



# 데이터 조작용-DML




배재대학교 컴퓨터공학과  
김 창 수

# Contents

 데이터 입력 - INSERT 문

 데이터 수정 - UPDATE문

 데이터 삭제 - DELETE문

 트랜잭션 관리

# 데이터 조작어

## ❖ 데이터 조작어 (DML:Data Manipulation Language)

- 테이블에 새로운 데이터를 입력하거나 기존 데이터를 수정 또는 삭제하기 위한 명령어

## ❖ 종류

- INSERT : 새로운 데이터 입력 명령어
- UPDATE : 기존 데이터 수정 명령어
- DELETE : 기존 데이터 삭제 명령어
- MERGE : 두개의 테이블을 하나의 테이블로 병합하는 명령어

## ❖ 트랜잭션

- 여러 개의 명령문을 하나의 논리적인 작업단위로 처리하는 기능
- 트랜잭션 관리 명령어
  - COMMIT : 트랜잭션의 정상적인 종료를 위한 명령어
  - ROLLBACK : 트랜잭션의 비정상적인 종단을 위한 명령어

# 데이터 입력

## ❖ 개요

- 테이블에 데이터를 입력하기 위한 명령인 INSERT 명령문 사용
- 데이터 입력 방법
  - 단일 행 입력 : 한번에 하나의 행을 테이블에 입력하는 방법
  - 다중 행 입력 : 서브쿼리를 이용하여 한번에 여러 행을 동시에 입력하는 방법

# 단일 행 입력

## ❖ 단일 행 입력 방법

- INTO 절에 명시한 칼럼에 VALUES 절에서 지정한 칼럼 값을 입력
- INTO 절에 칼럼을 명시하지 않으면 테이블 생성시 정의한 칼럼 순서와 동일한 순서로 입력
- 입력되는 데이터 타입은 칼럼의 데이터 타입과 동일해야 함
- 입력되는 데이터의 크기는 칼럼의 크기보다 작거나 동일해야 함
- CHAR, VARCHAR2, DATE 타입의 입력 데이터는 단일인용부호('')로 묶어서 입력

## ❖ 사용법

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

# 단일 행 입력 예

## ❖ 사용 예

- 학생 테이블에 홍길동 학생의 데이터를 입력하여라

```
SQL> desc student;
```

이름

성?

성

데이터 입력전에 칼럼이름, 칼럼순서, 제약 조건, 데이터 타입을 확인한다.

STUDNO

NUMBER(5)

NAME

VARCHAR2(10)

USERID

VARCHAR2(10)

GRADE

VARCHAR2(1)

IDNUM

VARCHAR2(13)

BIRTHDATE

DATE

TEL

VARCHAR2(13)

HEIGHT

NUMBER(5,2)

WEIGHT

NUMBER(5,2)

DEPTNO

NUMBER(4)

PROFNO

NUMBER(4)



# 단일 행 입력 예

```
SQL> INSERT INTO student
  2 VALUES(10110, '홍길동', 'hong', '1','8501011143098','85/01/01', '041)630-3114',
  3          170, 70, 101, 9903);
```

칼럼의 순서에 따라 데이터를 입력한다

1 개의 행이 만들어졌습니다.

```
SQL> SELECT studno, name
  2 FROM student
  3 WHERE studno = 10110;
```

STUDNO	NAME
--------	------

10110	홍길동
-------	-----

```
SQL> COMMIT;
```

커밋이 완료되었습니다.

데이터베이스에 영구적으로 데이터를 저장하기 위하여 COMMIT 명령문을 실행

# NULL의 입력

## ❖ NULL 입력

- 데이터를 입력하는 시점에서 해당 컬럼 값을 모르거나, 미확정
- 묵시적인 방법
  - INSERT INTO 절에 해당 칼럼 이름과 값을 생략
  - 해당 칼럼에 NOT NULL 제약조건이 지정된 경우 불가능
- 명시적 방법
  - VALUES 절의 칼럼 값에 NULL , '' 사용



# 묵시적으로 NULL을 입력하는 예

## ❖ 사용 예

- INSERT 명령문에서 묵시적인 방법을 이용하여 부서 테이블의 부서 번호와 부서 이름을 입력하고 나머지 칼럼은 NULL을 입력하여라.

```
SQL> INSERT INTO department(deptno, dname)
2 VALUES (300, '생명공학부');
```

