Chapter 7

DML(Data Manipulation Language)

INSERT

UPDATE

DELETE

MERGE

CTAS

- DML 문장은 다음의 경우에 실행됩니다.
  - 테이블에 새로운 행을 추가할 때
  - 테이블에 있는 기존의 행을 변경할 때
  - 테이블로부터 기존의 행을 제거할 때

DML(Data Manipulation Language)은 SQL의 핵심 부분입니다. 데이터베이스에 데이터를 추가, 갱신 또는 삭제하고자 한다면 DML 문장을 실행합니다.

DML 문은 다음 3가지 구문을 포함합니다.

- 새로운 행을 데이터베이스에 추가하는 INSERT 문
- 기존 행을 수정하기 위한 UPDATE 문
- 기존 행을 삭제하기 위한 DELETE 문

DML 구문의 실행을 완전히 보장해야 할 필요가 있습니다. 이를 위해 트랜잭션이 필요합니다. 작업의 논리적인 단위 형태인 DML 문장의 모음을 트랜잭션이라고 합니다.

예를 들어 은행 데이터베이스를 고려해 봅시다. 은행 고객이 저축성 예금을 당좌 예금으로 전달할 때, 트랜잭션은 세 가지의 분리되는 작업으로 구성됩니다.

- 저축성 예금을 감소시킨다.
- 당좌 예금을 증가시킨다.
- 트랜잭션 일지에 트랜잭션을 기록한다.

오라클 서버는 올바르게 예금을 유지하기 위해 세 가지 SQL 문장 모두가 수행되도록 해야 합니다. 누군가가 실행 시에 트랜잭션의 문장 중의 하나를 막아 버린다면, 그 트랜잭션 내의 다른 문장은 취소되어야 합니다. 예를 들며 저축성 예금을 감소시키는 것이 성공한 다음 당좌예금을 증가시키는 도중 에러가 발생하여 당좌예금의 증가가 실패했다면 이미 실행된 저축성 예금의 감소를 취소해야 합니다.

이 장에서는 데이터 조작에 대해서만 설명합니다. 트랜잭션은 9장에서 자세하게 설명합니다.

Chapter 7

DML(Data Manipulation Language)

INSERT

UPDATE

DELETE

MERGE

CTAS

새로운 행

270	Data Analytics	200	1700

"새로운 행을 DEPARTMENT 테이블에 삽입..."

DEPTARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MAMAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
...	...	...	...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MAMAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
...	...	...	...
270	Data Analytics	200	1700

INSERT 문장을 실행하여 테이블에 새로운 행을 추가할 수 있습니다. VALUES 절을 가지는 이 문장은 테이블에 한번에 오직 하나의 행만을 추가합니다.

```
INSERT INTO table [(column1 [, column2, ... ])]
VALUES             (value1 [, value2, ... ]);
```

```
SQL> DESC departments;
 이름              널       유형
 --------------- -------- ------------
 DEPARTMENT_ID   NOT NULL NUMBER(4)
 DEPARTMENT_NAME NOT NULL VARCHAR2(30)
 MANAGER_ID               NUMBER(6)
 LOCATION_ID              NUMBER(4)
```

EMPLOYEES

열 | 데이터 | Model | 제약 조건 | 권한 부여 | 통계 | 트리거 | 플래시백 | 종속성 | 세부 정보 | 분할 영역 | 인덱스 | SQL

작업...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	Primary key of employees table.
2	FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	First name of the employee. A no
3	LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3	Last name of the employee. A not
4	EMAIL	VARCHAR2(25 BYTE)	No	(null)	4	Email id of the employee
5	PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5	Phone number of the employee; in
6	HIRE_DATE	DATE	No	(null)	6	Date when the employee started o
7	JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7	Current job of the employee; for
8	SALARY	NUMBER(8,2)	Yes	(null)	8	Monthly salary of the employee.
9	COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9	Commission percentage of the emp
10	MANAGER_ID	NUMBER(6,0)	Yes	(null)	10	Manager id of the employee; has
11	DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11	Department id where employee wor

- 각각의 열에 대한 값을 포함하는 새로운 행을 삽입합니다.
- 테이블에 있는 열의 디폴트 순서로 값을 나열합니다.
- INSERT 절에서 열을 선택적으로 나열합니다.
- 문자와 날짜 값은 단일 따옴표 내에 둡니다.

