# Java Web Programming 입문 16

(Oracle PL/SQL #08)

# 오늘의 키워드

- TCL (Transaction Control Language)

# TCL (Transaction Control Language)

- Transaction
  - 실행된 연속적인 SQL문의 집합
  - 일련의 작업처리를 위해 연관된 DML 들은 TCL 에 의해 Table에 반영됨
  - DML문은 여러 개의 문장이 하나의 Transaction 으로 생성
  - DDL문과 DCL문은 하나의 문장이 하나의 Transaction
- Transaction의 시작
  - Database 접속직후
  - 한 개 이상의 DML 실행 후 TCL이 실행된 직후
- Transaction의 종료
  - TCL (COMMIT, ROLBACK) 실행될 때
  - DDL, DCL문이 실행될 때
  - (비)정상적으로 Database가 종료될 때

# TCL (Transaction Control Language)

- A은행 : 천만원, B은행 : 백만원
  - A은행에서 B은행으로 백만원을 이체

```
SQL> UPDATE A계좌_Table
        SET 잔액_Column = 9,000,000
```

```
SQL> UPDATE A계좌_Table
        SET 잔액_Column = 9,000,000
      WHERE 계좌번호_Column = XXXX-XXXX-XXXXXX
```

```
SQL> UPDATE 계좌_Table
        SET 잔액_Column = 2,000,000
      WHERE 계좌MP_Column = YYYY-YYYY-YYYYYY
```

# TCL (Transaction Control Language)

- All or Nothing!
  - All : COMMIT
  - Nothing : ROLLBACK

- Auto?

| Auto COMMIT | Auto ROLLBACK |
| --- | --- |
| DDL,DCL<br>EXIT 명령 사용과같은 정상종료 | 비정상종료 |

# TCL (Transaction Control Language)

- Session
  - Database 접속 후 여러 SQL문을 실행하고 종료하게 되는 전체 단계
  - 한 명의 사용자가 여러 개의 윈도우를 통해 여러 번 Database에 접속하면 여러 개의 Session 을 사용
  - 원격접속으로 동일 계정을 사용하여 여러 곳에서 접속을 한다면 각각의 접속은 Session이 다름

# TCL (Transaction Control Language)

- COMMIT
  - 사용자가 실행한 SQL문의 결과를 영구적으로 반영
  - COMMIT이 제대로 실행되지 않으면 작업은 취소됨

| Session1 | Session2 |
| --- | --- |
| 1) SQL> SELECT * FROM DEPT; | |
| 2) SQL> INSERT INTO DEPT<br>VALUES (11, '전산과', '서울'); | |
| 3) SQL> SELECT * FROM DEPT; | |
| | 4) SQL> SELECT * FROM DEPT; |
| 5) SQL> COMMIT; | |
| | 6) SQL> SELECT * FROM DEPT; |

# TCL (Transaction Control Language)

- ROLLBACK
  - 사용자가 실행한 SQL 문의 결과를 취소
  - ROLLBACK 범위가 정해져 있지 않다면, 이전 Transaction 직전까지 작업 취소

```
SQL> DELETE FROM DEPT;
```

```
SQL> SELECT * FROM DEPT;
```

```
SQL> ROLLBACK;
```

```
SQL> SELECT * FROM DEPT;
```

# TCL (Transaction Control Language)

- Transaction 처리

```
SQL> UPDATE EMP
        SET SAL = 100
      WHERE ENAME = 'SCOTT'
```

  - 위 UPDATE문이 실행되면

    - 본래  SAL 값에 저장되어있는 값이 Rollback Segment(Undo Segment) 복사
    - 실제 Table 에 Data를 변경
    - COMMIT 이 실행되면 Transaction 종료
    - ROLLBACK이 실행되면 복사해둔 본래 값을 가져와서 덮어쓴 후 Transaction 종료

