SQL의 이해

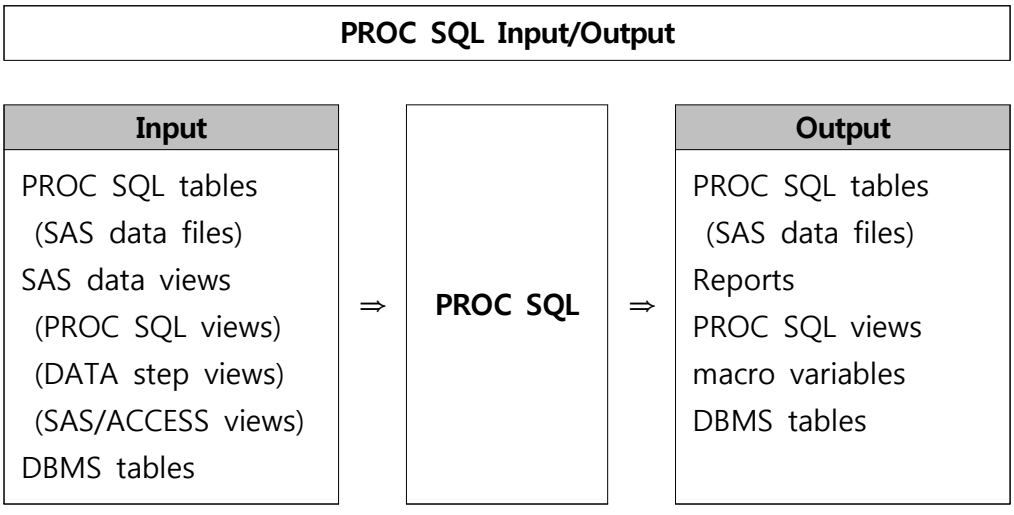
1 SQL 이해하기

가. SQL

구조적 질의 언어(Structured Query Language, 이하 SQL) 약어로써 관계형 테이블과 데이터베이스 테이블에서 검색과 갱신하는 데 널리 사용되는 표준 언어이다. 관계형이라는 개념은 수학에서의 집합의 개념과 유사한 것으로 컬럼(column)과 행(row)으로 배열된 2차원 테이블을 나타낸다. SQL은 IBM에서 처음 개발되었으며 현재 MS-SQL이나 ORACLE-SQL와 같은 여러 버전이 존재하지만 유사한 형식으로 인해 한 버전의 문법만 익힌다면 다른 버전은 쉽게 사용할 수 있다.

나. PROC SQL

PROC SQL은 SAS/BASE를 구성하고 있는 요소로 SAS내에서 SQL을 가능하게 해주는 프로시저이다. SAS dataset[table] 처리가 가능하며 SAS 프로시저와 DATA step 기능의 일부도 수행할 수 있다. 특히 SAS에서 사용하는 옵션, 함수 그리고 포맷 등을 자유롭게 사용할 수 있어 SAS DATA step과 프로시저와 더불어 사용한다면 강력한 데이터 처리 및 분석이 가능하다. 아래 그림은 PROC SQL에서 가능한 작업을 도식한 것이다.



<그림 1> PROC SQL의 기능

구체적으로 서술해보면 다음과 같다.

◦ 요약 통계량 산출/보고서 작성

◦ table 또는 view에 있는 데이터 검색

◦ table의 행/컬럼 추가, 수정, 삭제

◦ 여러 table 또는 view에 있는 데이터 조합

◦ table, view와 index 생성

◦ table에 있는 데이터 갱신

◦ DBMS table에 있는 데이터 검색 및 갱신

다. Table

PROC SQL의 table은 SAS data 파일과 동일하다. 즉, DATA 형태의 SAS 파일이라고 볼 수 있다. PROC SQL의 table은 행과 열로 구성되며 행은 SAS data 파일의 관측개체, 열은 변수와 대응된다. 데이터베이스 용어와 같이 비교해 보면 아래 표와 같이 대응된다.

