



## 03. DML

- 1) SQL 명령어 분류 와 DML
- 2) INSERT
- 3) UPDATE
- 4) DELETE
- 5) SQL Script file
- 6) N개 Session 생성
- 7) TRANSACTION

## ● 1. SQL 명령어 분류 와 DML

분류	대상	구문
Query	데이터	SELECT(조회)
DML(Data Manipulation Language)	데이터	INSERT(입력), UPDATE(수정), DELETE(삭제), MERGE(INSERT+UPDATE)
TCL(Transaction Control Language)	트랜잭션	COMMIT(저장), ROLLBACK(취소), SAVEPOINT(중간 저장점)
DDL(Data Definition Language)	Object	CREATE(생성), ALTER(변경), DROP(Object 삭제) , TRUNCATE(절삭)
DCL(Data Control Language)	권한	GRANT(부여), REVOKE(취소)

Google → [oracle insert syntax 21c](#) → INSERT(클릭) → SYNTAX(클릭)

## ● 2. INSERT

### ❑ INSERT

- \* 테이블에 새로운 행(Row , Record)삽입

### ❑ INSERT Syntax Diagram

#### Syntax

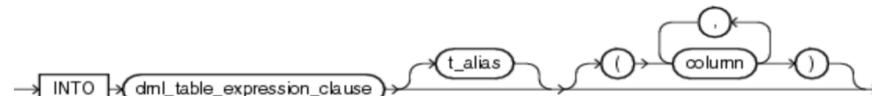
**insert ::=**



Description of the illustration insert.gif

*(single\_table\_insert ::=, multi\_table\_insert ::=)*

**insert\_into\_clause ::=**



Description of the illustration insert\_into\_clause.gif

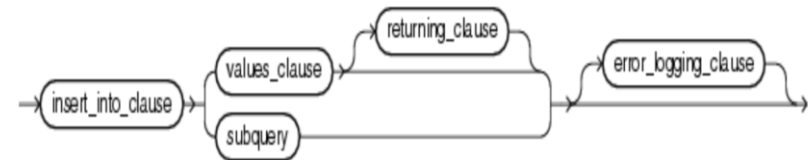
*(DML\_table\_expression\_clause ::=)*

**values\_clause ::=**



Description of the illustration values\_clause.gif

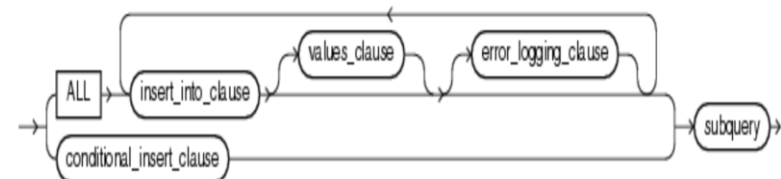
**single\_table\_insert ::=**



Description of the illustration single\_table\_insert.gif

*(insert\_into\_clause ::=, values\_clause ::=, returning\_clause ::=, subquery::=, error\_logging\_clause ::=)*

**multi\_table\_insert ::=**



Description of the illustration multi\_table\_insert.gif

*(insert\_into\_clause ::=, values\_clause ::=, conditional\_insert\_clause ::=, subquery::=, error\_logging\_clause ::=)*

## ● 2. INSERT

### ❏ INSERT

\* 테이블에 새로운 행(Row , Record)삽입

- ① INSERT INTO DEPT VALUES(50,'연구소1','서울'); // 컬럼명 생략시 전체컬럼대상
- ② INSERT INTO DEPT(DEPTNO,DNAME,LOC) VALUES(51,'연구소2','대전'); // 좋은방식의 SQL 작성은?  
- 컬럼명과 VALUE(값)을 1:1로 매핑, 테이블에 정의된 컬럼 순서 필요(X)
- ③ SELECT \* FROM DEPT; // 신규 삽입 데이터 조회
- ④ INSERT INTO DEPT VALUES('중부영업점','대구'); // ERROR의 이유는?  
- DESC DEPT → 3개 컬럼 정의 , 2개의 데이터만 삽입시 에러 발생
- ⑤ INSERT INTO DEPT(DNAME,LOC) VALUES('중부영업점','대구'); // 여전히 ERROR인 이유는?  
- ORA-12899: "SCOTT"."DEPT"."DNAME" 열에 대한 값이 너무 큼(실제: 15, 최대값: 14)  
- 한글 저장시 1글자당 3 Bytes 할당시 , 중부영업점은 **15 Bytes 할당**으로 컬럼의 최대 저장 길이 초과 (DNAME **VARCHAR2(14)** )
- ⑥ INSERT INTO DEPT(DNAME,LOC) VALUES('중부지점','대구');  
- INSERT시 생략된 컬럼(ex DEPTNO)은 NULL  
- DEPTNO NOT NULL 인경우 " ORA-01400: NULL을 ("SCOTT"."DEPT"."DEPTNO") 안에 삽입할 수 없습니다. " 발생  
  
// INSERT시 특정 COLUM에 NULL 삽입방법 (명시적)  
⑦ INSERT INTO DEPT(DEPTNO,DNAME,LOC) VALUES(52, '북부지점',NULL); // 'NULL' 과 다른점은?  
INSERT INTO DEPT(DEPTNO,DNAME,LOC) VALUES(53, '남부지점','');  
// INSERT시 특정 COLUM에 NULL 삽입방법 (암시적)  
⑧ INSERT INTO DEPT(DEPTNO,DNAME) VALUES(54,'서부지점'); // 대상 컬럼 생략  
⑨ SELECT DEPTNO,DNAME,NVL(LOC,'미지정지역') AS LOC FROM DEPT; // 결과 조회  
⑩ COMMIT; // **변경사항 DB에 영구히 반영**

