**1. 데이터(DML) 조작어**

DML(Data Manipulation Language) 명령은 데이터를 입력,수정,삭제하는 SQL명령어이다. 데이터베이스에 영구적으로 반영되지 않은 데이터 조작 명령어들을 TRANSACTION이라고 하며 오라클에서는 이를 하나의 논리적 작업 단위로 사용한다.

|  |  |
| --- | --- |
| 명 령 어 | 설 명 |
| INSERT | 테이블에 새로운 행 추가 |
| UPDATE | 테이블의 행 내용을 변경 |
| DELETE | 테이블의 행 삭제 |
| COMMIT | 저장되지 않은 모든 변경 사항을 Database에 저장 |
| SAVEPOINT | savepoint 설정 |
| ROLLBACK | 저장되지 않은 보든 변경 사항을 취소 |

**1.1 INSERT 문장**

테이블에 사용하여 테이블에 새로운 행을 삽입(INSERT)할 수 있다.

**1.1.1 Syntax**

|  |
| --- |
| **INSERT INTO** table\_name [(column1[, column2, . . . . . ])]  **VALUES**  (value1[, value2, . . . . . . ]); |

**1.1.2 사용 예**

* 모든 column에 대해 값을 갖는 새로운 행을 삽입한다.

|  |
| --- |
| SQL> **DESC emp**  Name Null? Type  ------------------------------- -------- ----  EMPNO NOT NULL NUMBER(4)  ENAME VARCHAR2(10)  JOB VARCHAR2(9)  MGR NUMBER(4)  HIREDATE DATE  SAL NUMBER(7,2)  COMM NUMBER(7,2)  DEPTNO NOT NULL NUMBER(2)  SQL> **INSERT INTO emp**  2 **VALUES (1111,'YOONJB','',NULL,SYSDATE,3000,NULL,10);**  1 row created. |

* INSERT 절의 column은 선택적으로 기입할 수 있다. 이럴 경우 열중 NOT NULL 제약 조건이 있는 열은 반드시 포함하여야 한다.

|  |
| --- |
| SQL> **INSERT INTO emp(empno,ename,hiredate,deptno)**  2 **VALUES (2222,'홍길동',SYSDATE,10);**  1 row created. |

* 문자와 날짜 값은 단일 따옴표('')를 둘러싼다.

|  |
| --- |
| SQL> **INSERT INTO emp(empno,ename,job,hiredate,deptno)**  2 **VALUES (3333,'HONGKD','SALESMAN',**  3 **to\_date('19990215213812','YYYYMMDDHH24MISS'),10);**  1 row created. |
| SQL> **INSERT INTO emp(empno,ename,job,hiredate,deptno)**  2 **VALUES (4444,'JBY','ANALYST',’13-FEB-99’,20);**  1 row created. |

☞ Guidelines

1. VALUES절을 가지는 INSERT문장은 한번에 오직 하나의 행만을 추가한다.
2. 모든 행에 값을 갖는 새로운 행을 삽입할 수 있기 때문에 column list는 INSERT절에 필요하지 않다. 하지만 테이블에 정의된 순서에 따라 값을 나열해야 한다.
3. 명확성을 위해 INSERT절에 column list를 사용하면 좋다.(권장)
4. 문자와 날짜는 단일 따옴표 안에 쓰나, 수치 값은 사용하지 않는다.

**1.1.3 NULL 값을 새로운 행에 추가**

자료형에 관계없이 사용 가능하다.

**가) 암시적 방법**

|  |
| --- |
| SQL> **INSERT INTO dept(deptno,dname)**  2 **VALUES (50,'DEVELOPMENT');**  1 row created. |

**나) 명시적 방법**

|  |
| --- |
| SQL> **INSERT INTO dept**  2 **VALUES(60,'',NULL);**  1 row created. |

**1.1.4 특정 날짜 값 삽입**

형식 DD-MON-YY는 항상 날짜 값을 입력할 때 사용한다. 이 형식은 현재 세기에 대한 DEFAULT 세기를 다시 호출한다. 또한 날짜가 시간 정보를 포함하므로 DEFAULT 시간은 자정(00:00:00)이다. 날짜를 다른 세기로 입력하거나 또는 특정 시간을 요구 한다면 TO\_DATE함수를 사용하여라.