1 개의 행이 만들어졌습니다.

```
SQL> COMMIT;
```

커밋이 완료되었습니다.

```
SQL> SELECT *
2 FROM department
3 WHERE deptno = 300;
```

DEPTNO	DNAME	COLLEGE	LOC
300	생명공학부		

deptno, dname 칼럼을 제외한 나머지 칼럼은 NULL이 입력된다

# 명시적으로 NULL을 입력하는 예

## ❖ 사용 예

- INSERT 명령문에서 명시적인 방법을 이용하여 부서 테이블의 부서 번호와 부서 이름을 입력하고 나머지 칼럼은 NULL을 입력하여라.

```
SQL> INSERT INTO department  
2 VALUES (301, '환경보건학과', '', '');
```

1 개의 행이 만들어졌습니다.

```
SQL> SELECT *  
2 FROM department  
3 WHERE deptno = 301;
```

DEPTNO	DNAME	COLLEGE	LOC
301	환경보건학과		

# 날짜 데이터 입력 방법

## ❖ NULL 입력

- 해당 시스템에서 요구하는 기본 날짜 형식으로 입력
- UNIX 기본 날짜 형식 : 'DD-MON-YY'
- 퍼스널 오라클 : 'YY/MM/DD'
- 필요에 따라서 TO\_DATE 함수 사용

# 날짜 형식 입력하는 예

## ❖ 사용 예

- 교수 테이블에서 입사일을 2013년 3월1일로 입력하여라.

```
SQL> insert into professor(profno, name, position, hiredate, deptno)
  2  values(9920, '김창수', '조교수',
  3         to_date('2013/03/01', 'yyyy/mm/dd'), 102);
```

1 개의 행이 만들어졌습니다.

```
SQL> commit;
```

커밋이 완료되었습니다.

```
SQL> select *
  2  from professor
  3  where profno = 9920;
```

hiredate의 입력 형식과 상관없이 출력 형식은 기본 날짜 형식으로 출력된다

PROFNO	NAME	USERID	POSITION	SAL	HIREDATE
9920	김창수		조교수		13/03/01
	102				

# SYSDATE 함수를 이용한 현재 날짜 입력

## ❖ SYSDATE 함수

- 현재 시점의 날짜 값을 자동적으로 입력
- 시스템에 저장된 현재 날짜 데이터를 반환하는 함수

## ❖ 사용 예

- 교수 테이블에서 새로운 행을 입력할 때 입사일을 현재 날짜로 입력 하여라.

```
SQL> insert into professor
2 values(9910,'정 회 경','hkjung','교수',200,SYSDATE,10,101);
```

1 개의 행이 만들어졌습니다.

```
SQL> commit;
```

커밋이 완료되었습니다.

```
SQL> select *
2 from professor
3 where profno = 9910;
```

PROFNO	NAME	USERID	POSITION	SAL	HIREDATE
9910	정 회 경	hkjung	교수	200	16/02/01

# 다중 행 입력

## ❖ 다중 행 입력 방법

- INSERT 명령문에서 서브쿼리 절을 이용
- INSERT 명령문에 의해 한번에 여러 행을 동시에 입력
- 9i 버전 이후 부터
  - unconditional INSERT ALL
  - conditional INSERT ALL
  - conditional FIRST INSERT
  - pivoting INSERT지원

# 단일 테이블에 다중 행 입력

## ❖ 단일 테이블에 다중 행 입력 방법

- INSERT 명령문에서 서브쿼리 절을 이용하여 자신이나 다른 테이블에 데이터를 복사하여 여러 행 동시 입력
- INSERT 명령문의 VALUES절 대신 서브쿼리에서 검색된 결과 집합을 한꺼번에 입력
- 서브쿼리의 결과 집합은 INSERT 명령문에 지정된 칼럼 개수와 데이터 타입이 일치해야 함
- 서브쿼리를 이용한 다중 행 입력시 테이블에 기본 키, 고유 키 제약조건이 중복되지 않도록 주의
- 제약 조건을 위반할 경우 입력되지 않고 오류 발생