다음 구문은 DEPARTMENTS 테이블에 새로운 행을 추가합니다. 새로 추가되는 부서의 아이디는 280, 부서이름은 Data Analytics, 매니저 아이디는 Null, 지역 아이디는 1700(미국 시카고)입니다.

```
SQL> INSERT INTO departments
  2  VALUES (280, 'Data Analytics', null, 1700);
1 행 이(가) 삽입되었습니다.
```

새로운 행을 삽입할 때 열을 지정하지 않을 경우 VALUES 절에는 테이블을 정의할 때의 순서와 타입에 맞게 모든 열들에 대한 입력정보가 포함되어 있어야 합니다.

다음 작업을 위해 ROLLBACK; 명령으로 삽입했던 행의 저장을 취소하세요.

Null을 포함하는 열이 있을 경우 다음과 같이 선택적으로 열을 지정할 수 있습니다. 위 구문과 아래 구문의 같습니다. 위 구문이 실행되었다만 다음 구문을 실행할 필요 없습니다.

```
SQL> INSERT INTO departments
  2    (department_id, department_name, location_id)
  3  VALUES
  4    (280, 'Data Analytics', 1700);
```

다음 구문을 통해 280번 부서정보가 입력되었는지 확인해 보세요.

```
SQL> SELECT *
  2  FROM departments
  3  WHERE department_id=280;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	280	Data Analytics	(null)	1700

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
290	디자이너	null	1700
300	DB관리자	null	1800
310	데이터분석가	null	1800
320	퍼블리셔	200	1800
330	서버관리자	200	1800

다음을 추가해보세요
rollback으로 취소해보세요

- 서브쿼리로 INSERT 문장을 작성합니다.
- VALUES 절을 사용하지 않습니다.
- 서브쿼리의 열 수와 INSERT 절의 열 수는 일치해야 합니다.

기존의 테이블로부터 값을 가져와 테이블에 추가하기 위해서 INSERT 문장을 사용할 수 있습니다. VALUES 절에서 서브쿼리를 사용할 수 있습니다.

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

구문에서...

- *select_sub_query* : 행을 리턴할 서브쿼리입니다. INSERT 절의 열 목록에서 열의 개수와 데이터타입은 서브쿼리에 있는 값의 개수와 그들의 데이터타입과 일치해야 합니다.

```
SQL> CREATE TABLE managers AS
  2    SELECT employee_id, first_name, job_id, salary, hire_date
  3    FROM   employees
  4    WHERE  1=2;

Table MANAGERS이(가) 생성되었습니다.
```

다음 구문은 EMPLOYEES 테이블로부터 SELECT 구문으로 조회한 결과를 MANAGERS 테이블에 저장합니다. MANAGERS 테이블에 저장한 데이터는 직무가 매니저(PU_MAN, ST_MAN, SA_MAN, MK_MAN)인 사원들입니다.

```
SQL> INSERT INTO managers
  2    (employee_id, first_name, job_id, salary, hire_date)
  3    SELECT employee_id, first_name, job_id, salary, hire_date
  4    FROM   employees
  5    WHERE  job_id LIKE '%MAN';

12개 행 이(가) 삽입되었습니다.
```

Chapter 7

DML(Data Manipulation Language)

INSERT

UPDATE

DELETE

MERGE

CTAS

- UPDATE 문장으로 기존의 행을 갱신합니다.
- 필요하다면 하나 이상의 행을 갱신할 수 있습니다.

UPDATE 문장을 사용하여 기존의 행을 수정할 수 있습니다. **업셋!**

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

위의 구문형식에서...

- *table* : 테이블의 이름입니다.
- *column* : 테이블의 열 이름입니다.
- *value* : 열에 대한 관련 값이나 서브쿼리입니다.
- *condition* : 갱신할 행을 명시하고, 열 이름, 표현식, 상수, 서브쿼리 그리고 비교 연산자로 구성됩니다.

다음 구문은 UPDATE 구문을 연습하기 위해 EMPLOYEES 테이블의 임시 사본을 생성합니다. ALTER TABLE 구문은 제약조건을 추가합니다. 제약조건은 11장에서 설명합니다.

```
SQL> CREATE TABLE emps AS SELECT * FROM employees;

Table EMPS이(가) 생성되었습니다.

SQL> ALTER TABLE emps
  2  ADD (CONSTRAINT emps_emp_id_pk PRIMARY KEY (employee_id),
  3       CONSTRAINT emps_manager_fk FOREIGN KEY (manager_id)
  4                  REFERENCES emps(employee_id)
  5  );

Table EMPS이(가) 변경되었습니다.
```

103번 사원의 급여를 10% 인상하는 테이블 행 갱신 구문을 실습하기 전에 변경되기 전 데이터를 확인해 보겠습니다.

다음 구문은 103번 사원의 사원아이디와 이름 그리고 급여를 출력합니다.