## ● 2. INSERT

---

### ❑ Multi table INSERT

\* 테이블에 N행(Row , Record)삽입

#### ① INSERT ALL

```
INTO DEPT(DEPTNO,DNAME,LOC) VALUES(55,'Multi','INSERT1')
```

```
INTO DEPT(DEPTNO,DNAME,LOC) VALUES(56,'Multi','INSERT2')
```

```
INTO DEPT(DEPTNO,DNAME,LOC) VALUES(57,'Multi','INSERT3')
```

```
SELECT 1 FROM dual;
```

- 문법적으로 Subquery로 종료, 다양한 유형의 Subquery 응용 가능, INTO 이하에 서로 다른 테이블 사용 가능

```
SELECT * FROM DEPT;
```

// N개 Row 1 Insert

#### ② ROLLBACK;

```
SELECT * FROM DEPT;
```

// 변경사항 취소

### ❑ Conditional INSERT

#### ① CREATE TABLE BONUS\_2 AS SELECT \* FROM BONUS WHERE 1=2;

// Row=0인 빈 테이블 생성

```
DESC BONUS_2
```

```
SELECT * FROM BONUS_2;
```

#### ② INSERT ALL

```
WHEN COMM > 0 THEN INTO BONUS
```

// 조건에 따라 INSERT

```
WHEN COMM IS NULL THEN INTO BONUS_2
```

```
SELECT ename,job,sal,comm FROM emp WHERE job IN ('CLERK','SALESMAN');
```

#### ② SELECT \* FROM bonus;

```
SELECT * FROM bonus_2;
```

#### ③ ROLLBACK;

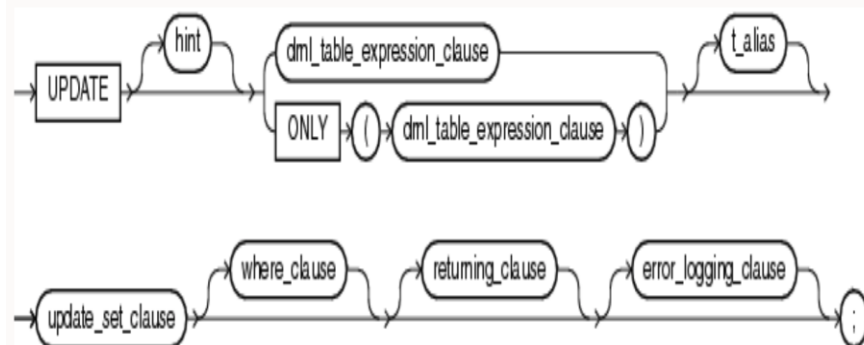
### ● 3. UPDATE

#### ❑ UPDATE

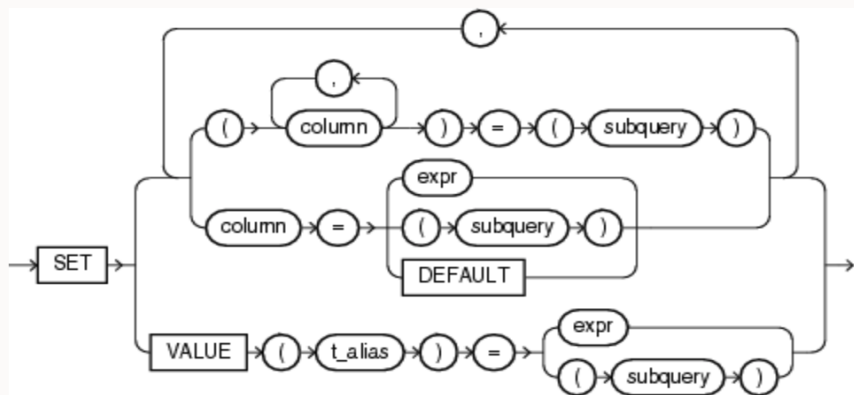
\* 테이블내에 저장된 기존 컬럼(들)(Column , Field) 수정

#### ❑ UPDATE Syntax Diagram

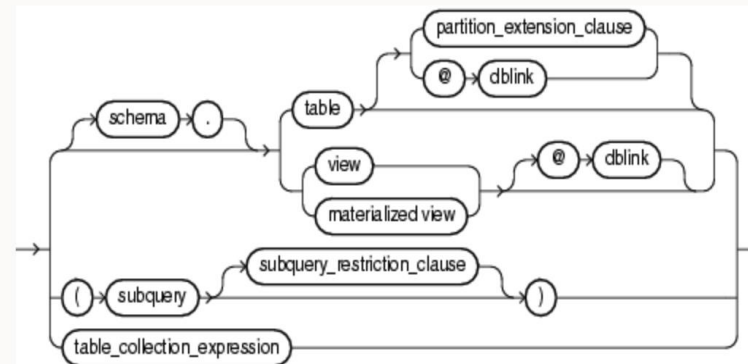
**update::=**



**update\_set\_clause::=**



**DML\_table\_expression\_clause::=**



**where\_clause::=**



### ● 3. UPDATE

---

// 조직 명칭 변경

50번 조직    연구소1 → M연구소

51번 조직    연구소2 → T연구소, 대전 → 인천 변경.