# TCL (Transaction Control Language)

- SAVEPOINT : 어디까지 돌려?

```
SQL> INSERT INTO DEPT VALUES (1, 'A', 'B'); -- 첫번째 DML 실행
SQL> SAVEPOINT A1;                          -- 이곳에 Savepoint 지정
SQL> INSERT INTO DEPT VALUES (2, 'B', 'C'); -- 두번째 DML 실행
SQL> SAVEPOINT B1;                          -- 이곳에 Savepoiint 지정
SQL> INSERT INTO DEPT VALUES (3, 'C', 'D');  -- 세번째 DMl 실행 (잘못 넣은 값)
SQL> ROLLBACK TO B1; -- B1 까지 돌아간다. B1 바로 뒤
SQL> ROLLBACK TO A1; -- A1 까지 돌아간다. A1 바로 뒤
SQL> COMMIT;
```

- 이 산이 아닌가벼 – O, 아까 그 산이 맞는가벼 – X

# TCL (Transaction Control Language)

- 읽기 일관성 (Read Consistency)

```
Session1                                        Session2
-----------------------------------------------------------------------
1) SQL> SELECT * FROM DEPT;

2) SQL> INSERT INTO DEPT
        VALUES (11, '전산과', '서울');

3) SQL> SELECT * FROM DEPT;

                                        4) SQL> SELECT * FROM DEPT;

5) SQL> COMMIT;

                                        6) SQL> SELECT * FROM DEPT;
```

  - Data의 검색이 일관되게 해줌
  - DML 문장이 실행될 때 ROLLBACK을 위해 Oracle Server가 변경 전후 Data를 모두 보관
  - Table을 직접 변경하고 있는 Session에서만 변경중인 Data 확인 가능
    -> 타 Session 에서는 변경 중인 Data 확인 불가
  - 한 Session에서 변경중인 Row는 다른 Session에서 변경이 불가
    (변경중인 Session에서 COMMIT, ROLLBACK 이 실행후에 변경 가능)

# TCL (Transaction Control Language)

- LOCK
  - DML문 실행 시 해당 Transaction에 의해 발생한 Data가 다른 사용자에 의해 변경이 발생하지 않도록 접근을 보류시킴 (화장실!)
  - COMMIT, ROLLBACK 문이 실행되면 해제

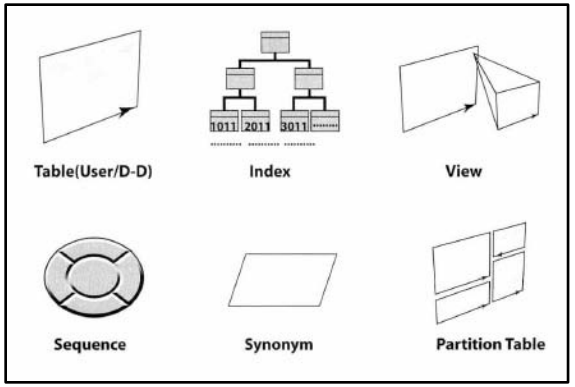
```
Session1                                        Session2
-----------------------------------------------------------------------
1) SQL> SELECT * FROM DEPT;

2) SQL> INSERT INTO DEPT
        VALUES (13, '전산과', '서울');

3) SQL> SELECT * FROM DEPT;

                                        4) SELECT * FROM DEPT;

                                        5) INSERT INTO DEPT
                                           VALUES (13, '전산과', '서울');

6) SQL> COMMIT;

                                        7) 5) INSERT 문이 실행됨

8) SELECT * FROM DEPT
```

# TCL (Transaction Control Language)

- LOCK
  - Row-Level Lock
    - 어떤 Session에서 실행되고 있는 DML문에 의해 변경이 진행중인 Row 에 Lock 이 발생. Row-Level Lock은 변경이 진행중인 Row가 완료될때까지 보호
    - Session, Transaction 예제
  - Table-Level Lock
    - 변경중인 Data가 Table 전체일 경우 Table-Level Lock 발생
    - Table 전체 DELETE