<표 1> 데이터의 구조

SQL	Base SAS	Data Base
테이블(table)	데이터세트(dataset)	파일(file)
행(row)	관측개체(observation)	레코드(record)
열(column)	변수(variable)	필드(field)

Data Step와 PROC SQL을 이용하여 생성하거나 갱신한 table은 별도의 변환없이 사용가능해 호환성이 높고 SAS의 어느 프로시저에서도 사용가능하다.

라. View

PROC View는 table처럼 실제 데이터를 가지고 있는 것이 아니라 select 구나 query를 저장해 가지고 있는 일종의 가상 table이다. SAS 프로시저나 DATA step에서 특정한 View를 호출하면 저장된 select 구나 query가 실행되고 기존의 테이블이나 다른 View로부터 데이터를 가져와 보여준다. PROC SQL View를 이용하면 테이블과는 별도로 SAS Data Set를 물리적으로 생성되지 않기 때문에 저장 공간을 줄일 수는 장점이 있다. 하지만 단점도 가지고 있다. 예를 들어, View 안에 정렬을 하는 문장을 가지고 있다면 View를 호출할 때 마다 정렬을 수행해야하기 때문에 처리시간이 많이 걸릴 수 있다. 이런 경우에는 View를 사용하는 것보다는 table을 사용하는 것이 유리하다.

마. NULL 값

일반적인 SQL에서는 결측값(missing value)은 Null 값으로 0(zero)값이나 공백(blank)값과는 다르다. 그러나 PROC SQL에서는 SAS의 다른 부분과의 호환을 위해서 세 개의 값을 동일하게 취급한다.

2 SQL 프로시저의 기본문법

SQL 프로시저의 기본문법은 다음과 같다.

```
RSUBMIT;
PROC SQL options;
   CONNECT TO ODBC(DSN=QUERY USER=XXXXXX PW=XXXXXX SCHEMA=DM);
   CREATE TABLE[VIEW] table_name1[view_filename] AS
   SELECT * CONNECTION TO ODBC(
        SELECT   expression 1,
                 expression 2
          INTO   #table_name3
        FROM     table_names
        WHERE    condition statements
        GROUP    BY column_name1, column_name2
        Having   condition statements
        ORDER    BY column_name1 [DESC], column_name2 [DESC]);
QUIT;
ENDRSUBMIT;
```

SAS 프로시저는 PROC SQL; 다음에 위치하며 마지막 질의어에만 세리콜론(;)을 붙인다. 다른 프로시저처럼 RUN;이 없어도 실행이 가능하며 통상적으로 마지막에 QUIT;로 끝내는 것이 좋다. 가장 기본적인 구성요소는 SELECT와 FROM으로만 구성되어 있고 나머지 WHERE, GROUP BY, HAVING 그리고 ORDER BY는 각 상황에 맞게 사용된다. 다만 기본문법에 적힌 순서는 반드시 지켜야 한다.

가. SELECT/FROM

PROC SQL의 가장 기본적인 형태로 SELECT/FROM 구만으로도 충분히 유용한 결과를 얻을 수 있다. PROC SQL QUERY에 반드시 필요한 부분이고 다음과 같은 내용이 기재된다.

- ◦ SELECT 구 : 컬럼명이 기재되는 부분
- ◦ FROM 구 : SELECT에 기재된 컬럼이 있는 Table이 기재되는 부분

```
SELECT   고객번호
 FROM    dm.월개인본인회원정보;
```

나. WHERE

WHERE는 Table에서 특정한 조건을 만족하는 행만 검색 가능하도록 한다. 따라서, PROC SQL의 OUTPUT은 WHERE 다음에 오는 조건을 만족하는 행을 이용한 결과가 나타난다.

```
SELECT   고객번호
 FROM    dm.월개인본인회원정보
WHERE    정상카드수>1;
```

다. ORDER BY

Table을 하나 또는 여러 개의 컬럼순으로 정렬한다. 문자는 알파벳순으로, 숫자는 크기순으로 올림차순과 내림차순 둘 다 가능하다. 올림차순이 기본으로 지정되어있다.

```
SELECT   고객번호
 FROM    dm.월개인본인회원실적
WHERE    정상카드수>1 AND 기준년월='201001'
ORDER    BY 이용금액_CA DESC;
```

