Chapter 9. 데이터 조작

이번 장에서는 DML 문장을 이용하여 테이블에 새로운 행을 입력하고 기존에 입력되어 있는 행을 수정 및 삭제하는 방법을 알아본다. 또한, Oracle 9i 버전부터 새롭게 추가된 행의 머지(Merge) 구문과 트랜잭션을 제어하는 방법을 설명한다.

데이터 조작 언어(DML)

데이터 조작 언어(DML : Data Manipulation Language)란 테이블에 새로운 행을 입력하고, 기존에 입력된 행을 수정 및 삭제하는 명령이다. 여기서, 한개 이상의 DML 문장들은 하나의 트랜잭션(Transaction)으로 구성되는데, 트랜잭션은 데이터베이스의 논리적 작업 단위를 의미한다. 예를 들어, 은행에서의 계좌이체 작업을 생각해보자. 즉, A 구좌에서 B 구좌로의 계좌이체 작업은 전체 3가지 작업으로 완료 된다. 첫 번째는 A 구좌의 잔고를 감소시키고, 두 번째는 A 구좌의 감소된 잔고만큼 B 구좌의 잔고를 증가시킨 다음, 마지막으로 계좌이체 작업을 기록하는 것이다. 만약에 이 세 가지 작업 중에 하나의 작업이라도 실패하게 되었다면 이러한 모든 계좌 이체 작업은 전부 취소되어야 한다. 즉, 이 세 가지 데이터 조작 작업은 하나의 논리적 작업 단위로서 트랜잭션으로 처리되어야 한다는 것이다.

INSERT

INSERT는 테이블에 새로운 행을 입력하는 문장이다. INSERT 구문의 문법은 다음과 같다.

```
INSERT INTO table [(column [, column ...])]
VALUES (value [, value ...])
```

INSERT 구문은 한번에 하나의 행만을 입력할 수 있다.

```
SQL> INSERT INTO DEPT(DEPTNO, DNAME, LOC)
  2  VALUES(90, '인사과', '서울');

1 개의 행이 만들어졌습니다.
```

위 문장에서 테이블명 뒤에 컬럼들을 기술하지 않으면 테이블내의 컬럼 순서(DESC 명령으로 출력되는 컬럼 순서)에 일치하도록 입력할 값들을 기술해야한다. 즉, 위 문장은 아래 문장과 동일하다.

```
INSERT INTO DEPT VALUES(90, '인사과', '서울');
```

입력할 값이 문자 타입이나 날짜 타입인 경우는 ' '를 붙여준다.

■ NULL 값 입력

NULL 값을 갖는 행을 입력하는 방법은 두 가지 방법이 있으며, 첫 번째는 NULL 값이

입력될 컬럼을 INSERT 문장에 기술하지 않는 방법이다.

```
SQL> INSERT INTO DEPT(DEPTNO, DNAME)
  2  VALUES(91, '총무과');

1 개의 행이 만들어졌습니다.
```

두 번째 방법은 NULL 값 대신에 NULL이라는 키워드를 기술하는 방법이다.

```
SQL> INSERT INTO DEPT
  2  VALUES(92, '경리과', NULL);

1 개의 행이 만들어졌습니다.
```

■ 특수 값 입력

INSERT 문장에 SYSDATE, USER와 같은 특수 함수를 이용하면 테이블에 시스템의 날짜 또는 현재 사용자의 계정을 입력 할 수 있다.

```
SQL> INSERT INTO EMP(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
  2  VALUES(9000, USER, '연구원', 7839, SYSDATE, 5000, NULL, 90);

1 개의 행이 만들어졌습니다.
```

■ 날짜 입력

테이블에 날짜 타입을 입력할 때는 INSERT 문장에서 날짜를 RR/MM/DD 형식으로 입력하여야 날짜 타입으로 인식되어 저장되지만, DD-MON-RR과 같은 형식으로 입력하면 문자 타입으로 인식되어 올바르게 저장되지 않는다. 이러한 경우, TO_DATE 함수를 사용하여 날짜 타입으로 변환해야만 한다.

```
SQL> INSERT INTO EMP(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
  2  VALUES(9001, '홍길동', '상담원', 7839, TO_DATE('19-JUN-04', 'DD-MON-YY'),
  3  3000, NULL, 30);

1 개의 행이 만들어졌습니다.
```

