 부서 (서브쿼리))
;


--인라인뷰: FROM 절에 서브쿼리가 사용된 경우

<서브쿼리1> 판매월, 평균판매금액 조회 >> 월평균판매액
조건1 2000년 1월~ 2000년 12월까지 판매된
조건2 이탈리아에서 판매된

SELECT  sales_month, ROUND (AVG(amount_sold)) month_avg
FROM    sales s, countries t, customers c
WHERE   sales_month BETWEEN '200001' AND '200012'
   AND country_name = 'Italy'
```

```
   AND s.cust_id = c.cust_id
   AND t.country_id = c.country_id
GROUP BY sales_month
;
```

문제> sales 와 countries 테이블에서 조인할 수있는 공통된 컬럼이 없다!
풀이> 다른테이블을 조인을 위한 연결고리로 만든다

<서브쿼리2> 판매테이블로부터 평균판매금액만 조회 >> 년평균판매액
조건1 2000년 1월~ 2000년 12월까지 판매된,
조건2 이탈리아에서 판매된

```
SELECT  ROUND (AVG(s.amount_sold)) year_avg
FROM    sales s, countries t, customers c
WHERE   sales_month BETWEEN '200001' AND '200012'
   AND country_name = 'Italy'
   AND s.cust_id = c.cust_id
   AND t.country_id = c.country_id
;

--<메인쿼리> 2000년 이탈리아 평균 매출액(연평균)보다 큰 매출액을 달성한 월 평균 매출액을 조회
--
--SELECT  판매월, 평균판매액
--FROM    테이블
--WHERE   월평균판매액 > 연평균판매액

--SELECT  a.sales_month, a.month_avg 이렇게 써도 된다
SELECT  a.*
FROM    (SELECT  sales_month, ROUND (AVG(amount_sold)) month_avg
            FROM    sales s, countries t, customers c
            WHERE   sales_month BETWEEN '200001' AND '200012'
               AND country_name = 'Italy'
               AND s.cust_id = c.cust_id
               AND t.country_id = c.country_id
```

```
                GROUP BY sales_month
                ) a,
                 (SELECT  ROUND (AVG(s.amount_sold)) year_avg
                   FROM    sales s, countries t, customers c
                    WHERE   sales_month BETWEEN '200001' AND '200012'
                     AND country_name = 'Italy'
                     AND s.cust_id = c.cust_id
                     AND t.country_id = c.country_id ) b
WHERE   a.month_avg > b.year_avg
;

COMMIT;
```

6장 1번

```
SELECT e.employee_id, e.emp_name, js.job_title, j.start_date, j.end_date, d.department_name
FROM employees e, job_history j, departments d, jobs js
WHERE e.employee_id = 101
   AND e.department_id = d.department_id
   AND e.employee_id = j.employee_id
   AND js.job_id = j.job_id
 ;
```

6장 2번

```
SELECT a.employee_id, a.emp_name, b.job_id, b.department_id
FROM employees a,
    job_history b
WHERE a.employee_id = b.employee_id(+) -- (+)외부조인은 한 쪽으로만
 AND a.department_id(+) = b.department_id;
```

=> 두 개의 테이블을 outer-join 할 수 없습니다

```
SELECT  department_id
FROM    employees
GROUP BY    department_id
```

```
ORDER BY    department_id;

SELECT  department_id
FROM   job_history
GROUP BY    department_id
ORDER BY    department_id;
-- employees 테이블에 값이 더 나와서 기준컬럼으로 준다.
--외부조인은 값이 적은 job_histroy 에 준다



6장 3번
외부조인을 할 때(+)연산자를 같이 사용할 수 없는데
IN절에 사용하는 값이 한 개이면 사용할 수 있다. 그 이유는?
--IN 연산자에 대한 개념

SELECT  employee_id
FROM    employees
WHERE   e.department_id IN (10, 20, 30);

--동일 연산 수행 코드
SELECT  e.employee_id
FROM    employees e, departments d
WHERE   e.department_id IN (10, 20, 30);

SELECT  e.employee_id
FROM    employees e, departments d
WHERE   e.department_id = 10
  OR    e.department_id = 20
  OR    e.department_id = 30
;



SELECT  e.employee_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id(+) = d.department_id
    OR e.department_id IN (10, 20, 30)
;
```

--괄 조인 운영 (+)는 OR 또는 IN의 연산수를 허용하지 않습니다
<예시>
1. 외부조인을 IN과 함께 쓰는 경우(오류)

```
SELECT  e.employee_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id IN ( e.department_id (+), 200)
;
```

--비정상~ ()괄호안에 값이 한개만 넣어야 값이 나옴

2. 같은 의미의 코드(오류)

```
SELECT  e.employee_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id(+) = d.department_id
   OR e.department_id IN  = 200
;
```

3. IN절 데이터를 1개만 넣는 경우 (정상)

```
SELECT  e.employee_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id(+) = d.department_id
   AND e.department_id IN (10)
;
```

4.3번과 같은 코드 (정상)

```
SELECT  e.employee_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id(+) = d.department_id
 AND e.department_id IN (10)
;
```

답) IN절에 사용하는 값이 1개인 경우
의미상 OR을 사용한 것이 아니므로 외부조인 가능
--외부조인시 (+) 기호를 어떤 컬럼에 붙여야 할까?