예) 아래의 INSERT문장은 날짜를 입력하는 방법이다.

|  |  |
| --- | --- |
| INSERT명령 | 입력된 입사일자의 결과 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (5555,TO\_DATE('1999','YYYY'),30); | 1999/02/01 00:00:00 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (6666,TO\_DATE('99','YY'),20); | 1999/02/01 00:00:00 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (7777,TO\_DATE('02','MM'),20); | 1999/02/01 00:00:00 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (8888,TO\_DATE('13','DD'),30); | 1999/02/13 00:00:00 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (9999,TO\_DATE('10','HH24'),20); | 1999/02/01 10:00:00 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (1122,TO\_DATE('10','MI'),20); | 1999/02/01 00:10:00 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (2211,TO\_DATE('10','SS'),20); | 1999/02/01 00:00:10 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (1133,TO\_DATE('JUN 3,99','MON DD,YY'),10); | 1999/06/03 00:00:00 |
| SQL> INSERT INTO emp(empno,hiredate,deptno)  2 VALUES (3311,TO\_DATE('2000/02/01 17:35:10',  3 'YYYY/MM/DD HH24:MI:SS'),20); | 2000/02/01 17:35:10 |

**1.1.5 치환 변수를 사용하여 값 입력**

SQL\*Plus의 치환 변수를 사용하여 사용자가 상호 작용으로 값을 추가할 수 있다. 날짜와 문자 값은 단일 인용 부호를 사용하여 감싸는 것이 사용하기에 편하고 SCRIPT로 작성하여 사용하면 입력시 매우 유용하다.

예) SCRIPT 안에서 치환 변수 사용하는 방법이다.

|  |
| --- |
| SQL> **ed dept\_insert**  dept\_insert SCRIPT안에 다음의 명령어를 작성하여 저장하고 실행 한다.  **INSERT INTO dept (deptno,dname,loc)**  **VALUES (&department\_id,'&department\_name','&location');**  SQL> **@dept\_insert**  Enter value for department\_id: **70**  Enter value for department\_name: **EDUCATION**  Enter value for location: **ATLANTA**  old 2: VALUES (&department\_id,'&department\_name','&location')  new 2: VALUES (70,'EDUCATION','ATLANTA')  1 row created. |

**1.1.6 다른 테이블로부터 행 복사**

기존의 테이블로부터 값을 가져와 테이블에 추가하기 위해서 INSERT문장을 사용할 수 있다. 즉 VALUES절에서 subquery를 사용할 수 있다.

**가) Syntax**

|  |
| --- |
| **INSERT INTO** table\_name [column1[,column2, . . . .]] **subquery**; |

문제1) EMP 테이블에서 EMPNO,ENAME,SAL,HIREDATE의 COLUMN만 선택하여 EMP\_10 테이블을 생성한 후 10번 부서만 선택하여 이에 대응하는 값을 EMP\_10테이블에 입력하여라.

|  |
| --- |
| SQL> **CREATE TABLE emp\_10(id,name,salary,hiredate)**  2 **AS SELECT empno,ename,sal,hiredate**  3 **FROM emp**  4 **WHERE 1 = 2;**  Table created.  SQL> **INSERT INTO emp\_10**  2 **SELECT empno,ename,sal,hiredate**  3 **FROM emp**  4 **WHERE deptno = 10;**  7 rows created. |

♣ 참고

INSERT절의 열의 개수와 서브쿼리의 열의 개수가 좌측부터 1대1대응하며 자료형과 길이가 같아야 한다.