## ❖ 사용법

```
INSERT INTO table [(column1, column2,...)]  
subquery;
```

# 다중 행 입력 - INSERT ALL

## ❖ INSERT ALL(unconditional INSERT ALL) 명령문

- 서브쿼리의 결과 집합을 조건없이 여러 테이블에 동시에 입력
- 서브쿼리의 컬럼 이름과 데이터가 입력되는 테이블의 칼럼이 반드시 동일해야 함

## ❖ 사용법

```
INSERT ALL | FIRST  
INTO [table1] VLAUES[(column1, column2,...)]  
INTO [table2] VLAUES[(column1, column2,...)]  
INTO [table3] VLAUES[(column1, column2,...)]  
subquery;
```

- ALL : 서브쿼리의 결과 집합을 해당하는 INSERT절에 모두 입력
- FIRST : 서브쿼리의 결과 집합을 해당하는 첫번째 INSERT절에 입력
- subquery : 입력 데이터 집합을 정의하기 위한 서브쿼리



# 다중 행 입력 - INSERT ALL 예

## ❖ 다중 행 입력을 위한 height\_info, weight\_info 예제 테이블 생성

```
SQL> CREATE TABLE height_info (  
2 studno    number(5),  
3 name      varchar2(10),  
4 height    number(5,2));
```

테이블이 생성되었습니다.

```
SQL> CREATE TABLE weight_info (  
2 studno    number(5),  
3 name      varchar2(10),  
4 height    number(5,2));
```

테이블이 생성되었습니다.

# 다중 행 입력 - INSERT ALL 예

## ❖ 사용 예

- 학생 테이블에서 2학년 이상의 학생을 검색하여 height\_info 테이블에는 학번, 이름, 키, weight\_info 테이블에는 학번, 이름, 몸무게를 각각 입력하여라.

```
SQL> INSERT ALL
  2 INTO height_info VALUES (studno, name, height)
  3 INTO weight_info VALUES (studno, name, weight)
  4 SELECT studno, name, height, weight
  5 FROM student
  6 WHERE grade >= '2';
```

20 개의 행이 만들어졌습니다.

```
SQL> COMMIT;
```

커밋이 완료되었습니다.

```
SQL> SELECT *
  2 FROM height_info;
```

STUDNO	NAME	HEIGHT
10101	이민준	176
10103	이민준	170
10201	이민준	164
10104	이민준	161
10202	이민준	177
10105	이민준	171
10204	이민준	171
10107	이민준	175
20103	이민준	166
10108	이민준	162

# 다중 행 입력 - Conditional INSERT ALL

## ❖ Conditional INSERT ALL 명령문

- 서브쿼리의 결과 집합에 대해 WHEN 조건절에서 지정한 조건을 만족하는 행을 해당되는 테이블에 각각 입력
- 서브쿼리에서 검색된 행을 만족하는 조건이 여러 개 일 경우 해당 테이블에 모두 입력
- ALL : WHEN~THEN~ELSE의 조건을 만족하는 서브쿼리의 모든 검색 결과를 입력하기 위한 옵션
- WHEN 조건절 THEN : 서브쿼리의 결과 집합에 대한 비교 조건
- 서브쿼리의 결과 집합 중에서 조건절1을 만족하는 결과 행은 table1에 입력, 조건절 2을 만족하는 결과 행은 table2에 입력, 그리고 어느 조건절도 만족하지 않는 행은 table3에 입력

# 다중 행 입력 - Conditional INSERT ALL

## ❖ 사용법

```
INSERT ALL  
[WHEN 조건절1 THEN  
INTO [table1] VLAUES[(column1, column2,...)]  
[WHEN 조건절2 THEN  
INTO [table2] VLAUES[(column1, column2,...)]  
[ELSE  
INTO [table3] VLAUES[(column1, column2,...)]  
subquery;
```

# 다중 행 입력 - Conditional INSERT ALL 예

## ❖ weight\_info, height\_info 테이블 데이터 모두 삭제

```
SQL> DELETE FROM height_info;
```

10 행이 삭제되었습니다.

```
SQL> DELETE FROM weight_info;
```

10 행이 삭제되었습니다.

```
SQL> COMMIT;
```

커밋이 완료되었습니다.

```
SQL> SELECT * FROM height_info;
```

선택된 레코드가 없습니다.

```
SQL> SELECT * FROM weight_info;
```

선택된 레코드가 없습니다.