① UPDATE DEPT SET DNAME = ' M연구소' WHERE DEPTNO = 50;                      // 단일 컬럼 변경

② UPDATE DEPT SET DNAME = ' T연구소', LOC='인천' WHERE DEPTNO = 51;        // 복수 컬럼 변경

③ SELECT \* FROM DEPT WHERE DEPTNO IN (50,51);                                      // 변경내역조회

④ COMMIT;    // 변경사항 반영(저장)

⑤ UPDATE DEPT SET LOC='미개척지';    // WHERE절 생략시 전체 ROW 대상

⑥ SELECT \* FROM DEPT;

⑦ ROLLBACK;    // 해당 변경사항 영구히 취소

⑧ SELECT \* FROM DEPT;    // 결과 확인

⑨ select dname, replace(dname,' ','\*') from dept;                                      // dname 컬럼에 공백문자 저장

update dept set dname=**trim**(dname);    // set 절에 함수 사용가능

select dname, replace(dname,' ','\*') from dept;

⑩ commit;



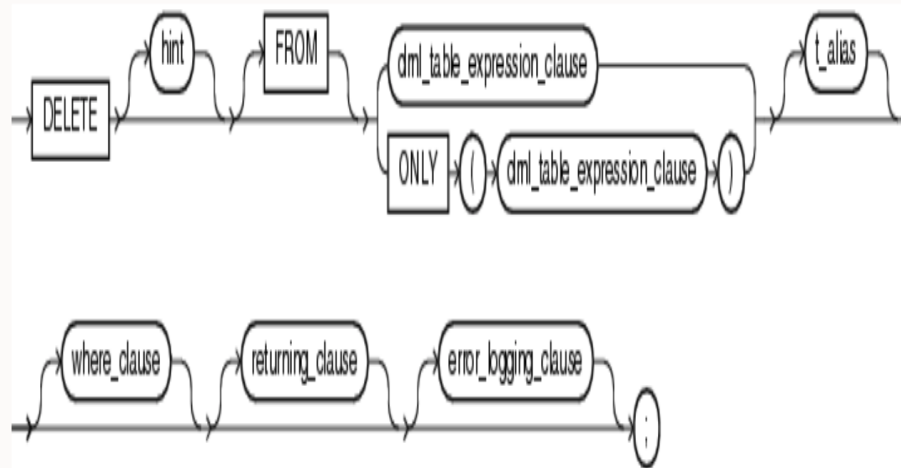
## ● 4. DELETE

### ❑ DELETE

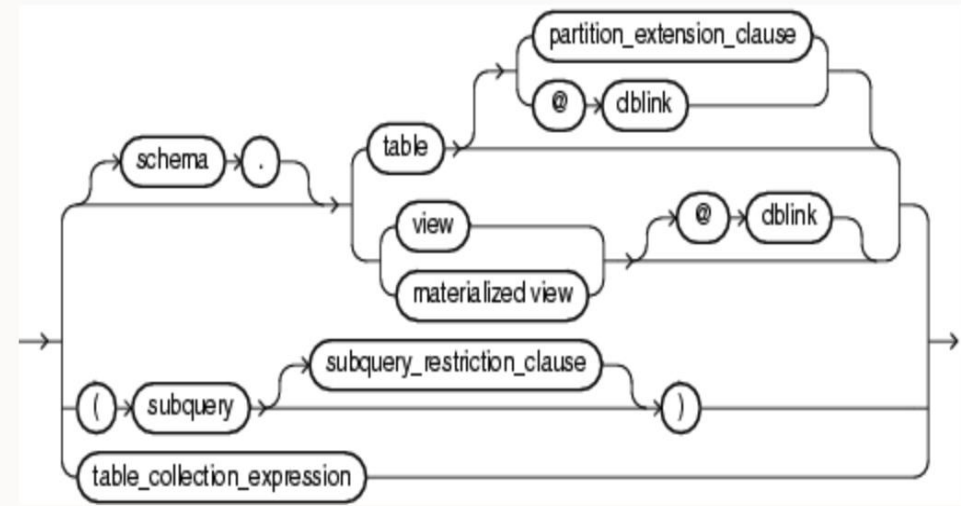
\* 테이블내에 저장된 기존 행(Row, Record) 삭제

### ❑ DELETE Syntax Diagram

***delete::=***



***DML\_table\_expression\_clause::=***



***table\_collection\_expression::=***



***where\_clause::=***



## ● 4. DELETE

---

// 미개척 지역을 폐쇄

① DELETE FROM DEPT WHERE **LOC IS NULL or DEPTNO IS NULL**;  
SELECT \* FROM DEPT;  
commit;

② DELETE DEPT;  
- FROM 생략 가능 , Delete Syntax Diagram을 읽어 확인  
SELECT \* FROM DEPT;

// **WHERE절 생략시 전체 ROW 삭제**

③ ROLLBACK;  
SELECT \* FROM DEPT;

// 해당 변경사항 취소

## ● 5. SQL Script file

### ❑ SQL Script file

- \* 텍스트 파일 포맷으로 저장되는 SQL 명령어 모음들
- \* 용도 : N개의 SQL 명령어들을 순차적으로 실행시 사용  
반복적으로 수행하는 작업에 사용

