14장. 데이터 조작어(DML)

DML은 Data Manipulation Language의 약자로 데이터 조작어를 뜻합니다. 정의된 데이터베이스에 입력된 레코드를 조회, 수정, 삭제하는 등의 역할을 합니다. 즉, 테이블에 있는 행과 열을 조작하는 언어입니다. 데이터베이스 사용자가 질의어를 통하여 저장된 데이터를 실질적으로 처리하는데 사용합니다. 데이터 조작어에는 SELECT, INSERT, UPDATE,DELETE가 있습니다.

14.1 INSERT

INSERT문을 사용하여 테이블에 행을 삽입합니다.
또한 일반적으로, 단순한 수치 이외의 정수는 예시와 같이, 작은따옴표(')로 묶어줘야 합니다.

```
INSERT INTO weather VALUES ('San Francisco', 46, 50, 0.25, '1994-11-27');
```

지금까지의 구문에서는 열의 순서를 기억해 두어야 했습니다. 다음과 같은 다른 구문을 사용하면 열목록을 명시적으로 제공할수 있습니다.

```
INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');
```

목록의 열은 원하는 순서대로 지정 할 수 있습니다. 또한 일부 열을 생략 할 수 있습니다.
예를들어, 강수량을 모르는 경우 다음과 같이 할 수 있습니다.

```
INSERT INTO weather (date, city, temp_hi, temp_lo)
VALUES ('1994-11-29', 'Hayward', 54, 37);
```

또한 COPY를 사용하여 대량의 데이터를 일반 텍스트 파일에서 로드할수있습니다. COPY명령은 INSERT보다 유연하지는 않지만 일반적으로 목적에 특화 되어 있기때문에 일반적으로 더 빠릅니다.

예를 들면 다음과 같습니다.

```
COPYweather FROM '/home/user/weather.txt';
```

14.2 UPDATE

UPDATE는 조건을 만족하는 모든 행에서 지정된 열의 값을 변경합니다. 수정할 열만 SET절에서 언급하면 됩니다. 명시적으로 수정되지 않은 열은 이전 값을 유지합니다.

예시로 다음 명령을 사용하여 데이터를 수정 할 수 있습니다.

```
UPDATE weather
        SET temp_hi = temp_hi -2, temp_lo = temp_lo -2
        WHEREdate > '1994-11-28';
```

14.3 DELETE

DELETE 명령을 사용하여 테이블에서 행을 삭제할 수 있습니다. DELETE WHERE에 지정된 테이블에서 절을 만족하는 행을 삭제합니다.절이 WHERE없으면 결과는 테이블의 모든 행을 삭제합니다.

예시로 다음 명령을 사용하여 테이블에서 행을 삭제할 수 있습니다.

```
DELETE FROM weather WHERE city = 'Hayward';
```

Hayward 에 대한 기상데이터는 모두 삭제 되었습니다.

```
DELETEFROM tablename;
```

조건이 없으면 DELETE는 지정된 테이블의 모든 행을 삭제하고 테이블을 비웁니다. 시스템은 삭제 전엔 확인을 요구한 는 일은 하지 않습니다.

14.4 SELECT

SELECT는 데이터 조회문입니다. 선택 목록과 테이블 목록 및 선택적 조건으로 나눌 수 있습니다. 예를 들어, 날씨의 모든 행을 검색하려면 아래 조회문을 보십시오.

```
SELECT * FROM weather;
```

다음 조회문도 같은 결과가 나옵니다.

```
SELECT city, temp_lo, temp_hi, prcp, date FROM weather;
```

선택 목록에는 단순한 열 참조 뿐만 아니라, 식을 지정 할 수도있습니다.

예를 들어, 다음 조회문도 수행할 수 있습니다.

```
SELECT city, (temp_hi+temp_lo)/2 AS temp_avg, date FROM weather
```

AS 절을 사용하여 출력 열의 레이블 지정 부분에 유의하십시오.(AS 절은 생략할 수 있음).

필요한 행이 무엇인지 지정 하는 WHERE절을 추가하여 쿼리에 "조건화" 할 수 있습니다. WHERE절은 논리(참값) 표현식을 가지 며 이 논리 표현식 이 참인 행만 리턴합니다. 자주 사용되는 논리 연산자(AND, OR, NOT)를 조건부로 사용할 수 있습니다.