# 다중 행 입력 - Conditional INSERT ALL 예

## ❖ 사용 예

- 학생 테이블에서 2학년 이상의 학생을 검색하여 height\_info 테이블에는 키가 170보다 큰 학생의 학번, 이름, 키를 입력하고 weight\_info 테이블에는 몸무게가 70보다 큰 학생의 학번, 이름, 몸무게를 각각 입력하여라.

```
SQL> INSERT ALL
2  WHEN height > 170 THEN
3      INTO height_info VALUES (studno, name, height)
4  WHEN weight > 70 THEN
5      INTO weight_info VALUES (studno, name, weight)
6  SELECT studno, name, height, weight
7  FROM student
8  WHERE grade >= '2';
```

10 개의 행이 만들어졌습니다.

# 다중 행 입력 – Conditional-First INSERT

## ❖ Conditional-First INSERT 명령문

- 서브쿼리의 결과 집합에 대해 WHEN 조건절에서 지정한 조건을 만족하는 첫번째 테이블에 우선적으로 입력하기 위한 명령문
- 서브쿼리의 결과 집합중에서 조건을 만족하는 첫 번째 WHEN절에서 지정한 테이블에만 입력하고 나머지 WHEN절 무시

## ❖ 사용법

```
INSERT FIRST  
[WHEN 조건절1 THEN  
INTO [table1] VLAUES[(column1, column2,...)]  
[WHEN 조건절2 THEN  
INTO [table2] VLAUES[(column1, column2,...)]  
[ELSE  
INTO [table3] VLAUES[(column1, column2,...)]  
subquery;
```

# 다중 행 입력 – Conditional-First INSERT 예

## ❖ weight\_info, height\_info 테이블 데이터 모두 삭제

```
SQL> DELETE FROM height_info;
```

10 행이 삭제되었습니다.

```
SQL> DELETE FROM weight_info;
```

10 행이 삭제되었습니다.

```
SQL> COMMIT;
```

커밋이 완료되었습니다.

```
SQL> SELECT * FROM height_info;
```

선택된 레코드가 없습니다.

```
SQL> SELECT * FROM weight_info;
```

선택된 레코드가 없습니다.



# 다중 행 입력 – Conditional-First INSERT 예

## ❖ 사용 예

- 학생 테이블에서 2학년 이상의 학생을 검색하여 height\_info 테이블에는 키가 170보다 큰 학생의 학번, 이름, 키를 입력하고 weight\_info 테이블에는 몸무게가 70보다 큰 학생의 학번, 이름, 몸무게를 각각 입력하여라. 단, 키가 170보다 작고, 몸무게가 70보

```
SQL> INSERT FIRST
  2  WHEN height > 170 THEN
  3      INTO height_info VALUES (studno, name, height)
  4  WHEN weight > 70 THEN
  5      INTO weight_info VALUES (studno, name, weight)
  6  SELECT studno, name, height, weight
  7  FROM    student
  8  WHERE   grade >= '2';
```

# 다중 행 입력 – Conditional-First INSERT 예

```
SQL> SELECT * FROM weight_info;
```

STUDNO	NAME	HEIGHT
10103	김영균	88
10108	류민정	72

```
SQL> SELECT * FROM height_info;
```

STUDNO	NAME	HEIGHT
10101	전인하	176
10202	오유석	177
10105	임유진	171
10204	윤진욱	171
10107	이광훈	175

전인하, 오유석, 이광훈 학생은 몸무게가 70보다 크지만 height\_info 테이블에만 입력된다



# 다중 행 입력 – PIVOTING INSERT

## ❖ PIVOTING INSERT 명령문

- OLTP(OnLine Transaction Processing) 업무에서 사용되는 데이터를 데이터웨어하우스 업무에서 사용되는 분석용 데이터로 변환하는 경우에 유용
- 하나의 행을 여러 개의 행으로 나누어서 입력하는 기능
- Unconditional INSERT ALL 명령문과 거의 동일
- INTO 절에서 하나의 테이블만 지정
- 예를 들면, 5개의 칼럼으로 구성된 요일별 판매 실적 데이터를 하나의 칼럼으로 통합할때 하나의 칼럼으로 통합된 판매 데이트의 요일을 구분하기 위하여 요일 구분 칼럼을 추가