라. GROUP BY

GROUP BY는 그룹(행들의 집합)별 결과를 보여준다. GROUP BY를 사용하면 SELECT 또는 HAVING에 aggregate 함수(SUM, COUNT, MAX 등)를 이용해 그룹별 결과를 얻을 수 있다 (aggregate 함수에 대한 자세한 내용은 이후에 다루기로 하겠다). SELECT 구에 aggregate 함수가 없는데도 불구하고 GROUP BY 구를 사용하면 PROC SQL은 ORDER BY 구처럼 취급하고, GROUP BY 구가 없는데도 불구하고 SELECT 구에 aggregate 함수를 사용하면 전체 데

이터에 적용한다.

```
SELECT   성별,
         SUM(이용금액_신판) AS 신판이용금액
 FROM    dm.월개인본인회원실적
WHERE    정상카드수>1 AND 기준년월='201001'
GROUP    BY 성별;

SELECT   고객번호,
         성별,
         이용금액_신판
 FROM    dm.월개인본인회원실적
WHERE    정상카드수>1 AND 기준년월='201001'
GROUP    BY 성별;

SELECT   성별,
         SUM(이용금액_신판) AS 신판이용금액
 FROM    dm.월개인본인회원실적
WHERE    정상카드수>1 AND 기준년월='201001'
```

마. HAVING

HAVING 구는 GROUP BY 구와 함께 사용되며, 특정한 조건을 만족하는 그룹만 검색가능하도록 한다. 따라서, PROC SQL은 데이터를 그룹화하고 aggregate 함수를 적용한 후 HAVING의 조건을 적용한다.

```
SELECT   성별,
         리스크등급,
         SUM(이용금액_신판) AS 신판이용금액
 FROM    dm.월개인본인회원실적
WHERE    정상카드수>1 AND 기준년월='201001'
GROUP    BY 성별, 리스크등급
HAVING   avg(이용금액_CA)>1000000000;
```

2 SQL 프로시저의 사용법

Table에 있는 모든 컬럼 선택하기

SELECT구에 *(asterisk)을 사용하면 table에 있는 모든 컬럼을 선택할 수 있다. 단, Table에 있는 모든 컬럼을 선택할 때는 table에 저장된 순서로 컬럼을 나타낸다.

```
proc sql outobs=50
       select *
         from sashelp.prdsale;
quit;
```

NOTE) 'outobs=*n*'는 출력문에 나오는 행의 개수를 제한하는 옵션으로 data set 옵션의 OBS와 유사하다.

Table에 있는 특정한 컬럼 선택하기

Table에 있는 특정한 컬럼을 선택하고자 할 때는 SELECT 구에 해당 컬럼명을 기재하면 된다. 여러 개의 컬럼을 선택할 때는 반드시 ,(comma)로 구분해야하며 출력문에 나타나는 컬럼의 순서는 Table 저장된 컬럼의 순서에 관계없이 SELECT 구에 표기된 순서대로 나열된다.

```
proc sql
       select country,
              region,
              division
         from sashelp.prdsale;
quit;
```

NOTE) SELECT 구에 특정 컬럼명을 표기한 후 *을 사용하면 전체 컬럼을 선택할 수 있다. *만 사용할 때와의 차이점은 출력문에 나타나는 컬럼의 순서를 변경할 수 있다.

NOTE) SELECT 구의 기술방식에서 SQL과 SAS DATA step의 기존 table[dataset]의 컬럼[변수]의 처리방식에 대해 차이점을 나타낸다. 적당한 방식을 통한다면 같은 결과를 얻을 수 있지만, SQL문은 특정한 컬럼의 선택을 기본으로 하고, SAS DATA step은 모든 컬럼[변수]의 선택을 기본으로 한다는 것에 차이점이 있다.

