제 8장 데이터 조작(DML)

제 9장. 데이터 조작

1) 목 적
2) 데이터 조작어(**DML 문**)
3) 테이블에 새로운 행 추가(**INSERT 문**)
4) 테이블로부터 기존의 행 변경(**UPDATE 문**)
5) 테이블로부터 기존의 행 삭제(**DELETE 문**)
6) 데이터베이스 트랜잭션
7) 요 약

1) 목 적

- 각 **DML** 문장을 기술.
- 테이블에 행을 삽입(**INSERT** 문).
- 테이블에 행을 갱신(**UPDATE** 문).
- 테이블로부터 행을 삭제(**DELETE** 문).
- 트랜잭션 제어

2) 데이터 조작어

❖ **DML** 문장을 실행하는 경우

- 테이블에 새로운 행을 추가.
- 테이블에 있는 기존의 행 변경.
- 테이블로부터 기존의 행 제거.

❖ **트랜잭션**은 작업의 논리적인 단위 형태인 **DML의 모음**으로 구성됨.

3) 테이블에 새로운 행 추가(삽입)

"...새로운 행을 **DEPT** 테이블에 삽입..."

DEPT 테이블

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERRATIONS	BOSTON

새로운 행

50	DEVELOPMENT	DETROIT

DEPT 테이블

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERRATIONS	BOSTON
50	DEVELOPMENT	DETROIT

3-1) INSERT 문장

❖ **INSERT 문장**을 사용하여 **테이블에 새로운 행을 추가** 함.

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

❖ 이 구문 형식은 **한번에 오직 하나의 행만이 삽입**됨.

3-2) 새로운 행 삽입

- 각각의 열에 대한 값을 포함하는 새로운 행을 삽입함.
- 테이블에 있는 열의 디폴트 순서로 값을 나열함.
- INSERT 절에서 열을 선택적으로 나열함.
- 문자와 날짜 값은 단일 따옴표(‘’) 내에 둠.

```
SQL> INSERT INTO dept (deptno, dname, loc)
     2> VALUES (50, ‘DEVELOPMENT’, ‘DETROIT’);
1 row created.
```

3-3) Null 값을 가진 새로운 행 추가

1) 암시적 방법

- 열 목록으로부터 열을 생략함.

```
SQL> INSERT INTO  dept (deptno, dname)
     2> VALUES  (60, 'MIS');
1 row created.
```

2) 명시적 방법

- NULL 키워드를 명시함.

```
SQL> INSERT INTO  dept
     2> VALUES  (70, 'FINANCE', NULL);
1 row created.
```

3-4) 특수 값 삽입

❖ SYSDATE 함수

- 현재 날짜와 시간을 기록함.

```
SQL> INSERT INTO   emp (empno, ename, job, mgr, hiredate, sal, comm, deptno)
    2> VALUES (7196, 'GREEN', 'SALESMAN', 7782, SYSDATE, 2000, NULL, 10);
1 row created.
```

3-5) 특정 날짜 값 삽입

❖ 새로운 종업원 추가 함.

```
SQL> INSERT INTO   emp
  2> VALUES (2296, 'AROMANO', 'SALESMAN', 7782, TODATE('FEB 3, 97', 'MON DD, YY'), 1300, NULL, 10);
1 row created.
```

❖ 추가 내용을 확인

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
2296	AROMANO	SALESMAN	7782	03-FEB-97	1300		10

3-6) 치환 변수를 사용하여 값 삽입

❖ SQL*PLUS 치환 매개 변수를 사용하여 상호작용 스크립트를 생성.

```
SQL> INSERT INTO   dept  (deptno, dname, loc)
     2> VALUES ( &department_id, '&department_name', '&location' );
1 row created.
```

```
Enter values for department_id : 80
Enter values for department_name : EDUCATION
Enter values for location : ATLANTA
1 row created.
```

3-8) 다른 테이블로부터 행 복사

- 서브쿼리로 INSERT 문장을 작성함.
- VALUES 절을 사용하지 않음.
- 서브쿼리의 열 수와 INSERT 절의 열수를 일치함.

```
SQL> INSERT INTO managers(id, name, salary, hiredate)
  2>          SELECT empno, ename, sal, hiredate
  3>          FROM   emp
  4>          WHERE  job = 'MANAGER';
```

4) 테이블에 있는 기존의 행 변경(갱신)

EMP

EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7566	JONES	MANAGER	20
…			

“ …EMP 테이블의 행을 갱신…”

EMP

EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	20
7566	JONES	MANAGER	20
…			

4-1) UPDATE 문장

❖ UPDATE 문장으로 기존의 행을 갱신함.