# 다중 행 입력 – PIVOTING INSERT

## ❖ 사용 예

- PIVOTING INSERT 를 실습하기 위한 예제 테이블

```
SQL> CREATE TABLE sales (  
2 sales_no number(4),  
3 week_no number(2),  
4 sales_mon number(7,2),  
5 sales_tue number(7,2),  
6 sales_wed number(7,2),  
7 sales_thu number(7,2),  
8 sales_fri number(7,2));
```

테이블이 생성되었습니다.

```
SQL> INSERT INTO sales VALUES(1101, 4, 100, 150, 80, 60, 120);
```

1 개의 행이 만들어졌습니다.

```
SQL> INSERT INTO sales VALUES(1102, 5, 300, 300, 230, 120, 150);
```

1 개의 행이 만들어졌습니다.

```
SQL> CREATE TABLE sales_data (  
2 sale_no number(4),  
3 week_no number(2),  
4 day_no number(2),  
5 sales number(7,2));
```

# 다중 행 입력 – PIVOTING INSERT

## ❖ 사용 예

- PIVOTING INSERT 명령문을 사용하여 SALES 테이블의 요일별 데이터를 통합하여 SALES\_DATA 테이블에 하나의 행으로 입력하라.

```
SQL> INSERT ALL
  2 INTO sales_data VALUES(sales_no, week_no, '1', sales_mon)
  3 INTO sales_data VALUES(sales_no, week_no, '2', sales_tue)
  4 INTO sales_data VALUES(sales_no, week_no, '3', sales_wed)
  5 INTO sales_data VALUES(sales_no, week_no, '4', sales_thu)
  6 INTO sales_data VALUES(sales_no, week_no, '5', sales_fri)
  7 SELECT sales_no, week_no, sales_mon, sales_tue, sales_wed,
  8        sales_thu, sales_fri
  9 FROM sales;
```

# 다중 행 입력 – PIVOTING INSERT

```
SQL> SELECT * FROM sales;
```

SALES_NO	WEEK_NO	SALES_MON	SALES_TUE	SALES_WED	SALES_THU	SALES_FRI
1101	4	100	150	80	60	120
1102	5	300	300	230	120	150

```
SQL> SELECT * FROM sales_data
2 ORDER BY sale_no;
```

SALE_NO	WEEK_NO	DAY_NO	SALES
1101	4	1	100
1101	4	2	150
1101	4	4	60
1101	4	5	120
1101	4	3	80
1102	5	1	300
1102	5	2	300
1102	5	4	120
1102	5	5	150
1102	5	3	230

# 데이터 수정

## ❖ 데이터 수정 개요

- UPDATE 명령문은 테이블에 저장된 데이터 수정을 위한 조작용어
- WHERE 절을 생략하면 테이블의 모든 행을 수정

## ❖ 사용법

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

- WHERE 절을 생략하면 테이블의 모든 행을 수정
- Condition : 컬럼이름 , 표현식, 상수, 서브쿼리, 비교 연산자

# 데이터 수정 예

## ❖ 사용 예

- 교수 번호가 9903인 교수의 현재 직급을 '부교수' 로 수정하여라

```
SQL> SELECT profno, name, position
2   FROM professor
3   WHERE profno = 9903;
```

PROFNO	NAME	POSITION
9903	성연희	조교수

```
SQL> UPDATE professor
2   SET position = '부교수'
3   WHERE profno = 9903;
```

1 행이 갱신되었습니다.

```
SQL> COMMIT;
```

커밋이 완료되었습니다.

```
SQL> SELECT profno, name, position
2   FROM professor
3   WHERE profno = 9903;
```

PROFNO	NAME	POSITION
9903	성연희	부교수



# 서브쿼리를 이용한 데이터 수정

## ❖ 서브쿼리를 이용한 데이터 수정 개요

- UPDATE 명령문의 SET 절에서 서브쿼리를 이용
- 다른 테이블에 저장된 데이터 검색하여 한꺼번에 여러 칼럼수정
- SET 절의 칼럼 이름은 서브쿼리의 칼럼 이름과 달라도 됨
- 데이터 타입과 칼럼 수는 반드시 일치

## ❖ 사용법

```
UPDATE table1
SET (column1, column2, ...)=(SELECT s_column1, s_column2, ...
                              FROM table2
                              [WHERE condition2])
[WHERE condition1];
```