컬럼의 중복값 제거

'distinct + 컬럼명'을 사용하면 컬럼의 값 중 다른 값과 구별되는 값 목록을 구할 수 있다.

```
proc sql;
       select distinct country
         from sashelp.prdsale;
quit;
```

NOTE) distinct를 SELECT 구에 나오는 모든 컬럼에 적용하면 각 컬럼별 최소단위 값의 모든 가능한 조합으로 나타난다.

```
proc sql;
       select distinct country,
              region
         from sashelp.prdsale;
quit;
```

Table 구조 파악

DESCRIBE TABLE은 Table을 구성하고 있는 모든 컬럼명 및 컬럼속성을 파악하고자 할 때 사용한다.

```
proc sql;
describe table sashelp.prdsale;
quit;
```

OUTPUT에 문자열 추가하기

string 표현식과 literal 표현식을 이용해서 OUTPUT에 문자열을 추가할 수 있다. OUTPUT에 컬럼명이 그대로 출력되는 것을 방지하기 위해서는 컬럼 라벨의 시작을 특수문자로 하면된다. PROC SQL에서는 컬럼에 라벨을 지정하면 OUTPUT에 컬럼명 대신 라벨명이 나타나고 특수문자로 시작되는 라벨명을 주면 아무것도 출력하지 않는다.

```
proc sql;
       select distinct "Country ",country,",",
              region, "Region"
         from sashelp.prdsale;
quit;

proc sql;
```

```
        select distinct 'Country', country label='#',',',
               region label='지역','Region'
          from sashelp.prdsale;
quit;
```

수치연산

Table에 저장되어 있는 숫자형 컬럼들을 산술연산자와 수치함수를 통해 수치연산이 가능하다. 수치연산 결과 새로운 컬럼이 생성된다.

```
proc sql;
        select *,
               height*2.54  format=4.1,
               weight*0.453592 format=4.1
          from sashelp.class;
quit
```

컬럼에 별명 지정

컬럼 별명(aliasing) 지정을 이용하여 PROC SQL QUERY내의 어떠한 컬럼이라도 새로운 컬럼명을 부여할 수 있다. 새로운 컬럼명은 SAS 변수명을 지정 규칙을 따라야 하며 별명 지정 시 컬럼 다음 'AS + 별명'을 이용한다.

```
proc sql
        select *,
               height*2.54 as height_inch format=4.1,
               weight*0.453592 as weight_kg format=4.1
          from sashelp.class;
quit
```

별명을 이용해 연산된 값 불러오기

별명을 이용해 PROC SQL QUERY내의 연산된 값을 다시 이용할 때, CALCULATED 키워드를 사용해야 한다. CALCULATED 키워드는 SELECT구나 WHERE구에서만 사용할 수 있다.

```
proc sql;
        select *,
```

```
            height*2.54 as height_cm format=4.1,
            weight*0.453592 as weight_kg format=4.1,
            22*(calculated height_cm/100)**2 as weight_sd format=4.1
        from sashelp.class;
quit;
```

조건문

특정한 조건에 따라 컬럼의 값을 결정하고자 할 때 CASE 문을 사용하며 DATA step에서 IF/THEN 문과 대응된다. CASE문은 반드시 END 문으로 끝내야 하며 END 이후에 컬럼명이나 컬럼속성을 부여할 수 있다. 이는 IF/THEN 문은 THEN 후에 컬럼명을 지정하는 것과 차이점이 있다.

< 표 2-2 > IF/THEN 문과 CASE WHEN 문 비교표

IF ~ THEN ~ 문	CASE WHEN 문
IF *conditions* THEN *result-expression*; ELSE IF *conditions* THEN *result-expression*; ELSE *result-expression*;	CASE WHEN *conditions* THEN *result-expression* WHEN *conditions* THEN *result-expression* ELSE *result-expression* END

```
proc sql;
      select *,
            height*2.54 as height_cm format=4.1,
            weight*0.453592 as weight_kg format=4.1,
            case when sex='M' then 22*(calculated  height_cm/100)**2
                 when sex='F' then 21*(calculated  height_cm/100)**2
            end as weight_sd format=4.1
        from sashelp.class;
quit;
```

Missing Value 대체하기

COALESCE 함수를 이용하면 컬럼에 있는 Missing 값을 새롭게 지정한 값으로 대체가능하다. COALESCE 함수는 CASE WHEN 조건문을 사용하여 Missing 값을 대체하는 문을 작성하면 동일한 결과를 얻을 수 있다.

```
proc sql;
       select *,
              case when age>15 then ' '
              else 'Available' end as Join_yn,
              coalesce(calculated Join_yn, 'Not Available') as Join_yn
         from sashelp.class;
quit;
```

컬럼속성 부여

컬럼에 대한 설명, 입·출력 형식 그리고 길이를 부여하고자 할 때 컬럼 뒤 label, informat, format, length를 사용하여 부여할 수 있다. 사용법은 다음과 같다.