```
SQL> SELECT employee_id, first_name, salary
  2  FROM   emps
  3  WHERE  employee_id=103;
```

	EMPLOYEE_ID	FIRST_NAME	SALARY
1	103	Alexander	9000

다음 구문은 103 사원의 급여를 10%인상합니다.

```
SQL> UPDATE emps
  2  SET    salary=salary*1.1
  3  WHERE  employee_id=103;

1 행 이(가) 업데이트되었습니다.
```

```
UPDATE  table
SET     (column1, column2, ...) =
                          (SELECT column1, column2, ...
                           FROM   table
                           WHERE  condition)
WHERE   condition
```

UPDATE 문에도 서브쿼리를 사용할 수 있습니다. SET 절의 값 위치에 서브쿼리를 이용한 결과를 넣을 수 있습니다.

다음 구문은 서브쿼리가 여러 개 열을 반환하는 다중 열 서브쿼리를 이용해서 UPDATE 구문을 작성한 예입니다.

```
SQL> UPDATE emps
  2  SET (job_id, salary, manager_id) =
  3      (SELECT job_id, salary, manager_id
  4       FROM   emps
  5       WHERE  employee_id = 108)
  6  WHERE employee_id=109;

1 행 이(가) 업데이트되었습니다.
```

Chapter 7

DML(Data Manipulation Language)

INSERT

UPDATE

DELETE

MERGE

CTAS

- DELETE 문장을 사용하여 테이블로부터 기존의 행을 제거할 수 있습니다.
- 참조 무결성 제약 조건에 주의해야 합니다. 다른 테이블에서 참조되고 있는 레코드가 존재할 경우를 주의해야 합니다.

DELETE 문장을 사용하여 기존의 행을 제거합니다.

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

구문 형식에서...

- *table* : 테이블의 이름입니다.
- *condition* : 삭제된 행을 명시하고, 열 이름, 표현식, 상수, 서브쿼리 그리고 비교 연산자로 구성됩니다.

다음 구문은 DELETE 구문을 연습하기 위해 EMPLOYEES 테이블의 임시 사본을 생성합니다. ALTER TABLE 구문은 제약조건을 추가합니다. 제약조건은 11장에서 설명합니다. UPDATE 구문을 실습할 때 이미 테이블이 만들어져 있다면 실행할 필요 없습니다.

```
SQL> CREATE TABLE emps AS SELECT * FROM employees;

Table EMPS이(가) 생성되었습니다.

SQL> ALTER TABLE emps
  2  ADD (CONSTRAINT emps_emp_id_pk PRIMARY KEY (employee_id),
  3       CONSTRAINT emps_manager_fk FOREIGN KEY (manager_id)
  4                  REFERENCES emps(employee_id)
  5  );

Table EMPS이(가) 변경되었습니다.
```

행 삭제는 실행하기 전에 반드시 확인하는 습관을 가지세요. ROLLBACK 명령으로 실행 취소가 가능하지만 그럴 경우 이전 트랜잭션 완료 이후 모든 변경 내용이 취소될 수 있습니다. 잘못된 DELETE 구문은 원치 않는 데이터를 삭제할 수 있으므로 주의하세요.

104번 사원의 정보를 삭제하고 싶습니다. 다음 구문은 104번 사원의 모든 정보를 삭제합니다.

```
SQL> DELETE FROM emps
  2  WHERE employee_id=104;

1 행 이(가) 삭제되었습니다.
```

다른 테이블의 값을 근거로 테이블로부터 행을 삭제하기 위해 서브쿼리를 사용할 수 있습니다.

실습을 위해 DEPARTMENTS 테이블의 사본 테이블을 생성합니다.

```
SQL> CREATE TABLE depts AS
  2    SELECT * FROM departments;

Table DEPTS이(가) 생성되었습니다.
```

테이블의 구조를 확인해 보세요. 어떤 열들이 어떤 타입으로 선언되어 있는지 알 수 있습니다.

```
SQL> DESC depts;

이름             널        유형
--------------- -------- ------------
DEPARTMENT_ID             NUMBER(4)
DEPARTMENT_NAME NOT NULL VARCHAR2(30)
MANAGER_ID                NUMBER(6)
LOCATION_ID               NUMBER(4)
```

다음 구문은 부서이름이 Shipping이 부서의 모든 사원 정보를 삭제합니다. 서브쿼리는 Shipping 부서에 대해 부서 번호를 알기 위해서 DEPTS 테이블을 검색합니다. 그런 다음에 서브쿼리는 이 부서 번호를 근거로 하는 EMPS 테이블로부터 데이터의 행을 삭제하는 메인 쿼리로 부서 번호를 넘겨줍니다.

