```
REM Author :
REM Date :
REM Objective : Chapter 2. Basic SELECT
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM SELECT의 기능
1. Selection : 조건검색, Row에 대한 필터링
2. Projection : column 에 대한 필터링
3. Join : 여러 테이블에서의 검색


REM SELECT Syntax

   SELECT [DISTINCT | ALL]  {* | column1, column2 [AS [alias]] | expr}
   FROM table_name
   WHERE condition
   ORDER BY column [ASC | DESC];

1. SELECT 절 다음에 질의하고 싶은 칼럼을 차례대로 나열한다. 이 때 여러 개의 칼럼 구분은 쉼표(,)로 한다.
2. FROM 절 다음에는 조회할 테이블 이름을 적는다.
3. * : 모든 칼럼을 조회한다.
4. ALL : 모든 결과 ROW를 보여준다.(기본값)
5. DISTINCT : 중복된 ROW 를 제외한 ROW를 보여준다.
6. expr : SQL 함수를 사용하거나, 수학 연산을 포함한 표현식
7. alias : 칼럼에 대한 별칭 사용.
8. Default Column Heading : column 명이 대문자로 Display 된다.
9. Default Data Justification : Number 값은 오른쪽 정렬, Character 와 Date 값은 왼쪽 정렬된다.

REM 모든 열 선택
   SELECT * FROM dept;

   SELECT *
   FROM emp;

   SELECT * FROM emp;

   SQL> SET pagesize 1000   -- 1000줄을 한 페이지로 설정하는 SQL*Plus 명령 실행
   SQL> /  -- SQL*Plus buffer 에 들어있는 SQL 문장을 다시 실행
   SQL> COL[UMN] mgr FOR[MAT] 9999  --숫자 칼럼의 크기를 4자리로 조정하는 SQL*Plus 명령 실행
   SQL> COL ename FORMAT A8  -- 문자칼럼의 크기를 8자리로 조정하는 SQL*Plus 명령 실행


REM 특정 열 선택
1. 각 열의 구분은 "," 로 한다.
   SELECT empno, ename, sal
   FROM emp;

   SELECT empno, ename, job, mgr FROM emp;


REM 산술연산자 : 수학 연산 표현식
1. +, - : 음수, 혹은 양수를 나타내는 기호. 단항 연산자.
2. *(multiply), /(divide) : 곱하기, 나누기를 의미. 이항 연산자.
3. +(add), -(subtract) : 더하기, 빼기를 의미. 이항 연산자.
4. 연산자의 우선순위가 있다.   1 --> 2 --> 3
5. 우선순위가 높은 연산 먼저 수행하며, 같은 우선순위의 연산자들을 왼쪽에서 오른쪽으로 순서대로 계산해 나간다.
6. 괄호를 사용하여 우선순위를 조절할 수 있다.

   SELECT empno, ename, sal, sal + 100
   FROM emp;

   SELECT sal, -sal FROM emp;
   SELECT sal, sal * 1.1 FROM emp;
   SELECT sal, comm, sal + comm FROM emp;
   SELECT sal, -sal + 100 * -2 FROM emp;
   SELECT sal, (-sal + 100) * -2 FROM emp;
   SELECT empno, ename, sal, sal * 12 FROM emp;
   SELECT empno, ename, sal, sal * 12 + comm FROM emp;
   SELECT empno, ename, sal, sal + comm * 12 FROM emp;
   SELECT empno, ename, sal, (sal + comm) * 12 FROM emp;


REM NULL value
1. NULL 이란?
```

```
   1) 특정 행, 특정 열에 대한 아직 값을 알 수 없는 상태, 의미가 없는 상태를 표현
   2) 이용할 수 없거나, 지정되지 않거나, 알 수 없거나, 적용할 수 없는 값
   3)아직 정의되지 않은 미지의 값
   4)현재 데이터를 입력하지 못하는 경우

2. NULL(ASCII 0)은 0(zero, ASCII 48) 또는 공백(blank, ASCII 32)과 다르다.

3. 연산의 대상에 포함되지 않는다.
4. NULL 값을 포함한 산술 연산 식의 결과는 언제나 NULL 이다.
5. NOT NULL 또는 Primary Key 제약조건이 걸린 칼럼에서는 NULL VALUE가 나타날 수 없다.
6. NULL 인 칼럼은 Length 가 0 이므로 data를 위한 물리적 공간을 차지 하지 않는다.

   SELECT empno, job, comm FROM emp;
   --NULL 인 값은 비어있는 것으로 표현된다.
   --job이 salesman인 사원들에게만 커미션이 적용되며, 사번 7844인 사원의 커미션은 0이다.

   SELECT empno, ename, sal * 12 + comm
   FROM emp;
   --comm 값이 NULL 인 경우 연봉은 얼마인가? 연봉 계산한 수식의 column heading은 어떻게 나타나는가?
   --comm 값이 NULL 인 row 의 경우 (sal + comm) * 12를 하면 결과도 모두 NULL 이 된다.
   --또한, expression 전체가 column heading 으로 나타난다.


REM IFNULL function
1. NULL 값을 어떤 특정한 값으로 치환할 때 사용
2. 치환할 수 있는 값의 형태는 숫자형, 문자형, 날자형 모두 가능

3. Syntax
   IFNULL(expr1, expr2)
   --If expr1 is not NULL, IFNULL() returns expr1; otherwise it returns expr2.
   --expr1 : NULL
   --expr2 : 치환값
   --expr1값이 NULL 아니면 expr1 값을 그대로 사용
   --만약 expr1 값이 NULL이면, expr2 값으로 대체

4. 예
   SELECT IFNULL(1, 0);  --> 1
   SELECT IFNULL(NULL, 10);  --> 10
   SELECT IFNULL(1/0, 10);   --> 10
   SELECT IFNULL(1/0, 'yes') ---> yes

   IFNULL(comm, 0)
   IFNULL(hiredate, '12/09/04')
   IFNULL(job, 'No Job')

   --위에서 연봉을 구하는 Query 를 NVL 함수를 사용하여 제대로 나올 수 있도록 고쳐보자.
   SELECT empno, ename, sal, comm, sal * 12 + IFNULL(comm, 0)
   FROM emp;

   SELECT empno, comm, IFNULL(comm, 0)
   FROM emp;

   SELECT IFNULL(mgr, 'No Manager')
   FROM emp;


REM Alias 별칭
1. column header 에 별칭을 부여 할 수 있다.
2. SELECT 절에 expression 을 사용할 때 도움이 된다.
3. 열 이름 바로 뒤에 기술한다. 또는 열이름과 별칭 사이에 AS를 사용할 수 있다.
4. 별칭에 공백이나 특수문자나 한글사용할 때, 대소문자를 기술할 때(기본값은 모두 대문자)에는 "" 로 기술한다.

   SELECT empno 사번 FROM emp;
   SELECT sal * 12 연봉 FROM emp;
   SELECT sal * 12 annual_salary FROM emp;
   SELECT sal * 12 Annual_Salary FROM emp;
   SELECT sal * 12 Annual Salary FROM emp;  --Error 발생
   SELECT ename  "Name" , sal AS "Salary", sal * 12 + IFNULL(comm, 0)  AS "Annual Salary"
   FROM emp;


REM Concatenation Operator (연결 연산자)
1. Oracle에서는 문자열 리터럴을 이을 때에는 '||' 를 사용한다.
2. 하지만 MySQL에서는 연결연산자(||)가 없기 때문에 CONCAT()를 사용한다.
```

3. **character** string 들을 여결하여 하나의 결과 string 을 만들어 낸다.

```
SELECT CONCAT(empno, ' ', ename) FROM emp;
SELECT CONCAT(empno, ' ', ename, ' ', hiredate) FROM emp;
--number 이나 date값은 default 형태의 character 값으로 자동 변환한 후 연결된다.
```

REM Literals (상수)
1. Literal 은 상수 값을 의미.
2. **Character** literal 은 작은 따옴표로 묶고, Number literal 은 따옴표 없이 표현한다.
3. **Character** literal을 작은 따옴표로 묶어 주어야 MySQL Server 는 keyword나 객체 이름을 구별할 수 있다.

```
SELECT CONCAT('Emp# of ', ename, ' is ', empno) FROM emp;
SELECT CONCAT(dname, ' is located at ', loc) FROM dept;
SELECT CONCAT(ename, ' is a ', job) AS "Employee" FROM emp;
SELECT CONCAT(ename, ' ', sal) FROM emp;
SELECT CONCAT(ename, ' is working as a ', job) FROM emp;
SELECT 'Java is a language.' FROM emp;  --14번 출력
SELECT 'Java is a language.' FROM dept;  --4번 출력
SELECT 'Java is a language.';
```

REM Duplicate **Values**(중복 행 제거하기)
1. 일반 Query는 **ALL** 을 사용하기 때문에 중복된 행이 출력된다.
2. **DISTINCT** 를 사용하면 중복된 행의 값을 제거한다.
3. **DISTINCT** 는 **SELECT** 바로 뒤에 기술한다.
4. **DISTINCT** 다음에 나타나는 **column**은 모두 **DISTINCT** 에 영향을 받는다.

```
SELECT job FROM emp;
SELECT ALL job FROM emp;
SELECT DISTINCT job FROM emp;
SELECT deptno FROM emp;
SELECT DISTINCT deptno FROM emp;
SELECT deptno, job FROM emp;
SELECT DISTINCT deptno, job FROM emp;
```

REM **WHERE** 절
1. 사용자들이 자신이 원하는 자료만을 검색하기 위해서
2. Syntax

```
SELECT column...
FROM table_name
WHERE conditions;
```

3. **WHERE** 절을 사용하지 않으면 **FROM** 절에 명시된 **table**의 모든 **ROW**를 조회하게 된다.
4. **table**내의 특정 **row**만 선택하고 싶을 때 **WHERE** 절에 조건식을 사용한다.
5. MySQL Server 는 **table**의 **row**를 하나씩 읽어 **WHERE** 절의 조건식을 평가하여 **TRUE**로 만족하는 것만을 선택한다.
6. condition을 평가한 결과는 **TRUE, FALSE, NULL** 중의 하나이다.
7. condition 내에서 **character** 와 date 값의 literal은 작은 따옴표를 사용하고, NUMBER 값은 그대로 사용한다.
8. condition 에서 사용하는 **character** 값은 대소문자를 구별하지 않는다.
   1) **WHERE** ename **=** 'JAMES';
   2) **WHERE** ename **=** 'james';

9. date 타입의 변경은 **DATE_FORMAT()**를 사용한다.
10. **WHERE** 는 **FROM** 다음에 와야 한다.
11. **WHERE** 절에 조건이 없는 FTS(**Full Table** Scan) 문장은 **SQL** Tunning의 1차적인 검토 대상이 된다.
12. **WHERE** 조건절의 조건식은 아래 내용으로 구성된다.
    -**Column** 명(보통 조건식의 좌측에 위치)
    -비교 연산자
    -문자, 숫자, 표현식(보통 조건식의 우측에 위치)
    -비교 **Column**명 (**JOIN** 사용시)

REM 비교연산자
--<, >, <=, >=, =, !=, <>(같지 않다)

```
--직위가 CLERK 인 사원의 이름과 직위 및 부서번호를 출력하시오.
SELECT ename, job, deptno
FROM emp
WHERE job = 'CLERK';

SELECT empno, ename, job
FROM emp
```

```
    WHERE empno = 7934;

    SELECT empno, ename, job, hiredate
    FROM emp
    WHERE hiredate = '1981-12-03';

    SELECT empno, ename
    FROM emp
    WHERE ename = 'JAMES';

    SELECT empno, ename
    FROM emp
    WHERE ename = 'james';

    SELECT dname
    FROM dept
    WHERE deptno = 30;

    SELECT ename, sal
    FROM emp
    WHERE sal >= 1500;

    --1983년 이후에 입사한 사원의 사번, 이름, 입사일을 출력하시오.
    SELECT empno, ename, hiredate
    FROM emp
    WHERE hiredate >= '1983-01-01';

    --급여가 보너스(comm) 이하인 사원의 이름, 급여 및 보너스를 출력하시오
    SELECT ename, sal, comm
    FROM emp
    WHERE sal <= IFNULL(comm, 0);

    --10번 부서의 모든 사람들에게 급여의 13%를 보너스로 지급하기로 했다. 이름, 급여, 보너스 금액, 부서번호를 출력하시오.
    SELECT ename, sal, sal * 0.13, deptno
    FROM emp
    WHERE deptno = 10;

    --30번 부서의 연봉을 계산하여, 이름, 부서번호, 급여, 연봉을 출력하라. 단, 년말에 급여의 150%를 보너스로 지급한다.
    SELECT ename, deptno, sal, sal * 12 + IFNULL(comm, 0) + sal * 1.5 AS "년봉"
    FROM emp
    WHERE deptno = 30;

    --부서번호가 20인 부서의 시간당 임금을 계산하시오. 단, 1달의 근무일수는 12일이고, 1일 근무시간은 5시간이다. 출력양식은 이름, 급여,
    시간당 임금을 출력하라.
    SELECT ename, sal, sal / 12 / 5
    FROM emp
    WHERE deptno = 20;

    --모든 사원의 실수령액을 계산하여 출력하시오. 단, 이름, 급여, 실수령액을 출력하시오. (실수령액은 급여에 대해 10%의 세금을 뺀 금액)
    SELECT ename, sal, sal - sal * 0.1 AS "실수령액"
    FROM emp;

    --사번이 7788인 사원의 이름과 급여를 출력하시오.
    --급여가 3000이 넘는 직종을 선택하시오.
    --PRESIDENT를 제외한 사원들의 이름과 직종을 출력하시오.
    --BOSTON 지역에 있는 부서이 번호와 이름을 출력하시오.


REM 논리연산자
    --AND(&&), OR(||), NOT(!)

    --사원테이블에서 급여가 1000불이상이고, 부서번호가 30번인 사원의 사원번호, 성명, 담당업무, 급여, 부서번호를 출력하시오.
    SELECT empno, ename, job, sal, deptno
    FROM emp
    WHERE sal >= 1000 AND deptno = 30;

    --사원테이블에서 급여가 2000불이상이거나 담당업무가 매니저인 사원의 정보중 사원번호, 이름, 급여, 업무를 출력하시오.
    SELECT empno, ename, sal, job
    FROM emp
    WHERE sal >= 2000  OR job = 'MANAGER';


REM SQL 연산자
1. BETWEEN A AND B : A보다 같거나 크고, B보다 작거나 같은
```

2. **IN(list) :** LIST 안에 있는 멤버들과 같은
3. A **LIKE** B **[ESCAPE** 'C'**]:** A가 B의 **패**턴과 **일**치하면 **TRUE,** 보통 **%,** _ 연산자와 같이 사용**, escape** 을 사용하면 B의 **패**턴 중에서 C를 상수로 취급한다**.**
4. **IS NULL / IS NOT NULL : NULL** 여부**를** 테스트

1)**BETWEEN** A **AND** B

```
--사원테이블에서 월급이 1300불에서 1500불까지의 사원정보중 성명, 담당업무, 월급을 출력하시오.
SELECT ename, job, sal
FROM emp
-- WHERE sal >= 1300 AND sal <= 1500;
WHERE sal BETWEEN 1300 AND 1500;

SELECt ename, job, sal
FROM emp
WHERE sal BETWEEN 1500 AND 1300;
--반드시 작은 값이 먼저 나와야 한다.

SELECT ename FROM emp
WHERE hiredate BETWEEN '1982-01-01' AND '1982-12-31';

--급여가 2000 에서 3000 사이인 사원을 출력하시오.
SELECT ename, job, sal FROM emp
WHERE sal BETWEEN 2000 AND 3000;
```

2) **IN**

```
--사원테이블에서 업무가 회사원, 매니저, 분석가인 사원의 이름, 업무를 출력하시오.
SELECT ename AS "이름",  job
FROM emp
-- WHERE job = 'CLERK' OR job = 'MANAGER' OR job = 'ANALYST';
WHERE job IN('CLERK', 'MANAGER', 'ANALYST');

--관리자의 사원번호가 7902, 7566, 7788인 모든 사원의 사원번호, 이름, 급여 및 관리자의 사원번호를 출력하시오.

SELECT dname FROM dept WHERE deptno IN(10,20);

SELECT ename FROM emp
WHERE job IN('ANALYST', 'CLERK')

--BOSTON 이나 DALLAS 에 위치한 부서를 출력하시오.
SELECT dname, loc FROM dept
WHERE loc IN('BOSTON', 'DALLAS');

--30, 40번 부서에 속하지 않는 사원들을 출력하시오.
SELECT ename, deptno FROM emp
WHERE deptno NOT IN(30,40);

--DALLAS 의 20번 부서, 또는 CHICAGO의 30번 부서를 출력하시오.
SELECT * FROM dept
WHERE (deptno, loc) IN )(20, 'DALLAS'), (30, 'CHICAGO'));
```

3)**LIKE(%, _)**

```
--Wildcard : %(0개 이상의 문자 대표), _ (1개의 문자 대표)
--Wildcard 문자를 일반 문자로 사용하고 싶을 때 ESCAPE 을 사용한다.
--ESCAPE 문자 바로 뒤에 사용된 Wildcard 문자는 일반 문자로 취급한다.

SELECT ename, job, hiredate FROM emp
-- WHERE hiredate LIKE '1987%';
WHERE hiredate >= '1987-01-01';

SELECT dname FROM dept
WHERE dname LIKE 'A%';

--이름이 A로 시작하는 사원을 출력하시오.
SELECT ename FROM emp
WHERE ename LIKE 'A%';

--사번이 8번으로 끝나는 사원을 출력하시오.
SELECT empno, ename FROM emp
WHERE empno LIKE '%8';

--1982에 입사한 사원을 출력하시오.
SELECT ename, hiredate FROM emp
```

```
    -- WHERE hiredate LIKE '1982%';
    -- WHERE hiredate BETWEEN '1982-01-01' AND '1982-12-31';
    WHERE hiredate >= '1982-01-01' AND hiredate <= '1982-12-31';

    SELECT empno, ename
    FROM emp
    WHERE ename LIKE 'MILLE_';

    SELECT empno, ename
    FROM emp
    WHERE ename LIKE '%$_TEST' ESCAPE '$';


   4) IS NULL / IS NOT NULL
    --column 의 NULL 여부를 판단할 때에는 반드시 'IS NULL' 혹은 'IS NOT NULL' 연산자를 사용한다.
    SELECT ename FROM emp
    WHERE comm IS NULL;
    WHERE comm IS NOT NULL;

    --comm 지급 대상인 사원을 출력하시오.
    SELECT ename, comm FROM emp
    WHERE comm IS NOT NULL;

    SELECT ename, mgr
    FROM emp
    WHERE mgr IS NULL;


REM 연산자 우선순위
1. !
2. -, ~, 괄호 ()
3. ^
4. *, /, DIV, %, MOD : 산술연산자
5. -, + : 산술연산자,  ||
6. <<, >>
7. &
8. |
9. =, !=, >=, >, <=, <, <>, !=, IS, LIKE, IN
10. BETWEEN, CASE, WHEN, THEN, ELSE
11. NOT
12. AND, &&
13. XOR
14. OR, ||

    SELECT ename, job FROM emp
    WHERE NOT job = 'ANALYST';

    SELECT ename, sal, deptno FROM emp
    WHERE sal > 2500 AND deptno = 20;

    SELECT deptno, dname FROM dept
    WHERE deptno = 10 OR deptno = 20;

    --업무가 SALESMAN 이거나 업무가 MANAGER 이고, 급여가 1300불이상인 사람의 사원번호, 이름, 업무, 급여를 출력하시오.
    SELECT empno, ename, job, sal
    FROM emp
    WHERE (job LIKE 'S%' OR job LIKE 'M%') AND sal >= 1300;

    --직종이 CLERK 인 사원 중에서 급여가 1000 이상인 사원을 출력하시오.
    SELECT ename, job, sal FROM emp
    WHERE job = 'CLERK' AND sal >= 1000;


REM ORDER BY
1. 기본적으로 데이터는 정렬되지 않는다.
2. 같은 쿼리를 수행할 때마다 결과가 다르게 나올 수 있다.
3. 별칭은 정렬에 영향을 주지 않는다.
4. Syntax

       SELECT column_list
       FROM table
       [WHERE conditions]
       [ORDER BY column[, column] {ASC | DESC};
```