# Object

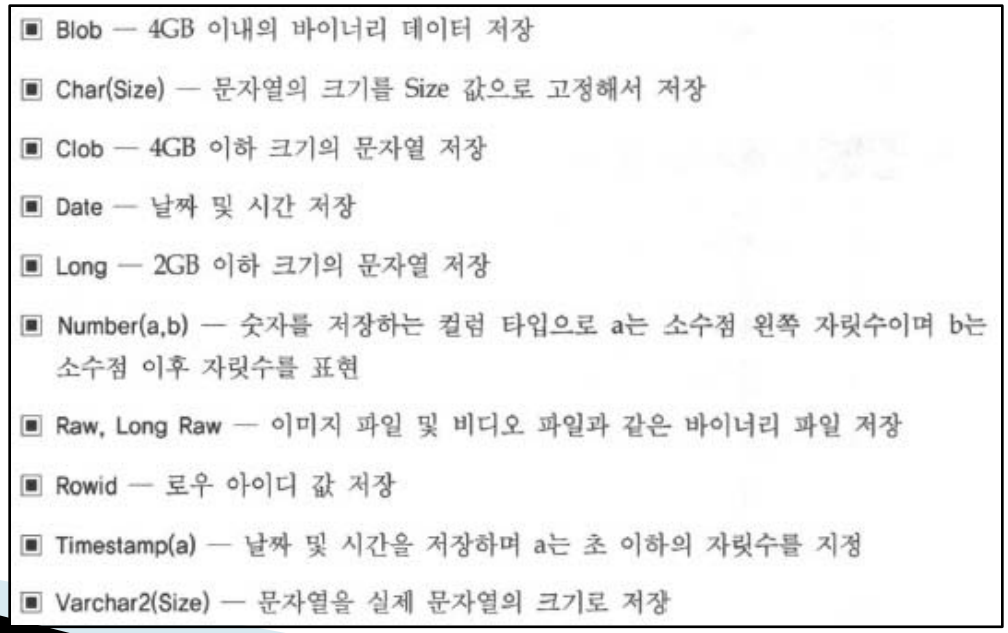
- Table
  - Data가 저장되는 Object
- Index
  - Table내 Data를 빠르게 검색하기 위해 사용되는 Object
  - 책에서의 목차, 색인과 동일한 기능




- View
  - 자주 참조하는 Data의 SELECT문을 미리 생성하여 검색속도를 향상시키는 Object
  - 물리적 공간을 추가로 차지하지 방식이 아님
- Sequence
  - 번호 생성기. 연속적인 일련번호를 자동으로 생성해주는 Object
- Synonym
  - Object의 별명
  - Table, Index 등의 이름이 너무 길어 SQL문 실행, 변경이 불편할 때 해당 Object를 지칭하는 또 하나의 이름을 지정해줄 수 있음

# Object

- Table
  - Database를 생성할 때 기본적으로 만들어지는 Data Dictionary Table 과 사용자의 Data를 저장/관리 하기 위해 생성되는 User Table 두 가지
  - Column
    - Table에 저장될 Data의 특성을 지정하는 Table 구성요소
  - Row
    - Column에 정의된 형식으로 저장되는 Data 한 건 한 건
  - Column Data Type



# Object

- Table
  - CHAR VS VARCHAR2
    - CHAR : 고정형 (2000)
    - VARCHAR2 : 가변형 (4000)
    - VARCHAR : 가변형 (거의 사용하지 않음)
    - 확인해보자

```
CHAR (8)
----------------------
 A | B | C | | | | | |
----------------------
```

```
VARCHAR2 (8)
-----------
 A | B | C
-----------
```

```
SQL> CREATE TABLE COLTEST
         (ID1 CHAR(3),
          ID2 VARCHAR2(3));
```

```
SQL> DESC COLTEST
```

```
SQL> INSERT INTO COLTEST
     VALUES ('A', 'A');
```

```
SQL> SELECT *
       FROM COLTEST
```

```
SQL> SELECT VSIZE(ID1), VSIZE(ID2)
       FROM COLTEST;
```

    - 언제 쓸까?
      - 조인 조건 사용시 비교되는 Table Column이 CHAR 타입일 경우
      - Column 값이 자주 변경되어 Row Migration이 많이 발생할 경우
      - 고정된 크기의 문자만 저장하는 Column의 경우
      - 빼고 다 VARCHAR2 (사실 무조건 사용해도 상관없는 수준)

