SQL #3. DDL

I. DDL

```
================================================
# DICTIONARY
데이터베이스의 각종 정보를 확인할 수 있다.
수정이나 삭제는 불가능하다. 주로 BASE table에 대한 view이다.
================================================
```

 # 사용자에게 속한 테이블

SELECT table_name FROM user_tables ;

 # 사용자에게 속한 객체 타입

SELECT DISTINCT object_type FROM user_objects ;

 # 사용자의 스키마 객체

SELECT * FROM user_catalog ;

```
================================================
# Create table (NAMING RULES)
글자로 시작하고, 대소문자를 구분하지는 않으나 실질적으로는 Dictionary에 대문자로 저장된다.
명시적으로 대소문자를 구분한 경우 대소문자를 따르게 된다.
================================================
```

 # scott 으로 접속

SQL> conn scott

 # 다음을 실습해 보시오.

```
SQL> CREATE TABLE   cre_tab1
                (   db_user   VARCHAR2(30) DEFAULT USER,
                    issue_date DATE DEFAULT   SYSDATE,
               type_operation NUMBER(3));
SQL> SELECT * FROM tab;
```

(* 현재 user가 소유하고 있는 table 리스트가 출력됨. cre_tab1 테이블명이 CRE_TAB1로 보임.)

```
SQL> INSERT INTO CRE_TAB1(type_operation) VALUES (100);
SQL> SELECT * FROM cre_tab1;
```

SQL> SELECT * FROM Cre_Tab1;
(* FROM 절에 대소문자를 섞어도 같은 table을 조회한다.)

 # table 명에 대소문자를 섞어서 사용하는 예이다. 다음을 실습해 보시오.
SQL> CREATE TABLE "Cre_tab2" (a NUMBER, b CHAR);
SQL> SELECT * FROM cre_tab2;
(* Error가 발생하는 이유를 적으시오.)

SQL> SELECT * FROM tab;
(* Cre_tab2 테이블 명이 대소문자 구별되어 보임.)

SQL> SELECT * FROM "Cre_tab2";
SQL> SELECT * FROM Cre_tab2;

```
================================================
# Data Type
================================================
```

 # 다음은 CHAR와 VARCHAR2 type에 대한 실습이다. 각 문장의 수행 결과를 예측하시오.
```
SQL> CREATE TABLE test_char
            ( id   NUMBER,
              a CHAR(2),
              b VARCHAR2(2));

SQL> INSERT INTO test_char VALUES (1, 'x','x');
```
(x 뒤에 공백 없이)

```
SQL> INSERT INTO test_char VALUES (2, 'x ','x ');
```
(x 뒤에 공백 하나씩)

```
SQL> SELECT * FROM test_char   WHERE   a = 'x';
```
 # (x 뒤에 공백 없이)
 # 수행 결과는 어떻게 되는가?

```
SQL> SELECT * FROM test_char   WHERE   a = 'x ';
```
(x 뒤에 공백 하나)
 # 수행 결과는 어떻게 되는가?

SQL> SELECT * FROM test_char WHERE b = 'x';
(x 뒤에 공백 없이)
 # 수행 결과는 어떻게 되는가?

SQL> SELECT * FROM test_char WHERE b = 'x ';
(x 뒤에 공백 하나)
 # 수행 결과는 어떻게 되는가?

NUMBER TEST

 # 다음 숫자가 입력이 되는지, 입력이 된다면 결과는 어떻게 나오는지 확인하시오.

SQL> CREATE TABLE test_num (a NUMBER (3), b NUMBER (3, 1));
SQL> insert into test_num values (100, 10);
SQL> insert into test_num values (100, 0.1);
SQL> insert into test_num values (100, 0.5);
SQL> insert into test_num values (100, 0.51);
SQL> insert into test_num values (100, 0.55);
SQL> select * from test_num;