5. 특징
  1) 기본적으로 오름차순정렬한다.
    --숫자인경우 (1 --> 999)
    --날짜인경우 (옛날 --> 최근)
    --문자인겨우 (알파벳순서, 유니코드순)
  2) NULL 은 오름차순일 경우는 제일 처음에, 내림차순인 경우에는 제일 마지막에 출력

6. ORDER BY 절에 정렬의 기준이 되는 column 을 여러개 지정할 수 있다. 첫번째 column 으로 정렬한 다음, 그 column 값이 같은 row 들에 대해서는 두 번째 column 값으로 정렬한다.
7. 오름차순(ASC) 정렬이 기본이며, 내림차순으로 정렬하고자 할 때에는 DESC를 사용한다.
8. ORDER BY 절에 column 이름 대신 positional notation 을 사용할 수도 있다. Position number 는 SELECT 절에서의 column 순서를 의미한다.

```
--입사일자 순으로 정렬하여 사원번호, 이름, 입사일자를 출력하시오.
SELECT empno, ename, hiredate
FROM emp
ORDER BY hiredate DESC;

--부서번호가 20번인 사원의 연봉 오름차순으로 출력하시오.
SELECT empno, ename, sal, comm, sal * 12 + IFNULL(comm, 0) AS "Annual"
FROM emp WHERE deptno = 20
ORDER BY "Annual" ASC;

--부서번호로 정렬한 후, 부서번호가 같을 경우 급여가 많은 순으로 사원번호, 사원이름, 업무, 부서번호, 급여를 출력하시오.
SELECT empno, ename, job, deptno, sal
FROM emp
ORDER BY deptno ASC, sal DESC;
```

```
REM Author :
REM Date :
REM Objective : Chapter 3. Built-in Function
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM   SQL function
-A function is a stored program that you can pass parameters into and then return a value.
1. Built Function(내장함수)
2. Stored Function(사용자 정의 함수)

REM 단일행 함수(Single Row function)
1. Syntax
   function_name(column | expression [ arg1, arg2...])

2. 종류
  1)제어흐름 함수
  2)숫자 함수
  3)날자시간 함수
  4)문자열 함수
  5)집합 함수
  6)변환 함수
  7)기타 함수


REM 제어 흐름 함수(Flow Control Functions)
1. IF()
   1)Definition
     -Returns a value if a condition is TRUE, or another value if a condition is FALSE.

   2)Syntax
     IF(expr1, expr2, expr3)

   3)만일 expr1이 참이면, expr2를 리턴한다.
   4)그렇지 않으면 expr3을 리턴한다.

   SELECT IF(1 > 2, 2, 3);   --> 3
   SELECT IF(1 < 2, 'yes', 'no') --> 'yes'


2. CASE
   1)Definition
     -Goes through conditions and return a value when the first condition is met.
     -like an IF-THEN-ELSE statement.
     -So, once a condition is true, it will stop reading and return the result.
     -If no conditions are true, it will return the value in the ELSE clause.
     -If there is no ELSE part and no conditions are true, it returns NULL.

   2)Syntax
     CASE
        WHEN compare_value1 THEN result1
        WHEN compare_value2 THEN result2
        WHEN compare_value3 THEN result3
        ...
        ELSE resultN
     END

   SELECT job, sal,
     CASE   WHEN  job = 'ANALYST' THEN sal * 1.1
            WHEN  job = 'CLERK'    THEN sal * 1.15
            WHEN  job = 'MANAGER'  THEN  sal * 1.2
            ELSE  sal
     END  AS "SALARY"
   FROM emp;


3. IFNULL
   1)Definition
     -Returns a specified value if the expression is NULL.
     -If the expression is NOT NULL, this function returns the expression.

   2)Syntax
     IFNULL(expr1, expr2)
        -If expr1 is not NULL, IFNULL() returns expr1; otherwise it returns expr2.
        -expr1 : NULL
```

```
        -expr2 : 치환값
        -expr1값이 NULL 아니면 expr1 값을 그대로 사용
        -만약 expr1 값이 NULL이면, expr2 값으로 대체


4. NULLIF
   1)Definition
     -Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned.

   2)Syntax
     NULLIF(expr1, expr2)

   SELECT NULLIF(1,1);  --> NULL
   SELECT NULLIF(1,2);  --> 1
   SELECT NULLIF("Hello", "world"); --> 'Hello'



REM 숫자 함수(Numeric Functions)

1. ABS
   1) 숫자 값을 절대값으로 바꾼다.
   2)Syntax
     ABS(expression)

   SELECT ABS(-15)


2. CEIL(CEILING)
   1)Returns the smallest integer value that is bigger than or equal to a number.
   2)Syntax
     CEIL(number)

   SELECT CEIL(15.7)


3. DEGREES
   1)Convert radians to degrees
   2)Syntax
     DEGREES(number)

   SELECT DEGREES(PI()*2);  --> 360
   SELECT DEGREES(PI());  --> 180
   SELECT DEGREES(PI() / 2);  --> 90


4. FLOOR
   1)Returns the largest integer value that is smaller than or equal to a number.
   2)Syntax
     FLOOR(number)

   SELECT FLOOR(15.7)


5. MOD
   1)Returns the remainder of a number divided by another number.
   2)Syntax
     MOD(m, n)
       -m MOD n
       -m % n

   SELECT ename, sal, comm, MOD(sal, comm)
   FROM emp
   WHERE job = 'SALESMAN';

   SELECT 10 / 3, MOD(10, 3);
   SELECT sal, MOD(sal, 30);


6. PI
   SELECT PI();


7. POW(POWER)
```

1)Returns the value of a number raised to the power of another number.

```
SELECT POWER(3,2)
```

8. RADIANS
1)Converts a degree value into radians.
2)Syntax
RADIANS(number)

```
SELECT RADIANS(-45);  -->  -0.7853981633974483
SELECT RADIANS(90);  -->  1.5707963267949
```

9. RAND
1)Returns a random number between 0 (inclusive) and 1 (exclusive).
2)Syntax
RAND(seed)

```
SELECT RAND();  --> 0.26097273012713784
```

10. ROUND
1)Rounds a number to a specified number of decimal places.
2)Syntax
ROUND(column | expression, n)
3) 열, 표현식, 값을 소수점 n째 자리로 반올림
4) n을 지정하지 않은 경우 소수점 이하 값이 없어짐
5) n이 음수이면 소수점 왼쪽 수가 반올림

```
SELECT ROUND(45.925, 2), ROUND(45.925, 0), ROUND(45.925, -1);
SELECT ROUND(-1.23);
SELECT ROUND(-1.58);
SELECT ROUND(1.298, 1);
SELECT ROUND(1.298, 0);
```

11. SIGN
1) 주어진 수가 양수이면 1, 0이면 0, 음수이면 -1

```
SELECT SIGN(-12);
```

12. SQRT
1)Returns the square root of a number.

```
SELECT SQRT(13);
```

13. TRUNCATE
1)Truncates a number to the specified number of decimal places.
2)열, 표현식, 값을 소수점 n째 자리까지 남기고 버린다.
3)Syntax
TRUNC (column | expression, n)

```
SELECT TRUNCATE(345.156, 0);  --> 345
SELECT TRUNCATE(1.223,1);
SELECT TRUNCATE(1.999,1);
SELECT TRUNCATE(122, -2);
```

REM 날짜 함수

1. 날짜데이터
1)MySQL은 표준 출력 형식으로 주어진 날짜 또는 시간 유형에 대한 값을 검색하지만 사용자가 제공하는 입력 값에 대한 다양한 형식을 해석하려고 시도한다.
2)다른 형식의 값을 사용하면 예측할 수 없는 결과가 발생할 수 있다.
3)MySQL은 여러 형식으로 값을 해석하려고 시도하지만 날짜 부분은 항상 월-일-년 또는 일-월-보다는 년-월-일 순서(예: '98-09-04')로 지정해야 한다.
4)다른 곳에서 일반적으로 사용되는 연도 순서(예: '09-04-98', '04-09-98'), 다른 순서의 문자열을 년-월-일 순서로 변환하려면 STR_TO_DATE() 함수가 유용할 수 있다.
5)2자리 연도 값을 포함하는 날짜는 세기를 알 수 없기 때문에 모호하다.
6)MySQL은 다음 규칙을 사용하여 2자리 연도 값을 해석한다.

```
    -Year values in the range 70-99 become 1970-1999.
    -Year values in the range 00-69 become 2000-2069.


2. ADDDATE
   1)Adds a time/date interval to a date and then returns the date.
   2)Syntax
   ADDDATE(date, INTERVAL value addunit)
   OR
   ADDDATE(date, days)

   SELECT ADDDATE("2017-06-15 09:34:21", INTERVAL 15 MINUTE);    --> 2017-06-15 09:49:21
   SELECT ADDDATE("2017-06-15 09:34:21", INTERVAL -3 HOUR);      --> 2017-06-15 06:34:21
   SELECT ADDDATE("2017-06-15", INTERVAL -2 MONTH);              --> 2017-04-15
   SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);              --> '2008-02-02'
   SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);               --> '2008-02-02'
   SELECT ADDDATE('2008-01-02', 31);                            --> '2008-02-02'


3. ADDTIME
   1)Adds a time interval to a time/datetime and then returns the time/datetime.
   2)Syntax
   ADDTIME(datetime, addtime)

   --Add 5 seconds and 3 microseconds to a time and return the datetime:
   SELECT ADDTIME("2017-06-15 09:34:21.000001", "5.000003");   --> 2017-06-15 09:34:26.000004

   --Add 2 hours, 10 minutes, 5 seconds, and 3 microseconds to a time and return the datetime:
   SELECT ADDTIME("2017-06-15 09:34:21.000001", "2:10:5.000003"); --> 2017-06-15 11:44:26.000004

   -Add 5 days, 2 hours, 10 minutes, 5 seconds, and 3 microseconds to a time and return the datetime:
   SELECT ADDTIME("2017-06-15 09:34:21.000001", "5 2:10:5.000003"); --> 2017-06-20 11:44:26.000004

   --Add 2 hours, 10 minutes, 5 seconds, and 3 microseconds to a time and return the time:
   SELECT ADDTIME("09:34:21.000001", "2:10:5.000003"); --> 11:44:26.000004


4. CURDATE
   1)Returns the current date.
   2)The date is returned as "YYYY-MM-DD" (string) or as YYYYMMDD (numeric).
   3)This function equals the CURRENT_DATE() function.
   4)Syntax
     CURDATE()

   SELECT CURDATE() + 1;  --> 20210831
   SELECT CURDATE();       --> '2021-08-30'
   SELECT CURDATE() + 0;  --> 20210830


5. CURRENT_DATE
   1)Returns the current date.
   2)Syntax
     CURRENT_DATE()

   SELECT CURRENT_DATE() + 1;   --> 20210831


6. CURRENT_TIME
   1)Returns the current time.
   2)The time is returned as "HH-MM-SS" (string) or as HHMMSS.uuuuuu (numeric).
   3)This function equals the CURTIME() function.
   4)Syntax
     CURRENT_TIME()

   SELECT CURRENT_TIME() + 1;   --> 224909
   SELECT CURTIME();  --> --> '22:49:58'
   SELECT CURTIME() + 0;  --> 224958.000000


7. CURRENT_TIMESTAMP
   1)Returns the current date and time.
   2)The date and time is returned as "YYYY-MM-DD HH-MM-SS" (string) or as YYYYMMDDHHMMSS.uuuuuu (
   numeric).

```

```
   SELECT CURRENT_TIMESTAMP();  --> '2021-08-30 22:52:13'
   SELECT CURRENT_TIMESTAMP() + 1 --> 20210830225329


8. DATE
   1)Extracts the date part from a datetime expression.
   2)Syntax
      DATE(expression)

   SELECT DATE("2017-06-15 09:34:21");  --> '2017-06-15'


9. DATEDIFF
   1)Returns the number of days between two date values.
   2)Syntax
      DATEDIFF(date1, date2)

   SELECT DATEDIFF("2017-06-25 09:34:21", "2017-06-15 15:25:35");  --> 10
   SELECT DATEDIFF("2017-01-01", "2016-12-24"); --> 8


10. DATE_FORMAT
   1)Formats a date as specified.
   2)Syntax
      DATE_FORMAT(date, format)

   SELECT DATE_FORMAT("2017-06-15", "%M %d %Y");  --> June 15 2017
   SELECT DATE_FORMAT("2017-06-15", "%W %M %e %Y"); --> Thursday June 15 2017


11. DAY
   1)Returns the day of the month for a given date (a number from 1 to 31).
   2)This function equals the DAYOFMONTH() function.
   3)Syntax
      DAY(date)

   SELECT DAY("2017-06-15 09:34:21");  --> 15
   SELECT DAY(CURDATE()); --> 30


12. DAYNAME
   1)Returns the weekday name for a given date.
   2)Syntax
      DAYNAME(date)

   SELECT DAYNAME("2017-06-15 09:34:21");  --> Thursday
   SELECT DAYNAME(CURDATE());  --> Monday


13. LAST_DAY
   1)Extracts the last day of the month for a given date.
   2)Syntax
      LAST_DAY(date)

   SELECT LAST_DAY("2017-02-10 09:34:00");   --> 2017-02-28


14. MAKEDATE
   1)Creates and returns a date based on a year and a number of days value.
   2)Syntax
      MAKEDATE(year, day)

   SELECT MAKEDATE(2017, 175);  --> 2017-06-24


15. MAKETIME
   1)Creates and returns a time based on an hour, minute, and second value.
   2)Syntax
      MAKETIME(hour, minute, second)

   SELECT MAKETIME(16, 1, 0);  --> 16:01:00


16. NOW
```

```
   1)Returns the current date and time.

   SELECT NOW();


17. PERIOD_ADD
   1)Adds a specified number of months to a period.
   2)Return the result formatted as YYYYMM.
   3)Syntax
      PERIOD_ADD(period, number)

   SELECT PERIOD_ADD(201703, 15);  --> 201806


18. PERIOD_DIFF
   1)Returns the difference between two periods. The result will be in months.
   2)Syntax
      PERIOD_DIFF(period1, period2)

   SELECT PERIOD_DIFF(201703, 201803);  --> -12
   SELECT PERIOD_DIFF(1703, 1612);  --> 3


19. QUARTER
   1)Returns the quarter of the year for a given date value (a number from 1 to 4).
   2)Syntax
      QUARTER(date)

   SELECT QUARTER("2017-01-01 09:34:21");   --> 1


20. STR_TO_DATE
   1)Returns a date based on a string and a format.

   SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');  --> '2013-05-01'
   SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');  --> '2013-05-01'



REM 문자 함수
1. ASCII, CHAR
   1)Returns the ASCII value for the specific character.
   2)Returns the String value for the specific ASCII code.
   3)Syntax
      ASCII(str)
      CHAR(number)

   SELECT ASCII('2');  --> 50
   SELECT CHAR(77,121,83,81,'76'); --> 'MySQL'


2. BIT_LENGTH
   1)Returns the length of the string str in bits.
   2)Syntax
      BIT_LENGTH(str)

   SELECT BIT_LENGTH('hello');  --> 40
   SELECT BIT_LENGTH('안녕');   --> 48


3. CHAR_LENGTH
   1)Returns the length of the string str, measured in characters.
   2)Syntax
      CHAR_LENGTH(str)

   SELECT CHAR_LENGTH("SQL Tutorial");   --> 12
   SELECT CHAR_LENGTH("안녕");   --> 2


4. LENGTH
   1)Returns the length of a string (in bytes).
   2)Syntax
      LENGTH(str)
```

```
   SELECT LENGTH("SQL Tutorial");  --> 12
   SELECT CHAR_LENGTH("안녕");   --> 6


5. FORMAT
   1)The FORMAT() function formats a number to a format like "#,###,###.##", rounded to a specified
   number of decimal places, then it returns the result as a string.
   2)Syntax
      FORMAT(number, decimal_places)

   SELECT FORMAT(250500.5634, 0);  --> '250,501'
   SELECT FORMAT(12332.123456, 4);  --> '12,332.1235'
   SELECT FORMAT(12332.1,4);  --> '12.332.1000'
   SELECT FORMAT(12332.2,0);  --> '12,332'
   SELECT FORMAT(12332.2,2,'de_DE');  --> '12.332,20'
      -If no locale is specified, the default is 'en_US'


6. LOWER
   1) 소문자로 변환
   2) Syntax
   LOWER(column | expression)

   SELECT empno, ename
   FROM emp
   WHERE LOWER(ename) = 'scott';


7. UPPER
   1) 대문자로 변환
   2) Syntax
   UPPER (column | expression)

   SELECT empno, ename, deptno
   FROM emp
   WHERE ename = 'blake';

   SELECT empno, ename, deptno
   FROM emp
   WHERE ename = UPPER('blake');


8. CONCAT
   1)Adds two or more expressions together.
   2)Syntax
   CONCAT(expression1, expression2, expression3,...)

   SELECT CONCAT("SQL ", "Tutorial ", "is ", "fun!")


9. SUBSTR[ING]
   1)Extracts a substring from a string (starting at any position).
   2)Syntax
   SUBSTR(string, start, length)

   SELECT SUBSTRING('Quadratically',5);  --> 'ratically'
   SELECT SUBSTRING('foobarbar' FROM 4); --> 'barbar'
   SELECT SUBSTRING('Quadratically',5,6);  --> 'ratica'
   SELECT SUBSTRING('Sakila', -3); --> 'ila'
   SELECT SUBSTRING('Sakila', -5, 3);  --> 'aki'


10. INSTR
   1)Returns the position of the first occurrence of substring substr in string str.
   2)Syntax
   INSTR(str,substr)

   SELECT INSTR('foobarbar', 'bar');    --> 4
   SELECT INSTR('xbar', 'foobar');      --> 0


11. LPAD | RPAD
   1)Left-pads a string with another string, to a certain length.
   2)Syntax
```

```
   LPAD(string, length, lpad_string)

   SELECT LPAD("SQL Tutorial", 20, "ABC");  --> ABCABCABSQL Tutorial


12. LTRIM | RTRIM
   1)Removes leading spaces from a string.
   2)Syntax
   LTRIM(string)

   SELECT LTRIM("    SQL Tutorial");  --> SQL Tutorial


13. REPLACE
   1)Replaces all occurrences of a substring within a string, with a new substring.
   2)Syntax
   REPLACE(string, substring, new_string)

   SELECT REPLACE("SQL Tutorial", "SQL", "HTML");  --> HTML Tutorial


14. REPEAT
   1)Repeats a string as many times as specified.
   2)Syntax
   REPEAT(string, number)

   SELECT REPEAT("SQL Tutorial", 3);  --> SQL TutorialSQL TutorialSQL Tutorial


15. REVERSE
   1)Reverses a string and returns the result.
   2)Syntax
   REVERSE(string)

   SELECT REVERSE("SQL Tutorial");  --> lairotuT LQS


16. SPACE
   1)Returns a string of the specified number of space characters.
   2)Syntax
      SPACE(number)

   SELECT SPACE(6);   --> '      '



REM 변환함수
1. CAST
   1)Converts a value (of any type) into the specified datatype.
   2)Syntax
      CAST(value AS datatype)

   SELECT CAST(150 AS CHAR);  --> '150'
   SELECT CAST("14:06:10" AS TIME);  --> 14:06:10


2. CONVERT
   1)Converts a value into the specified datatype or character set.
   2)Syntax
      CONVERT(value, type)
      OR
      CONVERT(value USING charset)

   SELECT CONVERT(150, CHAR);   --> '150'



REM Information Functions
1. DATABASE
   1)Returns the default (current) database name as a string in the utf8 character set.
   2)Syntax
      DATABASE()

   SELECT DATABASE();
```

2. **USER(SESSION_USER, SYSTEM_USER)**
   1)**Returns** the **current** MySQL **user** name **and host** name **as** a string **in** the utf8 **character set.**
   2)Syntax
      USER()

   SELECT USER();

3. VERSION
   1)**Returns** a string that indicates the MySQL server version**.**
   2)The string uses the utf8 **character set.**
   3)Syntax
      VERSION**()**

   SELECT VERSION**();**

REM Author :
REM Date :
REM Objective : Chapter 4. **Aggregate Function**
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM 여러 행(그룹) 함수
1. 개념
   1)그룹 함수는 행 집합에 작용하여 그룹 당 하나의 결과를 생성한다.
   2)이 집합은 전체 테이블 또는 그룹으로 분류된 테이블이다.