**1.1.7 INSERT시 무결성 제약 조건의 오류**

무결성 제약 조건(5가지)이 위배되면 INSERT시 ERROR가 발생합니다.

|  |
| --- |
| SQL> **INSERT INTO emp**  2 **VALUES (7788,'YOON','MANAGER',NULL,SYSDATE,NULL,NULL,10);**  INSERT INTO emp  \*  ERROR at line 1:  ORA-00001: unique constraint (SCOTT.EMP\_PRIMARY\_KEY) violated  SQL> **INSERT INTO emp**  2 **VALUES (1144,'YOON','MANAGER',NULL,SYSDATE,NULL,NULL,91);**  INSERT INTO emp  \*  ERROR at line 1:  ORA-02291: integrity constraint (SCOTT.EMP\_FOREIGN\_KEY) violated - parent key not found  SQL> **INSERT INTO emp**  2 **VALUES (1144,'YOON','MANAGER',1234,SYSDATE,NULL,NULL,10);**  INSERT INTO emp  \*  ERROR at line 1:  ORA-02291: integrity constraint (SCOTT.EMP\_SELF\_KEY) violated - parent key not found  SQL> **INSERT INTO emp (empno,ename,job)**  2 **VALUES (1144,'YOON','MANAGER');**  INSERT INTO emp (empno,ename,job)  \*  ERROR at line 1:  ORA-01400: mandatory (NOT NULL) column is missing or NULL during insert |

**1.2 UPDATE 문장**

1. UPDATE문장으로 기존의 행을 갱신합니다.
2. 일반적으로 단일 행을 식별하기 위해서 기본 키(primary key)를 사용합니다. 다른 열을 사용하면 원하지 않는 여러 행이 갱신될 수 있습니다.
3. 특정 열이나 행은 WHERE절을 이용하여 갱신할 수 있다.

**1.2.1 Syntax**

|  |
| --- |
| **UPDATE** table\_name  **SET** column1 = value1 [,column2 = value2, . . . . . . .]  [**WHERE** condition]; |

♣ 참고

DEMO TABLE를 새로 생성하여 실습을 하자

|  |
| --- |
| SQL> **@c:\orawin95\dbs\demobld** |

문제2) EMP 테이블에서 사원 번호가7788인 사원의 부서를 10번으로 변경하여라.

|  |
| --- |
| SQL> **UPDATE emp**  2 **SET deptno = 10**  3  **WHERE empno = 7788;**  1 row updated. |

문제3) EMP 테이블에서 사원 번호가7788인 사원의 부서를 20, 급여를 3500으로 변경하여라.

|  |
| --- |
| SQL> **UPDATE emp**  2 **SET deptno = 20, sal = 3500**  3 **WHERE empno = 7788;**  1 row updated. |

문제4) EMP 테이블에서 부서를 모두 10으로 변경하여라.

|  |
| --- |
| SQL> **UPDATE emp**  2 **SET deptno = 10;**  14 rows updated. |

**1.2.2 다중 열 SUBQUERY로 갱신**

다중 열 SUBQUERY는 UPDATE문장의 SET절로 구현할 수 있다.

**가) Syntax**

|  |
| --- |
| **UPDATE** table\_name  **SET** (column1, column2, . . . . ) =  ( **SELECT**  column1,column2, . . .  **FROM**  table\_name  **WHERE**  coundition)  [**WHERE** condition]; |

문제4) EMP 테이블에서 SCOTT의 업무와 급여가 일치하도록 JONES업무와 급여를 개인하여라

|  |
| --- |
| SQL> **UPDATE emp**  2 **SET (job,sal) = (SELECT job,sal**  3 **FROM emp**  4  **WHERE ename = 'SCOTT')**  5  **WHERE ename = 'JONES';**  1 row updated. |

**1.2.3 다른 테이블을 근거로 한 행 갱신**

다른 테이블의 값을 근거로 하는 테이블에서 행을 갱신하기 위해 UPDATE문장에서 SUBQUERY를 사용한다.

문제4) EMP 테이블을 근거로 EMPLOYEE 테이블을 생성한 후 7902의 업무에 해당하는 사원의 부서번호를 7902의 부서번호로 갱신하여라.

|  |
| --- |
| SQL> **@c:\orawin95\dbs\demobld**  SQL> **UPDATE employee**  2 **SET deptno = (SELECT deptno**  3 **FROM emp**  4 **WHERE empno = 7902)**  5 **WHERE job = (SELECT job**  6 **FROM emp**  7 **WHERE empno = 7902);**  2 rows updated. |

**1.2.4 UPDATE시 무결성 제약 조건 ERROR**

무결성 제약 조건이 위배되는 값으로 UPDATE할 경우 ERROR가 발생한다.