### ❑ SQL Script file 생성


#### 1) SQL-Developer에서 Script 파일 생성

파일 > 새로 만들기 > 데이터베이스 파일 > SQL 파일 > 확인

파일 이름 : test.sql

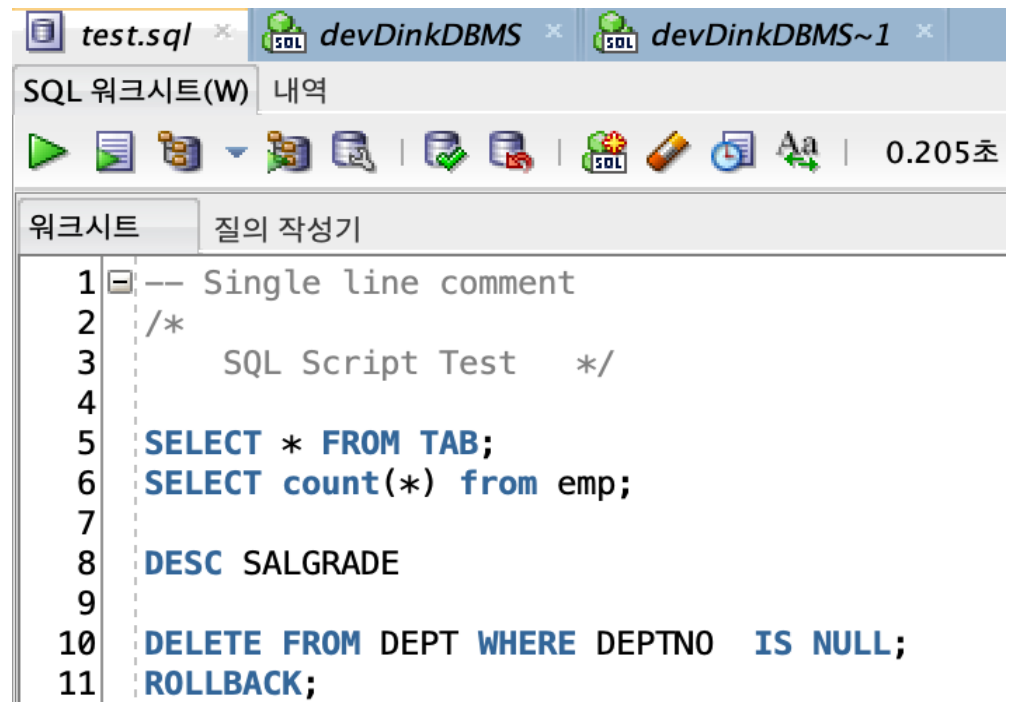
디렉토리 : c:\WSQLDEV

#### 2) SQL-Developer에서 Script 파일 실행

- SQL 명령어(들) 작성후 저장
-  스크립트 실행 버튼 클릭 or F5
- 접속선택 : devDinkDBMS 선택후 확인
- 5개 SQL 명령어가 순차적인 실행결과 확인
- 파일 > 닫기

#### 3) SQL-Developer에서 Script 파일 열기

- 파일 > 열기 > c:\WSQLDEV\test.sql



```
1 -- Single line comment
2 /*
3     SQL Script Test */
4
5 SELECT * FROM TAB;
6 SELECT count(*) from emp;
7
8 DESC SALGRADE
9
10 DELETE FROM DEPT WHERE DEPTNO IS NULL;
11 ROLLBACK;
```

## ● 6. N개 Session 생성

---

### ❑ SQL-Developer에서 N개 세션(Session) 생성

- \* 도구 > SQL 워크시트 > devDinkDBMS 선택 > 확인버튼 클릭
- \* 2번 반복
- \* 각각 SQL 워크시트에서 세션 ID 확인하는 SQL 실행

```
select sys_context('userenv','sid') from dual;
```

// 동일한 SID이면 **동일 세션 & 다른 SQL 워크시트**

- \* **비공유 SQL 워크시트 2번 클릭**



- \* 신규 생성된 2개의 **비공유 SQL 워크시트**에서 세션 ID 확인하는 SQL 실행

```
select sys_context('userenv','sid') from dual;
```

// 다른 SID이면 **다른 세션**

## ● 7. TRANSACTION

---

**“거래를 완료할 수 없습니다.”**

**“처리가 수행되지 않았습니다.”**

## ● 7. TRANSACTION

### □ Transaction

- \* 데이터베이스의 존재 목적은 데이터를 제공하고 데이터를 처리하는데 있으며 데이터베이스에서 **데이터를 처리하는 논리적인 단위**를 트랜잭션(Transaction)이라 한다.

사전적 의미	① (업무,거래 따위의) <b>처리</b> ② <b>거래</b> , 매매
데이터베이스	- <b>논리적 일의 단위(A logical unit of work)</b>  - 1개의 논리적인 일은 N개의 물리적인 행위(Activity)의 묶음으로 데이터베이스는 SQL을 통해서 일을 수행하고 행위(Activity)가 발생하므로 <b>일련의 SQL 묶음이 트랜잭션(Transaction)</b>

- \* 트랜잭션은 트리플 A이다.