2. 종류
   1)**AVG**
   2)**COUNT**
   3)**MAX**
   4)**MIN**
   5)**SUM**
   6)**STDDEV**
   7)**VARIANCE**

3. 사용 지침
   1)**DISTINCT**를 지정하면 함수는 중복되지 않는 값만 검토하고 **ALL**을 지정하면 중복 값을 포함한 모든 값을 검토한다. 기본은 **ALL**
   2)인수에 대한 데이터 유형은 CHAR, VARCHAR, NUMBER 또는 DATE이며 expression 형식으로 나열됨
   3)**COUNT(*)**를 제외한 모든 그룹 함수는 **NULL** 값을 무시

4. **AVG**
   1)**Returns** the average **value of** expr.
   2)The **DISTINCT option** can be used **to return** the average **of** the **distinct values of** expr.
   3)Syntax
      AVG([DISTINCT | ALL] expression**)**
   4)expression 값의 평균
   5)**NULL** 무시

```
SELECT AVG(sal), MAX(sal), MIN(sal), SUM(sal)
FROM emp
WHERE job LIKE 'SALES%';

SELECT AVG(comm)
FROM emp;

SELECT AVG(NVL(comm, 0))
FROM emp;
```

5. **COUNT**
   1)**Returns** a **count of** the number **of** non-**NULL values of** expresssion.
   2)Syntax
      COUNT({ * | [DISTINCT | ALL] expression **})**
   3)행 수, expression은 **NULL**을 제외한 값을 계산
   4)*을 사용하면 중복 행 및 **NULL** 값을 가진 행을 포함하여 선택한 행 모두를 계산

```
SELECT COUNT(*)
FROM emp
WHERE deptno = 30;

SELECT COUNT(comm)
FROM emp
WHERE deptno = 30;

SELECT COUNT(DISTINCT (deptno))
FROM emp;
```

6. **MAX**
   1)**Returns** the maximum **value in** a **set of values.**
   2)Syntax
      MAX([DISTINCT | ALL] expression**)**
   3)expression의 최대값이며 **NULL** 값을 무시

7. **MIN**
   1)**Returns** the minimum **value in** a **set of values.**

2)Syntax
MIN([DISTINCT | ALL] expression**)**
3)expression의 최소값이며 **NULL** 값을 무시

```
SELECT MIN(hiredate), MAX(hiredate)
FROM emp;

SELECT MIN(ename), MAX(ename)
FROM emp;
```

8. **SUM**
1)Calculates the **sum of** a **set of values.**
2)Syntax
SUM(expression**)**
3)**NULL** 값 무시

9. **STDDEV**
1)**Returns** the population standard deviation **of value.**
2)Syntax
STDDEV([DISTINCT | ALL] expression**)**
3)**NULL** 값을 무시
4)분산의 제곱근

```
SELECT STDDEV(sal)
FROM emp;
```

10. **VARIANCE**
1)**Returns** the population standard **variance of value.**
2)Syntax
VARIANCE([DISTINCT | ALL] expression**)**
3)**NULL** 값을 무시
4)**편차** 제곱의 평균

REM **GROUP BY**
1. 지금까지는 테이블을 하나의 대형 정보 그룹**으**로 취급했음
2. 테이블 정보**를** 더 작은 그룹**으**로 나눠**야** 할 경우 **GROUP BY**절을 사용
3. **GROUP BY** 절을 사용하여 테이블 행을 그룹**으**로 나눈 후 그룹 함수**를** 사용하여 각 그룹에 대한 요**약** 정보**를** 반환 가능
4. 지침
1)**GROUP BY**절에 열을 포함시켜**야** 한다**.**
2)**GROUP BY**절엔 열 별칭을 사용할 수 없다**.**
3)기본적**으**로 **GROUP BY**목**록**에 포함된 열은 오름**차**순**으**로 정**렬**된다**.** 무시하려면 **ORDER BY**사용
5. Syntax

```
SELECT
FROM
WHERE
GROUP BY;
```

1. **SELECT** 절
1)**SELECT** 절에서 **GROUP** 함수**(**복수행 함수**)**와 **column**이 같이 기술되면**,** 반드시 **GROUP BY** 절이 기술되어**야** 한다**.**

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

SELECT deptno, MAX(sal)
FROM emp
GROUP BY deptno
ORDER BY deptno;

SELECT deptno, MAX(sal), MIN(sal), SUM(sal), AVG(sal)
FROM emp
GROUP BY deptno
ORDER BY deptno DESC;
```

2)그러나 **SELECT** 절에 복수행함수만 기술되고**, column** 을 사용하지 않았다면 **GROUP BY** 를 필요로 하지 않는다**.**

```
SELECT MAX(sal), MIN(sal), SUM(sal), AVG(sal)
```

```
    FROM emp

  3)Multiple Grouping
    -부서별, 업무별로 그룹하여 결과를 부서번호, 업무, 인원수, 급여의 평균, 급여의 합을 구하시오.

    SELECT deptno, job, COUNT(*), AVG(sal), SUM(sal)
    FROM emp
    GROUP BY deptno, job
    ORDER BY deptno ASC, job DESC;

  4)여러 열을 기준으로 분류
    -하나 이상의 GROUP BY 열 나열
    -열 순서에 따라 결과의 기본 정렬 순서를 결정

    SELECT deptno, job, SUM(sal)
    FROM emp
    GROUP BY deptno, job;



REM HAVING
1. WHERE 절에서는 복수행 함수를 사용할 수 없다.

  SELECT deptno, COUNT(*), SUM(sal)
  FROM emp
  WHERE COUNT(*) >= 4
  GROUP BY deptno;


2. GROUP BY의 조건절은 HAVING 이다.

  SELECT deptno, COUNT(*), SUM(sal)
  FROM emp
  GROUP BY deptno
  HAVING COUNT(*) >= 4;

  -사원테이블에서 업무별 급여의 평균이 3000불 이상인 업무에 대해, 업무명, 평균급여, 급여의 합을 구하시오.
    SELECT job, AVG(sal), SUM(sal)
    FROM emp
    GROUP BY job
    HAVING AVG(sal) >= 3000;

  -사원테이블에서 전체 월급이 5000불을 초과하는 각 업무에 대해 업무이름과 월 급여의 합계를 출력하라. 단, 판매원은 제외하고 월급여 합계의 내림차순으로 출력하라.
    SELECT job, SUM(sal)
    FROM emp
    WHERE job NOT LIKE 'SA%'
    GROUP BY job
    HAVING SUM(sal) > 5000
    ORDER BY SUM(sal) DESC;

    SELECT deptno, AVG(sal)
    FROM emp
    GROUP BY deptno;


3. GROUP BY열은 SELECT 목록에 포함시키지 않아도 된다. BUT 별 의미 없음
  SELECT AVG(sal)
  FROM emp
  GROUP BY deptno;


4. ORDER BY 절 사용 가능
  SELECT deptno, AVG(sal)
  FROM emp
  GROUP BY deptno
  ORDER BY AVG(sal);



REM HAVING 절을 사용한 분류된 행을 포함 또는 제외
1. SQL-92버전 및 이전 버전에서는 SELECT 목록의 열 또는 표현식 중 집계 함수가 아닌 것은 GROUP BY 절에 포함시켜야 한다.

  SELECT deptno, COUNT(ename)
```

```
   FROM emp;  --> SQL92 및 이전버전에서는 Error, 이후 버전은 가능

   --SQL92 및 이전버전에서 수정
   SELECT deptno, COUNT(ename)
   FROM emp
   GROUP BY deptno;


2. 그룹 결과 제외 : HAVING 절
   1)WHERE 를 사용하여 행을 제한하는 것과 같이 HAVING 절을 사용하여 그룹을 제한
   2)그룹 함수의 결과를 기반으로 행을 제한할 경우 GROUP BY 절 및 HAVING 절이 모두 있어야
   3)주의할 점 : WHERE 절로 그룹을 제한할 수 없음

   SELECT deptno, MAX(sal)
   FROM emp
   GROUP BY deptno
   HAVING MAX(sal) > 2900;

   SELECT deptno, AVG(sal)
   FROM emp
   WHERE AVG(sal) > 2000
   GROUP BY deptno;   ==> Error

   SELECT deptno, AVG(sal)
   FROM emp
   GROUP BY deptno
   HAVING AVG(sal) > 2000;

   SELECT deptno, COUNT(*), SUM(sal)
   FROM emp
   GROUP BY deptno
   HAVING COUNT(*) > 2;



REM ROLLUP
1. GROUP BY절과 함께 사용
2. GROUP BY절에 의해서 그룹핑 된 집합 결과에 대해 좀 더 상세한 정보를 반환하는 기능을 수행
3. 보통 SELECT 절에 ROLLUP을 사용함으로써 보통의 SELECT된 데이터와 그 데이터의 총계를 구할 수 있다.

   SELECT job, SUM(sal)
   FROM emp
   GROUP BY job;

   --ROLLUP을 사용해서 직무별로 급여 합계와 총계를 구한다.
      SELECT job, SUM(sal)
      FROM emp
      GROUP BY job
      WITH ROLLUP;

4. GROUP BY 칼럼이 두 개 이상인 경우 합계 및 소계까지 계산되어 표시된다.
      SELECT job, deptno, SUM(sal)
      FROM emp
      GROUP BY job, deptno
      WITH ROLLUP;
```

```
REM Author :
REM Date :
REM Objective : Chapter 5. Join
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM 조인(JOIN)
1. 한개 이상의 테이블로부터 데이터를 조회하는 것
2. 주로 Primary-Key와 Foreign-Key의 관계를 가진 컬럼을 소유하고 있는 테이블을 통한 검색 시 사용


REM CROSS JOIN
1. Cartesian Product(카티시안 곱)
2. 조인 조건이 부적합하거나 조인조건을 완전히 생략한 경우 행의 모든 조합을 표시
3. 첫번째 테이블의 모든 행이 두번째 테이블의 모든 행에 조인됨
4. 너무 많은 행을 생성하기 때문에 결과를 사용할 수 없다.
5. 모든 행을 조합해야 하는 경우가 아니라면 반드시 WHERE절에 적합한 조인 조건을 포함시켜야

   SELECT empno, ename, dname
   FROM emp, dept;

   SELECT empno, ename, dname
   FROM emp CROSS JOIN dept;



REM NATURAL JOIN
1. EQUI Join (등가조인), SIMPLE JOIN(단순조인), INNER JOIN(내부조인)
2. 2개 이상의 테이블이 공통되는 컬럼에 의해 논리적으로 결합되는 조인기법
3. WHERE절에 사용된 공통된 컬럼들이 동등 연산자(=)에 의해 비교

   SELECT empno, ename, dname
   FROM emp, dept
   WHERE emp.deptno = dept.deptno;

   --사원이름 KING 의 부서이름과 근무지를 출력하시오.
   SELECT empno, ename, dname
   FROM emp, dept
   WHERE emp.deptno = dept.deptno AND ename = 'KING';


4. 테이블 별칭 사용
   1)별칭의 Guideline
   2)열 이름을 지정하는 경우 시간이 많이 걸리며 테이블 이름이 길때는 더욱 오래 걸림
   3)테이블 이름 대신 테이블 별칭을 사용가능
   4)SQL 코드를 적게 작성해도 되므로 메모리 사용이 줄어듬
   5)FROM 절에서 특정 테이블이름에 대해 별칭을 사용했다면 SELECT 문에서도 테이블 이름을 대신한다.
   6)의미있는 이름을 주자.
   7)별칭은 현재 SELECT에서만 유효하다.

   SELECT e.empno, e.ename, d.dname
   FROM emp e, dept d
   WHERE e.deptno = d.deptno;

   SELECT empno, enam, dname
   FROM emp NATURAL JOIN dept;

   SELECT empno, ename, job, dname, loc
   FROM emp NATURAL JOIN dept
   WHERE empno = 7900;



REM JOIN ~ USING
1. USING을 사용하지 않았다면 MySQL Server는 각 테이블에 공통된 컬럼을 자동으로 검색하여 비교
2. USING을 사용하면 USING절에 정의된 컬럼을 기준으로 Natural 조인이 발생한다.

   SELECT empno, ename, dname
   FROM emp JOIN dept USING (deptno)
   WHERE empno = 7900;



REM JOIN ~ ON
1. EQUI JOIN에서는 WHERE절에서 작성했던 조인 조건을 ON절에서 작성한다.
```

```
75
76      SELECT empno, ename, dname
77      FROM emp JOIN dept
78      ON (emp.DEPTNO = dept.DEPTNO)
79      WHERE empno = 7900;
80
81
82
83  REM NON-EQUI JOIN (비등가조인)
84  1. emp table과 salgrade table간의 관계
85  2. 두개의 테이블사이에 직접 대응하는 열이 없다.
86  3. 이 관계는 =을 제외한 연산자를 사용하여 형성
87
88      SELECT e.ename, e.sal, s.grade
89      FROM emp e, salgrade s
90      WHERE e.sal BETWEEN s.losal AND s.hisal;
91
92
93
94  REM OUTER JOIN
95  1. 포괄조인
96  2. 일치하는 항목이 없는 레코드도 질의할 수 있다.
97
98      SELECT e.ename, e.deptno, d.dname
99      FROM emp e, dept d
100     WHERE e.deptno = d.deptno;
101     --부서 OPERATIONS는 그 부서에서 일하는 사원이 없다. 그래서 표시되지 않는다.
102
103
104 3. 하나이상의 널 행을 생성하여 완전한 테이블의 하나 이상의 행과 조인할 수 있다.
105
106     SELECT e.ename, e.deptno, d.dname
107     FROM emp e RIGHT OUTER JOIN dept d
108     ON e.deptno= d.deptno;
109
110     --LEFT OUTER JOIN을 수행하기 위해 DEPT에 없는 부서 번호 50번 사원을 입력한다.
111     CREATE TABLE emp1
112     AS
113     SELECT * FROM emp;
114
115     INSERT INTO emp(empno, ename, sal, job, deptno)
116     VALUES(8282, 'JACK', 3000, 'ANALYST', 50);
117
118     SELECT e.ename, e.job, e.sal, d.loc, d.dname
119     FROM emp1 e LEFT OUTER JOIN dept d
120     ON (e.deptno = d.deptno)
121
122
123
124 REM SELF JOIN(자체조인)
125 1. 하나의 테이블이 2번 이상 반복적으로 사용되고 참조해야 할 컬럼이 자신의 테이블에 있을 때
126 2. 각 사원의 관리자의 이름을 찾을 때
127 3. 예를 들어, Blake의 관리자 이름을 찾으려면 다음을 수행할 것이다.
128     1)EMP 테이블의 ename 열에서 BLAKE를 찾는다.
129     2)mgr 열에서 BLAKE의 관리자 번호를 찾는다. BLAKE의 관리자 번호는 7839이다.
130     3)ename 열에서 empno 7839인 관리자 이름을 찾는다. King의 사원 번호가 7839이므로 King이 Blake의 관리자이다.
131     4)이 프로세스에서는 테이블을 두 번 검색한다. 첫 번째는 테이블의 ename 열에서 BLAKE와 mgr 값 7839를 찾고 두 번째는 empno
        열에서 7839를 찾고 ename 열에서 KING을 찾는다.
132
133     SELECT worker.ename, manager.ename
134     FROM emp worker, emp manager
135     WHERE worker.mgr = manager.empno;
136
137
138     --1.사원 이름 및 사원 번호를 해당 관리자 이름 및 관리자 번호와 함께 표시하고 열 머리글을 각각 "사원이름", "사원번호", "관리자이름",
        "관리자번호"로 표시하시오.
139     SELECT employee.ename AS "사원이름",
140                 employee.empno AS "사원번호",
141                 employer.ename AS "관리자이름",
142                employer.empno AS "관리자번호"
143     FROM  emp employee, emp employer
144     WHERE employee.mgr = employer.empno AND employee.deptno = 10;
145
146
```

```
    --2. emp table에서 self join하여 관리자를 출려하되, 아래의 형식에 맞게 출력하시오.
    --BLAKE 의 관리자는 KING 이다.


    --3.사원테이블에서 그들의 관리자보다 먼저 입사한 사원에 대해 이름, 입사일, 관리자 이름, 관리자 입사일을 출력하시오.