■ 대체 변수를 이용한 입력

INSERT 문장을 반복해서 사용하는 경우, 대체 변수를 사용하면 좀 더 편리하게 데이터 입력 작업을 수행 할 수 있다.

```
SQL> INSERT INTO DEPT
  2  VALUES(&DEPT_NO, '&DEPT_DNAME', '&DEPT_LOC');
dept_no의 값을 입력하십시오: 93
dept_dname의 값을 입력하십시오: 홍보과
dept_loc의 값을 입력하십시오: 대전
구   2: VALUES(&DEPT_NO, '&DEPT_DNAME', '&DEPT_LOC')
신   2: VALUES(93, '홍보과', '대전')

1 개의 행이 만들어졌습니다.
```

■ 다른 테이블로부터 행 복사

INSERT 문장은 한번에 하나의 행만 입력할 수 있으나, INSERT INTO ... SELECT 문

장을 이용하면 다른 테이블에 저장된 복수개의 행들을 한번에 입력할 수 있다.

다음과 같이 실습용 테이블을 생성한 후, 사원 테이블에서 부서코드가 10번인 사원 데이터들을 실습용 테이블 EMP10에 입력해보자.

```
SQL> CREATE TABLE EMP10
  2  (EMPNO NUMBER(4),
  3  ENAME VARCHAR2(10),
  4  JOB VARCHAR2(10),
  5  SAL NUMBER(7,2));

테이블이 생성되었습니다.

SQL> INSERT INTO EMP10(EMPNO, ENAME, JOB, SAL)
  2  SELECT EMPNO, ENAME, JOB, SAL
  3  FROM EMP
  4  WHERE DEPTNO = 10;

3 개의 행이 만들어졌습니다.
```

■ INSERT 문장에서 서브 쿼리 사용

INSERT 문장에서 테이블명 대신 서브 쿼리를 사용할 수도 있다.

```
SQL> INSERT INTO
  2  (SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO
  3   FROM EMP
  4   WHERE DEPTNO = 30)
  5  VALUES (9002, '콩쥐', '사무원', 7934, '04/01/01', 3500, NULL, 30);

1 개의 행이 만들어졌습니다.
```

위 문장에서 서브 쿼리에 WITH CHECK OPTION을 사용하면 서브 쿼리의 WHERE 조건에 만족하는 행만을 입력할 수 있다. 다음과 같이 WHERE 절의 조건과 입력 데이터가 일치 되지 않으면 오류가 발생한다.

```
SQL> INSERT INTO
  2  (SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO
  3   FROM EMP
  4   WHERE DEPTNO = 30 WITH CHECK OPTION)
  5  VALUES (9003, '팥쥐', '보조원', 7934, '04/04/05', 1500, NULL, 40);
FROM EMP
     *
3행에 오류:
ORA-01402: 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다
```

UPDATE

UPDATE는 테이블에 저장되어 있는 행들을 변경하는 문장이다. UPDATE 구문의 문법은 다음과 같다.

```
UPDATE table
SET column = value [, column = value, ... ]
[WHERE condition];
```

UPDATE 구문은 한번에 여러 개의 행을 변경 할 수 있다.

```
SQL> UPDATE DEPT
  2  SET LOC = '부산'
  3  WHERE DEPTNO = 90;

1 행이 갱신되었습니다.
```

위 문장에서 WHERE 절이 생략되면 테이블 내의 모든 행이 변경되므로 주의할 필요가 있다.

■ 서브 쿼리를 이용한 복수 컬럼 변경

UPDATE 문장에서 여러 개의 컬럼을 한번에 변경 할 수 있으며, 서브 쿼리가 포함 될 수도 있다.

```
SQL> UPDATE EMP
  2  SET JOB = (SELECT JOB
  3             FROM EMP
  4             WHERE EMPNO = 7900),
  5      SAL = (SELECT SAL
  6             FROM EMP
  7             WHERE EMPNO = 7844)
  8  WHERE EMPNO = 9000;

1 행이 갱신되었습니다.
```

■ 다른 테이블을 기반으로 테이블의 행 변경

UPDATE 문장의 서브 쿼리에서 참조하는 테이블과 변경하고자 하는 테이블이 다를 수도 있다.

```
SQL> UPDATE EMP10
  2  SET JOB = (SELECT JOB
  3             FROM EMP
  4             WHERE EMPNO = 7844)
  5  WHERE EMPNO = (SELECT EMPNO
  6                 FROM EMP
  7                 WHERE ENAME = 'MILLER');

1 행이 갱신되었습니다.
```