```
SELECT  e.employee_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id(+) = d.department_id
```

```
   OR e.department_id =10
   OR  e.department_id =20
   OR  e.department_id =30
;
```

6장 4번

```
SELECT  a.department_id, a.department_name
FROM    departments a
INNER JOIN  employees b
ON  a.department_id = b.department_id
WHERE b.salary > 3000
ORDER BY a.department_name;
```

6장 5번

EXISTS > 조인조건 유: 연관성 있는 서브쿼리
IN > 조인조건 무: 연관성 없는 서브쿼리

```
SELECT  d.department_id, d.department_name
FROM    departments d, job_history j
WHERE   d.department_id =  j.department_id;

SELECT  d.department_id, d.department_name
FROM    departments d
WHERE   d.department_id IN (SELECT department_id
                    FROM job_history)
 ;
```

6장 6번
연도별 이탈리아 최소매출사원명 최소매출액

```
SELECT  SUBSTR(s.sales_month,1,4), c.country_name, e.emp_name, MIN(s.amount_sold)
FROM    sales s, employees e, countries c
```

```
WHERE   c.country_name = 'Italy'
   AND s.employee_id = e.employee_id
GROUP BY SUBSTR(s.sales_month,1,4), e.emp_name, c.country_name
ORDER BY 1;

--풀이방법
SELECT SUBSTR(a.sales_month, 1, 4) as years,
            a.employee_id,
            SUM(a.amount_sold) AS amount_sold
         FROM sales a,
            customers b,
            countries c
        WHERE a.cust_id = b.CUST_ID
          AND b.country_id = c.COUNTRY_ID
          AND c.country_name = 'Italy'
        GROUP BY SUBSTR(a.sales_month, 1, 4), a.employee_id;
--이거 본다음
--결과는 이렇게 만들면 되요
SELECT  years,
             MAX(amount_sold) AS max_sold,
             MIN(amount_sold) AS min_sold
        FROM ( SELECT SUBSTR(a.sales_month, 1, 4) as years,
                    a.employee_id,
                    SUM(a.amount_sold) AS amount_sold
                FROM sales a,
                    customers b,
                    countries c
               WHERE a.cust_id = b.CUST_ID
                 AND b.country_id = c.COUNTRY_ID
                 AND c.country_name = 'Italy'
               GROUP BY SUBSTR(a.sales_month, 1, 4), a.employee_id
             ) K
        GROUP BY years;
```

8장 **PL/SQL**

/* */ 여러줄 생략

좌항 [:=] 우항: 우항의 값을 좌항에 할당
:= > 대입연산자

**블록
[이름부] -- CREATE로 이름생성
[IS(AS)] -- 프로시저, 사용자함수의 변수/상수 선언
  변수명 데이터타입;
DECLARE -- 일반 프로그램의 변수/상수 선언
  선언부1
  선언부2
BEGIN
  실행부1;
  실행부2;
[EXCEPTION
  예외처리부]
END;

함수: 입력값을 활용해 특정 연산을 수행 후 결과 값 반환
프로시저: 특정 로직을 처리하고 결과 값은 반환하지 않음. 서브프로그램
*/

--익명블록

```
DECLARE
   vi_num  NUMBER;       --vi_num 이름으로 변수 선언(생성)
BEGIN                    --여기서부터 프로그램을 시작하겠다
    vi_num := 100;       --어떤 값이던지 넣을 수 있다
   DBMS_OUTPUT.PUT_LINE(vi_num);  --가로가 있으면 함수, 함수(기능): 로그확인
END;                                    (입력값)
```

DBMS_OUTPUT.PUT_LINE: 오라클 sqp developer 도구에 있는 DBMS에 output 해주는 line

/ :프로그램 작성이 끝났음 SQL*PLUS 설정 여부에 따라 선택사용
PL/SQL 프로시저가 성공적으로 완료되었습니다. => 컴파일 + 실행완료
/
DECLARE