# 서브쿼리를 이용한 데이터 수정 예

## ❖ 사용 예

- 서브쿼리를 이용하여 학번이 10201인 학생의 학년과 학과 번호를 10103 학번 학생의 학년과 학과 번호와 동일하게 수정하여라.

```
SQL> SELECT studno, grade, deptno  
2 FROM student  
3 WHERE studno = 10201;
```

STUDNO	G	DEPTNO
--------	---	--------

10201	2	102
-------	---	-----

```
SQL> SELECT studno, grade, deptno  
2 FROM student  
3 WHERE studno = 10103;
```

STUDNO	G	DEPTNO
--------	---	--------

10103	3	101
-------	---	-----

# 데이터 삭제

## ❖ 데이터 삭제 개요

- DELETE 명령문은 테이블에 저장된 데이터 삭제를 위한 조작어
- WHERE 절을 생략하면 테이블의 모든 행 삭제

## ❖ 사용법

```
DELETE [FROM] table  
[WHERE condition1];
```

# 단일 행 삭제 예

## ❖ 사용 예

- 학생 테이블에서 학번이 20103인 학생의 데이터를 삭제하여라.

```
SQL> DELETE  
  2 FROM student  
  3 WHERE studno = 20103;
```

1 행이 삭제되었습니다.

```
SQL> COMMIT;
```

커밋이 완료되었습니다.

```
SQL> SELECT *  
  2 FROM student  
  3 WHERE studno = 20103;
```

선택된 레코드가 없습니다.

학번이 20103인 행이 삭제되어 출력 결과가 없다

# 서브쿼리를 이용한 데이터 삭제

## ❖ 서브쿼리를 이용한 데이터 삭제 개요

- WHERE 절에서 서브쿼리 이용
- 다른 테이블에 저장된 데이터를 검색하여 한꺼번에 여러행의 내용을 삭제 함
- WHERE 절의 칼럼 이름은 서브쿼리의 칼럼 이름과 달라도 됨
- 데이터 타입과 칼럼 수는 일치

## ❖ 사용법

```
DELETE FROM table
WHERE (column1, column2, ...)=(SELECT s_column1, s_column2, ...
                                FROM table2
                                [WHERE condition2]);
```

# 서브쿼리를 이용한 데이터 삭제 예

## ❖ 사용 예

- 학생 테이블에서 컴퓨터공학과에 소속된 학생을 모두 삭제하여라.

```
SQL> DELETE FROM student
2 WHERE deptno = (SELECT deptno
3                 FROM department
4                 WHERE dname = '컴퓨터공학과');
```

10 행이 삭제되었습니다.

```
SQL> SELECT *
2 FROM student
3 WHERE deptno = (SELECT deptno
4                 FROM department
5                 WHERE dname = '컴퓨터공학과');
```

선택된 레코드가 없습니다.

컴퓨터공학과 학생 데이터가 모두 삭제되었다.

# MERGE

## ❖ MERGE 개요

- 구조가 같은 두개의 테이블을 비교하여 하나의 테이블로 합치기 위한 데이터 조작어
- WHEN 절의 조건절에서 결과 테이블에 해당 행이 존재하면 UPDATE 명령문에 의해 새로운 값으로 수정, 그렇지 않으면 INSERT 명령문으로 새로운 행을 삽입
- 대량의 데이터를 분석하기 위한 업무에 유용
- 예를 들면,
  - 전자상거래 회사에서 하루 수만건의 데이터를 평소에는 판매 데이터를 월 단위로 분리하여 별도의 테이블에서 관리하다가 연말에 판매 실적 분석을 위해 하나의 테이블로 합치는 경우

# MERGE 사용법

## ❖ 사용법

```
MERGE INTO [table][alias]
USING [table | view | subquery]alias
ON[join condition]
WHEN MATCHED THEN
    UPDATE SET .....
WHEN NOT MATCHED THEN
    INSERT INTO .....
    VALUES .....;
```

- MERGE INTO : 하나의 테이블로 합치기 위한 결과 테이블
- USING : 테이블, 뷰, 서브쿼리에 대한 별명 지정
- ON : 조인 조건 지정
- WHEN MATCHED THEN : ON 절의 조인 조건을 만족하는 행 존재하면지정된 값으로 행을 UPDATE
- WHEN NOT MATCHED THEN:ON 절의 조인 조건을 만족하지 않을 경우 새로운 행으로 INSERT
- WHEN MATCHED THEN 절과 WHEN NOT MATCHED THEN 절에서는 테이블이나 뷰 이름 대신에 USING 절에서 지정한 별명 사용