REM UNION
1. 2개의 쿼리를 위아래로 이어붙여 출력하는 쿼리

    SELECT job, deptno
    FROM emp
    WHERE sal >= 3000

    UNION

    SELECT job, deptno
    FROM emp
    WHERE deptno = 10

2. 주의할 점
    1)위쪽 쿼리와 아래쪽 쿼리 칼럼의 갯수가 동일해야 한다.
    2)위쪽 쿼리와 아래쪽 쿼리 칼럼의 데이터타입이 동일해야 한다.
    3)결과로 출력되는 칼럼명은 위쪽 쿼리의 컬럼명으로 출력된다.
    4)ORDER BY절은 제일 아래쪽 쿼리에서만 작성할 수 있다.



REM UNION ALL
1. 위아래의 쿼리 결과를 하나의 결과로 출력하는 집합 연산

    SELECT job, deptno
    FROM emp
    WHERE sal >= 3000

    UNION ALL

    SELECT job, deptno
    FROM emp
    WHERE deptno = 10

2. 주의할 점은 UNION과 동일
3. UNION과 UNION ALL의 차이점
    1)중복된 데이터를 하나의 고유한 값으로 출력한다.
    2)첫 번째 칼럼의 데이터를 기준으로 내림차순으로 정렬하여 출력한다.
```

```
REM Author :
REM Date :
REM Objective : Chapter 6. Subquery
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0


REM Subquery
1. 다른 SELECT 문의 절에 삽입된 SELECT 문
   -테이블 자체의 데이터에 종속된 조건을 사용해서 테이블에서 행을 선택할 때 유용
2. WHERE 절, SELECT 절, FROM 절에서 사용

3. Syntax
      main query
         (sub query)

      SELECT column_name...
      FROM table_name
      WHERE expression  operator  (SELECT column_name...FROM table_name);

4. Guideline
   1)서브쿼리는 괄호로 묶는다.
   2)서브쿼리는 비교 연산자 오른쪽에 넣는다.
   3)서브쿼리에서는 ORDER BY 를 사용할 수 없다.
   4)ORER BY절은 메인 SELECT 문 마지막에 넣는다.
   5)서브쿼리에서 사용되는 비교연산자는 단일 행 연산자 및 여러 행 연산자를 모두 사용가능

   --사번 7566의 급여보다 많이 받는 사원의 이름
   SELECT ename
   FROM emp
   WHERE sal > (SELECT sal
                FROM emp
                WHERE empno = 7566);

   --만일 Subquery가 없다면
   --먼저 7566 사원의 급여를 구하고
      SELECT sal
      FROM emp
      WHERE empno = 7566   --> 2975

   --그 다음
      SELECT ename
      FROM emp
      WHERE sal > 2975;


   SELECT ename
   FROM emp
   WHERE sal > (SELECT sal
                FROM emp
                WHERE empno = 7566)
   ORDER BY ename ASC;



REM Sub Query 의 종류
1. 단일 행 서브 쿼리 : 내부 SELECT 문에서 한 행만 반환하는 질의
2. 여러 행 서브 쿼리 : 내부 SELECT 문에서 여러 행만 반환하는 질의
3. 여러 열 서브 쿼리 : 내부 SELECT 문에서 여러 열만 반환하는 질의


REM 단일 행 서브 쿼리
1. 내부 SELECT 문에서 하나의 행을 반환
2. 단일 행 연산자(비교 연산자 : =, <>, <, >, <=, >=)를 사용

   --사번 7369번과 직무가 동일한 사원들의 이름과 직무를 표시하시오.
   SELECT ename, job
   FROM emp
   WHERE job = (SELECT job
                FROM emp
                WHERE empno = 7369);


REM 여러 행 서브 쿼리
```

1. 여러 행을 반환하는 서브쿼리
2. 여러 값을 처리하는 함수(연산자, IN, ANY, ALL)사용
3. ANY와 ALL
   1)< ANY : 최대값보다 작은
   2)> ANY : 최소값보다 큰
   3)= ANY : IN 과 동일
   4)< ALL : 최소값보다 작은
   5)> ALL : 최대값보다 큰
   6)NOT 은 모든 연산자와 함께 사용가능

```
--부서에서 최소 급여를 받는 사원
SELECT ename, sal, deptno
FROM emp
WHERE sal IN (SELECT MIN(sal) FROM emp GROUP BY deptno);


--급여가 사무원보다 적으면서 직무가 사무원이 아닌 사원
SELECT empno, ename, job
FROM emp
WHERE sal < ANY (SELECT sal
                 FROM emp
                 WHERE job='CLERK')
AND job <> 'CLERK';


--급여가 모든 부서의 평균 급여보다 많은 사원
SELECT empno, ename, job
FROM emp
WHERE sal > ALL (SELECT AVG(sal)
                 FROM emp
                 GROUP BY deptno);
```

REM Sub Query 의 일반적인 오류
1. 단일 행 서브 쿼리에 대해 여러 행이 반환되는 것

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

2. 서브 쿼리의 결과 값이 널인 경우 결과를 반환하지 않는다.

```
SELECT ename, job
FROM emp
WHERE job = (SELECT job
             FROM emp
             WHERE ename='SMYTHE');
```

REM 여러 열 서브 쿼리
1. 두 개 이상의 열을 비교
2. 논리 연산자를 사용하여 혼합 WHERE 절을 작성
3. 여러 열 서브 쿼리를 사용하면 중복된 WHERE 조건을 단일 WHERE 절로 결합할 수 있다.

```
--사원번호 7396, 7499과 같은 상사와 부서번호를 갖는 모든 사원의 번호와 상사번호 및 부서번호를 출력. 단 7369, 7499는 제외한다.
SELECT empno, mgr, deptno
FROM emp
WHERE (mgr, deptno) IN
                    (SELECT mgr, deptno
                     FROM emp
                     WHERE empno IN (7396, 7499))
AND empno NOT IN (7369, 7499);
```

REM FROM 절에서의 서브 쿼리

```
--해당 부서의 평균 급여보다 급여를 많이 받는 모든 사원의 이름, 급여, 부서 번호 및 평균 급여를 표시.
SELECT a.ename, a.sal, a.deptno, b.salavg
FROM emp a, (SELECT deptno, AVG(sal) salavg
             FROM emp
```

```
149                     GROUP BY deptno) b
150     WHERE a.deptno = b.deptno
151     AND a.sal > b.salavg;
```

```
REM Author :
REM Date :
REM Objective : Chapter 7. DML
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM  데이터 조작(DML)
-데이터베이스에 데이터를 추가, 갱신, 삭제할 때 사용


REM INSERT 문
1. 테이블에 새 행 삽입할 때 사용
2. Syntax

   INSERT INTO table_name [ (column1[, column2[,...]])]
   VALUES (value1, [, value2[,...]]);

3. GuideLine
   1)VALUES 절에서는 동시에 한개의 레코드만 삽입된다.
   2)테이블 이름 뒤에 특정 칼럼을 지정하지 않으면 반드시, 스키마 순서대로 VALUES에서 사용해야 한다.
   3)명확한 Query문을 위해 가급적이면 컬럼이름을 지정하는 것이 좋다.(권장)
   4)문자형과 날짜형은 반드시 단일 따옴표를 사용하고, 숫자형은 단일 따옴표를 사용하지 않는다.
   5)각 열의 값을 포함하는 새 행을 삽입할 수 있으므로 INSERT 절에 열 목록이 필요 없지만 열 목록을 사용하지 않는 경우에는 값을 테이블의
   기본 열 순서에 따라 나열해야 한다.
   6)테이블 리스트에 있는 칼럼 갯수와 VALUES 절의 값 갯수는 일치해야 한다.
      INSERT INTO dept(deptno, dname)
      VALUES (99, '총무과', '서울');

   7)입력될 값의 데이터 타입은 칼럼의 데이터 타입과 일치해야 한다.
      INSERT INTO dept(deptno, dname)
      VALUES ('99', 총무과);

   8)입력될 값의 크기는 칼럼의 크기보다 크지 않아야한다.
      INSERT INTO dept(deptno, dname)
      VALUES (999, '총무과');

   9)NULL 값에 주의하자.
   10)Foreign Key에 주의하자.
   11)오직 한번에 하나의 행만 입력할 수 있다.
      INSERT INTO dept
      VALUES (99, '총무과', '서울', 98, '인사과', '대전');

      INSERT INTO dept(deptno, dname, loc)
      VALUES(50, 'DEVELOPMENT', 'SEOUL');



REM NULL값을 갖는 행 삽입
1. 암시적방법
   INSERT INTO dept(deptno, dname)
   VALUES (60, 'MIS');

2. 명시적 방법
   INSERT INTO dept
   VALUES (70, 'FINANCE', NULL);



REM NOW 함수 또는 CURDATE(), CURTIME()를 사용하여 현재 날짜 및 시간 삽입
   INSERT INTO emp (empno, ename, job, mgr, hiredate, sal, comm, detpno)
   VALUES (7196, 'GREEN', 'SALESMAN', 7782, CURDATE(), 2000, NULL, 10);

   CREATE TABLE emp_copy
   AS
   SELECT *
   FROM emp;

   --Database의 Character set 변경, Table Collation 변경 --> utf8_general_ci

   INSERT INTO emp_copy(empno, ename, job, mgr, hiredate, sal, comm, deptno)
   VALUES(9999, 'Sujan', '', NULL, SYSDATE, 3000,NULL , 10);

   INSERT INTO emp_copy
   VALUES(8888, '홍길동', '', NULL, CURDATE(), 4000,NULL , 20)

```

```
   INSERT INTO emp_copy(empno, ename, hiredate, deptno)
   VALUES(7777, '백두산', CURDATE(), 40);

   INSERT INTO emp_copy(empno, ename, hiredate, deptno)
   VALUES (6666, '한라산', STR_TO_DATE('20080501', '%Y%m%d'), 30);



REM 다른 테이블로부터 행 복사
   CREATE TABLE emp_clone
   AS
   SELECT empno, ename, sal, hiredate
   FROM emp
   WHERE 1 < 0;


   --사원테이블에서 부서번호 10번의 레코드만 emp_clone 으로 복사하시오.
   INSERT INTO emp_clone(empno, ename, sal, hiredate)
   SELECT empno, ename, sal, hiredate
   FROM emp
   WHERE deptno = 10;



REM  UPDATE 문
1. 기본 행 수정
2. 필요한 경우 한번에 여러 행 갱신 가능
3. Syntax

   UPDATE table_name
   SET column = value [, column = value, ...]
   [WHERE condition];


   UPDATE emp
   SET deptno = 20
   WHERE empno = 7782;

   UPDATE emp
   SET deptno = 20;



REM 무결성 제약 조건 오류
-무결성 제약 조건의 영향을 받는 값을 포함하는 레코드를 갱신하면 오류가 발생
   UPDATE emp
   SET deptno = 55
   WHERE deptno = 10;



REM  DELETE 문
1. 기존 행 제거
2. Syntax
   DELECT [FROM] table_name
   [WHERE condition];

   DELECT FROM dept
   WHERE dname = 'DEVELOPMENT';

   DELECT FROM emp;



REM 무결성 제약 조건 오류
-무결성 제약 조건의 영향을 받는 값을 포함하는 레코드를 삭제하면 오류가 발생

   DELETE FROM dept
   WHERE deptno = 10;


   --emp_copy테이블에서 1987년에 입사한 사원을 제거하시오.
   DELETE FROM emp_copy
   WHERE YEAR(hiredate) = '1987';
```

REM DML과 **TRANSACTION**

1. DDL 명령어인 경우에는 직접 **Database**의 **table**에 영향을 미치기 때문에 DDL명령어**를** 입**력**하는 순간 명령어에 해당하는 작업이 즉시**(** AUTO **COMMIT)** 완료된다**.**
2. 하지만**,** DML 명령어의 경우**,** 조작하려는 **table**을 memory buffer에 **올**려놓고 작업을 하기 때문에 실시간**으**로 **table**에 영향을 미치는 것이 아니다**.**
3. 따**라**서 buffer에서 처리한 DML 명령어가 실제 **table**에 반영되기 위해서는 **COMMIT** 명령어**를** 입**력**하여 **Transaction**을 종료해**야** 한다**.**
4. **table**의 전체 **data를** 삭제하는 경우**,** System 활용 **측**면에서는 삭제된 **data를 log**로 저장하는 **DELETE TABLE** 보다는 System 부하가 적은 **TRUNCATE TABLE**을 권고한다**.**
5. 단**,** **TRUNCATE TABLE**의 경우 삭제된 **data**의 **log**가 없**으**므로 **ROLLBACK**이 불가능하므로 주의해**야** 한다**.**

REM Author :
REM Date :
REM Objective : Chapter 8. TCL
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM **Transaction**

1. **Transaction** 이란?
   1)한 개 이상의 실행될 **SQL** 문장의 집합
   2)**Database**의 논리적 연산단위.
   3)밀접히 관련되어 분리될 수 없는 한 개 이상의 **Database** 조작.
   4)하나의 **Transaction**에는 하나 이상의 **SQL** 문장이 포함된다.
   5)분할할 수 없는 최소의 단위.
   6)데이터의 일관성을 보장
   7)행의 **LOCK** 처리를 기본으로 함
   8)데이터의 변경시 융통성을 제공하고, 사용자의 프로세스가 예기치 않게 중단되거나, 시스템 장애가 발생하여 데이터의 일관성이 어렵게 된 경우에도 일관성을 보장하기 위한 시스템적 응용
   9)전부 적용하거나 전부 취소한다.
   10)즉 **ALL OR** NOTHING의 개념.
   11)Oracle에서는 **Transaction**의 대상이 되는 **SQL** 문장을 실행하면 자동으로 시작되고, **COMMIT**또는 **ROLLBACK**을 실행한 시점에서 종료된다.

2. **Transaction**의 특성
   1)원자성(Atomicity) : **Transaction**에서 정의된 연사들은 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않은 상태로 남아 있어야 한다.(**all or** nothing)
   2)일관성(Conststency) : **Transaction**이 실행되기 전의 **Database** 내용이 잘못 되어 있지 않다면 **Transaction**이 실행된 이후에도 **Database**의 내용에 잘못이 있으면 안 된다.
   3)고립성(**Isolation**) : **Transaction**이 실행되는 도중에 다른 **Transaction**의 영향을 받아 잘못된 결과를 만들어서는 안된다.
   4)지속성(Durability) : **Transaction**이 성공적으로 수행되면 그 **Transaction**이 갱신한 **Database**의 내용은 영구적으로 저장된다.

3.TCL(Tracnsaction Control **Language**)
   1)**COMMIT** : 보류중인 데이터를 영구적인 데이타베이스로 변경사항을 저장하고 현재의 **TRANSACTION** 을 종료한다.
   2)**ROLLBACK [TO SAVEPOINT** name]: 보류중인 데이터의 변경사항을 모두 되돌리고 현재의 트랜젝션을 종료한다. 만일 특정지점을 지정하지 않으면 모든 트랜젝션 취소한다.
   3)**SAVEPOINT** name :현재의 트랜젝션의 저장점을 표시한다.

4.**Transaction** 의 범위
   -아래의 사항들은 자동으로 **TRANSACTION** 이 적용된다.
   1)DDL 문이 실행된 경우
   2)DCL 문이 실행된 경우
   3)명시적으로 **commit** 이나 **rollback** 이 실행되지 않은 상태에서 **exit** 를 해서 sqlplus 를 종료할 때
   4)강제적으로 **Transaction** 이 적용되는 경우
   5)시스템 장애가 발생할 경우

5.트랜잭션 암시적 처리
   1)자동 **COMMIT**은 아래의 사항에 발생
      -DDL 문이 실행된 경우
      -DCL 문이 실행된 경우
      -명시적으로 **COMMIT** 또는 **ROLLBACK** 이 실행되지 않은 채 **SQL***Plus 에서 정상 종료(**exit**)한 경우
   2)-자동 **ROLLBACK**
      -**SQL***Plus 의 비정상 종료 시 또는 시스템 장애 발생시

6. **COMMIT**이나 **ROLLBACK** 이전의 **Data**의 상태
   1)단지 Memory Buffer에만 영향을 받았기 때문에 **Data**의 변경 이전 상태로 복구 가능하다.
   2)현재 사용자는 **SELECT** 문장으로 결과를 확인 가능하다.
   3)다른 사용자는 현재 사용자가 수행한 명령의 결과를 볼 수 없다.
   4)변경된 행은 잠금(LOCKING)이 설정되어서 다른 사용자가 변경할 수 없다.

REM **COMMIT**
1. 변경 사항을 영구히 저장

```
UPDATE emp SET deptno = 10 WHERE empno = 7782;
COMMIT;
```

2. **COMMIT** 이후의 **Data**의 상태
   1)**Data**에 대한 변경 사항이 **Database**에 반영된다.
   2)이전 **Data**는 영원히 잃어버리게 된다.

3)모든 사용자는 결과를 볼 수 있다.
4)관련된 행에 대한 잠금(LOCKING)이 풀리고, 다른 사용자들이 행을 조작할 수 있게 된다.

REM **ROLLBACK**
1. 보류 중인 변경 내용을 모두 되돌림
   -Data 변경 사항이 취소되어 **Data**의 이전 상태로 복구되며, 관련된 행에 대한 잠금(LOCKING)이 풀리고 다른 사용자들이 **Data**의 변경을 할 수 있게 된다.

2. **ROLLBACK**이후의 상태
   1)**Data**에 대한 변경 사항은 취소된다.
   2)이전 **Data**는 다시 재저장된다.
   3)관련된 행에 대한 잠금(LOCKING)이 풀리고, 다른 사용자들이 행을 조작할 수 있게 된다.

```
DELETE FROM emp;
ROLLBACK;
```

REM **COMMIT**과 **ROLLBACK**의 효과
1. **Data** 무결성 보장
2. 영구적인 변경을 하기 전에 **Data**이 변경 사항 확인 가능
3. 논리적으로 연관된 작업을 그룹핑하여 처리 가능.

REM **SAVEPOINT**
1. 현 시점에서 **SAVEPOINT**까지 **Transaction**의 일부만 **ROLLBACK** 할 수 있다.
2. 따라서 복잡한 대규모 **Transaction**에서 Error가 발생했을 때 **SAVEPOINT**까지의 **Transaction**만 **Rollback**하고 실패한 부분에 대해서만 다시 실행할 수 있다.
3. 복수의 저장점을 정의할 수 있으며, 동일이름으로 저장점을 정의했을 때는 나중에 정의한 저장점이 유효하다.
4. Syntax
   SAVEPOINT savepoint_name;
5. 저장점까지 **rollback**할 때는 **ROLLBACK** 뒤에 저장점 명을 지정한다.
   ROLLBACK TO savepoint_name;
6. **Rollback**에 **SAVEPOINT** 명을 부여하여 실행하면 저장점 설정 이후에 있었던 **Data** 변경에 대해서만 원래 **Data** 상태로 되돌아가게 된다.
7. 저장점 지정없이 **ROLLBACK**을 실행했을 경우에는 반영안된 모든 변경 사항을 취소하고 **Transaction** 시작 위치로 되돌아간다.

REM 저장점까지 변경 내용 **ROLLBACK**
1. **SAVEPOINT** 사용하여 현재 트랜잭션에 표시자를 생성하여 트랜잭션을 더 작은 부분으로 나눈 후 **ROLLBACK TO SAVEPOINT** 문을 사용해 보류 중인 변경 내용을 해당 표시자까지 되돌림.