문제5) EMP 테이블에서 10번 부서의 사원을 모두 91번 부서로 갱신하여라.

|  |
| --- |
| SQL> **UPDATE emp**  2 **SET deptno = 91**  3 **WHERE deptno = 10;**  UPDATE emp  \*  ERROR at line 1:  ORA-02291: integrity constraint (SCOTT.EMP\_FOREIGN\_KEY) violated - parent key not found |

문제6) DEPT 테이블에서 부서 번호 10을 15로 갱신하여라.

|  |
| --- |
| SQL> **UPDATE dept**  2 **SET deptno = 15**  3 **WHERE deptno =10;**  UPDATE dept  \*  ERROR at line 1:  ORA-02292: integrity constraint (SCOTT.EMP\_FOREIGN\_KEY) violated - child record found |

**1.3 DELETE 문장**

1. DELETE 문장을 사용하여 테이블로부터 기존의 자료를 삭제할 수 있다.
2. WHERE절을 명시하여 특정 행이나 행들을 삭제할 수 있다.
3. WHERE절을 생략하면 테이블의 모든 행이 삭제 된다.

**1.3.1 Syntax**

|  |
| --- |
| **DELETE** [**FROM**] table\_name  [**WHERER**  condition]; |

문제7) EMP 테이블에서 사원번호가 7499인 사원의 정보를 삭제하여라.

|  |
| --- |
| SQL> **DELETE emp**  2 **WHERE empno = 7499;**  1 row deleted. |

문제8) EMP 테이블에서 입사일자가 83년인 사원의 정보를 삭제하여라.

|  |
| --- |
| SQL> **DELETE emp**  2 **WHERE TO\_CHAR(hiredate,'YY') = '83';**  1 row deleted. |

**1.3.2 다른 테이블을 근거로 한 행 삭제**

다른 테이블을 값을 근거로 행을 삭제하기 위해서는 SUBQUERY를 사용하여야 한다.

문제9) EMP 테이블의 자료 중 부서명이 'SALES'인 사원의 정보를 삭제하여라.

|  |
| --- |
| SQL> **DELETE emp**  2 **WHERE deptno = (SELECT deptno**  3  **FROM dept**  4 **WHERE dname = 'SALES');**  5 rows deleted. |

**1.3.3 무결성 제약 조건 ERROR**

무결성 제약 조건을 위반하도록 행을 삭제하면 ERROR가 발생한다.

문제10) EMP 테이블에서 사원 번호가 7902인 사원의 정보를 삭제하여라.

|  |
| --- |
| SQL> **DELETE emp**  2 **WHERE empno = 7902;**  DELETE emp  \*  ERROR at line 1:  ORA-02292: integrity constraint (SCOTT.EMP\_SELF\_KEY) violated - child record found |

문제11) DEPT 테이블에서 부서명이 ‘ACCOUNTING’ 부서의 정보를 삭제하여라.

|  |
| --- |
| SQL> **DELETE dept**  2  **WHERE dname = 'ACCOUNTING';**  DELETE dept  \*  ERROR at line 1:  ORA-02292: integrity constraint (SCOTT.EMP\_FOREIGN\_KEY) violated - child record found |

**1.4 데이터베이스 TRANSACTION**

ORACLE SERVER는 TRANSACTION을 근거로 하는 데이터의 일관성을 보증한다. TRANSACTION은 데이터 변경시에 보다 나은 융통성과 제어를 제공하며 그들은 사용자 프로세스 실패나 시스템 실패 같은 이벤트에서 데이터의 일관성을 책임집니다. TRANSACTION은 데이터를 일관되게 변경하는 DML문장으로 구성됩니다. 예를 들면, 하나의 예금에 대한 차변과 똑같은 금액이 있는 다른 예금에 대한 대변을 포함해야 하는 두 개의 예금 사이에 예금을 전달합니다. 액션은 둘 모두 실패하거나 둘 모두 성공해야 합니다. 대변은 차변 없이는 인증될 수 없습니다.

