```
REM Author : Henry
REM Date : 2024.06.12
REM Objective :
REM Environment : Ubuntu Server 22.04 LTS, MySQL Workbench 8.0 CE, MySQL Community Server 8.0.37-0ubuntu0.22.04.3 (ubuntu)


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.

테이블을 바라보는 관점을 전체가 아닌 일부로 제한.
같은 테이블을 호출할 때 모든 테이블이 아닌 일부만 가져와
마치 원래 그런 테이블이 있던 것 처럼 View로 구성.
>> 이후로는 FROM Table 대신 FROM View로 쿼리한다

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
```

단순 뷰를 통해 갖온 테이블은 view를 이용해 수정할 수 있다.

복합 뷰는 기본적으로 JOIN을 이용해 테이블을 가져온다.
Read Only인 뷰

with check option은 where 절에서 조건으로 지정한 컬럼은
바뀌지 않도록 제한하는 것.

```
    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;
```

VIEW에 INSERT된 레코드가 기존 테이블에도 등록되는 것을 볼 수 있다.

view가 이미 있는 view라면 수정, 아니라면 생성

```
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;

                INFORMATION_SCHEMA.VIEWS는 관리자용 DB로 VIEWS 테이블은 mysql 내에서 생성된 view를 출력하는 구문

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
                                  연결된 dept 테이블에 50번이 없어서 오류


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;
```
에러 메세지 : CHECK OPTION Failed 출력

REM **Limit**

ORDER BY로 최상위, 최하위를 구분해 출력 갯수를 N개로 제한

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.

데이터가 많아져 한번에 탐색할 범위가 넓은 경우
>> INDEX로 쪼개 관리하는 것이 효율적이다.

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
PRIMARY KEY를 제외한 UNIQUE한 컬럼으로 인덱스를 생성할 때 선언

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);
```
UNIQUE를 뺀 INDEX

3)Single column index : 하나의 컬럼에만 인덱스를 부여

```
CREATE INDEX i_emp_ename ON emp(ename);
```
일반적으로 하나의 컬럼에 하나의 인덱스

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.