■ 행 변경 오류 : 무결성 제약조건 위반

다른 테이블을 참조하는 테이블의 행을 변경하는 경우, 외래키 제약조건에 의해 오류가 발생 될 수 있다. 예를 들어, 다음 UPDATE 문장은 사번이 9001번인 사원의 부서코드를 50번으로 변경하는 문장인데, 부서 테이블에는 50번 부서가 존재하지 않기 때문에 외래키 제약조건에 위배되어 해당 행이 변경되지 않는다.

```
SQL> UPDATE EMP
  2  SET DEPTNO = 50
  3  WHERE EMPNO = 9001;
UPDATE EMP
*
1행에 오류:
ORA-02291: 무결성 제약조건(SCOTT.FK_DEPTNO)이 위배되었습니다- 부모 키가 없습니
다
```

DELETE

DELETE는 테이블에 저장되어 있는 행들을 삭제하는 문장이다. DELETE 구문의 문법은 다음과 같다.

```
DELETE [FROM] table
[WHERE condition];
```

DELETE 구문은 한번에 여러 개의 행을 삭제 할 수 있다.

```
SQL> DELETE FROM DEPT
  2  WHERE DEPTNO = 93;

1 행이 삭제되었습니다.
```

위 문장에서 WHERE 절이 생략되면 테이블 내의 모든 행이 삭제되므로 주의할 필요가 있다.

■ 다른 테이블을 기반으로 테이블의 행 삭제

DELETE 문장의 서브 쿼리에서 참조하는 테이블과 삭제하고자 하는 테이블이 다를 수도 있다.

```
SQL> DELETE FROM EMP
  2  WHERE DEPTNO = (SELECT DEPTNO
  3                  FROM DEPT
  4                  WHERE DNAME = '인사과');

1 행이 삭제되었습니다.
```

■ 행 삭제 오류 : 무결성 제약조건 위반

다른 테이블에 의해 참조되는 테이블의 행을 삭제하는 경우, 외래키 제약조건에 의해 오류가 발생 할 수 있다. 예를 들어, 다음 DELETE 문장은 부서 테이블에서 부서코드가 10번인 행을 삭제하는 문장인데, 10번 부서는 사원 테이블에서 사원들이 참조하고 있는 부서이므로 외래키 제약조건에 위배되어 해당 행이 삭제되지 않는다.

```
SQL> DELETE FROM DEPT
  2  WHERE DEPTNO = 10;
DELETE FROM DEPT
*
1행에 오류:
ORA-02292: 무결성 제약조건(SCOTT.FK_DEPTNO)이 위배되었습니다- 자식 레코드가 발
견되었습니다
```

DEFAULT

DEFAULT는 INSERT와 UPDATE 문장에서 입력 또는 변경하고자 하는 값 대신에 기술할 수 있는 키워드이다. DEFAULT가 사용되면 변경하고자 하는 테이블의 컬럼에 지정된 디폴트 값으로 입력 또는 변경된다. DEFAULT는 SQL : 1999 표준과의 호환을 위해서 Oracle 9i에서 새롭게 추가된 키워드이다. 만약, 테이블의 컬럼에 디폴트가 지정되어 있지 않은 경우에 DEFAULT 키워드를 사용하면 NULL 값이 적용된다. 다음과 같이 실습용 테이블을 만들고 DEFAULT를 이용하여 데이터를 입력해보자

```
SQL> CREATE TABLE TEST1
  2  (NO NUMBER(4),
  3  NAME VARCHAR2(10) DEFAULT 'UNKNOWN',
  4  AGE NUMBER(4) DEFAULT 0);

테이블이 생성되었습니다.

SQL> INSERT INTO TEST1 VALUES(1, '길동', DEFAULT);

1 개의 행이 만들어졌습니다.

SQL> SELECT * FROM TEST1;

        NO NAME              AGE
---------- ---------- ----------
         1 길동                0
```