예를 들어, 다음은 샌프란시스코에서 비오는 날씨 기상 데이터를 검색합니다.

```
SELECT* FROM weather
WHERE city = 'San Francisco' AND prcp > 0.0;
```

쿼리 결과를 정렬하고 반환하도록 지정 할 수 있습니다.

```
SELECT * FROM weather
ORDERBY city;
```

이 조회문에서는 정렬 순서가 충분히 지정 되지 않았습니다. 따라서 San Francisco의 행은 순서가 다를수있습니다.

그러나 다음과 같이 하면 항상 위의 결과가 됩니다.

```
SELECT* FROM weather
ORDERBYcity, temp_lo;
```

쿼리 결과에서 중복 행을 제외하도록 지정 할 수 있습니다.

```
SELECT DISTINCT city FROM weather;
```

14.4.1 JOIN

지금까지의 쿼리는 한번에 하나의 테이블에만 액세스하는 것이었습니다. 쿼리는 한 번에 여러 테이블에 액세스하거나 테이블의 여러 행을 동시에 처리하는 방식으로 단일 테이블에 액세스할 수 있습니다. 한 번에 동일한 테이블이나 다른 테이블의 여러 행에 액세스하는 쿼리를 조인 쿼리라고 합니다.

예를 들어, 모든 기상 데이터를 관련 도시의 위치정보와 함께 표시하려는 경우를 들 수 있습니다. 그렇게 하려면 weather 테이블의 각 행의 city 열을 cities 테이블의 모든 행의 name 열과 비교하고 두 값이 일치하는 조합을 선택해야 합니다.

이 작업은 다음 쿼리를 통해 수행할 수 있습니다.

```
SELECT*FROM weather, cities
WHERE city=name;
```

```
city | temp_lo | temp_hi | prcp | date | name | location
---------------+---------+---------+------+------------+---------------+-----------

San Francisco | 46 | 50 | 0.25 | 1994-11-27 | San Francisco | (-194,53) 57 | 0 | 1994-11-29
San Francisco |42                                    |   San Francisco |     (-194,53)
```

- Hayward 시에 대한 결과행이 없습니다. 이것은 cities 테이블에는 Hayward와 일치하는 항목이 없기 때문에 조인 시 weather테이블의 일치하지 않는 행은 무시됩니다. 이것을 어떻게 해결할 수 있는지 잠시 후에 설명합니다.

- 도시 이름을 가진 두 개의 열이 있습니다. weather 테이블과 cities 테이블의 목록이 연결되었습니다.

또한 *를 사용하지 않고 명시적으로 출력 열 목록을 지정하겠습니다.

```
SELECT city, temp_lo, temp_hi, prcp, date, location
FROM weather, cities
WHERE city=name;
```

열이 모두 다른 이름이었기 때문에, 파서는 자동적으로 어느 테이블의 열인가를 찾을 수 있었습니다. 두 테이블에서 컬럼 이름이 중복되는 경우, 다음과 같이 표시할 칼럼을 표시하기 위해 컬럼 이름을 규정해야 합니다.

```
SELECT weather.city, weather.temp_lo, weather.temp_hi,
       weather.prcp, weather.date, cities.location
FROM weather, cities
WHEREcities.name = weather.city;
```

조인 쿼리는 일반적으로 모든 열 이름을 한정하는 것이 좋습니다. 그러면 나중에 테이블 중 하나에 이름이 중복된 열이 추가되어도 쿼리가 실패하지 않습니다.

여기까지 나타낸 것과 같은 조인 쿼리는 다음과 같이 내부조인 형태로 나타낼 수 있습니다.

```
SELECT *
 FROM weather INNER JOIN cities ON (weather.city = cities.name);
```

수행하려는 쿼리는 날씨를 스캔하고 각 행에 대해 cities 행과 일치하는 행을 찾습니다.일치하는 행이 없으면 cities 테이블의 열 부분을 일부 "빈 값"으로 바꾸고, 이 유형의 쿼리를 외부 조인이라고 합니다

```
SELECT * FROM weather LEFT OUTER JOIN cities ON (weather.city = cities.name);



city | temp_lo | temp_hi | prcp | date | name | location

--------------+---------+---------+------+------------+--------------+----------

Hayward | 36      |  54 | | 1994-11-29 | 50 | 0.25 |
San Francisco | 46 |1994-11-27 | San Francisco | (-194,53) 57 | 0 | 1994-11-29 |
San Francisco | 43 |San Francisco | (-194,53)
```

이 쿼리를 LEFT OUTER JOIN이라고 합니다. 조인 연산자의 왼쪽에 지정한 테이블의 각 행이 최저라도 한 번 출력되고, 우측의 테이블에서는 왼쪽의 테이블의 행과 일치하는 것만이 출력됩니다. 오른쪽 테이블과 일치하지 않는 왼쪽 테이블의 행을 출력할 때 오른쪽 테이블의 열은 빈 값 (NULL)으로 바뀝니다.