**1.4.1 TRANSACTION의 유형**

|  |  |
| --- | --- |
| 유 형 | 설 명 |
| DML | 작업의 논리적인 단위로 취급하는 임의의 수의 DML문장으로 구성됩니다. |
| DDL | 오직 하나의 DDL문장으로 구성합니다. |
| DCL | 오직 하나의 DCL문장만으로 구성합니다. |

**1.4.2 TRANSACTION의 시작과 종료**

**가) TRANSACTION의 시작**

1. 실행 가능한 SQL문장이 제일 처음 실행될 때

**나) TRANSACTION의 종료**

1. COMMIT이나 ROLLBACK
2. DDL이나 DCL문장의 실행(자동 COMMIT)
3. 기계 장애 또는 시스템 충돌(crash)
4. deadlock 발생
5. 사용자가 정상 종료

**1.5.2 COMMIT과 ROLLBACK의 장점**

1. 데이터의 일관성 제공
2. 데이터를 영구적으로 변경하기 전에 데이터 변경을 확인하게 한다.
3. 관련된 작업을 논리적으로 그룹화 할 수 있다.

**1.4.3 TRANSACTION 제어**

COMMIT, SAVEPOINT,ROLLBACK 문장으로 TRANSACTION의 논리를 제어할 수 있다.

|  |  |
| --- | --- |
| 명령어 | 설 명 |
| COMMIT | 모든 미결정 데이터를 영구적으로 변경 함으로서 현재 TRANSACTION을 종료합니다. |
| SAVEPOINT name | 현재 TRANSACTION 내에 savepoint를 표시합니다. |
| ROLLBACK [TO SAVEPOINT name] | ROLLBACK은 모든 미결정 데이터 변경을 벼림으로써 현재의 TRANSACTION을 종료합니다.ROLLBACK TO SAVEPOINT name은 savepoint와 모든 연이은 변경을 버립니다. |

♣ 참고

하나의 TRANSACTION이 끝난 후에 다음의 실행 가능한 SQL문장은 자동적으로 다음 TRANSACTION을 시작할 것이다. DDL과 DCL은 자동적으로 COMMIT되므로 TRANSACTION을 임시적으로 종료한다. SAVEPOINT는 ANSI 표준 SQL이 아니다.

**1.4.3 암시적 TRANSACTION 처리**

1. 자동 COMMIT은 다음의 경우 발생
2. DDL,DCL문장이 완료시
3. 명시적인 COMMIT이나 ROLLBACK없이 SQL\*Plus를 정상 종료
4. 자동 ROLLBACK은 다음의 경우 발생
5. SQL\*Plus를 비정상 종료 또는 시스템 실패

**1.4.4 COMMIT이나 ROLLBACK 이전의 데이터 상태**

1. 데이터 이전의 상태로 북구가 가능하다.
2. 현재 사용자는 SELECT문장으로 DML작업의 결과를 확인할 수 있다.
3. 다른 사용자는 SELECT문장으로 현재 사용자 사용한 DML문장의 결과를 확인할 수 없다.
4. 변경된 행은 LOCK이 설정되어서 다른 사용자가 변경할 수 없다.

**1.4.5 COMMIT이후의 데이터 상태**

1. 데이터베이스에 데이터를 영구적으로 변경
2. 데이터의 이전 상태는 완전히 상실
3. 모든 사용자가 결과를 볼 수 있다.
4. 변경된 행의 LOCK이 해제되고 다른 사용자가 변경할 수 있다.
5. 모든 SAVEPOINT는 제거된다.

문제12) EMP 테이블에서 7788인 사원의 부서번호를 30번 부서로 갱신한 후 자료를 확정하여라.

|  |
| --- |
| SQL> **UPDATE emp**  2 **SET deptno = 30**  3 **WHERE empno = 7788;**  1 row updated.  SQL> **commit;**  Commit complete. |

**1.4.6 ROLLBACK이후의 데이터 상태**

1. 데이터의 변경이 취소
2. 데이터의 이전 상태로 복구
3. 변경된 행들의 LOCK이 해제되어 다른 사용자가 변경할 수 있다.

|  |
| --- |
| SQL> **DELETE emp;**  14 rows deleted.  SQL> **ROLLBACK;**  Rollback complete. |