UPDATE 문장에도 DEFAULT를 사용할 수 있다.

```
SQL> UPDATE TEST1 SET NAME = DEFAULT WHERE NO = 1;

1 행이 갱신되었습니다.

SQL> SELECT * FROM TEST1;

        NO NAME              AGE
---------- ---------- ----------
         1 UNKNOWN             0
```

MERGE

MERGE는 조건에 따라 데이터를 입력하거나 변경 할 수 있는 문장이다. 즉, 입력할 데이터가 대상 테이블에 존재하지 않으면 INSERT가 수행되며, 동일한 데이터가 대상 테이블에

존재하면 UPDATE가 수행된다. MERGE 문장의 문법은 다음과 같다.

```
MERGE INTO table_name table_alias
  USING (table|view|sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
    col1 = col1_val,
    col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

EMP10 테이블의 데이터를 EMP 테이블의 데이터로 갱신해보자. 즉, EMP 테이블에 저장된 행이 EMP10 테이블에 존재하면 EMP10 테이블의 해당 행을 EMP 테이블에 저장된 행으로 UPDATE하고, 존재하지 않으면 EMP10 테이블에 해당 행을 INSERT하는 것이다.

```
SQL> MERGE INTO EMP10 N
  2     USING EMP O
  3     ON (N.EMPNO = O.EMPNO)
  4     WHEN MATCHED THEN
  5       UPDATE SET
  6       N.ENAME = O.ENAME,
  7       N.JOB = O.JOB,
  8       N.SAL = O.SAL
  9     WHEN NOT MATCHED THEN
 10       INSERT (N.EMPNO, N.ENAME, N.JOB, N.SAL)
 11       VALUES (O.EMPNO, O.ENAME, O.JOB, O.SAL);