❖ 필요하다면 하나 이상의 행을 갱신함.

```
UPDATE    table
SET       column = value [, column = value]
[WHERE]   condition1;
```

4-2) 테이블 행 갱신

❖ 특정 열 or 행은 WHERE절을 명시함으로써 수정될 수 있음.

```
SQL> UPDATE emp
     2> SET        deptno = 20
     3> WHERE   empno = 7782;
1 row updated.
```

❖ WHERE절을 생략하면 테이블에 있는 모든 행이 수정됨.

```
SQL> UPDATE employee
     2> SET        deptno = 20
1 row updated.
```

4-3) 다중 열 서브 쿼리로 갱신

❖ 종업원 7499이 업무와 부서에 일치하도록

종업원 7698의 업무와 부서를 갱신함.

```
SQL> UPDATE    emp
  2> SET       (job, deptno) = ( SELECT job, deptno
  3>                             FROM   emp
  4>                             WHERE empno =7499)
  5> WHERE empno = 7698;
1 row updated.
```

4-4) 다른 테이블을 근거로 한 행 갱신

❖ UPDATE 문장에서 서브 쿼리를 사용함.

```
SQL> UPDATE   emp
    2>  SET         deptno = ( SELECT deptno
    3>                           FORM    emp
    4>                           WHERE empno = 7788)
    5> WHERE      job =      ( SELECT job
    6>                           FROM    emp
    7>                           WHERE empno = 7788);
2 rows updated.
```

5) 테이블로부터 기존의 행 제거(삭제)

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERRATIONS	BOSTON
50	DEVELOPMENT	DETROIT
60	MIS	
…		

“ …DEPT 테이블의 행을 삭제…”

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERRATIONS	BOSTON
60	MIS	
…		

5-1) DELETE 문장

❖ DELETE 문장을 사용하여 **테이블로부터 기존의 행을 제거**함.

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

5-2) 테이블로부터 행 삭제

❖ WHERE절을 명시하여 특정 행 or 행들을 삭제함.

```
SQL> DELETE FROM  department
  2>  WHERE       dname = 'DEVELOPMENT';
1 row deleted.
```

❖ WHERE절을 생략하면 테이블의 모든 행이 삭제됨.

```
SQL> DELETE FROM  department;
1 row deleted.
```

5-3) 다른 테이블을 근거로 한 행 삭제

❖ DELETE 문장에서 서브 쿼리를 사용함.

```
SQL> DELETE FROM    emp
  2> WHERE          deptno = ( SELECT  deptno
                               FROM    dept
                               WHERE   dname = 'SALES');
6 rows deleted.
```

6) 데이타베이스 트랜잭션

- 트랜잭션은 다음 문장 중 하나로 구성됨.
  - 데이터를 일관되게 변경하는 하나 이상의 DML 문장.
  - 하나의 DDL 문장.
  - 하나의 DCL 문장.
- 실행 가능한 첫 번째 SQL 문장이 실행될 때 시작함.
- 다음 이벤트 중 하나로 종료 됨.
  - COMMINT 또는 ROLLBACK
  - DDL 또는 DCL 문장 실행(자동 커밋)
  - 사용자 종료
  - 시스템 파손(crash)