# MERGE 사용 예

## ❖ 사용 예

- professor 테이블과 professor\_temp 테이블을 비교하여 professor 테이블에 있는 기존 데이터는 professor\_temp 테이블의 데이터에 의해 수정하고, professor 테이블에 없는 데이터는 신규로 입력한다.

```
SQL> CREATE TABLE professor_temp AS  
2 SELECT *  
3 FROM professor  
4 WHERE position = '교수';
```

professor 테이블에서 직급이 '교수' 인 데이터를 검색하여 professor\_temp 테이블에 저장

테이블이 생성되었습니다.

```
SQL> UPDATE professor_temp  
2 SET position = '명예교수'  
3 WHERE position = '교수';
```

professor\_temp 테이블의 '교수' 직급을 '명예교수' 로 수정

2 행이 갱신되었습니다.

```
SQL> INSERT INTO professor_temp  
2 VALUES(9999, '김도경', 'arom21',
```

professor\_temp 테이블에 새로운 데이터 입력

1 개의 행이 만들어졌습니다.

# MERGE 사용 예

## ❖ 사용 예

```
SQL> MERGE INTO professor p
2 USING professor_temp f
3 ON (p.profno = f.profno)
4 WHEN MATCHED THEN
5     UPDATE SET p.position = f.position
6 WHEN NOT MATCHED THEN
7     INSERT VALUES(f.profno, f.name, f.userid, f.position,
8                   f.sal, f.hiredate, f.comm, f.deptno);
```

professor 테이블과 professor\_temp 테이블을 병합한다.

3 행이 병합되었습니다.

```
SQL> select distinct * from professor;
```

PROFNO	NAME	USERID	POSITION	SAL	HIREDATE	COMM	DEPTNO
9901	김도훈	capool	명예교수	500	82/06/24	20	101
9902	이재우	sweat413	조교수	320	95/04/12		201
9903	성영희	Pascal	부교수	360	93/05/17	15	101
9904	성영희	Blue77	전임강사	240	98/12/02		102
9905	권영일	refresh	명예교수	450	86/01/08	25	102
9906	이만식	Pocari	부교수	420	88/09/13		101
9907	전지현	totoro	전임강사	210	01/06/01		101
9908	강지현	Bird13	부교수	400	90/11/18	17	202
9910	백미선	white	전임강사	200	06/11/02	10	101
9920	최아진		조교수		06/01/01		102
9999	김도경	arom21	전임강사	200	06/11/05	10	101

11 개의 행이 선택되었습니다.



# 트랜잭션 관리

## ❖ 트랜잭션 개요

- 관계형 데이터베이스에서 실행되는 여러 개의 SQL 명령문을 하나의 논리적 작업 단위로 처리하는 개념
- COMMIT : 트랜잭션의 정상적인 종료
- ROLLBACK : 트랜잭션의 전체 취소

### 명시적인 트랜잭션 제어 명령문

명령문	의미
COMMIT	트랜잭션내의 모든 SQL 명령문에 의해 변경된 작업 내용을 디스크에 영구적으로 저장하고 트랜잭션을 종료
ROLLBACK	트랜잭션내의 모든 SQL 명령문에 의해 변경된 작업 내용을 전부 취소하고 트랜잭션을 종료

# COMMIT

## ❖ COMMIT 개요

- 하나의 트랜잭션에서 실행되는 모든 SQL 명령문의 처리 결과가 하드디스크에 안전하게 보장되는 것을 보장
- 처리 결과를 디스크에 영구적으로 저장
- 해당 트랜잭션에 할당된 CPU, 메모리 같은 자원이 해제
- 서로 다른 트랜잭션을 구분하는 기준
- COMMIT 명령문 실행하기 전에 하나의 트랜잭션 변경한 결과를 다른 트랜잭션에서 접근할 수 없도록 방지하여 일관성 유지

# ROLLBACK

## ❖ ROLLBACK 개요

- 하나의 트랜잭션에서 실행된 SQL 명령문의 처리결과를 취소
- CPU,메모리 같은 해당 트랜잭션에 할당된 자원을 해제, 트랜잭션을 강제 종료