16 행이 병합되었습니다.
```

위의 예를 보면 EMP10 테이블과 EMP 테이블에서 동일한 행의 존재 여부는 두 테이블의 EMPNO 컬럼에 동일한 값이 존재하는지 여부로 판단한다. 즉, 동일한 사번이 있으면 EMP 테이블의 데이터로 EMP10 테이블을 변경하고, 동일한 사번이 없으면 EMP 테이블의 데이터를 입력한다.

트랜잭션(Transaction)

Oracle 데이터베이스는 트랜잭션에 의해 데이터의 일관성이 보장된다. 트랜잭션은 데이터 변경시 사용자에게 유연성 및 제어성을 부여해주고 사용자 프로세스 또는 시스템 오류가 발생 했을 때, 데이터의 일관성을 지켜주는 역할을 한다. 트랜잭션은 데이터의 일관성 있는 변경을 수행하는 여러 개의 DML 문장으로 구성된다. 예를 들어, 은행에서 사용자 구좌 간에 발생하는 계좌이체는 차변과 대변 값이 정확하게 일치해야하기 때문에 각 계좌에서 발생하는 작업은 모두 성공하거나 실패 되도록 하여야 한다.

트랜잭션의 타입은 다음과 같다.

표 9-1. 트랜잭션의 타입

타입	설명
DML(Data Manipulation Language)	데이터베이스의 논리적 작업 단위로서 여러 개의 DML 문장으로 구성
DDL(Data Definition Language)	하나의 DDL 문장으로 구성
DCL(Data Control Language)	하나의 DCL 문장으로 구성

그렇다면 트랜잭션이 시작되는 시점과 종료되는 시점을 알아보자. 트랜잭션은 첫 번째 DML 문장이 실행되면 시작되고 다음과 같은 상황에서 트랜잭션은 종료된다.

· 사용자가 COMMIT 또는 ROLLBACK 명령을 실행한 경우
· CREATE 명령과 같은 DDL 문장을 실행한 경우
· DCL 문장을 실행한 경우
· 사용자가 SQL*Plus 또는 iSQL*Plus를 종료한 경우
· 하드웨어 고장 또는 시스템 오류

사용자가 COMMIT 또는 ROLLBACK 명령을 사용하여 트랜잭션을 직접 제어함으로써 다음과 같은 장점을 얻을 수 있다.

· 데이터의 일관성을 보장해준다.
· 데이터의 변경사항을 데이터베이스에 영구히 반영하기 전에 데이터 변경사항을 미리 볼 수 있다.
· 논리적으로 연관된 작업들을 그룹화 할 수 있다.

■ 트랜잭션 제어 명령

트랜잭션을 제어하는 명령은 다음과 같다.

표 9-2. 트랜잭션 제어 명령

명령	설명
COMMIT	현재 진행 중인 트랜잭션을 종료하며 모든 변경사항을 데이터베이스에 영구히 반영
SAVEPOINT *name*	현재 진행 중인 트랜잭션의 중간에 저장점 *name*을 표시
ROLLBACK	현재 진행 중인 트랜잭션을 종료하며 모든 변경사항을 취소
ROLLBACK TO *name*	현재 진행 중인 트랜잭션의 저장점으로 복귀하고 저장점 이후의 모든 데이터 변경사항을 취소