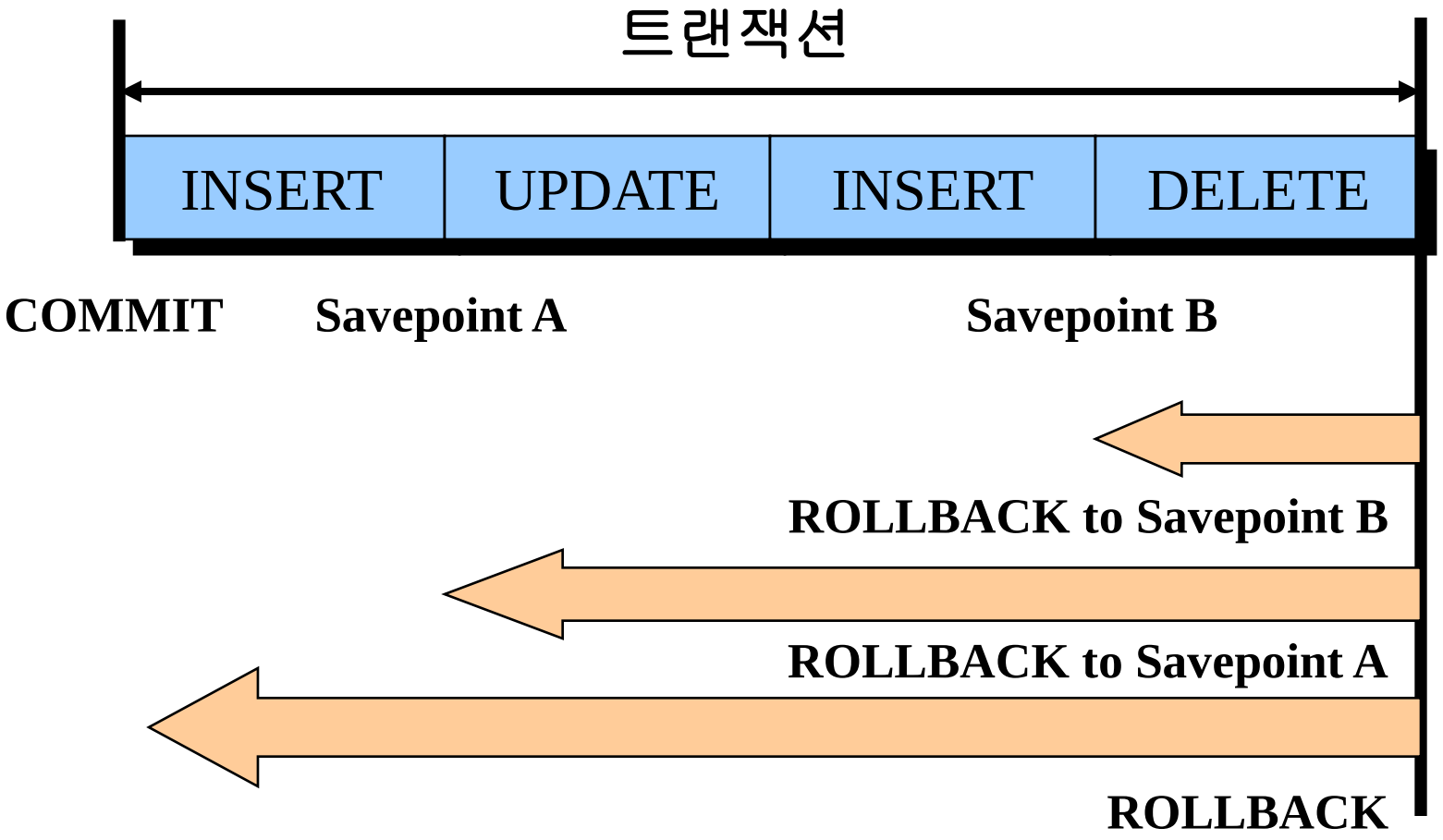
6-1) COMMIT과 ROLLBACK의 장점

❖ 데이터 일관성을 제공함.

❖ 데이터의 영구적으로 변경하기 전에
데이터 변경을 미리 보도록 함.

❖관련된 작업을 논리적으로 그룹화 함.

6-2) 트랜잭션 제어


트랜잭션
INSERT
UPDATE
INSERT
DELETE
COMMIT
Savepoint A
Savepoint B
ROLLBACK to Savepoint B
ROLLBACK to Savepoint A
ROLLBACK

6-3) 암시적 트랜잭션 처리

❖ 자동적인 COMMIT은 다음 환경 하에서 발생

- DDL 문장이 완료 시
- DCL 문장이 완료 시
- 명시적인 COMMIT 또는 ROLLBACK 없이 SQL*PLUS를 그냥 종료 시킬 때

❖ 자동적인 ROLLBACK은

SQL*PLUS를 비정상적으로 종료 or

시스템 실패 하에서 발생 함.