```
START TRANSACTION;

UPDATE emp SET deptno = 10 WHERE empno = 7782;
SAVEPOINT a;

INSERT INTO emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES (7999, 'TOM', 'SALESMAN', 7782, CURDATE(), 2000, 2000, 10);

ROLLBACK TO a;
```

REM Author :
REM Date :
REM Objective : Chapter 9. DDL
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM DDL(Data Definition Language)
-데이타베이스의 Object 구조를 생성, 변경, 삭제하는 명령어
-CREATE, ALTER, DROP, RENAME, TRUNCATE

REM Naming Convension
출처: https://taptorestart.tistory.com/entry/MySQL-데이터베이스명-테이블명-컬럼명은-어떻게-지어야-할까
1. General
1)Ensure the name is unique and does not exist as a reserved keyword.
2)Keep the length to a maximum of 30 bytes—in practice this is 30 characters unless you are using a multi-byte character set.
3)Names must begin with a letter and may not end with an underscore.
4)Only use letters, numbers and underscores in names.
5)Avoid the use of multiple consecutive underscores—these can be hard to read.
6)Use underscores where you would naturally include a space in the name (first name becomes first_name).
7)Avoid abbreviations and if you have to use them make sure they are commonly understood.

2. Tables
1)Use a collective name or, less ideally, a plural form. For example (in order of preference) staff and employees.
2)Do not prefix with tbl or any other such descriptive prefix or Hungarian notation.
3)Never give a table the same name as one of its columns and vice versa.
4)Avoid, where possible, concatenating two table names together to create the name of a relationship table. Rather than cars_mechanics prefer services.

3. Columns
1)Always use the singular name.
2)Where possible avoid simply using id as the primary identifier for the table.
3)Do not add a column with the same name as its table and vice versa.
4)Always use lowercase except where it may make sense not to such as proper nouns.

REM Database 생성
1. Database
-Table이나 View, Index 등 데이터베이스 내에 정의하는 모든 것
2. The CREATE DATABASE statement is used to create a new SQL database.
3. Syntax
```
CREATE DATABASE [IF NOT EXISTS] db_name
[create_option] ...

create_option: [DEFAULT] {
    CHARACTER SET [=] charset_name | COLLATE [=] collation_name
}
```
4. Example
```
CREATE DATABASE testDB;
CREATE DATABASE study_db default CHARACTER SET UTF8;
```


REM Database 삭제
1. The DROP DATABASE statement is used to drop an existing SQL database.
2. Syntax
```
DROP DATABASE [IF EXISTS] db_name
```
3. Example
```
DROP DATABASE testDB;
```


REM 테이블의 생성
1. CREATE TABLE
-The CREATE TABLE statement is used to create a new table in a database.
2. Syntax
```
    CREATE TABLE [IF NOT EXISTS] table_name
    (
        column_name     column_definition
        [CONSTRAINT] PRIMARY KEY |
        [CONSTRAINT] UNIQUE |
        [CONSTRAINT] FOREIGN KEY |
```

```
        CHECK (expression)
    )
    column_definiction : {
        data_type [NOT NULL | NULL] [DEFAULT default_value]
          [AUTO_INCREMENT] [UNIQUE][PRIMARY KEY]
          [COMMENT 'string']
    }
    table_option : {
        AUTO_INCREMENT [=] value |
        [DEFAULT] CHARACTER SET [=] charset_name |
        COMMENT [=] 'string'
    }

1)일반적 생성 방법
    CREATE TABLE table_name ( column_name    datatype, ...);

    CREATE TABLE IF NOT EXISTS dept1
    (
        deptno    TINYINT,
        dname     VARCHAR(13),
        loc       VARCHAR(14) DEFAULT 'Seoul'
    )DEFAULT CHARACTER SET UTF8

2)SUB_QUERY 에 의해 검색된 테이블과 동일한 구조로 생성
    -테이블 복사, CTAS(Create Table ~ As Select ~)방법
    -주의할 점은 기존 테이블의 제약조건이 모두 없어진다는 점

    CREATE TABLE dept2
    AS
    SELECT * FROM dept;

    CREATE TABLE emp2
    AS
    SELECT EMPNO, ENAME FROM EMP;

    CREATE TABLE emp3
    AS
    SELECT * FROM emp
    WHERE DEPTNO = 10;

    CREATE TABLE emp4
    AS
    SELECT * FROM emp where 1=0;  <-- 구조만 복사

3)Guide Lines
    -지정한 열 이름을 사용하여 테이블을 생성하며 SELECT 문에 의해 검색된 행을 테이블에 삽입.
    -열 사양이 제공되는 경우 열 수는 하위 질의 SELECT 목록의 열 수와 동일해야 한다.
    -열 사양이 제공되지 않는 경우 해당 테이블의 열 이름은 하위 질의의 열 이름과 동일.

4)조회
    SELECT table_name
    FROM user_tables;

    SELECT object_name, object_type
    FROM user_objects;

    CREATE TABLE dept30
    AS
    SELECT empno, ename, sal * 12 "Annual Salary", hiredate
    FROM emp
    WHERE deptno = 30;

5)사용예
    --부서별로 인원수, 평균급여, 급여의 합, 최소급여, 최대급여를 포함하는 emp_calcu table을 생성하라.
    CREATE TABLE emp_calcu
    AS
    SELECT deptno, COUNT(*) AS cnt, AVG(sal) AS salavg, SUM(sal) AS salsum,
           MIN(sal) AS minsal, MAX(sal) AS maxsal
    FROM emp
    GROUP BY deptno;


    --사원번호, 이름, 업무, 입사일자, 부서번호만 포함하는 emp_temp table을 생성하는데, 자료는 포함하지 않고 스키마만 생성하시오.
    CREATE TABLE emp_temp
```

```
    AS
    SELECT empno, ename, job, hiredate, deptno
    FROM emp
    WHERE 1 < 0;


    --테이블을 다음챠트를 기반으로 DEPARTMENT 테이블을 생성하고, 생성한 후 테이블 구조를 확인하시오.
    --열이름 :          id          name
    --데이타유형 :      INT       VARCHAR
    --길이 :             7           25
    CREATE TABLE department
    (
       id       INT(7),
       name      VARCHAR(25)
    );

    DESC department


    --테이블을 다음챠트를 기반으로 EMPLOY 테이블을 생성하고, 생성한 후 테이블 구조를 확인하시오.
    --열이름 :      id          last_name     first_name     dept_id
    --데이타유형 :  INT         VARCHAR       VARCHAR        INT
    --길이 :         7            25             25            7
    CREATE TABLE employ
    (id       INT(7),
       last_name      VARCHAR(25),
       first_name VARCHAR(25),
       dept_id      INT(7)
    );

    DESC employ



REM TABLE 삭제
1. The DROP TABLE statement is used to drop an existing table in a database.
   -table의 모든 구조와 데이터가 삭제
   -DDL이기 때문에 TRANSACTION 이 COMMIT된다.
   -영구히 삭제, 되돌릴 수 없다.
2. Syntax
    DROP TABLE table_name;

    DROP TABLE emp30;
    DROP TABLE dept;

3. Guideline
   1)삭제하려는 테이블의 기본 키나 고유 키를 다른 테이블에서 참조해서 사용하는 경우에는 해당 테이블을 제거할 수 없다.
   2)이런 경우에는 참조하는 테이블을 먼저 제거한 후 해당 테이블을 삭제해야 한다.



REM ALTER TABLE
1. The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
2. The ALTER TABLE statement is also used to add and drop various constraints on an existing table.
   -테이블의 스키마 변경
   -Column을 추가/삭제하거나 제약조건을 추가/삭제하는 작업

3. Column Add
   -열 추가
   -새 열은 마지막 열, 위치 지정 불가능
   -이미 행을 포함하고 있는 테이블이라면 새로 들어갈 열의 모든 행은 초기에 널 값을 갖는다.
   -Syntax
      ALTER TABLE table_name
      ADD  [COLUMN] col_name column_definition
         [FIRST | AFTER col_name]
            [, column datatype]...);

    ALTER TABLE dept30
    ADD job VARCHAR(9);

    ALTER TABLE dept30
    ADD sal INT AFTER ename;

    ALTER TABLE dept1
```

```
      ADD bigo   VARCHAR(15) FIRST;


2. Column Modify
   -Column의 데이터 유형, Default값, NOT NULL 제약조건에 대한 변경
   -Syntax
      ALTER TABLE table_name
      MODIFY  [COLUMN] column_name column_definition
         [FIRST | AFTER col_name]

   -MODIFY Guide Lines
      --숫자 열의 너비 또는 전체 자릿수를 증가가능.
      --열이 널 값만 포함하고 테이블에 행이 없는 경우 열의 너비를 줄일 수 있다.
      --열이 널 값을 포함하면 데이터 유형을 변경할 수 있다.
      --열의 기본값을 변경하면 변경 이후에 테이블에 삽입되는 항목에만 영향을 준다.
      --테이블의 구조 변경(데이터 타입, 길이), 만일 기존 데이터가 있을 경우에는 CHAR와 VARCHAR 사이의 타입 변경만 가능
      --컬럼의 크기 변경 역시 기존에 저장된 데이터의 길이와 같거나 클 경우에만 변경이 가능
      --즉 해당 컬럼에 자료가 없을 경우에는 데이터타입과 컬럼의 크기를 변경가능
      --해당 컬럼에 자료가 있을 경우에는 데이터타입 변경 불가능하고 크기는 늘릴 수만 있음.

      ALTER TABLE dept30
      MODIFY ename VARCHAR(15);

      ALTER TABLE dept1
      MODIFY bigo   VARCHAR(30);


   -사용예
      --위 예제에서 생성한 EMPLOY테이블의 last_name열에 긴 성을 가진 사원의 성을 저장할 수 있도록 길이를 50으로 수정한 후, 내용을
      확인하시오


3. Column Remove
   -열 삭제
   -Guide Lines
      --열은 데이터를 포함하거나 포함하지 않거나 삭제 할 수 있다.
      --한 번에 하나의 열만 삭제할 수 있다.
      --테이블에는 최소 하나의 열이 있어야 하므로 열을 삭제하려면 테이블에 열이 둘 이상 있어야 한다.
      --삭제된 열은 복구할 수 없다.
   -Syntax
      ALTER TABLE table_name
      DROP [COLUMN] col_name

      ALTER TABLE emp1
      DROP COLUMN JOB;


4. CHANGE COLUMN
   -Column의 이름을 변경
   -Syntax
      ALTER TABLE table_name
      CHANGE [COLUMN] old_col_name new_col_name column_dinifition
         [FIRST | AFTER col_name]

      ALTER TABLE dept1
      CHANGE COLUMN bigo bigo1 VARCHAR(15);


REM TABLE TRUNCATE
1. Empties a table completely.
2. ROLLBACK 불가능
3. DELETE는 모든 행을 제거할 수 있지만, 저장공간을 해제할 수 없다.
4. 모든 행을 제거할 때는 DELETE보다 TRUNCATE TABLE문을 사용하자.
5. Syntax
      TRUNCATE [TABLE] table_name

      TRUNCATE TABLE emp30;

6. DROP TABLE은 테이블 자체를 제거하지만, TRUNCATE는 테이블은 존재하면서 데이터만 제거하기에 구조는 남아있다.


REM Rename TABLE
```

1. **table**의 이름 변경
2. Syntax
   RENAME TABLE old_table_name **TO** new_table_name

      RENAME TABLE dept2 **TO** dept3**;**

REM 테이블에 주석 문 추가
1. **Table Comment**
   -CREATE TABLE 또는 **ALTER TABLE**에 table_option**으**로 추가
      COMMENT [=] 'string'

   ALTER TABLE dept1
      COMMENT = '부서 정보 테이블입니다'**;**

2. **Column Comment**
   -CREAET **TABLE** 또는 **ALTER TABLE**에 column_definition**으**로 추가
      COMMENT 'string'

   ALTER TABLE dept1
      MODIFY COLUMN deptno TINYINT **COMMENT** '부서코드'

REM **DATA TYPE**
1. CHAR
   1**)**고정길이의 문자 데이타
   2**)**기본값 및 최소크기 **:** 1Byte
   3**)**최대크기 125Bytes
   4**)**나머지 공간을 여백**으**로 채워서 처음 정의된 공간을 모두 사용하는 타입

2. VARCHAR
   1**)Variable Character**의 **약**자
   2**)**가변길이의 문자 데이타
   3**)**기본값 및 최소 크기 **:** 1Byte
   4**)**최대크기 **:** 65535Bytes
   5**)**여백**으**로 채우지 않고 필요한 공간만 사용

   ABCDE --> char(8) --> ABCDE_ _ _
   ABCDE --> varchar(8) --> ABCDE

3. 정수
   1**)**TINYINT**(**1**),** SMALLINT**(**2**),** INT**(**4**),** BIGINT**(**8**)**
   2**)**부호없는 정수**를** 저장할 때는 UNSIGNED 예**약**어**를** 뒤에 붙여준다**.**

4. 실수
   1**)숫**자값을 -38자리 **~** 38자리 저장
   2**)**FLOAT**(**4**),** DOUBLE**(**8**),** DECIMAL**(**m**,** d**)**

5. DECIMAL**(**m**,** d**)**
   1**)**전체자리수**(**m**)**중 소수점 이하 자리수**(**d**)**
   2**)**전체자릿수는 십진수의 총 **갯**수이고**,** 소수점 이하 자릿수는 소수점의 오른쪽에 있는 **숫**자 **갯**수**.**

6. DATE
   1**)**A**.**D 1000년 1월 1**일 ~** A**.**D**.** 9999년 12월 31**일**까지의 날짜
   2**)SELECT** CURDATE**();**
   3**)**'YYYY-MM-DD' 형식

7. DATETIME
   1**)**'1000-01-01 00:00:00.000000' **to** '9999-12-31 23:59:59.999999'
   2**)**'YYYY-MM-DD HH:MM:SS' 형식

```
REM Author :
REM Date :
REM Objective : Chapter 10. Constraints
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0


REM Column 의 옵션
   1. DEFAULT OPTION
   2. Constraints


REM DEFAULT OPTION
   1. 열에 기본값을 부여할 수 있다.
   2. 이 옵션은 열 값이 없는 행을 삽입할 경우 열에 널 값이 입력되는 것을 방지
   3. 리터럴, 표현식을 기본값으로 사용할 수 있다.
   4. 기본 표현식은 해당 열의 데이터 유형과 일치해야 한다.
   5. ex)
      hiredate  DATETIME DEFAULT CURRENT_TIMESTAMP  <--- NOW() 불가

   6. 테이블을 생성할 때 혹은 스키마를 변경할 때 칼럼에 직접 적용
      CREATE TABLE table_name
      (column   datatype [DEFAULT  expression][,...]);

   7. 실습1

      CREATE TABLE Department
      AS
      SELECT * FROM dept
      WHERE 0 > 1;

      ALTER TABLE Department
      ADD COLUMN  hiredate     DATETIME     DEFAULT   CURRENT_TIMESTAMP;

      INSERT INTO Department(deptno, hiredate)
      VALUES(10, DEFAULT);

   8. 실습2

      CREATE TABLE Jusorok
      (
         bunho   SMALLINT,
         gender      CHAR(6)     DEFAULT   '남자'
      );

      INSERT INTO Jusorok VALUES (1, '여자');
      INSERT INTO Jusorok VALUES (2, DEFAULT);
      SELECT * FROM jusorok;


REM Constraints(제약조건)
   -사용자가 원하는 조건의 데이터만 유지하기 위한 즉, 데이터의 무결성을 유지하기 위한 데이터베이스의 보편적인 방법
   -테이블의 특정 칼럼에 설정하는 제약이다.

      INSERT INTO dept
      VALUES(10, 'TEST', 'SEOUL');  --무결성 제약조건 위배
      --Dept 테이블에 이미 10번 부서가 존재하고 있기 때문

      INSERT INTO dept
      VALUES(NULL, NULL, 'SEOUL'); --오류
      --NULL 을 부서번호에 삽입할 수 없다.