- ① **A logical unit of work**
- ② **Atomic unit**
- ③ **All or Nothing**

논리적인 일의 단위로 SQL의 묶음으로 모든(ALL) SQL이 성공적으로 수행되던가 모든 SQL이 실행되지 않는(Nothing) 쪼갤수 없는 하나의 원자적 단위(Atomic Unit)로 처리된다

- \* 트랜잭션은 4가지 특징(특성)

- ① **일관성(Consistency)** ② **원자성(Atomicity)** ③ **고립성(isolation)** ④ **지속성(Durability)**

## ● 7. TRANSACTION

### ❏ Transaction 시작

#### \* 실행 가능(Executable)한 첫번째 SQL 실행시 시작

(A transaction begins with the first executable SQL statement)

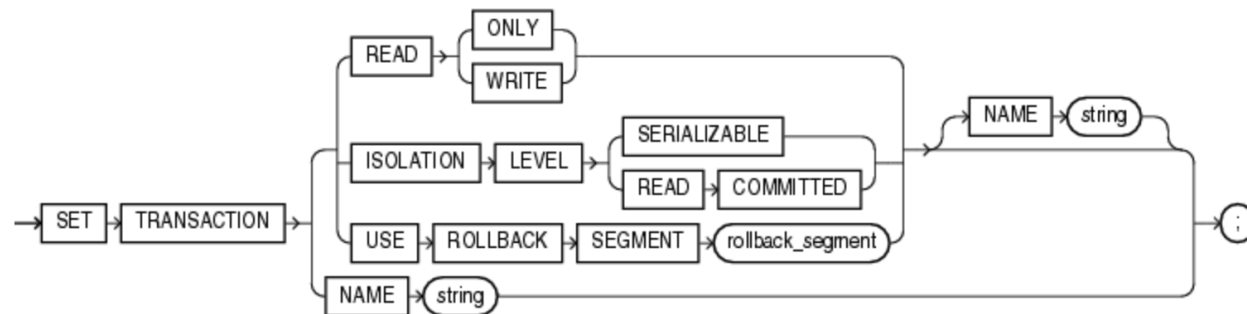
[참고1] 실행가능하지 않은 SQL은 없다. 여기서 의미는 **변경 가능한 SQL** 이라는 의미.

#### ① DML,DDL,DCL 실행시 Transaction 시작.

DML은 데이터(Data) 변경, DDL은 데이터베이스의 객체(Object) 변경, DCL은 권한(Privilege) 변경

#### ② SET TRANSACTION [ READ ONLY | READ WRITE];

**set\_transaction::=**



[참고2] MS SQL-SERVER는 BEGIN TRANSACTION 시작

## ● 7. TRANSACTION

### □ Transaction 종료

명시적 종료	<b>COMMIT, ROLLBACK</b>
암시적 종료	<p>① DDL,DCL 실행시</p> <ul style="list-style-type: none"> <li>- <b>DML은 N개 명령어 묶음이 1개의 트랜잭션 구성.</b></li> <li>- <b>DDL,DCL은 1개 명령어가 1개의 트랜잭션 구성.</b></li> </ul> <p>* DDL,DCL 명령어 시작시 트랜잭션을 시작하고 성공적으로 종료시에는 암시적(자동)으로 COMMIT을 실행하고 실패하면 암시적(자동)으로 ROLLBACK을 실행하여 트랜잭션 종료.</p> <p>② <b>비정상 종료시 자동으로 트랜잭션 ROLLBACK 수행</b></p> <ul style="list-style-type: none"> <li>- Client(ex SQL-Developer) 프로그램 비정상 종료</li> <li>- 네트워크 단절(Oracle DBMS 와 Client Program간)</li> <li>- DBMS 비정상종료(Instance Crash)</li> </ul>

### □ Transaction Control Language (제어 명령어)

<b>COMMIT</b>	트랜잭션 시작이후 발생한 모든 변경사항을 데이터베이스에 영구히 저장(지속성 Durability) 하고 LOCK 해제.
<b>ROLLBACK</b>	트랜잭션 시작 이전의 상태로 되돌린다. 트랜잭션 진행중 발생한 모든 입력/수정/삭제 모든 변경사항을 취소하고 LOCK 해제
SAVEPOINT	현재 시점부터 저장점(SAVEPOINT) 까지 트랜잭션의 일부만 ROLLBACK 할수 있도록 트랜잭션내 특정 위치에 대한 저장점(Savepoint)으로 복잡한 대규모 TRANSACTION에서 에러가 발생시 부분 ROLLBACK을 지원하기 위한 기능



## ● 7. TRANSACTION

### ❑ TRANSACTION 실습 - 시작과 종료

T.S	INSERT INTO DEPT(DEPTNO,DNAME,LOC) VALUES(90,'신사업부','경기도');		트랜잭션 시작
	UPDATE EMP SET DEPTNO = 90 WHERE DEPTNO = 30;	2	트랜잭션 진행중
	DELETE FROM DEPT WHERE DEPTNO = 30;	3	트랜잭션 진행중
	SELECT * FROM DEPT;		변경 진행중인
	SELECT * FROM EMP WHERE DEPTNO = 30;		데이터 조회 ①
T.E	ROLLBACK;		트랜잭션 종료
	SELECT * FROM DEPT; SELECT * FROM EMP		Rollback 취소범위확인

#### ① 트랜잭션의 고립성(isolation)

트랜잭션 진행중에 변경된 데이터는 취소(ROLLBACK)나 저장(COMMIT)될수 있는 데이터이기 때문에 트랜잭션을 수행중인 현재 세션(Session)에서는 조회가 가능하고 다른 세션에서는 변경이 진행중인 데이터를 조회할수 없고 변경 이전 상태의 데이터 조회가 가능하다. 비공유 SQL 워크시트를 생성후

① SQL을 실행하면 변경이 진행중인(Dirty) 데이터가 보이지 않는다.