SQL> insert into test_num values (999, 99);
SQL> insert into test_num values (999, 100);
SQL> insert into test_num values (99, 99.9);
SQL> insert into test_num values (99, 99.91);
SQL> insert into test_num values (99, 99.95);
SQL> insert into test_num values (1000, 10);
SQL> select * from test_num;

SQL> insert into test_num values (0.1, 10);
SQL> insert into test_num values (0.6, 10);
SQL> select * from test_num;

SQL> insert into test_num values ('A', 10);

==========
Create with Subquery

```
================================================
```

 # 다음 두 문장은 어떤 차이가 있나?

```
SQL> CREATE TABLE history1
            AS SELECT * FROM emp;
SQL> CREATE TABLE history2
            AS SELECT * FROM emp
        WHERE 1 = 0;

SQL> DESC history1;
SQL> DESC history2;

SQL> SELECT * FROM history1;
SQL> SELECT * FROM history2;
```

 # 테이블의 내용은 가져오지 않고, 테이블의 정의만 가져오려고 WHERE절에 항상 거짓이 되도록
 # 질의문을 작성하였다.

```
================================================
# Data Dictionary
================================================
```

 # 다음을 실습해 보시오.

```
SQL> SELECT * FROM user_users;
SQL> SELECT * FROM all_users;
SQL> SELECT * FROM dba_users;
SQL> SELECT * FROM dba_users;
```

 # 에러가 발생한 이유는 무엇일까?
 # 관리자로 접속해서 다시 수행해보자.

```
SQL> conn system
SQL> SELECT * FROM dba_users;
```

SQL> conn scott

 # DICTIONARY VIEW 찾기

```
SQL> SELECT table_name FROM dictionary
```

```
        WHERE table_name LIKE '%USER%';

SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE '%PRIV%';

SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE '%TABLE%'
        OR table_name LIKE '%COLUMN%';

SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE '%CONS%';

SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE '%AUDIT%';

SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE '%IND%';
```

```
================================================
# CREATE TABLE with Constraints
================================================
```

 # 테이블 생성: 다음이 의미하는 바는 무엇인가?
 # 테이블의 구조가 어떻게 되는지 확인하자.

```
SQL> CREATE TABLE book(
    id      NUMBER(5) CONSTRAINT book_id_pk PRIMARY KEY,
    name      VARCHAR2(20) CONSTRAINT book_name_not_null NOT NULL,
    price      NUMBER(12,2) CONSTRAINT book_price_check CHECK (price > 0),
    isbn      VARCHAR2(14) CONSTRAINT book_isbn_unique UNIQUE,
    pub_date DATE DEFAULT SYSDATE
     );

SQL> CREATE TABLE job(
    id      NUMBER(3) CONSTRAINT job_id_pk PRIMARY KEY,
    name      VARCHAR(5) NOT NULL
     );

SQL> CREATE TABLE author (
```

```
    id      NUMBER(5) CONSTRAINT author_id_pk PRIMARY KEY,
    name      VARCHAR2(20) CONSTRAINT author_name_not_null NOT NULL,
    gender      CHAR(1) DEFAULT 'M',
    age      NUMBER(2),
    job_id      NUMBER(3),
    CONSTRAINT author_gender_check CHECK (gender in ('M', 'F')),
    CONSTRAINT author_job_id_fk FOREIGN KEY (job_id) REFERENCES job(id)
     );

SQL> CREATE TABLE author_book (
    author_id      NUMBER(5),
    book_id            NUMBER(5),
    author_order      NUMBER(2) DEFAULT 1,
    CONSTRAINT authorbook_author_id_fk FOREIGN KEY (author_id) REFERENCES author(id) ON
DELETE CASCADE,
    CONSTRAINT authorbook_book_id_fk FOREIGN KEY (book_id) REFERENCES book(id) ON
DELETE CASCADE,
    CONSTRAINT authorbook_pk PRIMARY KEY (book_id, author_id, author_order)
     );
```

```
================================================
# 생성된 제약조건을 확인해보자.
================================================
```

```
# 왜 안나올까?
SQL> SELECT constraint_name, constraint_type,
    search_condition
    FROM user_constraints
    WHERE table_name = 'book';
```