1. 특징
   1) 사용자에 의해 발생한 잘못된 DML 문이 실행되지 않는다.
   2) 제약 조건에 대한 모든 정보가 자료사전에 저장된다.
   3) 원할 때는 언제든지 기능을 비활성화할 수 있고, 또한 활성화 할 수 있다.
   4) 처리 결과가 즉시 사용자에게 보여진다.
   5) 하나의 칼럼에 여러 개의 제약 조건을 설정할 수 있다.
   6) 부적절한 데이타의 입력, 수정, 삭제를 방지할 목적
   7) 제약조건은 테이블레벨/컬럼레벨 제약조건이 있다.
   8) 제약조건은 종속성이 있을 경우 삭제를 방지한다.
   9) 테이블에서 삽입, 수정, 삭제를 할 때 마다 조건에 대한 규칙을 적용한다.
   10) 제약조건은 각각의 RDBMS 마다 다를 수 있다.
   11) Oracle 에서는 제약 조건 이름을 지정하지 않으면, 자동으로 SYS-Cnumber 형식으로 생성한다.
   12) 사용자가 해당 제약조건에 위배되는 명령을 요청했을 때, 오라클은 제약조건 이름과 함께 에러를 내보낸다.
```

13) DESC 명령어는 NOT NULL 제약 조건을 확인할 수 있지만, 다른 제약조건을 확인할 수 없다.

2. 제약조건의 종류
1) Column Level Constraint
-컬럼단위로 제약조건을 부여할 때
-5가지의 제약조건이 모두 가능
-NOT NULL은 컬럼레벨에서만 부여가능
-Syntax
column_name data_type
[CONSTRAINT] constraint_type

id VARCHAR2(10) CONSTRAINT PRIMARY KEY,
id VARCHAR2(10) PRIMARY KEY

```
CREATE TABLE Student
(
   hakbun   CHAR(4) PRIMARY KEY,
   name     VARCHAR(20) NOT NULL,
   kor      TINYINT  NOT NULL
)

SELECT CONSTRAINT_NAME, constraint_type, TABLE_name
FROM information_schema.table_constraints
WHERE TABLE_NAME = 'student';
```

2) Table Level Constraint
-각각의 칼럼의 정의와 개별적으로 정의한다.
-하나 이상의 칼럼을 지정가능하다.
-NOT NULL을 제외한 나머지 제약조건의 정의가 가능
-Syntax
column_name data_type,
...,
...,
[CONSTRAINT constraint_name] constraint_type

```
id     VARCHAR2(10),
name   VARCHAR2(20),
age    NUMBER(3),
CONSTRAINT testpk_id_pk PRIMARY KEY(id)


CREATE TABLE Student
(
   hakbun    CHAR(4),
   name      VARCHAR(20)  NOT NULL,
   kor       TINYINT(3) NOT NULL,
   CONSTRAINT Student_hakbun_pk  PRIMARY KEY(hakbun)
);
```

3. 데이타 무결성 제약조건 5가지
-PRIMARY KEY(PK)
-FOREIGN KEY(FK)
-UNIQUE(UK)
-NOT NULL(NN)
-CHECK(CK)

4. Constraints Naming Convention guide lines
-제약조건의 이름은 Oracle과 달리 Query에서는 사용할 일은 없다.
-constraint_name : tablename_columnname_constrainttype
-ex : emp_empno_pk, emp_deptno_fk

REM PRIMARY KEY

EMPNO	ENAME	JOB	DEPTNO
7499	ALLEN	SALESMAN	30
	JONES	MANAGER	20
	JONES	SALESMAN	10

-현재 EMPNO 가 **UNIQUE** 로 설정된 상태임
-**UNIQUE** 는 **NULL** 이 허용되기 때문에 추가적으로 **NOT NULL**이 필요
-그래서 **PRIMARY KEY** 가 필요함.

1. 테이블에 저장된 행 데이터를 고유하게 식별하기 위한 기본키
2. 기본적으로 테이블마다 주 식별자는 있어야 한다.
3. 하나의 테이블에 하나의 기본키 제약만 정의할 수 있다.
4. **NOT NULL** 제약조건과 **NO** DUPLICATE 제약조건**(UNIQUE)**이 부여된다.
5. **UNIQUE INDEX** 가 자동으로 생성
6. 테이블 레벨 제약조건과 컬럼 레벨 제약조건 모두 가능
7. Syntax
   1) **Column Level Constraints**

```
CREATE TABLE Test
(
   id NUMBER(3)     PRIMARY KEY,
   name       VARCHAR2(20)
);
```

   2) **Table Level Constraints**

```
CREATE TABLE Test1
(
   id NUMBER(3),
   name    VARCHAR2(20),
   CONSTRAINT test1_id_pk PRIMARY KEY(id)
);
```

REM **FOREIGN KEY**

1. 관계형 데이터베이스에서 테이블 간의 관계를 정의하기 위해 기본키를 다른 테이블의 외래키로 복사하는 경우 생성.
2. 자식테이블(참조하는 쪽)에서 정의한다.
3. MASTER **TABLE**(부모테이블) vs DETAIL **TABLE**(자식테이블)
   -MASTER **TABLE**(parent) 은 참조 당하는 쪽(예:dept **table**)을 의미하고 DETAIL **TABLE**(child)은 참조하는 쪽(예:emp **table**)을 의미한다.
4. 외래키가 바라보는(참조하는) 부모테이블(마스터테이블)의 키는 **PRIMARY KEY, UNIQUE KEY**로 정의된 열을 지정할 수 있으며, 데이타타입이 일치해야 하고, **NULL** 일 수 있다.
5. **ON DELETE CASCADE**를 지정하면, 부모테이블의 레코드를 삭제할 때, 참조된 행을 삭제할 수 있다.
6. Syntax
   1) **Column Level Constraint**

```
column_name  data_type
[CONSTRAINT] FOREIGN KEY
REFERENCES table_name (column_name)
[ON DELETE CASCADE]

ex) deptno  NUMBER(2) FOREIGN KEY REFERENCES dept (deptno);
```

   2) **Table Level Constraint**

```
column_name  data_type,
...,
...,
[CONSTRAINT constraint_name]  FOREIGN KEY(column_name)
REFERENCES table_name (column_name) [ON DELETE CASCADE]

ex) deptno NUMBER(2),
      FOREIGN KEY(deptno)  REFERENCES dept (deptno)
```

7. **FOREIGN KEY** 의 주의점
   1) **FOREIGN KEY** 값은 MASTER **TABLE**에서 존재하는 값과 일치해야하거나 **NULL**이 되어야 한다.
   2) MASTER **TABLE**을 먼저 생성해야 한다.
   3) MASTER **TABLE**에 **PRIMARY KEY** 또는 **UNIQUE KEY** 로 설정된 열을 DETAIL **TABLE**에서 참조해야 한다.
   4) MASTER **TABLE** 과 DETAIL **TABLE** 에서 참조하는 쪽과 참조 당하는 쪽의 열은 자료형이 서로 일치해야 한다.

8. 실습

-부모 테이블 먼저 생성

```
CREATE TABLE dept10
(
   deptno TINYINT PRIMARY KEY,
```

```
    dname VARCHAR(15),
    LOCAL VARCHAR(1)
);

-자식테이블 생성
CREATE TABLE emp10
(
    empno INT PRIMARY KEY,
    ename VARCHAR(15),
    deptno TINYINT,
    CONSTRAINT FOREIGN KEY(deptno)  REFERENCES dept10(deptno)
);

CREATE TABLE dept_copy
(
    deptno     NUMBER(2),
    dname      VARCHAR2(14),
    loc        VARCHAR2(13),
    CONSTRAINT dept_copy_deptno_pk  PRIMARY KEY(deptno)
);

CREATE TABLE emp_copy
(
    empno  INT,
    ename  VARCHAR(10),
    hiredate    DATE,
    deptno  SMALLINT,
    CONSTRAINT emp_copy_empno_pk PRIMARY KEY(empno),
    CONSTRAINT emp_copy_deptno_fk  FOREIGN KEY(deptno)
       REFERENCES dept_copy (deptno)
);


REM UNIQUE KEY

   EMPNO      ENAME       JOB        DEPTNO
   7499       ALLEN       SALESMAN       30
   7499       JONES       MANAGER        20

1. 열 또는 열 집합 모든 값들의 유일성을 보장하기 위한 키
2. 중복된 값을 가질 수 없다는 것을 보증한다.
3. 하나의 테이블에서 여러 칼럼에 명시할 수 있다.
4. PRIMARY KEY와 유사하지만, NULL 허용이 된다는 것이 차이다.
5. 자동으로 INDEX가 부여된다.
6. Syntax
   1) Column Level Constraint
      column_name   data_type   [CONSTRAINT] UNIQUE,
   2) Table Level Constraint
      column_name   data_type,
      , ,  ,
      [CONSTRAINT constraint_name] UNIQUE(column_name)

7. 실습

   CREATE TABLE dept_clone
   (
      deptno  SMALLINT,
      dname      VARCHAR(20),
      loc        VARCHAR(20),
      CONSTRAINT   dept_clone_deptno_uk  UNIQUE(deptno)
   );

   CREATE TABLE unitest
   (
      deptno  SMALLINT UNIQUE,
      dname      CHAR(14),
      loc        CHAR(13)
   );

   --OR
   CREATE TABLE unitest
   (
      deptno  SMALLINT,
      dname    CHAR(14),
```

```
    loc     CHAR(13),
    CONSTRAINT unitest_deptno_uk UNIQUE (deptno),
);


REM NOT NULL
1. 값이 NULL 이 되지 않는다는 것을 보장한다.
2. INSERT, UPDATE 시 NULL 을 허용하지 않겠다는 의미
3. NOT NULL 제약조건이 없는 열은 기본적으로 NULL 이 허용된다.
4. 반드시 컬럼레벨 제약조건에서만 지정가능
5. PRIMARY KEY 는 기본적으로 NOT NULL 을 가지고 있음.
6. Syntax
    column datatype NOT NULL

7. 실습

CREATE TABLE dept_copy1
(
    deptno    SMALLINT,
    dname    VARCHAR(20) NOT NULL  DEFAULT  'Accounting',
    loc      VARCHAR(20),
      CONSTRAINT dept_copy1_deptno_pk PRIMARY KEY(deptno),
      CONSTRAINT dept_copy1_loc_uk UNIQUE(loc)
);

INSERT INTO dept_copy1(deptno, loc)
VALUES(20, 'Pusan');

CREATE TABLE dept14
(
    deptno SMALLINT,
    dname  VARCHAR(15),
    LOCAL CHAR(1),
    CONSTRAINT dept14_dname_nn NOT NULL(dname)
); -- ERROR 발생

CREATE TABLE NULLDemo
(
    deptno  SMALLINT,
    dname VARCHAR(10) NOT NULL,
    loc VARCHAR(10),
    CONSTRAINT nulldemo_deptno_pk PRIMARY KEY(deptno)
);

INSERT INTO nulldemo VALUES (1, 'aaa', '');
INSERT INTO nulldemo VALUES (2, 'bbb', null);
INSERT INTO nulldemo VALUES (3, NULL, 'seoul');


REM CHECK
1. 행이 만족해야 하는 조건을 정의하는 것이다.
2. Syntax
 1) Column Level Constraint
   column_name  data_type CHECK(condition)
 2) Table Level Constraint
  -column_name  data_type,
   ,,,,
   [CONSTRAINT constraint_name] CHECK(condition)

3. 실습
CREATE TABLE dept_copy
    (
    deptno   SMALLINT,
      dname   VARCHAR(20),
      loc      VARCHAR(20),
      CONSTRAINT dept_copy_deptno_pk  PRIMARY KEY(deptno),
      CONSTRAINT dept_copy_deptno_ck CHECK(deptno IN(10,20,30,40,50))
    );

--사원번호, 사원명, 급여, 성별 4개의 칼럼을 갖는 테이블을 생성하시오,
--단 사원번호가 기본키로, 사원명은 NOT NULL로, 급여는 500에서 5000사이의 값만
--저장할 수 있고, 성별은 남자는 M, 여자는 F 둘중의 하나만 저장할 수 있어야 한다.
CREATE TABLE emp2
(
```

```
    empno    INT  PRIMARY KEY,
    ename    VARCHAR(10)  NOT NULL,
    sal      DECIMAL(7,2)  CHECK(sal BETWEEN 500 AND 5000),
    gender   CHAR(1)  CHECK(gender IN('M', 'F'))
);

CREATE TABLE zippost
   (
      post1     CHAR(3),
      post2     CHAR(3),
      address     VARCHAR(100)  NOT NULL,
      CONSTRAINT zippost_post_pk  PRIMARY KEY(post1, post2)
   );

   CREATE TABLE member
   (
      id  INT,
      name       VARCHAR(10) NOT NULL,
      gender  CHAR(1),
      jumin1  CHAR(6),
      jumin2  CHAR(7),
      tel        CHAR(13),
      post1       CHAR(3),
      post2       CHAR(3),
      address VARCHAR(100),
      CONSTRAINT member_id_pk  PRIMARY KEY(id),
      CONSTRAINT member_gender_ck CHECK(gender IN ('1', '2')),
      CONSTRAINT member_tel_uk UNIQUE(tel),
      CONSTRAINT member_post_fk  FOREIGN KEY(post1, post2)
         REFERENCES zippost(post1, post2)
   );


REM DICTIONARY 에서 제약 조건 검색하기
1. DESC 명령어는 NOT NULL 제약 조건을 확인할 수 있지만, 다른 제약조건을 확인할 수 없다.
2. 제약조건 확인
   information_schema.table_constraints;
      -constaint_catalog, constraint_schema, constraint_name, table_schema, table_name, constraint_type

3. DESC information_schema.table_constraints
  --CONSTRAINT_TYPE
   --PRIMARY KEY
   --FOREIGN KEY
   --UNIQUE


REM 제약 조건 추가
1. 제약 조건의 추가, 삭제는 가능하지만, 변경은 불가능
2. 제약 조건의 활성화, 비활성화 가능
3. NOT NULL 제약 조건은 MODIFY 절을 이용하여 추가
4. NOT NULL 은 Column Level 에서만 가능
5. Syntax
   ALTER TABLE table_name
   ADD [CONSTRAINT constraint_name] constraint_type(column);

6. 실습

   DROP TABLE emp1;

   CREATE TABLE emp1
   AS
   SELECT empno, ename, job, deptno
   FROM emp;

   ALTER TABLE emp1
   ADD
      CONSTRAINT emp1_deptno_fk FOREIGH KEY(deptno)
      REFERENCES dept(deptno);

   CREATE TABLE emp_clone
   AS
   SELECT empno, ename, job
   FROM emp
   WHERE deptno = 10;
```

```
ALTER TABLE emp_clone
ADD CONSTRAINT emp_clone_empno_pk PRIMARY KEY(empno);

ALTER TABLE emp_clone
MODIFY ename  VARCHAR(10)  CONSTRAINT emp_clone_nn NOT NULL;
```

1. 아래와 같이 dept 및 emp 테이블의 제약조건의 이름을 나열하는 질의를 작성하시오.

```
CONSTRAINT_NAME
------------------------------
dept_deptno_pk
emp_empno_pk
emp_deptno_fk
emp_mgr_fk
```

2. 사원테이블을 emp_copy라는 테이블을 생성하시오. 사원이름에 UNIQUE키를 부여하시오.

```
CREATE TABLE emp_copy
AS
SELECT empno, ename, sal
FROM emp
WHERE deptno = 20;

ALTER TABLE emp_copy
ADD CONSTRAINT emp_copy_ename_uk UNIQUE (ename);
```

REM 제약조건의 삭제

1. ALTER TABLE의 DROP 절을 사용하여 기존의 테이블에 있는 제약조건을 삭제할 수 있다.
2. Syntax

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name
OR
DROP [{ PRIMARY KEY(column_name) | FOREIGN KEY(column_name) | UNIQUE(column_name) }]
```