# Object

- Data Dictionary

```
SQL> SELECT    *
          FROM DICTIONARY
      ORDER BY TABLE_NAME
```

```
SQL> SELECT    *
          FROM DICT
      ORDER BY TABLE_NAME
```

  - DBA_xxxxx
    - DB 관리 목적으로 참조하는 Table (DBA)
  - USER_xxxxx
    - 현재 접속된 계정이 생성한 Object 정보
  - ALL_xxxxx
    - 현재 접속된 계정이 생성한 Object 정보
    - 다른 계정에서 생성한 Object중 현재 계정이 볼 수 있는 Object 정보
  - V$_xxxxx
    - DB의 성능 분석/통계 정보를 제공
    - X$ Table의 VIEW
  - X$_xxxxx
    - DB의 성능 분석/통계 정보를 제공
    - 일반 사용자 / DBA 모두 직접 참조는 불가

접두사 + 관심 KEYWORD(의 복수형)

# Object

- Data Dictionary

```
SQL> SELECT *
       FROM DICT;
```

```
-- CONNECT 명령으로 사용자 전환
-- SYS : DB최고 관리자 계정
SQL> CONNECT SYS/ORACLE AS SYSDBA
```

```
-- 현재 접속한 계정명
SQL> SHOW USER
```

```
-- DBA_USERS
-- 현재 DB에 접근가능한 사용자 목록
SQL> DESC DBA_USERS
```

```
SQL> SELECT USERNAME, PASSWORD
       FROM DBA_USERS;
```

```
-- Connect를 줄여서 CONN 으로 사용
SQL> CONN SCOTT/TIGER
```

```
-- 현재 접속해있는 'SCOTT'계정이 생성한 Table 정보
SQL> DESC USER_TABLES

SQL> SELECT TABLE_NAME
       FROM USER_TABLES;
```

# DDL (Data Definition Language)

- DDL (Data Definition Language)
  - 정의
    - Database의 Object 구조를 생성, 변경, 삭제하는 명령
  - CREATE
    - Database 내 모든 Object 생성시 사용

```
SQL> CREATE TABLE EMP_TEST
        (EMPNO CHAR(3),
        ENAME VARCHAR2(15));
```

  - ALTER
    - 생성된 Object 구조 변경 시 사용

```
SQL> ALTER TABLE EMP_TEST
        MODIFY VARCHAR2(20);
```

# DDL (Data Definition Language)

- DDL (Data Definition Language)
  - DROP
    - 생성된 Object 삭제 시 사용

```
SQL> DROP TABLE EMP_TEST
```

  - RENAME
    - 생성된 Object명 변경 시 사용

```
SQL> RENAME EMP_TEST
          TO EMP_TEST_1;
```

  - COMMENT
    - Table 생성 후, Table및 Column에 대한 설명(주석) 작성시 사용

```
SQL> COMMENT ON
       TABLE EMP_TEST
          IS '사원정보';
```

```
SQL> COMMENT ON
       COLUMN EMP_TEST.EMPNO
          IS '사원번호';
```

  - TRUNCATE
    - Table에 저장된 모든 Row 삭제 시 사용 (DELETE?)

```
SQL> TRUNCATE TABLE EMP_TEST;
```

# DDL (Data Definition Language)

- DDL (Data Definition Language)
  - ALTER Table_Column
    - Table 생성

```
SQL> CREATE TABLE COPY_EMP
          AS SELECT *
               FROM EMP
```

    - 생성된 Table에 Column 추가

```
SQL> ALTER TABLE COPY_EMP
        ADD HP VARCHAR2(10);
```

    - Column 명 변경

```
SQL> ALTER TABLE COPY_EMP
     RENAME COLUMN HP TO MP;
```

    - Column 속성 변경

```
SQL> ALTER TABLE COPY_EMP
     MODIFY MP VARCHAR2(20);
```

    - Column 삭제

```
SQL> ALTER TABLE COPY_EMP
     DROP COLUMN MP;
```

```
SQL> DESC COPY_EMP
```

# DDL (Data Definition Language)

- DDL (Data Definition Language)
  - TRUNCATE VS DELETE
    - DELETE : DML
    - TRUNCATE : DDL

```
SQL> TRUNCATE TABLE TEMP_TABLE
```

```
SQL> DELETE FROM TEMP_TABLE
```

    - DDL, DML 차이
    - 속도?