 # DICTIONARY를 검사할때는 테이블 등의 이름은 항상 대문자로!!!

 # 제약조건의 이름과 종류, 검색 조건만 나온다.

```
SQL> SELECT constraint_name, constraint_type,
    search_condition
    FROM user_constraints
    WHERE table_name = 'BOOK';
```

 # 어느 컬럼에 걸렸는지만 나온다.

```
SQL> SELECT constraint_name, column_name
     FROM user_cons_columns
     WHERE table_name = 'BOOK';
```

 # 조인을 하면 원하는 것을 얻을 수 있다.

```
SQL> SELECT uc.constraint_name, uc.constraint_type, ucc.column_name, uc.search_condition
     FROM user_constraints uc, user_cons_columns ucc
     WHERE uc.constraint_name = ucc.constraint_name and uc.table_name = 'BOOK';
```

 # 다른 테이블도 확인해보자

```
SQL> SELECT uc.constraint_name, uc.constraint_type, ucc.column_name, ucc.position,
uc.search_condition
     FROM user_constraints uc, user_cons_columns ucc
     WHERE uc.constraint_name = ucc.constraint_name and uc.table_name = '&table_name';
```

(* SQL*PLUS에서는 &를 사용하면 값을 입력받도록 할 수 있다.)

```
SQL> /                    <-- 계속 수행해보자. 이름을 바꿔서 입력해보자
SQL> /
```

(* JOB 테이블에는 시스템이 지정한 이름을 지닌 제약조건이 있음을 확인해보자.)

```
================================================
# 생성한 테이블의 동작 테스트
================================================
```

```
SQL> INSERT INTO book VALUES (1, 'C++ INTRO', 20000, '15-222-22222', '07/01/02');
SQL> INSERT INTO book VALUES (2, 'JAVA PRIMER', 50000, '11-111-1111', DEFAULT);
SQL> SELECT * FROM book;
```

 # 다음은 수행될 수 있나? 안된다면 각각 어떠한 제약조건에 걸리는지 확인해보자.

```
SQL> INSERT INTO book VALUES (1, 'TEST BOOK', 50000, '33-333-3333', DEFAULT);
SQL> INSERT INTO book VALUES (3, NULL, 50000, '33-333-3333', DEFAULT);
SQL> INSERT INTO book VALUES (3, 'TEST BOOK', 50000, '11-111-1111', DEFAULT);
SQL> INSERT INTO book VALUES (3, 'TEST BOOK', -100, '33-333-3333', DEFAULT);
SQL> INSERT INTO book VALUES (3, 'TEST BOOK', 30000, '33-333-3333', NULL);
SQL> INSERT INTO book VALUES (4, 'TEST BOOK', 30000, '33-333-3333', NULL);
```

SQL> SELECT * FROM book;

 # UNIQUE컬럼에 NULL은 여러행이 있을 수 있나?

SQL> INSERT INTO book VALUES (4, 'TEST BOOK', 30000, NULL, DEFAULT);
SQL> INSERT INTO book VALUES (5, 'TEST BOOK', 30000, NULL, DEFAULT);
SQL> SELECT * FROM book;

 # VARCHAR(5)에 담을 수 있는 문자열의 길이는? 한글일때는 어떻게 다를까?

SQL> INSERT INTO job VALUES (1, '작가');
SQL> INSERT INTO job VALUES (2, '소설가');
SQL> INSERT INTO job VALUES (2, '교수');
SQL> INSERT INTO job VALUES (3, 'CEO');
SQL> INSERT INTO job VALUES (4, 'CEO12');
SQL> INSERT INTO job VALUES (5, 'CEO123');
SQL> SELECT * FROM job;