INSERT INTO EMP(EMPNO,ENAME,JOB,SAL)VALUES (1111,'ORACLE','DBA',3500);	트랜잭션 시작
UPDATE EMP SET SAL = SAL* 1.3 WHERE EMPNO= 1111;	트랜잭션 진행중
COMMIT;	트랜잭션 종료
ROLLBACK;	트랜잭션 종료 ②
SELECT * FROM EMP;	데이터 조회

② COMMIT을 통해 트랜잭션에서 발생한 모든 변경 사항을 데이터베이스에 저장(반영) 하고 트랜잭션을 종료해서 ROLLBACK 적용 대상이 없다.

## ● 7. TRANSACTION

### ❑ TRANSACTION 실습 – 문장 범위 ROLLBACK(Statement Level ROLLBACK)

ROLLBACK;	이전 트랜잭션 종료
DELETE FROM EMP WHERE EMPNO = 1111;	트랜잭션 시작
UPDATE EMP SET SAL = 123456789 WHERE EMPNO = 7788	진행중, 에러발생 ①
UPDATE EMP SET SAL = 1234 WHERE EMPNO = 7902;	실행 or SKIP ?
COMMIT;	트랜잭션 종료②
SELECT EMPNO,SAL FROM EMP WHERE EMPNO IN (1111,7788,7902);	트랜잭션 일부만 반영

① EMP.SAL NUMBER(7,2) 전체 7자리,정수 5자리, 소수점 이하 2자리  
9자리(123456789) 데이터 입력시 **자리수 초과 에러 발생**.

-1행에 오류: ORA-01438: 이 열에 대해 지정된 전체 자릿수보다 큰 값이 허용됩니다.

**DML 명령어 실행시 에러가 발생하면 해당 명령어에 의해 변경된 데이터만 자동으로 ROLLBACK 되는 현상이 Statement Level ROLLBACK**

② COMMIT의 적용 대상은 어디까지 ?

ⓐDELETE(성공) ⓑUPDATE(실패) ⓒUPDATE(성공)

SELECT로 결과를 확인해보면 ⓐ ⓒ 만 적용 되었다. ⓑ에서 에러가 발생해서 트랜잭션의 원자성( All or Nothing이 적용되어 전부 취소 처리가 되어야 하는데 ⓐⓒ만 성공처리

➔ **트랜잭션의 원자성(All or Nothing)을 보장하려면 어떻게 해야 하는가??**

## ● 7. TRANSACTION

### ❑ TRANSACTION 실습 – 트랜잭션 범위 ROLLBACK(Transaction Level ROLLBACK)

BEGIN	PL/SQL Block 시작
DELETE FROM EMP WHERE DEPTNO = 20;	트랜잭션 시작 ①
UPDATE EMP SET SAL = 123456789 WHERE EMPNO = 7499;	에러발생 ②
UPDATE EMP SET SAL = 1234 WHERE EMPNO = 7698;	실행 SKIP ③
COMMIT;	실행 SKIP ④
EXCEPTION	예외처리부 ⑤
WHEN OTHERS THEN	
ROLLBACK;	⑤ 트랜잭션 레벨 ROLLBACK
END;	PL/SQL Block 종료
/	Block 실행(Send to DBMS)
SELECT EMPNO,SAL FROM EMP WHERE DEPTNO = 20 or EMPNO IN (7499,7698);	결과확인

\* PL/SQL의 예외처리(EXCEPTION) 기능을 사용하여 트랜잭션의 원자성 보장.  
프로그래밍 언어의 예외처리(에러처리) 기능을 사용해야 올바른 트랜잭션 제어

- ② 정의된 자리수를 초과하는 **예외(런타임 에러)** 발생
  - ➔ ③ ④ 을 실행하지 않고 예외처리(EXCEPTON)영역으로 실행 흐름(제어)을 분기(이동)
- ⑤ 예외처리부내에서 예외의 종류에 따라 분기 위에서는 OTHERS로 분기.
- ⑥ 트랜잭션내에서 발생한 모든 변경 결과(①②)를 ROLLBACK 처리

[참고] 데이터베이스 프로그램 개발시 트랜잭션을 처리 하기 위해서는 해당 프로그래밍 언어의  
예외처리(에러처리)기능 사용 필요

- PL/SQL : EXCEPTION 구문
- JAVA : try ~ catch ~ finally 구문

## ● 7. TRANSACTION

try{

~ 생략

```
String insstr= "INSERT INTO BONUS_LARGE(YYYYMM,EMPNO,JOB,DEPTNO,SAL,BONUS)" +  
              " VALUES(?,?,?,?,?,?)";
```

```
stmt.executeQuery(sqlstr);
```