# DDL (Data Definition Language)

- DDL (Data Definition Language)
  - Table 명명 규칙
    - A~Z, a~z, 0~9, _, $, # 사용가능 (한글 가능)
    - 첫 글자는 영문자 (한글 가능)
    - 1 ~ 30자 까지 허용
    - 오라클 예약어 사용불가
    - 의미 있는 이름으로 (권장사항)
  - 다음 중 Table 명명이 올바른 것은?

```
SQL> CREATE TABLE ACUKSTUKUKS123
     ...(후략)

SQL> CREATE TABLE 9ACKSTUKVKST1369TK
     ...(후략)

SQL> CREATE TABLE AKSUKSCKS!!!
     ...(후략)

SQL> CREATE TABLE SELECT_FROM
     ...(후략)

SQL> CREATE TABLE ACKSTIIKSKSTKKKSTKTUKTAKTKTKTKKL
     ...(후략)
```

# DDL (Data Definition Language)

- Flashback
  - 오라클 내 휴지통 (쓸 일이 없는 게 제일 좋지만...)

```
SQL> DROP Table COPY_EMP -- Table을 지움
```

```
SQL> DESC COPY_EMP -- Desc 명령 안됨
```

```
SQL> ROLLBACK -- 헹여나 하는 맘으로 Rollback
```

```
SQL> DESC COPY_EMP -- 역시 안됨
```

```
SQL> SHOW RECYCLEBIN
```

```
SQL> FLASHBACK Table COPY_EMP
               TO BEFORE DROP
```

```
SQL> DESC COPY_EMP
```

```
SQL> SELECT * FROM COPY_EMP
```

# Pseudo Column

- 정의
  - Oracle Database에서 Table생성시 기본적으로 제공되는 Column
  - 대표적으로  ROWNUM, ROWID

- ROWNUM
  - SELECT Query시 명시된 Row의 순서번호

```
SQL> SELECT EMPNO, ENAME, ROWNUM
       FROM EMP;
```

- ROWID
  - ROW의 물리적 저장 주소 (형식 : 6,3,6,3)
  - Database 내 모든 Table에 모든 Row에 대한 유일 식별자
  - 어느 Data File, 어떤 Object, 어떤 Block, 몇 번째 Row 정보를 가짐
  - ROWID를 직접 지정해서 Row를 불러올 수도 있음
    - DB 내에서 Row를 불러오는 세상에서 가장 빠른 방법
    - INDEX 와 밀접하게 관련

```
SQL> SELECT EMPNO, ENAME, ROWID
       FROM EMP;
```

```
SQL> SELECT *
       FROM EMP
      WHERE ROWID = 'AAAR3sAAEAAAACXAAA'
```

# Pseudo Column

- EMP Table에서 급여가 많은 순으로 3명만 뽑고 싶다?
  - 급여가 많은 순

```
SQL> SELECT    ENAME, SAL
       FROM EMP
     ORDER BY SAL DESC;
```

```
SQL> SELECT EMPNO, ROWID, ROWNUM
       FROM EMP;
```

  - VIEW 를 생성해서?

```
SQL> CREATE VIEW E1
       AS SELECT    ENAME, SAL
            FROM EMP
          ORDER BY SAL DESC;
```

```
SQL> SELECT ENAME, SAL, ROWNUM
       FROM E1
      WHERE ROWNUM <= 3;
```

  - Sub-Query를 이용해서

```
SQL> SELECT ENAME, SAL, ROWNUM
       FROM (SELECT    ENAME, SAL
               FROM EMP
             ORDER BY SAL DESC)
      WHERE ROWNUM <= 3;
```