트랜잭션을 제어하는 명령 중에 SAVEPOINT란 다음 그림과 같이 트랜잭션이 진행중일 때, 임의 지점에 저장점을 기록함으로써 ROLLBACK TO 명령으로 언제든지 사용자가 원하는 저장점으로 데이터를 복원할 수 있도록 해준다. 여기서, ROLLBACK TO 명령은 진행 중인 트랜잭션의 저장점으로만 복원하는 것이지 트랜잭션을 종료시키지 않음을 주의해야 한다. 최종적으로 트랜잭션을 종료하려면 COMMIT 또는 ROLLBACK 명령을 사용해야 한다. 또한, 동일한 트랜잭션내에서 두 개 이상의 저장점을 지정하는 경우에 저장점의 이름이 중복되면 이전의 지정된 저장점은 삭제된다.

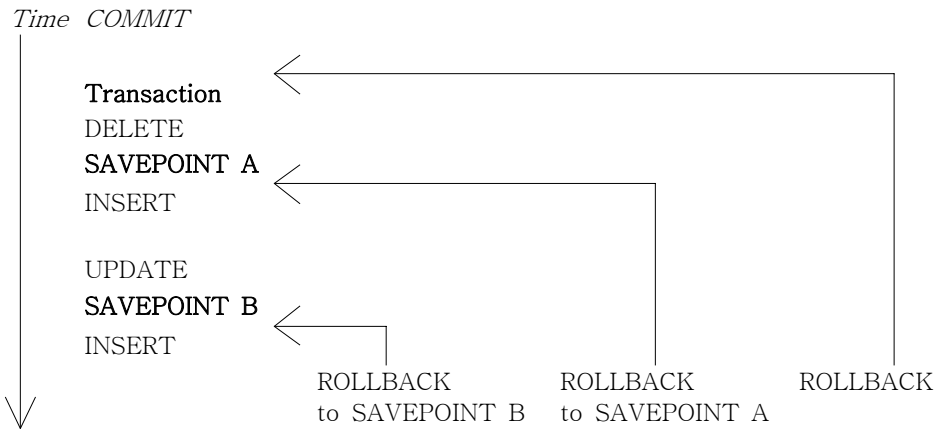

그림 9-1. 트랜잭션 제어 명령

■ 암시적 트랜잭션 처리(Implicit transaction processing)
암시적 트랜잭션에는 자동 커밋(Automatic commit)과 자동 롤백(Automatic rollback)이 있으며 다음과 같이 분류할 수 있다.

표 9-3. 암시적 트랜잭션 처리의 종류

종류	상황
Automatic commit	DDL 또는 DCL 문장의 실행시 SQL*Plus 또는 iSQL*Plus에서 COMMIT 또는 ROLLBACK 명령을 사용하지 않고 정상 종료
Automatic rollback	SQL*Plus 또는 iSQL*Plus에서 비정상 종료 또는 시스템 오류

위에서 SQL*Plus 또는 iSQL*Plus의 정상종료라 함은 SQL*Plus에서 EXIT 명령을 입력하거나 iSQL*Plus에서 로그아웃 링크를 클릭하여 종료하는 것을 의미한다.

■ 트랜잭션 종료 전의 데이터 상태
트랜잭션내의 모든 데이터 변경사항은 트랜잭션이 종료되기 이전까지는 임시적이다. COMMIT 또는 ROLLBACK 명령을 발생하기 전까지의 데이터 상태는 다음과 같다.

- 데이터 변경 작업은 주로 데이터베이스 버퍼에서 수행되므로, 데이터의 변경전 데이터는 복구 될 수 있다.
- 현재 사용자는 SELECT 문장을 이용하여 데이터 변경 후의 결과를 확인 할 수 있다.
- 다른 사용자들은 현재 사용자에 의해 변경된 데이터 결과를 확인할 수 없다. Oracle 데이터베이스는 다른 사용자들에게 해당 데이터의 가장 최근의 커밋된 결과를 보여줌으로써 데이터의 일관성을 보장한다.
- 변경된 행은 잠금(Lock)이 걸리며, 다른 사용자들은 해당 행들을 변경 할 수 없다.

COMMIT

COMMIT은 모든 데이터 변경사항을 데이터베이스에 영구히 반영시키는 명령으로 다음과

같은 작업이 수행된다.

· 데이터의 변경사항은 모두 데이터베이스에 기록된다.
· 변경 전의 데이터는 모두 잃게 된다.
· 모든 사용자들이 트랜잭션 종료 후의 결과를 볼 수 있다.
· 트랜잭션이 진행 중이었던 행들에 대한 잠금이 모두 해소되며, 다른 사용자에 의해서 변경이 가능해진다.
· 모든 SAVEPOINT는 삭제된다.

COMMIT 명령을 사용하는 방법은 다음과 같다.