~ 생략

```
while(rs.next()){      // 250 만번 WHILE LOOP 반복
```

```
    // [6단계] 부서별 보너스 계산
```

```
    if (rs.getInt(3) == 10 )
```

```
        Bonus = (int)(rs.getInt(4) * 0.3);
```

```
    else if(rs.getInt(3) == 20)
```

```
        Bonus = (int)(rs.getInt(4) * 0.1);
```

```
    else if(rs.getInt(3) == 30)
```

```
        Bonus = (int)(rs.getInt(4) * 0.05);
```

```
    else if(rs.getInt(3) == 40)
```

```
        Bonus = (int)(rs.getInt(4) * 0.2);
```

```
    else
```

```
        Bonus = 0;
```

```
    pstmt_ins.executeUpdate();      // INSERT 실행
```

```
}
```

```
conn.commit( );                  // 250만건 데이터 입력후 1번 commit  
                                / 종료 시간 측정
```

```
}catch(Exception e){
```

```
    conn.rollback( );             // Exception(Runtime Error) 발생시 rollback
```

```
    System.out.println("Oracle JDBC Exception");
```

```
    e.printStackTrace();
```

```
}finally{
```

```
    // [8단계] Resource 반납
```

```
    if(rs != null)    try{ rs.close();    } catch(SQLException sqle){}
```

```
    if(stmt != null)  try{ stmt.close();   } catch(SQLException sqle){}
```

```
    if(pstmt_ins != null) try{ pstmt_ins.close(); } catch(SQLException sqle){}
```

```
    if(conn != null)  try{ conn.close();   } catch(SQLException sqle){}
```

```
} // End try
```

## ● 7. TRANSACTION

---

### ❏ TRANSACTION 과 DDL

- |  |                      |
|--|----------------------|
| ① INSERT INTO EMP(EMPNO,ENAME,DEPTNO) VALUES(9999,'OCPOK',20); | // TRANSACTION START |
| ② ALTER TABLE EMP ADD( SEX CHAR(1) DEFAULT 'M');               | // DDL               |
| ③ ROLLBACK;  | // 취소 범위는?           |
| ④ DESC EMP;  |                      |
| ⑤ ALTER TABLE EMP DROP COLUMN SEX;                             | // DDL               |
| ⑥ ROLLBACK;  | // 취소 범위는?           |
| ⑦ DESC EMP   |                      |

## ● 7. TRANSACTION

❑ 트랜잭션(Transaction) 과 읽기 일관성(Read Consistency)

- \* **트랜잭션 고립성(Isolation)에 의해 변경이 진행중인 불안정한 상태의 데이터를 볼수없고(Dirty Read) 항상 안정적인 상태의 데이터를 조회하는 특성**
- \* **isolation: 실행중인 트랜잭션의 중간결과를 다른 트랜잭션이 접근할수 없는 성질**
- \* SQL-Developer에서 **비공유 SQL 워크시트** 생성하여 **2개의 세션**에서 실행

Session 1(첫번째 SQL 워크시트 )	Session 2(두번째 SQL 워크시트 )
① update emp set sal=0 where deptno= 10;	
	② select deptno,sal from emp where deptno = 10;
③ select deptno,sal from emp where deptno = 10;	
④ commit;	
	⑤ select deptno,sal from emp where deptno = 10;
⑥ select deptno,sal from emp where deptno = 10;	

- ② 트랜잭션의 고립성(Isolation)에 따라 변경 진행중인 데이터는 변경을 하는 세션에서만 접근 가능하여 Session2는 변경 진행중인 데이터를 볼수 없고 변경전 데이터(Before Image) 조회
- ③ Session1은 변경 진행중인 데이터 조회한다(sal = 0)
- ④ commit 이후 변경이 완료된 데이터를 다른 모든 세션에서 조회 가능
- ⑤ Session2에서 변경 완료된 데이터 조회 (sal = 0)
- ⑥ Session1에서 변경 완료된 데이터 조회 (sal = 0)

## ● 7. TRANSACTION

### ❑ 트랜잭션(Transaction) 과 Row Level Lock

\* DBMS에서는 동시 사용성 과 데이터 공유성을 지원하기 위해 Lock 매커니즘 사용.

Lock은 직렬성(Serialization)을 보장하는 기법중 하나로 다중 트랜잭션 수행시 데이터 무결성 보장.

\* Lock 생성: 트랜잭션 진행시 변경 대상 Row에 자동으로 Exclusive Row Level Lock생성후 데이터 변경.

Lock 해제: 트랜잭션 종료시(명시적,암시적) 해제

Lock의 범위에 따라 Row Level Lock 과 Table Level Lock으로 구분한다.

Session 1(첫번째 SQL 워크시트 )	Session 2(두번째 SQL 워크시트 )
① update emp set sal=9999 where deptno= 10;	
	② delete from emp where deptno = 20;
	③ delete from emp where deptno = 10;
④ commit;	⑤
	⑥ rollback;

① **Session1**에서 실행 가능한 첫번째 SQL(update)로 트랜잭션 시작

**10번 부서 데이터(3 Rows)에 Lock을 생성하고 직원 급여를 9999으로 update .**

② **Session2**에서 실행 가능한 첫번째 SQL(delete)로 트랜잭션 시작

**20번 부서 데이터(5 Rows)에 Lock을 생성하고 20번 부서 데이터를 delete .**

③ **Session2**에서 10번 부서 데이터(Row)에 Lock을 생성하려 하지만 이미 Session1 트랜잭션에서 10번 부서 데이터에 **Row Level Exclusive Lock**을 생성하였기 때문에 **Session 2에서는 Lock 해제시 까지 대기(Wait).**