```
    a INTEGER := 2**2*3**2;
    b NUMBER := 3+4;
BEGIN
       /*  DBMS_OUTPUT.PUT_LINE
    : DBMS 출력으로 입력값에 대한 로그 확인*/
   DBMS_OUTPUT.PUT_LINE('a='||TO_CHAR(a));
   DBMS_OUTPUT.PUT_LINE('b='||b);       --자동형변환
   DBMS_OUTPUT.PUT_LINE(b);
END;
/
DECLARE --선언부
--a는 2의 2승 곱하기 3의2승
    a INTEGER := 2**2*3**2;
    b NUMBER := 3+4;
    c VARCHAR2(30) := '우리나라';
    d BOOLEAN := false;
    e DATE := SYSDATE;
    f INT := 33;
BEGIN --실행부
    DBMS_OUTPUT.PUT_LINE('============================');
    DBMS_OUTPUT.PUT_LINE('a='||TO_CHAR(a));
    DBMS_OUTPUT.PUT_LINE('b='||b);       --자동형변환
    DBMS_OUTPUT.PUT_LINE(b);
    DBMS_OUTPUT.PUT_LINE(c);
    --불리안 값 출력방법
     DBMS_OUTPUT.PUT_LINE('d='|| CASE WHEN 'ture' ELSE 'false' END); --
DBMS_OUTPUT.PUT_LINE(d);
     DBMS_OUTPUT.PUT_LINE(e);
     DBMS_OUTPUT.PUT_LINE(f);
      DBMS_OUTPUT.PUT_LINE(b+f);
END;
/
DECLARE
    vs_emp_name VARCHAR2(80);   --잘못된 데이터타입 설정 위험!
    vs_dep_name VARCHAR2(80);   --변수선언
BEGIN
    SELECT e.emp_name, d.department_name
```

```
   INTO    vs_emp_name, vs_dep_name     --순서, 개수 일치
   FROM    employees e, departments d
   WHERE  e.department_id = d.department_id
      AND employee_id = 100
    ;
   DBMS_OUTPUT.PUT_LINE(vs_emp_name);
END;
/
DECLARE
   num CONSTANT NUMBER := 10;
BEGIN
   DBMS_OUTPUT.PUT_LINE('=======================================');
   DBMS_OUTPUT.PUT_LINE('num='||num);
END;
```

8장 1번

구구단 3단

```
DECLARE
  a  NUMBER := (3*1);
  b  NUMBER := (3*2);
  c  NUMBER := (3*3);
  d  NUMBER := (3*4);
  e  NUMBER := (3*5);
  f  NUMBER := (3*6);
 g   NUMBER := (3*7);
 h   NUMBER := (3*8);
  i  NUMBER := (3*9);
BEGIN
   DBMS_OUTPUT.PUT_LINE(a);
   DBMS_OUTPUT.PUT_LINE(b);
   DBMS_OUTPUT.PUT_LINE(c);
   DBMS_OUTPUT.PUT_LINE(d);
   DBMS_OUTPUT.PUT_LINE(e);
   DBMS_OUTPUT.PUT_LINE(f);
   DBMS_OUTPUT.PUT_LINE(g);
```

```
    DBMS_OUTPUT.PUT_LINE(h);
    DBMS_OUTPUT.PUT_LINE(i);
END;
```

<문제풀이>

유형1

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('==================================');
    DBMS_OUTPUT.PUT_LINE('3*1=3');
    DBMS_OUTPUT.PUT_LINE('3*2=6');
    DBMS_OUTPUT.PUT_LINE('3*3=9');
    DBMS_OUTPUT.PUT_LINE('3*4=12');
    DBMS_OUTPUT.PUT_LINE('3*5=15');
    DBMS_OUTPUT.PUT_LINE('3*6=18');
    DBMS_OUTPUT.PUT_LINE('3*7=21');
    DBMS_OUTPUT.PUT_LINE('3*8=24');
    DBMS_OUTPUT.PUT_LINE('3*9=27');
END;
```

유형2

```
DECLARE
    num NUMBER := 1;  --> 2 > 3
    dan NUMBER := 2;
BEGIN
    DBMS_OUTPUT.PUT_LINE('==================================');
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
```

```
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
    num := num + 1;
    DBMS_OUTPUT.PUT_LINE('3 * ' || num || ' = ' || 3*num);
END;
/
--상수
DECLARE
    num1 NUMBER :=10;
    num2 CONSTANT NUMBER := 10;
BEGIN
    DBMS_OUTPUT.PUT_LINE('======================');
    DBMS_OUTPUT.PUT_LINE('num1=' || num1);
    DBMS_OUTPUT.PUT_LINE('num2=' || num2);
    num1 := 20;
    DBMS_OUTPUT.PUT_LINE('num1=' || num1);
    DBMS_OUTPUT.PUT_LINE('num1+num2=' || (num1+num2));
END;                                                    – 문자가 아니고, 숫자. () 가로 써줌
/
```

**연산자(4) > 우선순위를 고려

수식연산자
문자연산자
비교연산자
논리연산자
대입연산자

DBMS_OUTPUT.PUT_LINE(): DBMS 출력창에 로그 출력

*****변수: 데이터의 저장 공간, 정장할 값을 변경 가능
**상수: 데이터 저장 공간, 초기 저장값을 변경 불가

SQL: 집합적 언어

PL/SQL: 절차적 언어(순서가 중요!)