 # 다음은 수행될 수 있나? 안된다면 각각 어떠한 제약조건에 걸리는지 확인해보자.

SQL> INSERT INTO author VALUES (1, 'KIM', 'M', 30, 1);
SQL> INSERT INTO author VALUES (2, 'LEE', 'A', 40, 2);
SQL> INSERT INTO author VALUES (3, 'CHOI', NULL, 40, 3); **<-- NULL인경우 check는?**
SQL> INSERT INTO author VALUES (4, 'PARK', 'F', 50, NULL);
SQL> INSERT INTO author VALUES (5, 'KWON', 'M', 33, 5);
SQL> SELECT * FROM author;

SQL> INSERT INTO author_book(author_id, book_id) VALUES (3, 1);
SQL> INSERT INTO author_book VALUES (1, 1, 2);
SQL> INSERT INTO author_book VALUES (2, 4, DEFAULT);
SQL> INSERT INTO author_book VALUES (3, 2, DEFAULT);
SQL> INSERT INTO author_book VALUES (6, 1, 1);

```
================================================
# 제약조건 추가
================================================
```

SQL> ALTER TABLE author_book
    ADD CONSTRAINT authorbook_bookorder_unique UNIQUE (book_id, author_order);

SQL> INSERT INTO author_book VALUES (1, 4, 2);
SQL> INSERT INTO author_book VALUES (1, 4, 3);

SQL> UPDATE author_book SET author_id = 1 WHERE book_id = 1 AND author_order = 3;

 # 다음 제약조건이 추가되는가? 왜 안되는가?

SQL> SELECT * FROM author_book;

SQL> ALTER TABLE author_book
ADD CONSTRAINT authorbook_bookauthor_unique UNIQUE (book_id, author_id);

SQL> SELECT * FROM author_book;

 # 중복된 행을 찾아서 수정하자.

SQL> SELECT book_id, author_id, count(*) as cnt FROM author_book GROUP BY book_id, author_id HAVING count(*) > 1;

SQL> UPDATE author_book SET author_id = 4 WHERE book_id = 1 AND author_order = 3;

SQL> SELECT * FROM author_book;

SQL> ALTER TABLE author_book
ADD CONSTRAINT authorbook_bookauthor_unique UNIQUE (book_id, author_id);

SQL> SELECT * FROM author_book;

```
================================================
# DELETE 수행시
================================================
```

 # 다음 delete가 잘 수행되는가? 안되는 이유는 무엇인가?

SQL> SELECT * FROM job;
SQL> delete from job where id = 2;
SQL> delete from job where id = 3;
SQL> SELECT * FROM job;

 # BOOK이나 AUTHOR 테이블의 행이 삭제될때 author_book 테이블은 어떻게 변하는가?
 # 이유는 무엇일까?

SQL> SELECT * FROM author_book;
SQL> delete from author where id = 4;

```
SQL> SELECT * FROM author_book;
SQL> delete from book   where id = 1;
SQL> SELECT * FROM author_book;
```

```
================================================
# TRUNCATE TABLE
================================================
```

```
SQL> SELECT * FROM TEST_NUM;
SQL> TRUNCATE TABLE TEST_NUM;
SQL> SELECT * FROM TEST_NUM;

SQL> SELECT * FROM job;
SQL> TRUNCATE TABLE job;
```

 # 수행되지 않는 이유는 무엇일까?

```
SQL> ALTER TABLE author
     DROP CONSTRAINT author_job_id_fk;
SQL> TRUNCATE TABLE job;
SQL> SELECT * FROM job;
```

```
================================================
# DROP TABLE
================================================
```

```
SQL> DROP TABLE job;
SQL> DROP TABLE author;
```

 # 에러가 나는 이유는 무엇인가?

```
SQL> DROP TABLE author CASCADE CONSTRAINTS;
SQL> DROP TABLE author_book;
SQL> DROP TABLE book;

SQL> DROP TABLE test_num;
SQL> DROP TABLE test_char;
SQL> DROP TABLE cre_tab1;
SQL> DROP TABLE "Cre_tab2";
```