**1.4.7 SAVEPOINT로 변경을 ROLLBACK**

1. SAVEPOINT를 사용하여 현재 TRANSACTION내에 표시자를 생성한다
2. ROLLBACK TO SAVEPOINT명령을 사용하여 표시자까지 ROLLBACK

|  |
| --- |
| SQL> **UPDATE emp**  2 **SET deptno = 30**  3 **WHERE empno = 7788;**  1 row updated.  SQL> **SAVEPOINT point\_1;**  Savepoint created.  SQL> **UPDATE emp**  2 **SET job = 'MANAGER';**  14 rows updated.  SQL> **ROLLBACK TO SAVEPOINT point\_1;**  Rollback complete. |

**1.4.8 문장 단위 ROLLBACK**

1. 실행 동안에 단일 DML문장이 실패하면 단지 그 문장만을 ROLLBACK한다.
2. ORACLE SERVER은 암시적 SAVEPOINT를 구현 한다.
3. 모든 다른 변경들은 유지된다.
4. 사용자는 COMMIT이나 ROLLBACK을 실행하여 명시적으로 TRANSACTION을 종료한다.

**1.5 읽기 일관성**

1. 읽기 일관성은 항상 데이터의 검색이 일관되게 보증한다.
2. 사용자에 의해 행해진 변경은 다른 사용자에 의해 행해진 변경과 충돌하지 않는다.
3. 데이터를 똑같게 보증한다.

**1.6 Locking**

1. 동시 TRANSACTION 사이의 상호 작용이 파괴되지 않도록 막아 줍니다.
2. 사용자 액션을 요구하지 않습니다.
3. 자동적으로 낮은 LEVEL의 제약 조건을 사용합니다.
4. TRANSACTION이 지속되도록 합니다.
5. 두가지 기본적인 모드를 가집니다.
6. Exclusive
7. Share

**1.6.1 Locking Mode**

|  |  |
| --- | --- |
| LOCK MODE | 설 명 |
| Exclusiv | 자원이 공유되는 것을 막아 줍니다. 자원을 배타적으로 lock하는 첫번째 TRANSACTION은 배타적 잠금이 해제되기 전까지는 자원을 변경할 수 있는 유일한 TRANSACTION입니다. |
| Share | 자원이 공유되도록 허용합니다. 데이터를 읽는 다중 사용자는 데이터를 공유하고, writer에 의해 동시 액세스 되는 것을 막기 위해 공유 잠금을 유지 합니다. 똑같은 지원상에서 여러 개의 TRANSACTION은 공유 잠금을 구할 수 있습니다. |

**◈ 연 습 문 제 ◈**

1. 아래의 구조를 만족하는 MY\_DATA 테이블을 생성하시오. 단 ID가 PRIMARY KEY이다.

|  |
| --- |
| SQL> DESC my\_data  Name Null? Type  ------------------------------- -------- ----  ID NOT NULL NUMBER(4)  NAME VARCHAR2(10)  USERID VARCHAR2(30)  SALARY NUMBER(10,2) |

2. 1번에 의해 생성된 테이블에 아래의 값을 입력하여라.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | USERID | SALARY |
| 1 | Scott | sscott | 10,000.00 |
| 2 | Ford | fford | 13,000.00 |
| 3 | Patel | ppatel | 33,000.00 |
| 4 | Report | rreport | 23,500.00 |
| 5 | Good | ggood | 44,450.00 |

3. 2번에서 입력한 자료를 확인 하여라.

4. 2번에서 입력한 자료를 모두 삭제하고 INSERT하기 위한 SCRIPT를 작성하여 대화식으로 입력하여라. 단 내용은 2번과 동일하다.

5. 4번에서 바꾼 자료를 영구적으로 데이터베이스에 등록하여라.

6. ID가 3번인 사람의 급여를 65,000.00으로 갱신하고 영구적으로 데이터베이스에 반영하여라.

7. 이름이 Ford인 사원을 영구 제명하여라.

8. 급여가 15,000이하인 사람의 급여를 15,000로 변경하여라.

9. 1번에서 생성한 테이블을 삭제하여라.