또한 테이블은 스스로 결합할 수 있습니다. 이것을 Self join이라고 합니다. 예를 들어, 다른 기상 데이터의 기온 범위 내에 있는 기상 데이터를 모두 검색하는 것을 생각해 봅시다. weather 각 행의 temp_lo와 temp_hi를 다른 weather 행의 temp_lo와 temp_hi 열을 비교해야 합니다. 다음 쿼리를 사용하여 수행할 수 있습니다

```
SELECT W1.city, W1.temp_lo ASlow, W1.temp_hi AShigh, W2.city,
      W2.temp_lo ASlow, W2.temp_hi AShigh FROM weather W1,
            weather  W2  WHERE
```

```
W1.temp_lo<
     W2.temp_lo AND W1.temp_lo
>
     W2.temp_hi;



city |low|high| city |low |high
--------------+-----+------+---------------+-----+------

San Francisco| 43 |57 | San Francisco | 46 |50
Hayward | 37 | (2 rows)54 | San Francisco | 46 |50
```

이제 조인의 왼쪽과 오른쪽을 구분할수 있도록 weather 테이블에 W1 과 W2라는 레이블이 붙어있습니다. 또한 입력량을 줄이 기위해 다른 쿼리에서도 이런 종류의 별칭을 사용할 수 있습니다.

예를 들면 다음과같습니다

```
SELECT* FROM weather w, cities c
WHEREw.city=c.name;
```

14.4.2 집계 함수

대부분의 다른 관계형 데이터베이스 제품과 마찬가지로 AgensSQL은 집계 함수를 지원합니다. 집계 함수는 여러 행으로 부터 하나의 결과를 계산합니다. SELECT 구문에서만 사용되며 주로 count(총수),max(최대), min(최소), sum(총합), avg(평균) 과같은 연산을 수행하는데 사용됩니다.

COUNT

특정 열의 행 개수를 세는 함수입니다. COUNT(*)로 작성하면 테이블에 존재하는 모든 행의 개수가 반환되고, 특정열에 대해서 COUNT를 수행하면 해당 열이 NULL이 아닌 행의 개수를 반환합니다.

```
SELECT COUNT(*) FROM tablename;
```

```
SELECT COUNT(Name) FROM tablename;
```

MIN/MAX

MIN과 MAX는 최솟값과 최댓값을 구하는 함수입니다. 사용법은 위의 COUNT와 동일합니다.

```
SELECT MAX(Age) FROM tablename;
```

GROUP BY절과 결합 된 집계도 매우 유용합니다. 예를 들어, 다음 명령을 사용하여 도시별로 최저 온도의 최대 값을 찾을 수 있습니다.

```
SELECT city, max(temp_lo)
    FROM weather
   GROUP BY city
City          |max
--------------+-----
Hayward    |37
San Francisco | 46
(2 rows)
```

각 집계 결과는 도시와 일치하는 전체 테이블 행에 대한 연산 결과입니다. 다음과 같이 HAVING을 사용하여 그룹화된 행을 필터링할 수 있습니다.

```
SELECT city, max(temp_lo)
FROM weather
GROUP BY city
HAVINGmax(temp_lo) < 40;

city | max
---------+-----
Hayward | 37
(1 rows)
```

또한 WHERE절도 같이 사용할 수 있습니다. 모든 temp_lo 의 값이 40 미만의 도시만을 출력하고 "S"로 시작 하는 이름의 도시만을 대상으로 하려면 다음을 보십시오.

```
SELECT city, max(temp_lo)
  FROM weather
  WHEREcity LIKE 'S%' 1
  GROUP BY city
  HAVING max(temp_lo) < 40
```

WHERE와 HAVING의 기본적인 차이점은 WHERE은 그룹과 집계함수를 계산하기 전에 입력 행을 먼저 선택합니다 HAVING은 그룹과 집계를 계산 한 후 그룹화 된 행을 선택한다는 것입니다. 따라서 WHERE 절은 집계 함수를 가질 수 없습니다. 반면에 HAVING 절에는 항상 집계 함수를 사용합니다.

앞의 예에서는 WHERE 내에 도시 이름 제한하여 적용할 수 있습니다. 왜냐하면 집계함수를 사용하지 않기때문입니다. 이것은 HAVING에 제한을 추가하는 것보다 효율적입니다.