```
SQL> INSERT INTO EMP
  2  VALUES (9003, '팥쥐', '상담원', 7698, '04/03/01', 3100, NULL, 30);

1 개의 행이 만들어졌습니다.

SQL> DELETE FROM DEPT
  2  WHERE DEPTNO = 92;

1 행이 삭제되었습니다.

SQL> COMMIT;

커밋이 완료되었습니다.
```

ROLLBACK

ROLLBACK은 모든 데이터 변경사항을 취소하는 명령어로 다음과 같은 작업이 수행된다.
· 데이터의 모든 변경 사항이 취소 된다.
· 변경 전의 데이터가 복원 된다.
· 트랜잭션이 진행 중이었던 행들에 대한 잠금이 모두 해소 된다,

ROLLBACK 명령을 사용하는 방법은 다음과 같다.

```
SQL> DELETE FROM EMP;

17 행이 삭제되었습니다.

SQL> ROLLBACK;

롤백이 완료되었습니다.
```

■ 문장 수준의 롤백
만약, DML 문장에 오류가 감지되면 트랜잭션의 일부분이 취소될 수 있다. 트랜잭션이 진행되는 중간에 단일 DML 문장이 실패하게 되면 해당 문장은 문장 수준의 롤백에 의해 취소되지만 해당 문장 이전의 DML 문장에 의해 발생된 변경사항은 취소되지 않는다.

한 가지 주의할 점으로 Oracle은 DDL 문장의 실행 전과 실행 후에 암시적 커밋을 발생시키는데, 해당 DDL 문장이 성공하지 못하더라고 해당 DDL 문장 전에 커밋이 발생하기 때문에 이전의 모든 데이터 변경사항은 롤백 되지 못한다.

읽기 일관성(Read Consistency)

데이터의 읽기 일관성이 필요한 이유는 다음과 같다.

- 데이터를 검색하는 사용자들과 변경하는 사용자들 사이에 일관적인 관점을 제공한다. 즉, 다른 사용자들이 변경 중인 데이터를 볼 수 없게 한다.
- 데이터를 변경하는 사용자들 사이에 일관적인 데이터베이스 변경 방법을 제공함으로써 동일한 데이터의 동시에 변경함으로써 발생 할 수 있는 혼란을 방지한다.

결과적으로 읽기 일관성의 목적은 각각의 사용자들에게 다른 사용자들의 DML 작업이 시작되기 이전의 데이터 즉, 가장 최근에 커밋 된 데이터를 보여주는 것이다.

■ 읽기 일관성의 구현 원리

읽기 일관성은 Oracle에 의해 자동으로 관리되며, 데이터의 변경 이전 값을 UNDO 세그먼트에 저장함으로써 구현된다. 즉, 데이터베이스에 INSERT, UPDATE, DELETE 문장이 실행되면 Oracle 데이터베이스는 변경 전 데이터를 UNDO 세그먼트에 임시적으로 저장한다. 이러한 상황에서 해당 데이터를 변경한 사용자를 제외한 모든 사용자들은 UNDO 세그먼트의 변경 전 데이터를 보게 된다.

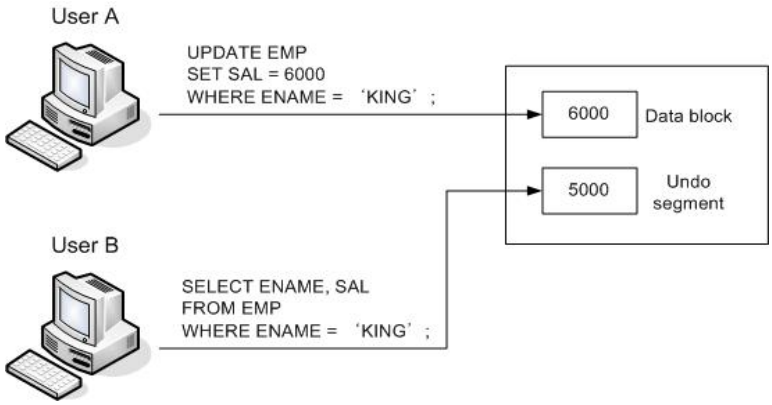

그림 9-2. 읽기 일관성의 원리

트랜잭션이 종료되기 전까지는 해당 데이터를 변경 중인 사용자만 변경 후의 결과를 보게 되며 다른 모든 사용자들은 모두 UNDO 세그먼트의 변경 전 데이터를 보게 된다. 즉, 이러한 메커니즘에 의해 사용자들은 가장 최근의 커밋된 데이터만을 보게 됨으로써 읽기 일관성이 유지된다.

DML 문장이 커밋되면 모든 변경사항은 데이터베이스에 영구히 반영되므로 모든 사용자들은 변경 후 데이터를 볼 수 있게 되며 UNDO 세그먼트에서 변경 전 데이터가 저장된 공간은 다른 트랜잭션을 위해 재활용 된다.

만약, 트랜잭션이 취소되면 모든 변경사항은 다음과 같은 작업에 의해 취소된다.
· UNDO 세그먼트내의 변경 전 데이터는 다시 테이블에 기록된다.
· 모든 사용자는 트랜잭션이 시작되기 이전의 데이터를 보게 된다.

잠금(Lock)

잠금이란 동일한 데이터를 변경하고 있는 트랜잭션 간에 발생 할 수 있는 데이터의 혼란을 사전에 예방하기 위한 메커니즘으로 Oracle 데이터베이스에 의해서 자동으로 수행되며 사용자의 개입이 불필요하다. 여기서, Oracle 데이터베이스에 의해 자동으로 진행되는 잠금을 암시적 잠금(Implicit locking)이라고 하며 SELECT를 제외한 모든 문장에서 잠금이 발생한다. 또한, 사용자가 직접 데이터에 잠금을 설정하는 것을 명시적 잠금(Explicit locking)이라고 부른다.

■ 암시적 잠금의 종류
DML 작업을 수행하면 다음과 같은 잠금이 발생한다.

· 공유 잠금(Share lock)은 DML 문장이 진행중인 테이블에 설정되고, 모든 트랜잭션들은 동일한 데이터에 공유 잠금을 설정할 수 있다.
· 배타 잠금(Exclusive lock)은 DML 문장이 변경 중인 각각의 행에 대하여 설정되고, 트랜잭션이 종료되기 전까지 다른 트랜잭션에 의해서 데이터가 변경되는 것을 방지한다. 또한, 배타 잠금은 같은 데이터를 여러 사용자가 동시에 변경 할 수 없도록 하고 커밋되지 않은 데이터가 다른 사용자에 의해 다시 변경되는 것을 방지한다.
· DDL 잠금은 테이블과 같은 데이터베이스 객체를 변경할 때 발생한다.

복습

1. 부서 테이블에 부서코드가 99, 부서명은 '관리과', 위치는 '대구'인 행을 입력하시오.

2. 부서 테이블에 99번 부서의 부서명을 '회계과'로 변경하시오.

3. 부서 테이블의 99번 부서 행을 삭제하시오.

4. 트랜잭션이 자동 커밋되는 시점을 모두 고르시오.
   a. DDL 문장의 실행
   b. DCL 문장의 실행
   c. 시스템 오류
   d. SQL*Plus의 정상 종료

5. 현재 진행 중인 트랜잭션을 종료하고 모든 데이터 변경사항을 취소하는 명령어는 ( )이며, 데이터의 변경사항을 데이터베이스에 영구히 반영하는 명령어는 ( )이다.