```
SQL> DELETE FROM emps
  2  WHERE department_id=
  3                      (SELECT department_id
  4                       FROM depts
  5                       WHERE department_name='Shipping');

45개 행 이(가) 삭제되었습니다.
```

모든 행이 삭제되지는 않습니다. 무결성 제약조건을 위반하도록 레코드를 삭제하고자 한다면 에러가 발생할 것입니다. 다음 구문은 실행 시 에러가 발생합니다. EMPLOYEES_ID 열은 MANAGER_ID 열에서 참조하고 있습니다. 104번 사원의 매니저는 103번 사원으로 지정되어 있기 때문에 103번 사원은 삭제되지 않습니다. 제약조건에 대해서 11장에서 자세하게 설명합니다.

```
SQL> DELETE FROM emps
  2  WHERE employee_id=103;

명령의 1 행에서 시작하는 중 오류 발생 -
DELETE FROM emps
WHERE employee_id=103
오류 보고 -
SQL 오류: ORA-02292: integrity constraint (HR.EMPS_MANAGER_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

Chapter 7

DML(Data Manipulation Language)

INSERT

UPDATE

DELETE

MERGE

CTAS(사본테이블)

회원의 아주 중요한 테이블이 있다고 가정하자, 매번 탈퇴, 수정되는 데이터를 따로 저장하는 테이블을 만들어 놓고, 일주일단위로 업데이트 한다.

데이터베이스에 INSERT 또는 UPDATE 할 때에 데이터가 존재하는지 여부를 체크를 하고 존재하면 UPDATE를 하고, 존재하지 않으면 INSERT를 수행할 수 있게 합니다.

```
MERGE INTO  table [alias]
  USING ( target | view | subquery ) [alias]
    ON ( join_condition )
  WHEN MATCHED THEN
    UPDATE SET column1=value1[, ... ]
  WHEN NOT MATCHED THEN
    INSERT (column_lists) VALUES (value_lists)
```

구문 형식에서...

- MERGE INTO : 테이블명을 선언합니다.
- USING : 두개 이상의 테이블을 조인할 때 사용합니다. 하나의 테이블만 사용한다면 DUAL 테이블을 사용할 수 있습니다.
- ON : join condition으로 where 절을 완성합니다.
- WHEN MATCHED THEN : 매칭되는 것이 있으면 update를 수행합니다.
- WHEN NOT MATCHED THEN 매칭되는 것이 없으면 insert를 수행합니다.

실습을 위해 EMPLOYEES 테이블과 구조만 같고, 데이터는 포함하지 않는 임시 테이블 EMPS_IT를 생성하세요. 이 테이블은 IT_PROG 직무를 갖는 사원 정보만 저장하기 위한 테이블입니다.

```
SQL> CREATE TABLE emps_it AS SELECT * FROM employees WHERE 1=2;

Table EMPS_IT이(가) 생성되었습니다.
```

EMPS_IT 테이블에 사원정보를 하나 추가합니다.

```
SQL> INSERT INTO emps_it
  2    (employee_id, first_name, last_name, email, hire_date, job_id)
  3  VALUES
  4    (105, 'David', 'Kim', 'DAVIDKIM', '06/03/04', 'IT_PROG');

1 행 이(가) 삽입되었습니다.
```

```
SQL> MERGE INTO emps_it a
  2    USING (SELECT * FROM employees WHERE job_id='IT_PROG') b
  3    ON (a.employee_id = b.employee_id)
  4  WHEN MATCHED THEN
  5    UPDATE SET
  6      a.phone_number = b.phone_number,
  7      a.hire_date = b.hire_date,
  8      a.job_id = b.job_id,
  9      a.salary = b.salary,
 10      a.commission_pct = b.commission_pct,
 11      a.manager_id = b.manager_id,
 12      a.department_id = b.department_id
 13  WHEN NOT MATCHED THEN
 14    INSERT VALUES
 15     (b.employee_id, b.first_name, b.last_name, b.email,
 16      b.phone_number, b.hire_date, b.job_id, b.salary,
 17      b.commission_pct, b.manager_id, b.department_id);

5개 행 이(가) 병합되었습니다.

SQL> SELECT * FROM emps_it;
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	105	David	Kim	DAVIDKIM	590.423.4569	05/06/25	IT_PROG	4800	(null)	103	60
2	106	Valli	Pataballa	VPATABAL	590.423.4560	06/02/05	IT_PROG	4800	(null)	103	60
3	103	Alexander	Hunold	AHUNOLD	590.423.4567	06/01/03	IT_PROG	9000	(null)	102	60
4	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/07	IT_PROG	4200	(null)	103	60
5	104	Bruce	Ernst	BERNST	590.423.4568	07/05/21	IT_PROG	6000	(null)	103	60

Chapter 7

DML(Data Manipulation Language)

INSERT

UPDATE

DELETE

MERGE

CTAS(사본테이블)

테이블을 생성할때도 서브쿼리 문장을 사용할 수있습니다.
사본테이블 생성에 사용.

CTAS(Create Table As Select) 구문은 현재 있는 테이블과 같은 구조를 갖는 테이블을 생성할 수 있도록 합니다. CTAS 구문을 이용한 테이블 복제 시 NOT NULL 제약조건을 제외한 다른 제약조건은 복사되지 않습니다.