6-4) COMMIT 또는 ROLLBACK 이전의 데이터 상태

- 데이터의 이전 상태는 복구될 수 있음.
- 현재 사용자는 SELECT 문장을 사용하여 DML 작업의 결과를 검토할 수 있음.
- 다른 사용자들은 현재 사용자에 의한 DML 문장의 결과를 볼 수 없음.
- 변경된 행들은 잠금 상태(Locked) 임.
  다른 사용자들은 변경된 행들 내의 데이터를 변경할 수 없음.

6-5) COMMIT 이후의 데이터 상태

❖ 데이터베이스에 영구적으로 데이터를 변경함.

❖ 데이터의 이전 상태는 완전히 상실됨.

❖ 모든 사용자가 결과를 볼 수 있음.

❖ 변경된 행들의 잠금 상태가 해제됨.

=> 잠금 상태가 해제된 행들은

다른 사용자가 조작하기 위해 이용 가능함.

❖모든 savepoint가 제거됨.

6-6) 데이터 COMMIT 작업

❖ 변경을 함.

```
SQL> UPDATE  emp
  2>  SET          deptno = 10
  3> WHERE     empno = 7782;
1 row updated.
```

❖ 변경을 커밋함.

```
SQL> COMMIT;
Commit complete.
```

6-7) ROLLBACK 이후의 데이터 상태

❖ ROLLBACK 문장을 사용하여 모든 결정되지 않은 변경들을 버림.

- 데이터 변경이 취소됨.
- 데이터는 이전상태로 복구 됨.
- 변경된 행들의 잠금 상태가 해제됨.

```
SQL> DELETE FROM  employee;
14 rows deleted.
SQL> ROLLBACK;
Rollback complete.
```

6-8) 표시자(Marker)으로 변경을 롤백

❖ SAVEPOINT 문장을 사용하여

현재 트랜잭션 내에 표시자(Marker)를 생성함.

❖ ROLLBACK TO SAVEPOINT 문장을 사용하여

표시자(marker)까지 롤백 함.

```
SQL> UPDATE…
SQL> SAVEPOINT update_done;
Savepoint created.
SQL> INSERT…
SQL> ROLLBACK TO update_done;
Rollback complete.
```