3. 실습

```
ALTER TABLE emp_copy
DROP CONSTRAINT emp_copy_ename_uk;

ALTER TABLE emp_copy
ADD PRIMARY KEY(empno);

ALTER TABLE emp_copy
DROP PRIMARY KEY;

ALTER TABLE emp1
DROP CONSTRAINT emp1_ename_nn;
```

```
REM Author :
REM Date :
REM Objective : Chapter 11. MySQL Objects
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0


REM MySQL Objects
   1. 종류
      TABLE, VIEW, INDEX, Stored Procedure, Stored Function, Trigger, Cursor

   2. Data Dictionary
      -INFORMATION_SCHEMA Database

   3. INFORMATION_SCHEMA
      -Database Metadata에 대한 액세스
      -Database 또는 Table 이름, 열의 데이터 유형 또는 액세스 권한과 같은 MySQL 서버에 대한 정보 제공
      -데이터 사전 및 시스템 카탈로그라고도 한다.

   4. SHOW Statements 가능
      -SHOW CHARACTER SET
      -SHOW COLLATION
      -SHOW COLUMNS
      -SHOW DATABASES
      -SHOW FUNCTION STATUS
      -SHOW INDEX
      -SHOW OPEN TABLES
      -SHOW PROCEDURE STATUS
      -SHOW STATUS
      -SHOW TABLE STATUS
      -SHOW TABLES
      -SHOW TRIGGERS
      -SHOW VARIABLES

      --SHOW CHARACTER SET;
      --SHOW CHARACTER SET WHERE `Default collation` LIKE '%korean%';

   5. INFORMATION_SCHEMA Tables
      1)CHARACTER SET
         -Available character sets

      2)COLUMN_PRIVILEGES
         -Privileges defined on columns

      3)COLUMNS
         -Columns in each table

      4)GLOBAL_VARIABLES
         -Global system variables

      5)PARAMETERS
         -Stored routine parameters and stored function return values

      6)ROUTINES
         -Stored routine information

      7)SCHEMA_PRIVILEGES
         -Privileges defined on schemas

      8)SESSION_VARIABLES
         -System variables for current session

      9)TABLE_CONSTRAINTS
         -Which tables have constraints

      10)TABLE_PRIVILEGES
         -Privileges defined on tables

      11)TABLES
         -Table information

      12)TRIGGERS
         -Trigger information

      13)USER_PRIVILEGES
```

```
        -Privileges defined globally per user

    14)VIEWS
        -View information



REM View
1. 테이블 뷰를 통한 데이터의 논리적 부분 집합 또는 조합
2. 논리 테이블
3. 자체적으로 데이터를 갖고 있지 않다.
4. 데이터를 보거나 변경할 수 있는 창이다.
5. 뷰의 기반이 되는 테이블을 기본 테이블이라 한다.
6. Data Dictionary에 SELECT문으로 저장
7. 공간을 차지하지도 않는다.


REM View 의 목적
1. 데이터베이스의 선택적인 내용을 보여줄 수 있기 때문에 데이터베이스에 대한 액세스를 제한한다. --> 보안에 도움이 된다.
2. 복잡한 질의어를 통해 얻을 수 있는 결과를 간단한 질의어를 써서 구할 수 있게 한다. --> 성능향상
3. 데이터 독립성을 허용한다.
4. 동일한 데이터의 다른 VIEW를 나타낸다.
5. 조인을 한 것처럼 여러 테이블에 대한 데이터를 VIEW를 통해 볼 수 있다.
6. 한개의 VIEW로 여러 테이블에 대한 데이터를 검색할 수 있다.


REM View 종류
1. 단순뷰(Simple View)
    1)오직 하나의 테이블에서만 데이터를 가져온다.
    2)그룹 이나 다중행 함수를 포함하지 않는다.
    3)뷰를 이용해서 DML 을 수행할 수 있다.
    4)DISTINCT 사용 불가능.

2. 복합뷰(Complex View)
    1)다중 테이블에서 데이터를 가져온다.
    2)그룹, 다중행 함수를 포함한다.
    3)DML 문장을 수행할 수 없다.
    4)DISTINCT 사용 가능.


REM View Syntax
    CREATE [OR REPLACE] VIEW view_name(alias,...)
    AS
    Subquery
    [WITH CHECK OPTION]

    --OR REPLACE : 기존에 존재하는 뷰가 있다면 삭제하고 새로 만든다.
    --WITH CHECK OPTION : 서브쿼리 내의 조건을 만족하는 행만 변경 가능


REM VIEW Guide Lines
1. 뷰를 정의하는 하위 질의는 조인, 그룹, 하위 질의 등의 복합 SELECT 구문을 포함할 수 있다.
2. 뷰를 정의하는 하위 질의는 ORDER BY 절을 포함할 수 없다. ORDER BY 절은 뷰에서 데이터를 검색할 때 지정.
3. View 를 수정할 때에는 ALTER를 사용하지 않고, OR REPLACE를 사용한다.
4. VIEW 의 구조를 볼 때는 DESC 사용.

    CREATE VIEW VIEW_TEST
    AS
    SELECT * FROM TEST;  --ERROR : TEST 테이블이 없음.


    CREATE VIEW empview10
    AS
    SELECT empno, ename, job
    FROM emp
    WHERE deptno = 10;

    DESC empview10   --VIEW 구조보기


    SELECT * FROM empview10;  --View를 이용한 데이터 조회


    CREATE VIEW EMP20
```

```
    AS
    SELECT EMPNO, ENAME, SAL
    FROM EMP
    WHERE DEPTNO = 20;

    DESC EMP20;

    SELECT * FROM EMP20;


    CREATE OR REPLACE VIEW EMP20(ENO, NAME, PAYROLL)
    AS
    SELECT EMPNO, ENAME, SAL
    FROM EMP
    WHERE DEPTNO = 20;



REM Data Dictionary에서 View정보보기
    DESC INFORMATION_SCHEMA.VIEWS;

    SELECT * FROM INFORMATION_SCHEMA.VIEWS
    WHERE TABLE_NAME = 'emp20';


    CREATE VIEW EMP_30_VU
    AS
    SELECT EMPNO, ENAME, SAL, DEPTNO
    FROM EMP
    WHERE DEPTNO = 30;

    DESC EMP_30_VU;

    INSERT INTO EMP_30_VU
    VALUES(1111, 'Jimin', 500, 30);

    SELECT * FROM EMP_30_VU;
    SELECT * FROM EMP;  --VIEW에 추가한 것이 실제 기본 테이블에도 반영됨.


REM View 수정
1. OR REPLACE 옵션을 사용
2. 이미 뷰가 있더라도 뷰를 생성하여 해당 뷰를 대체
    CREATE OR REPLACE VIEW empview10
    (employee_number, employee_name, job_title)
    AS
    SELECT empno, ename, job
    FROM emp
    WHERE deptno = 10;


REM VIEW 실습
1. 부서별로 부서명, 최소 급여, 최대 급여, 부서의 평균 급여를 포함하는 DEPT_SUM View 를 생성하라.
    CREATE OR REPLACE VIEW dept_sum(deptno, tmin, tmax, tavg)
    AS
    SELECT deptno, MIN(sal), MAX(sal), AVG(sal)
    FROM emp
    GROUP BY deptno;


2. emp table에서 사원번호, 이름, 업무를 포함하는 emp_view VIEW를 생성하시오.
    CREATE OR REPLACE VIEW emp_view(사원번호, 이름, 업무)
    AS
    SELECT empno, ename, job
    FROM emp;


3. 위 2번에서 생성한 VIEW를 이용하여 10번 부서의 자료만 조회하시오
    CREATE OR REPLACE VIEW emp_view
    (사원번호, 이름, 업무)
    AS
    SELECT empno, ename, job
    FROM emp
    WHERE deptno = 10;
```

```
4. 위 2번에서 생성한 VIEW를 Data Dictionary 에서 조회하시오.
   SELECT * FROM INFORMATION_SCHEMA.VIEWS
   WHERE TABLE_NAME = 'emp_view';


5. 이름, 업무, 급여, 부서명, 위치를 포함하는 emp_dept_name 이라는 VIEW를 생성하시오.
   CREATE OR REPLACE VIEW emp_dept_name
   AS
   SELECT ename, job, sal, dname, loc
   FROM emp, dept
   WHERE emp.deptno = dept.deptno;


6. 87년에 입사한 사람을 볼 수 있는 뷰
  CREATE OR REPLACE VIEW view_emp_87
   (sabun, name, hiredate)
   AS
   SELECT empno, ename, hiredate
   FROM emp
   WHERE YEAR(hiredate) = '1987';


7.부서별로 부서명, 최소급여, 최대급여, 부서별 평균급여를 포함하는 view_dept_sum 뷰를 생성하시오.
   CREATE OR REPLACE VIEW view_dept_sum(v_dname, v_min_sal, v_max_sal, v_avg_sal)
   AS
   SELECT dname, MIN(sal), MAX(sal), AVG(sal)
   FROM  emp, dept
   WHERE emp.deptno = dept.deptno
   GROUP BY dept.dname;


REM 복합뷰
--두개 이상의 테이블로 부터 값을 출력하고, 그룹함수를 포함하는 복잡한 VIEW

   --사원이름, 업무, 급여, 부서명, 위치를 포함하는 view_emp_dept 뷰를 생성하시오.
  CREATE OR REPLACE VIEW view_emp_dept
  AS
  SELECT ename AS "사원이름", job AS "업무", sal AS "급여", dname AS "부서명", loc AS "부서의 위치"
  FROM emp, dept
  WHERE emp.deptno = dept.deptno AND emp.deptno = 10;


   CREATE TABLE dept_clone
   AS
   SELECT *
   FROM dept;

   ALTER TABLE dept_clone
   ADD CONSTRAINT dept_clone_deptno_pk PRIMARY KEY(deptno);

   ALTER TABLE dept_clone
   MODIFY dname   VARCHAR(14)  NOT NULL;

   CREATE OR REPLACE VIEW view_dept_clone
   AS
   SELECT deptno, loc
   FROM dept_clone;

   INSERT INTO DEPT_CLONE
   VALUES(50, 'SEOUL');  --ERROR


REM WITH CHECK OPTION  절 사용하기
--사원테이블과 동일한 emp_20(20번부서만)이라는 뷰를 생성하되, WITH CHECK OPTION 을 사용해서 생성하시오.

   CREATE OR REPLACE VIEW emp_20
   AS
   SELECT * FROM emp
   WHERE deptno = 20
   WITH CHECK OPTION CONSTRAINT emp_20_ck;

   UPDATE emp_20
```

```
   SET deptno = 30
   WHERE empno = 7566;


REM Limit
1. 테이블에서 조건에 대한 최상위 레코드 N개 또는 최하위 레코드 N개를 표시
2. Syntax
   SELECT
   FROM
   LIMIT N;

   SELECT empno, ename, hiredate
   FROM   emp
   ORDER BY hiredate
   LIMIT 3;


   SET @ROWNUM :=0;
   SELECT @ROWNUM := @ROWNUM + 1 AS rank, empno, ename, hiredate
   FROM   emp
   ORDER BY hiredate
   LIMIT 3;

 --emp table에서 가장 최근에 입사한 5명의 사원번호, 사원명, 입사날짜를 출력하시오.


REM View  제거
1.뷰가 삭제돼도 기본 테이블의 데이터에는 영향이 없음.
2. Syntax
   DROP VIEW view_name;

   DROP VIEW empview10;



REM INDEX
1. 행에 대한 빠른 참조를 위해서, 테이블에 인덱스를 생성할 수 있다.

2. 장/단점
   1)장점
      -검색 속도가 빨라진다.
      -시스템에 부하를 줄여서 시스템 전체 성능을 향상시킨다.
   2)단점
      -인덱스를 위한 추가 공간이 필요하다.
      -인덱스를 생성하는 데 시간이 걸린다.
      -데이터의 변경작업이 자주 일어나는 경우에는 오히려 성능이 더 떨어진다.

3. 생성해야 할 조건
   1)테이블의 행의 수가 많다. 아주 작은 크기의 테이블에는 오히려 성능이 떨어진다.
   2)사용자의 SQL 문에서 WHERE 조건절에 자주 사용되는 칼럼이 대상이 된다.
   3)WHERE  조건에 의한 결과가 전체 행수의 비율(분포도) 2~4% 인 경우에 효과가 있다.
   4)분포도가 범위 이상이더라도 일부분의 데이터 검색이라면 적용가능
   5)JOIN 에 자주 사용되는 칼럼이나 NULL 을 포함하는 칼럼이 많은 경우

4. 생성하지 않아야 할 조건
   1) 테이블에 행이 적은 경우
   2) 컬럼이 WHERE 조건에 자주 사용되지 않을 때
   3) WHERE 조건에 의한 결과가 전체 행에 대해 10~15%의 결과보다 높게 리턴될 때
   4) 테이블이 자주 입력, 수정 , 삭제 될 때는 오히려 검색 속도가 더 떨어진다.

5. INDEX Type
   1)UNIQUE index : 지정된 열의 값이 고유해야 한다.
      CREATE UNIQUE INDEX index_name
      ON table_name(column1, column2, ...)

      -Index 확인
         SHOW INDEX FROM table_name;


      CREATE TABLE dept1
      AS
      SELECT * FROM dept
      WHERE 0 = 1;
```

```
    INSERT INTO dept1 VALUES(10, 'ACCOUNTING', 'SEOUL');
    INSERT INTO dept1 VALUES(20, 'SALES', 'PUSAN');
    INSERT INTO dept1 VALUES(30, 'OPERATION', 'PUSAN');
    INSERT INTO dept1 VALUES(40, 'IT', 'DAEJUN');

    CREATE UNIQUE INDEX idx_dept1_deptno ON dept1(deptno); --SUCCESS
    CREATE UNIQUE INDEX idx_dept1_loc ON dept1(loc); --ERROR

    CREATE INDEX idx_dept1_loc ON dept1(loc); --SUCCESS. UNIQUE 를 빼면 됨.


  2)Non-unique index : 가장 빠름을 보장하는 칼럼, 칼럼의 값이 고유하지 않을 때

    CREATE INDEX i_emp_ename ON emp(ename);


  3)Single column index : 하나의 컬럼에만 인덱스를 부여
    CREATE INDEX i_emp_ename ON emp(ename);

  4)Composite Index : 여러 컬럼에 인덱스부여가능
    CREATE INDEX I_emp_empno_ename ON emp(empno, ename);


6. INDEX 확인
  SHOW INDEX FROM table_name;

  SHOW INDEX FROM emp;


7. INDEX 의 제거
  ALTER TABLE table_name
  DROP INDEX index_name;

  OR
  DROP INDEX index_name ON table_name;


8. INDEX 의 수정
  1)인덱스는 수정할 수 없다.
  2)수정하기 위해서는 제거하고 새로 생성해야 한다.
```

REM Author :
REM Date :
REM Objective : Chapter 12. SQL Programming
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM SQL
1. 장점
   1)SQL 질의문 하나로 원하는 데이터를 검색 및 조작할 수 있다.
   2)사용자가 이해하기 쉬운 단어로 구성
   3)복잡한 로직을 간단하게 작성할 수 있다.
   4)ANSI 에 의해 문법이 표준화되어 있다.

2. 단점
   1)반복처리를 할 수 없다(LOOP)
   2)비교처리를 할 수 없다(IF).
   3)Error 처리를 할 수 없다(EXCEPTION).
   4)SQL 문을 캡슐화할 수 없다.
   5)변수선언을 할 수 없다.
   6)실행할 때마다 분석작업 후 실행한다.
   7)Network Traffic 을 유발한다.
   8)SQL 문 자체는 비 절차적 언어이므로, 여러 개의 질의문 사이에 연결이나 절차가 있어야 할 때에는 사용할 수 없다.
   9)실제 프로그래밍에서는 다른 언어를 사용해서 각각의 SQL 문들을 서로 연관되도록 하고 절차적 또는 순차적인 단계를 가지고 SQL문이 실행되도록 해야 한다.
   10)다른 언어를 이용해서 처리해도 되고, MySQL 자체적으로는 SQL Programming을 사용한다.
   11)다른 RDBMS에서는 사용할 수 없다.

REM SQL Programming
1. 개요
   1)SQL 문의 제한을 극복하기 위해 MySQL에서 추가적으로 만든, SQL 언어에 절차적인 프로그래밍 언어를 가미해 생성
   2)일반 프로그래밍의 언어적인 요소를 거의 다 가지고 있다.

2. 특징
   1)프로그램 개발시 모듈화
      -논리적 문장들을 그룹화
      -복잡한 프로그램 모듈을 그룹화가능
   2)변수 선언
   3)절차적 구조로 된 프로그래밍
      -조건문, 반복문
   4)ERROR 처리 가능

3. 구조
   delimiter --> block의 시작 (필수)
   DECLARE --> 변수의 선언 (선택)
   SET --> 변수의 선언 및 값 할당(선택)
   BEGIN --> 실행부 시작 (필수)
   END; --> block의 끝 (필수)

4. MySQL User-defined Variables
   1)MySQL은 또한 한 명령문에서 다른 명령문으로 값을 전달할 수 있는 사용자 정의 변수의 개념을 지원한다.
   2)MySQL의 사용자 정의 변수는 @var_name으로 작성
   3)여기서 var_name은 변수의 이름이며 영숫자 문자, ., _ 및 $로 구성될 수 있다.
   4)사용자 정의 변수는 세션에 따라 다르다.
      -한 클라이언트에서 정의한 변수는 다른 클라이언트와 공유되지 않으며 세션이 종료되면 이러한 변수는 자동으로 만료된다.
   5)변수 이름은 대소문자를 구별하지 않는다.
      -@mark or @Mark 같다.
   6)최대 이름의 길이는 64 글자이다.
   7)변수이름에는 특수문자 즉 !, #, ^, -, 등...이 포함될 수 있다.
      -단, 인용부호로 묶어야 한다.
      -@'var@1' or @"var^2" or @`var3`(백틱)
   8)이러한 변수는 선언할 수 없으며, 선언 시에만 초기화된다.
      -즉, 값을 할당해야 한다.
   9)선언되지 않은 변수는 SQL문장이 수행될 때 NULL로서 설정된다.
   10)변수 선언시 데이터타입(integer, floating-point, decimal, binary, nonbinary string or NULL value)을 지정
   11)Syntax
      SET @var_name = expression

   12)연산자 사용
      -SET @변수명 을 사용시 = 대입연산자를 사용한다
      -SELECT @변수명 을 사용시 := 과 같은 대입연산자를 사용한다.

      mysql>SET @var1 = 2+6;
      mysql>SET @var2 := @var1-2;

```
mysql>SELECT @var1, @var2;

   @var1    @var2
   ---------------------
   8             6


mysql>SELECT @var3;

   @var3
   -------------
   NULL


mysql>SELECT @var3 := 4;

   @var3 := 4
   ------------------
   4

mysql>SELECT @var4 = 5;

   @var4=5
   -----------
   NULL

mysql> SET @v1 = X'41';
mysql> SET @v2 = X'41'+0;
mysql> SET @v3 = CAST(X'41' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;

+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+


mysql> SET @v1 = b'1000001';
mysql> SET @v2 = b'1000001'+0;
mysql> SET @v3 = CAST(b'1000001' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+


SET @total_salary = (SELECT SUM(sal) FROM emp);
SELECT @total_salary;


13)변수의 종류
-지역변수
   --지역(로컬)변수는 프로시저(Procedure) 또는 트리거(Trigger) 내에서 로컬 변수 및 입력 매개 변수로 사용할 수 있다.
   --즉, Declares 내 지역(로컬)변수를 사용함을 의미 한다.
   ---Syntax
      DECLARE variable_name datatype(size) [DEFAULT default_value];

      DECLARE RTN_VAL VARCHAR(8);
      DECLARE total_price Oct(8,2) DEFAULT 0.0;
      DECLARE a,b,c INT DEFAULT 0;

      DELIMITER //
      Create Procedure Test()
         BEGIN
            DECLARE A INT DEFAULT 100;
            DECLARE B INT;
            DECLARE C INT;
            DECLARE D INT;
            SET B = 90;
            SET C = 45;
            SET D = A + B - C;
```

```
            SELECT A, B, C, D;
        END
        //
    DELIMITER ;

    CALL Test();


-시스템 변수
  --MySQL은 기본적으로 선언된 변수들이 존재한다. 이를 시스템 변수라 한다.
  --시스템 변수는 GLOBAL 또는 세션단위로 사용가능하다.
  --즉, 서버의 전체 작업과 클라이언트 연결 후 작업등 모든 부분에 영향을 준다.

  --시스템 변수 선언
    -- Syntax to Set value to a Global variable:
    SET GLOBAL sort_buffer_size=1000000;
    SET @@global.sort_buffer_size=1000000;

    -- Syntax to Set value to a Session variable:
    SET sort_buffer_size=1000000;
    SET SESSION sort_buffer_size=1000000;
    SET @@sort_buffer_size=1000000;
    SET @@local.sort_buffer_size=10000;

  --시스템 변수 확인
    --- 모든 변수 확인
      SHOW VARIABLES;

    --- 특정 변수 확인
    SELECT @@sort_buffer_size;


5. 조건문
  1)IF 문
    -Syntax
      IF 조건 THEN
        처리문;
      ENF IF;

      delimiter //
      CREATE PROCEDURE if_test()
      BEGIN
        DECLARE var INT;
        SET var = 52;
        IF var % 2 = 0 THEN
          SELECT 'Even Number';
        END IF;
      END
      //
      delimiter ;

      CALL if_test();


      IF 조건 THEN
        처리문1;
      ELSE
       처리문2;
      END IF;

      delimiter //
      CREATE PROCEDURE if_test()
      BEGIN
        DECLARE var INT;
        SET var = 51;
        IF var % 2 = 0 THEN
          SELECT 'Even Number';
        ELSE
          SELECT 'Odd Number';
        END IF;
      END
      //
      delimiter ;
```

```
        CALL if_test();


        IF 조건1 THEN
           처리문1;
        ELSEIF  조건2  THEN
           처리문2;
        ELSEIF  조건3  THEN
           처리문3;
        ...
        ELSE
           처리문N;
        END IF;

        delimiter //
        CREATE PROCEDURE if_test()
        BEGIN
           DECLARE season VARCHAR(20);
           SET season = '여름';
           IF season = '봄' THEN
              SELECT '진달래, 개나리';
           ELSEIF season = '여름' THEN
              SELECT '장미, 아카시아';
           ELSEIF season = '가을' THEN
              SELECT '코스모스, 백합';
           ELSE
              SELECT '동백, 매화';
           END IF;
        END
        //
        delimiter ;

        CALL if_test();


    --성적관리프로그램
       delimiter //
       CREATE PROCEDURE sungjukmgmt()
       BEGIN
          DECLARE irum VARCHAR(20);
          DECLARE hakbun CHAR(6);
          DECLARE kor, eng, mat, tot INT DEFAULT 0;
          DECLARE average DECIMAL(5, 2) DEFAULT 0.00;
          DECLARE hakjum CHAR(1) DEFAULT 'F';

          SET irum = '백두산';
          SET hakbun = '21-001';
          SET kor = 78, eng = 89, mat = 99;
          SET tot = kor + eng + mat;
          SET average = tot / 3;
          IF average <= 100 AND average >= 90 THEN
             SET hakjum = 'A';
          ELSEIF average < 90 AND average >= 80 THEN
             SET hakjum = 'B';
          ELSEIF average < 80 AND average >= 70 THEN
             SET hakjum = 'C';
          ELSEIF average < 70 AND average >= 60 THEN
             SET hakjum = 'D';
          ELSE
             SELECT hakjum = 'F';
          END IF;

          SELECT CONCAT('이름 ==> ', irum, CHAR(10), '학번 ==> ', hakbun, CHAR(10),
                 '국어 ==> ', kor, CHAR(10), '영어 ==> ', eng, CHAR(10),
                 '수학 ==> ', mat, CHAR(10), '총점==> ', tot, CHAR(10),
                 '평균 ==> ', average, CHAR(10), '평점 ==> ', hakjum);
       END
       //
       delimiter ;

       CALL sungjukmgmt()


    2)CASE문
```

```
296
297         CASE case_value
298           WHEN when_value THEN statement_list
299           [WHEN when_value THEN statement_list] ...
300           [ELSE statement_list]
301         END CASE
302
303         OR
304
305         CASE
306           WHEN search_condition THEN statement_list
307           [WHEN search_condition THEN statement_list] ...
308           [ELSE statement_list]
309         END CASE
310
311
312       --성적관리프로그램
313         delimiter //
314         CREATE PROCEDURE sungjukmgmt()
315         BEGIN
316           DECLARE irum VARCHAR(20);
317           DECLARE hakbun CHAR(6);
318           DECLARE kor, eng, mat, tot INT DEFAULT 0;
319           DECLARE average DECIMAL(5, 2) DEFAULT 0.00;
320           DECLARE hakjum CHAR(1) DEFAULT 'F';
321
322           SET irum = '백두산';
323           SET hakbun = '21-001';
324           SET kor = 78, eng = 89, mat = 99;
325           SET tot = kor + eng + mat;
326           SET average = tot / 3;
327
328           CASE
329             WHEN average >= 90 THEN
330               SET hakjum = 'A';
331             WHEN average >= 80 THEN
332               SET hakjum = 'B';
333             WHEN average >= 70 THEN
334               SET hakjum = 'C';
335             WHEN average >= 60 THEN
336               SET hakjum = 'D';
337             ELSE
338               SELECT hakjum = 'F';
339           END CASE;
340
341           SELECT CONCAT('이름 ==> ', irum, CHAR(10), '학번 ==> ', hakbun, CHAR(10),
342                 '국어 ==> ', kor, CHAR(10), '영어 ==> ', eng, CHAR(10),
343                 '수학 ==> ', mat, CHAR(10), '총점==> ', tot, CHAR(10),
344                 '평균 ==> ', average, CHAR(10), '평점 ==> ', hakjum);
345         END
346         //
347         delimiter ;
348
349         CALL sungjukmgmt()
350
351
352   10. 반복문
353     1)Syntax
354         WHILE search_condition DO
355           statement_list
356         END WHILE
357
358         --5,4,3,2,1
359         delimiter //
360         CREATE PROCEDURE dowhile()
361         BEGIN
362           DECLARE i INT DEFAULT 5;
363           DECLARE str VARCHAR(50);
364           SET str = ' ';
365
366           WHILE i > 0 DO
367             SET str = CONCAT(str, i, ', ');
368             SET i = i - 1;
369           END WHILE;
```

```
370
371         SELECT SUBSTRING(RTRIM(str), 1, LENGTH(str) - 2);
372     END
373     //
374     delimiter ;
375
376     CALL dowhile();
377
378     --구구단
379     CREATE TABLE tbl_gugudan
380     (
381         result VARCHAR(100)
382     );
383
384     delimiter //
385     CREATE PROCEDURE gugudan()
386     BEGIN
387         DECLARE i INT;
388         DECLARE j INT;
389         DECLARE str VARCHAR(100);
390
391         SET i = 1;
392         WHILE i < 10 DO
393             SET str = '';
394             SET j = 2;
395             WHILE j < 10 DO
396                 SET str = CONCAT(str, j, ' x ', i, ' = ', j * i, '  ');
397                 SET j  = j + 1;
398             END WHILE;
399             SET i = i + 1;
400             INSERT INTO tbl_gugudan VALUES(str);
401         END WHILE;
402     END
403     //
404     delimiter ;
405
406     CALL gugudan();
407
408     SELECT * FROM tbl_gugudan;
```

```
REM Author :
REM Date :
REM Objective : Chapter 13. Stored Objects
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0


REM Stored Programs
1. 잘 정리된 논리적 코드분할이름
2. 컴파일된 상태에서 데이터베이스에 저장되기 때문에 성능이 향상된다.
3. 테이블이름이나 컬럼의 이름을 명시하지 않기 때문에 보안에 도움이 된다.
4. 모듈화를 통한 관리 용이
5. SQL 문으로 구성된 본문이 있다.
6. 본문은 세미콜론 문자로 구분된 여러 SQL문으로 구성된다.
7. 종류
   1)Stored Procedures   <-----Java 에서 사용
   2)Stored Functions


REM Stored PROCEDURE(저장프로시저)
1. 목적 : 속도, 보안
2. compile 상태로 RDBMS 에 저장
3. 나중에 실행될 일련의 명령의 집합
4. Syntax
   DELIMITER //
   CREATE PROCEDURE procedure_name
   (
      [IN | OUT | IN OUT]  param_name  type
   )
   BEGIN
      SQL 문장들
   END
   //
   DELIMITER;

5. Parameter Mode  : 3가지
   -IN : 입력 매개변수
   -OUT : 출력 매개변수
   -IN OUT : 입력, 출력 매개변수

6. Examples

   delimiter //
   CREATE PROCEDURE helloworld()
   BEGIN
      SELECT 'Hello, World';
   END
   //
   delimiter ;

   CALL helloworld();


   delimiter //
   CREATE PROCEDURE test_proc()
   BEGIN
      SET @v_name = '백두산';
      SELECT CONCAT('My name is ', @v_name);
   END
   //
   delimiter ;

   CALL test_proc();


   -- emp 테이블이 모든 데이터를 삭제하는 Stored Procedure 를 작성하시오.
   delimiter //
   CREATE PROCEDURE del_all()
      BEGIN
   DELETE FROM emp_copy;
   END
   //
   delimiter ;

   CALL del_all();
```