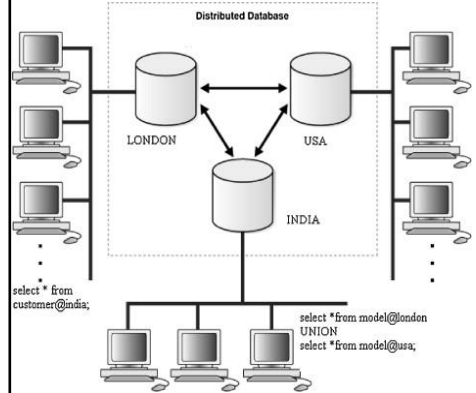
④ commit → 변경사항 반영후 → Lock 해제 →트랜잭션 명시적 종료 → 변경이 완료된 데이터 모든 세션에서 조회 가능

⑤ Session2에서 대기(Wait)상태를 풀고 10번 부서 데이터(Rows)에 Lock을 생성하고 10번 부서 데이터 delete.

⑥ 트랜잭션 종료 처리

## ● 7. TRANSACTION

### □ 유형

유형	내용	사례
<b>OLTP</b> (OnLine Transaction Processing)	<ul style="list-style-type: none"> <li>- <b>실시간처리(Real Time Processing)</b> <ul style="list-style-type: none"> <li>* 트랜잭션을 대기시간 없이 곧바로 처리,</li> <li>* 트랜잭션당 처리 비용 증가</li> </ul> </li> <li>- 사용자 중심 → 응답시간 최소화</li> </ul>	<ul style="list-style-type: none"> <li>- 은행 계정계 업무 ex) 계좌 신규/이체/입금/출금/해지</li> <li>- 항공사 예약</li> </ul>
<b>OLAP= DSS = Batch Processing</b> <b>Online Analytical Processing</b> <ul style="list-style-type: none"> <li>* Analytical</li> <li>* Decision Support System</li> </ul>	<ul style="list-style-type: none"> <li>- <b>일괄처리</b> (① 일정한 주기 ② 대용량 ③ 일괄)               <ul style="list-style-type: none"> <li>* 유사한 트랜잭션을 모아서(사전작업) 한번에 처리</li> <li>* 트랜잭션당 처리 비용 감소 → 단위시간당 처리되는 트랜잭션수 증가 → 시스템 자원 효율성 (성능) 증대</li> </ul> </li> <li>- <b>시스템 중심</b> → 효율적 시스템 자원 사용을 통해 전체 처리에 최적화</li> </ul>	<ul style="list-style-type: none"> <li>- 급여/요금 계산</li> <li>- 일별/월별 정산</li> <li>- 은행정보계 업무 ex) 신용평가, 경영정보, 통계</li> </ul>
<b>DTP</b> (Distributed Transaction Processing)	<ul style="list-style-type: none"> <li>- 지리적/물리적으로 분산되어 네트워크로 연결된 N개의 DBMS간 트랜잭션 처리, 논리적으로 단일 DBMS</li> <li>- 2단계 커밋(2 Phase Commit, Prepare/Commit)</li> <li>- DB Link, TPM(Transaction Processing Monitoring)</li> <li>- XA(eXtended Architecture) : X/Open DTP 모델에서 명시한 산업계 표준 스펙 ex) JTA(Java Transaction API)</li> </ul>	



## ● 7. TRANSACTION

### □ 유형별 성능 특징

유형	DBMS내 특징	SQL						
OLTP	<ul style="list-style-type: none"><li>- 작고 짜잔한 트랜잭션이 동시 다발적으로 발생</li><li>- <b>Read / Write</b> , 트랜잭션 처리 중심</li><li>- <b>Random Access</b></li><li>- <b>부분 범위 처리</b> EX) OPTIMIZER_MODE: FIRST_ROWS , FIRST_ROWS_N HINT : /*+ FIRST_ROWS(20) */</li></ul>	<table><tr><td>Parsing</td><td>.</td></tr><tr><td>Execute</td><td>.</td></tr><tr><td>Fetch</td><td>.</td></tr></table>	Parsing	.	Execute	.	Fetch	.
Parsing	.							
Execute	.							
Fetch	.							
OLAP= DSS = Batch Processing	<ul style="list-style-type: none"><li>- 굵고 거대한 트랜잭션이 순차적으로 작은 빈도로 발생</li><li>- <b>Read Only (주로)</b> , 주제중심 요약 데이터 생성 및 조회</li><li>- <b>Sequential Access</b></li><li>- <b>전체 범위 처리</b> EX) OPTIMIZER_MODE: ALL_ROWS HINT : /*+ ALL_ROWS */</li></ul>	<table><tr><td>Parsing</td><td>.</td></tr><tr><td>Execute</td><td>O</td></tr><tr><td>Fetch</td><td>O</td></tr></table>	Parsing	.	Execute	O	Fetch	O
Parsing	.							
Execute	O							
Fetch	O							

## ● 7. TRANSACTION

---

### □ 과제

- 1) SAVEPOINT를 사용하여 부분 롤백이 가능한지 증명하는 SQL Script 작성
- 2) AUTOCOMMIT 예를 재현
  - DDL 수행시
  - 정상적으로 접속(Connection) 종료시
  - 비정상 접속 (Connection) 종료시
  - DBMS 비정상 종료시
- 3) SELECT ~ FOR UPDATE를 사용하여
  - Repeatable Read 사례 구현 및 설명
  - 트랜잭션 시작/종료, Row Level Lock 설명