```
CREATE TABLE table AS SELECT statement
```

구문에서...

- AS SELECT *statement* : 테이블이 만들어질 서브쿼리를 입력합니다. SELECT 문장의 WHERE 절이 항상 FALSE일 경우 데이터는 포함하지 않고 구조만 갖는 테이블을 생성합니다. 서브쿼리이지만 괄호(( ))를 포함하지 않습니다.

다음 구문은 SELECT한 결과대로 테이블의 구조를 생성하고 데이터도 저장합니다.

```
SQL> CREATE TABLE emp2 AS SELECT * FROM employees;

Table EMP2이(가) 생성되었습니다.

SQL> SELECT COUNT(*) FROM emp2;

  COUNT(*)
----------
       107
```

다음 구문은 데이터는 갖지 않고 구조만 갖는 테이블을 생성합니다. 서브쿼리 WHERE 절 조건식의 결과는 항상 FALSE입니다.

```
SQL> CREATE TABLE emp3 AS SELECT * FROM employees WHERE 1=2;

Table EMP3이(가) 생성되었습니다.

SQL> SELECT COUNT(*) FROM emp3;

  COUNT(*)
----------
         0
```

모든 문제는 변경한 후 select문으로 조회로 확인한 후 commit합니다

문제 1.

DEPTS테이블의 다음을 추가하세요

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	개발	null	1800
290	회계부	null	1800
300	재정	301	1800
310	인사	302	1800
320	영업	303	1700

문제 2.

DEPTS테이블의 데이터를 수정합니다

1. **department_name 이 IT Support 인 데이터의 department_name을 IT bank로 변경**
2. **department_id가 290인 데이터의 manager_id를 301로 변경**
3. **department_name이 IT Helpdesk인 데이터의 부서명을 IT Help로 , 매니저아이디를 303으로, 지역아이디를 1800으로 변경하세요**
4. **이사, 부장, 과장, 대리 의 매니저아이디를 301로 한번에 변경하세요.**

문제 3.

삭제의 조건은 항상 primary key로 합니다, 여기서 primary key는 department_id라고 가정합니다.

1. **부서명 영업부를 삭제 하세요**
2. **부서명 NOC를 삭제하세요**

문제4

1. Depts 사본테이블에서 department_id 가 200보다 큰 데이터를 삭제하세요.
2. Depts 사본테이블의 manager_id가 null이 아닌 데이터의 manager_id를 전부 100으로 변경하세요.
3. Depts 테이블은 타겟 테이블 입니다.
4. Departments테이블은 매번 수정이 일어나는 테이블이라고 가정하고 Depts와 비교하여
   일치하는 경우 Depts의 부서명, 매니저ID, 지역ID를 업데이트 하고
   새로유입된 데이터는 그대로 추가해주는 merge문을 작성하세요.

문제 5

1. jobs_it 사본 테이블을 생성하세요 (조건은 min_salary가 6000보다 큰 데이터만 복사합니다)
2. jobs_it 테이블에 다음 데이터를 추가하세요

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DEV	아이티개발팀	6000	20000
NET_DEV	네트워크개발팀	5000	20000
SEC_DEV	보안개발팀	6000	19000

3. jobs_it은 타겟 테이블 입니다
4. jobs테이블은 매번 수정이 일어나는 테이블이라고 가정하고 jobs_it과 비교하여
   min_salary컬럼이 0보다 큰 경우 기존의 데이터는 min_salary, max_salary를 업데이트 하고 새로 유입된
   데이터는 그대로 추가해주는 merge문을 작성하세요