6-9) 문장 레벨 롤백

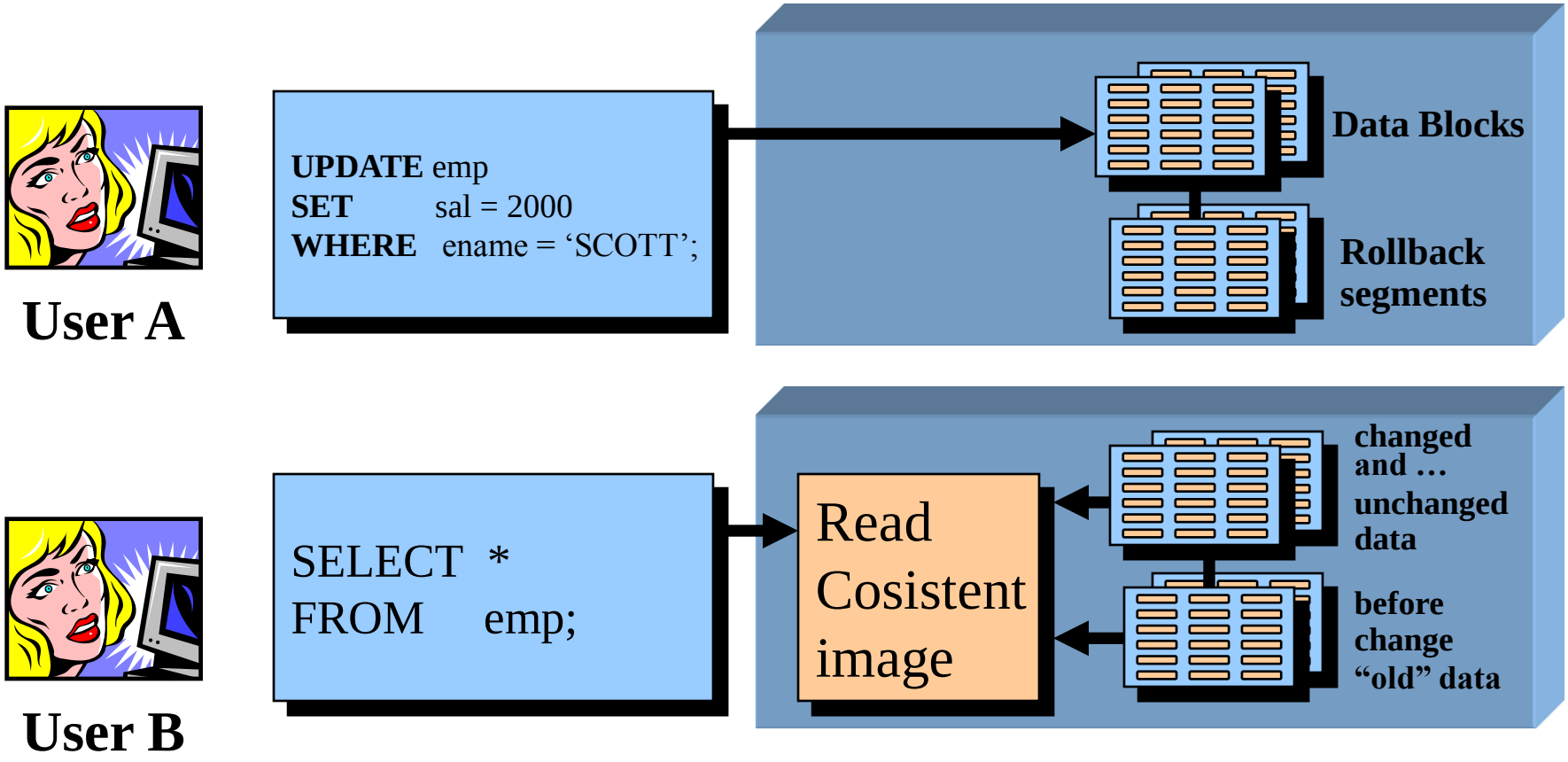
- 실행동안에 단일 DML문장이 실패하면, 단지 그 문장만을 롤백 함.
- 오라클 서버는 암시적인 savepoint를 구현함.
- 모든 다른 변경들은 보유 됨.
- 사용자는 COMMIT 또는 ROLLBACK 문장을 실행하여 명시적으로 트랜잭션을 종료함.

6-10) 읽기 일관성

- 읽기 일관성은 항상 데이터의 검색이 일관되게 보증함.
- 어떤 사용자에 의해 행해진 변경은

  다른 사용자에 의해 행해진 변경과 충돌하지 않음.
- 데이터를 똑같게 보증함.
  - Reader는 Writer에 대해서 기다리지 않음.
  - Writer는 Reader에 대해서 기다리지 않음.

6-11) 읽기 일관성 구현


User A
UPDATE emp
SET sal = 2000
WHERE ename = 'SCOTT';
Data Blocks
Rollback segments
User B
SELECT *
FROM emp;
Read Cosistent image
changed and … unchanged data
before change "old" data

6-12) 잠금(Locking)

- 동시 트랜잭션 사이의 상호 작용이 파괴를 막아 줌.
- 사용자 액션을 요구하지 않음.
- 자동적으로 가장 낮은 레벨이 제약 조건을 사용함.
- 트랜잭션이 지속되도록 함.
- 2 가지의 기본적인 모드를 가짐.
  - **Exclusive(독점)**
  - **Share(공유)**

7) 요 약

문 장	설 명
INSERT	테이블에 새로운 행을 추가(삽입)
UPDATE	테이블에 있는 기존의 행을 수정(갱신)
DELETE	테이블로부터 기존의 행을 제거(삭제)
COMMIT	모든 변경을 영구적으로 변경함.
SAVEPOINT	Savepoint marker 까지를 롤백
ROLLBACK	모든 확정되지 않은 변경를 바꿈.