7. 확인하기

```
DESC INFORMATION_SCHEMA.ROUTINES;

SELECT * FROM INFORMATION_SCHEMA.ROUTINES
WHERE specific_name = 'helloworld';
```

8. IN 매개변수

```
delimiter //
CREATE PROCEDURE test_proc_in(IN p_name VARCHAR(30))
BEGIN
   SELECT CONCAT('My name is ', p_name);
END
//
delimiter ;

CALL test_proc_in('백두산');


--사원번호와 봉급을 입력받아 업데이트하는 Procedure를 완성하시오.

delimiter //
CREATE PROCEDURE emp_sal_update(v_empno SMALLINT, v_sal FLOAT)
BEGIN
   UPDATE emp SET sal = v_sal
   WHERE empno = v_empno;
   COMMIT;
END
//
delimiter ;

CALL emp_sal_update(7369, 1000);


--사번을 받아 삭제하는 프로시저
delimiter //
CREATE PROCEDURE emp_del(v_empno SMALLINT)
BEGIN
   DELETE FROM emp
   WHERE empno = v_empno;
   COMMIT;
END
//
delimiter ;
CALL emp_del(7900);


--부서번호, 부서이름, 지역을 받아 삽입하는 프로시저
delimiter //
CREATE PROCEDURE sp_insert_dept(v_deptno TINYINT, v_dname VARCHAR(14), v_loc VARCHAR(13))
BEGIN
   INSERT INTO DEPT
   VALUES (v_deptno, UPPER(v_dname), UPPER(v_loc));
   COMMIT;
END
//
delimiter ;

CALL sp_insert_dept(50, 'marketting', 'Seoul')


--emp table에서 새로운 사원의 정보를 이름, 업무, 매니저, 급여를 입력받아 등록하는 emp_input 프로시저를 생성하라. 단, 부서번호는 매니저의 부서 번호와 동일하게 하고 보너스는 SALESMAN은 0을 그 외는 NULL을 입력하라.

delimiter //
CREATE PROCEDURE sp_emp_input
(
   v_empno  SMALLINT,
   v_ename  VARCHAR(10),
   v_job    VARCHAR(9),
   v_mgr    SMALLINT,
```

```
    v_sal   FLOAT
)
BEGIN
   DECLARE v_deptno TINYINT;

   SELECT deptno
   INTO   v_deptno
   FROM emp
   WHERE empno = v_mgr;

   IF UPPER(v_job) = 'SALESMAN' THEN
      INSERT INTO emp
      VALUES(v_empno, v_ename, UPPER(v_job), v_mgr, CURDATE(), v_sal, 0, v_deptno);
   ELSE
      INSERT INTO emp
      VALUES(v_empno, v_ename, UPPER(v_job), v_mgr, CURDATE(), v_sal, NULL, v_deptno);
   END IF;
END
//
delimiter ;

CALL sp_emp_input(8000, 'Sujan', 'salesman', 7902, 2000);
CALL sp_emp_input(8001, 'Sally', 'clerk', 7566, 3000);


-- 우편번호 검색하기
delimiter //
CREATE PROCEDURE sp_zipcode
(
   IN v_dong VARCHAR(100)
)
BEGIN
   SELECT zipcode, sido, gugun, dong, bunji
   FROM  zipcode
   WHERE dong LIKE CONCAT('%', v_dong, '%');
END //
delimiter ;

CALL sp_zipcode('역삼');


--이름을 입력받아서 그 사람의 업무가 MANAGER, ANALYST 이면 급여가 50% 가산하여 갱신하고, 업무가 MANAGER, ANALYST 가 아니면 20% 가산하는 SQL문을 작성하시오.


9. OUT 매개변수
delimiter //
CREATE PROCEDURE test_proc_out
(
   OUT v_name VARCHAR(30)
)
BEGIN
   DECLARE p_name VARCHAR(30);
   SET p_name = 'My name is 한라산';
   SELECT p_name INTO v_name;
END
//
delimiter ;

CALL test_proc_out(@t_name);    --binding 변수필요
SELECT @t_name;


--주어진 두개의 수 중 작은 수 구하기
delimiter //
CREATE PROCEDURE findMin
(
   IN x INT,   IN y INT,  OUT z INT
)
BEGIN
   DECLARE v_min INT;

   IF x < y THEN
      SET v_min = x;
```

```
    ELSE
       SET v_min = y;
    END IF;

    SELECT v_min INTO z;
END
//
delimiter ;

CALL findMin(23, 45, @t_min);
SELECT CONCAT('Minimum of (23,45) ==> ', @t_min);


--사번을 받아 사원이름과 봉급 검색
delimiter //
CREATE PROCEDURE sp_emp_select
(
    IN v_empno SMALLINT,
    OUT v_ename VARCHAR(10),
    OUT v_sal FLOAT
)
BEGIN
    SELECT ename, sal INTO v_ename, v_sal
    FROM emp
    WHERE empno = v_empno;
END
//
delimiter ;

CALL sp_emp_select(7788, @t_ename, @t_sal);
SELECT @t_ename, @t_sal;


--이름을 입력받아서 그 사원의 정보 중 부서명과 급여를 검색하는 프로시저를 완성하시오.
delimiter //
CREATE PROCEDURE emp_dept_sal_select
(
    IN v_ename VARCHAR(10),
    OUT v_dname  VARCHAR(14),
    OUT v_sal  FLOAT
)
BEGIN
    DECLARE v_deptno TINYINT;

    SELECT deptno, sal INTO v_deptno, v_sal
    FROM   emp
    WHERE ename = v_ename;

    SELECT dname INTO v_dname
    FROM   dept
    WHERE  deptno = v_deptno;
END
//
delimiter ;

CALL emp_dept_sal_select('SMITH', @t_dname, @t_sal);
SELECT CONCAT('SMITH의 부서명 ==> ', @t_dname, ', 봉급 ==> ', @t_sal);


10. IN OUT 파라미터
delimiter //
CREATE PROCEDURE test_proc_inout(INOUT v_name VARCHAR(30))
BEGIN
    DECLARE v_str VARCHAR(30);
    SET v_str = CONCAT('My name is ', v_name);

    SELECT v_str INTO v_name;
END
//
delimiter ;

SET @t_name = '북한산';
CALL test_proc_inout(@t_name);
SELECT @t_name;
```

11. Stored **Procedure ALTER**
-MySQL에서는 Stored **Procedure** 의 파라미터나 **body**를 수정할 수 있는 **ALTER Procedure** 는 지원하지 않는다.
-수정이 필요하면 프로시저 삭제 후 새로 생성해야 한다.

12. Stored **Procedure** Deletion
DROP PROCEDURE [IF EXISTS] sp_name;

13. Stored **Procedure**의 특징
1)MySQL의 성능을 향상시킨다.
2)모듈식 프로그래밍이 가능하다.
3)보안을 강화할 수 있다.
4)Programing **Language**에서 **Procedure**의 이름으로 호출할 수 있다.

REM Stored **FUNCTION**
1. MySQL에서 기본적으로 제공하는 함수 이외에 사용자가 필요에 따라 만든 함수
2. **Procedure**와 성격이 매우 비슷하지만, 반환값이 있느냐 없느냐가 가장 큰 차이
3. 실행시 반드시 하나의 값을 **RETURN**하기 위해 사용
4. 함수 선언에서 **Data Type**이 있는 **RETURN** 절을 추가하고 **body**에 **RETURN**문을 포함
5. 함수는 **IN** 파라미터만 사용한다.
6. Syntax

```
DELIMITER //
CREATE FUNCTION function_name
(
   param_name  type
)
RETURNS type
BEGIN
   SQL 문장들
   RETURN
END
//
DELIMITER;

--실행
SELECT function_name(argument_list);
```

```
delimiter //
CREATE FUNCTION chk_sal(v_sal FLOAT)
RETURNS FLOAT
BEGIN
   DECLARE t_sal FLOAT;
   SET t_sal = v_sal * 0.01;
   RETURN t_sal;
END
//
delimiter ;

SELECT empno, ename, sal, chk_sal(sal)
FROM emp
WHERE deptno = 10;
```

```
     EMPNO       SAL CHK_SAL(SAL)
---------- ---------- ------------
      7782       2450         24.5
      7839       5000           50
      7934       1300           13
```

```
SELECT SUM(sal), chk_sal(SUM(sal))
FROM emp
WHERE deptno = 10;
```

```
  SUM(SAL) CHK_SAL(SUM(SAL))
---------- ----------------
      8750             87.5
```

```
delimiter //
CREATE FUNCTION tax(v_value INT)
RETURNS INT
BEGIN
   RETURN v_value * 0.07;
END
//
delimiter ;

SELECT  sal, tax(sal)
FROM  emp
WHERE empno = 7902;

      SAL   TAX(SAL)
---------- ----------
      950       66.5


--사원명으로 검색하여 해당 사원의 직급을 얻어 오는 함수를 fun_sel_empname라는 이름으로 작성하시오.
delimiter //
CREATE FUNCTION fun_sel_empname(v_ename VARCHAR(10))
RETURNS VARCHAR(10)
BEGIN
   DECLARE v_job VARCHAR(9);
   SELECT job INTO v_job
   FROM emp
   WHERE ename = v_ename;

   RETURN v_job;
END
//
delimiter ;

SELECT fun_sel_empname('SCOTT');

--emp table에서 이름을 입력받아 부서번호, 부서명, 급여를 검색하는 함수(fun_emp_disp)을 작성하시오. 단 부서번호를 RETURN에 사용하시오.
   delimiter //
   CREATE FUNCTION fun_emp_disp(v_ename VARCHAR(10))
   RETURNS TINYINT
   BEGIN
      SELECT depno, dname, sal
      FROM emp NATURAL JOIN dept
      WHERE ename = v_ename;

      RETURN deptno;
   END
   //
   delimiter ;     ==> Error

   delimiter //
   CREATE FUNCTION fun_emp_disp(v_ename VARCHAR(10))
   RETURNS VARCHAR(100)
   BEGIN
      DECLARE v_deptno TINYINT;
      DECLARE v_dname  VARCHAR(14);
      DECLARE v_sal FLOAT;

      SELECT deptno, dname, sal INTO v_deptno, v_dname, v_sal
      FROM emp NATURAL JOIN dept
      WHERE ename = v_ename;

      RETURN CONCAT('Department Number ==> ', v_deptno, ', Department Name ==> ', v_dname, ',
      Salary ==> ', v_sal);
   END
   //
   delimiter ;

   SELECT fun_emp_disp('SCOTT');


7. 수정
   -Stored Procedure와 마찬가지로 수정은 할 수 없고 삭제 후 새로 생성해야 한다.
```

8. 삭제

```
DROP FUNCTION function_name;
```

9. Stored **Procedure** vs Stored **Function**

Stored **Procedure**	vs	Stored **Function**
파라미터 **IN, OUT, INOUT** 사용		입력용 파라미터만 사용가능
RETURNS 사용불가		반드시 **RETURN** 사용해야
CALL을 사용하여 호출		**SELECT** 문자에서만 사용
모든 **Statement** 사용 가능		**SELECT..INTO..**사용 가능
다양한 목적 사용		계산을 통한 하나의 값 반환시 사용

```
REM Author :
REM Date :
REM Objective : Chapter 14. DCL in MySQL
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0


REM Before you begin
1. All commands are executed inside the MySQL shell as root or administrative user.
2. The minimum privileges required to create user accounts and define their privileges is CREATE USER and GRANT.
3. To access the MySQL shell type the following command and enter your MySQL root user password when prompted:
   $ mysql -u root -p


REM Create a new MySQL User Account
1. A user account in MySQL consists of two parts: user name and host name.
2. To create a new MySQL user account, run the following command:
   mysql>CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'user_password';

3. To grant access from another host, change the hostname part with the remote machine IP.
   -For example, to grant access from a machine with IP 10.8.0.5 you would run:
   mysql>CREATE USER 'newuser'@'10.8.0.5' IDENTIFIED BY 'user_password';

4. To create a user that can connect from any host, use the '%' wildcard as a host part:
   mysql>CREATE USER 'newuser'@'%' IDENTIFIED BY 'user_password';


REM Grant Privileges to a MySQL User Account
1. There are multiple types of privileges that can be granted to a user account.
2. The most commonly used privilege are:
   1)ALL PRIVILEGES – Grants all privileges to a user account.
   2)CREATE – The user account is allowed to create databases and tables.
   3)DROP - The user account is allowed to drop databases and tables.
   4)DELETE - The user account is allowed to delete rows from a specific table.
   5)INSERT - The user account is allowed to insert rows into a specific table.
   6)SELECT – The user account is allowed to read a database.
   7)UPDATE - The user account is allowed to update table rows.

3. To grant specific privileges to a user account, use the following syntax:
   mysql>GRANT permission1, permission2 ON database_name.table_name TO 'database_user'@'localhost';

4. Examples:
   1)Grand all privileges to a user account over a specific database:
      mysql>GRANT ALL PRIVILEGES ON database_name.* TO 'database_user'@'localhost';

   2)Grand all privileges to a user account on all databases:
      mysql>GRANT ALL PRIVILEGES ON *.* TO 'database_user'@'localhost';

   3)Grand all privileges to a user account over a specific table from a database:
      mysql>GRANT ALL PRIVILEGES ON database_name.table_name TO 'database_user'@'localhost';

   4)Grant multiple privileges to a user account over a specific database:
      mysql>GRANT SELECT, INSERT, DELETE ON database_name.* TO database_user@'localhost';


REM Display MySQL User Account Privileges
-To find the privilege(s) granted to a specific MySQL user account, use the SHOW GRANTS statement:
   mysql>SHOW GRANTS FOR 'database_user'@'localhost';


REM Revoke Privileges from a MySQL User Account
-To revoke all privileges from a user account over a specific database, run the following command:
   mysql>REVOKE ALL PRIVILEGES ON database_name.* FROM 'database_user'@'localhost';


REM Remove an Existing MySQL User Account
-To delete a MySQL user account use the DROP USER statement:
   mysql>DROP USER 'user'@'localhost'


REM ALTER USER
1. The ALTER USER statement modifies MySQL accounts.
2. Syntax
```

```
   ALTER USER [IF EXISTS] user IDENTIFIED BY 'password_string';


REM User Check
   mysql>SELECT host, user, plugin, authentication_string, password_last_changed FROM mysql.user;


REM Saving Your Changes
-As a final step following any updates to the user privileges, be sure to save the changes by issuing the FLUSH
 PRIVILEGES command from the mysql prompt:
   mysql>FLUSH PRIVILEGES;

```