- label='*label*'
- informat=*informat*
- format=*format*
- length=*n*

컬럼을 이용한 table 정렬

원하는 컬럼의 값 순서대로 table을 정렬하고자 할 때 ORDER BY 뒤에 해당 컬럼을 지정하면 된다. 이 때 지정된 컬럼은 반드시 SELECT 구에 있을 필요는 없으며 정렬순서는 올림차순이 기본이다.

```
proc sql;
       select *
         from sashelp.class
        order by name;
quit;

proc sql;
       select *
         from sashelp.class
        order by height;
quit;
```

두 개 이상의 컬럼을 이용한 정렬

두 개 이상의 컬럼을 이용하여 정렬하고자 할 때 ORDER BY 문에 원하는 순서대로 컬럼을 지정한다. 단, 컬럼간에는 반드시 ,(comma)로 구분 해야된다. 만약 컬럼에 Missing 값이 있는 경우 올림차순에 의해 Missing 값이 먼저 출력된다.

```
proc sql;
       select *
         from sashelp.class
        order by sex, name;
quit;
```

정렬순서

정렬순서를 지정하기 위해서는 ASC(올림차순)이나 DESC(내림차순)을 표기해야한다. 표기방법은 ORDER BY 구에서 컬럼 다음에 표기하여 각 컬럼에 따라 다른 정렬순서를 지정할 수 있다. 첫 번째 나오는 컬럼의 정렬순으로 정렬하고 그 안에서 다시 두 번째 나오는 컬럼의 정렬순으로 정렬한다. 정렬순서는 올림차순이 기본으로 지정되어 있으므로 'ACS'는 꼭 기재할 필요는 없다.

```
proc sql;
       select *
         from sashelp.class
        order by sex, name desc;
quit;
```

생성된 컬럼을 이용한 정렬

PROC SQL QUERY내에서 새로 만들어진 컬럼값 순서로 정렬을 하고자 하는 경우 컬럼 별명을 ORDER By 구에 지정하면 된다.

```
proc sql;
       select *,
              height*2.54 as height_cm format=4.1,
              weight*0.453592 as weight_kg format=4.1,
              22*(calculated height_cm/100)**2 as weight_sd format=4.1
         from sashelp.class
```

```
        order by weight_sd desc;
quit;
```

컬럼 위치를 이용한 정렬

SELECT 구에 있는 컬럼을 이용하여 정렬하고자 할 때는 ORDER BY 다음에 컬럼명 대신 SELECT 구에 위치한 컬럼의 순서를 지정하여 정렬할 수 있다.

```
proc sql;
       select name,
              sex,
              height*2.54 as height_cm format=4.1,
              weight*0.453592 as weight_kg format=4.1,
              22*(calculated height_cm/100)**2 as weight_sd format=4.1
         from sashelp.class
        order by 2, 5 desc;
quit;
```

단순 WHERE 구 이용

특정한 조건을 만족하는 행을 검색하고자 할 경우 WHERE 구 다음에 해당 조건을 기재하며 SELECT 구에 지정하지 않는 컬럼을 사용해도 무방하다.

```
proc sql;
       select *
         from sashelp.class
        where sex='M';
quit;
```

비교연산자를 이용한 행 검색

WHERE 구에 비교연산자를 사용하여 특정 데이터를 선택할 수 있다.

<표 1> 비교연산자의 종류와 기능

연산자	약어	기능
=	EQ	같다.
^=	NE	같지않다.
>	GT	크다.
>=	GE	크거나 같다.
<	LT	작다.
<=	LE	작거나 같다.

```
proc sql;
       select *
         from sashelp.class
        where height>60;
quit;
```

여러 개의 조건을 이용한 행 검색

WHERE 구에 논리연산자, 비교연산자 등 두 개 이상의 조건을 사용하여 데이터를 선택할 수 있다. 가장 많이 사용하는 논리연산자는 다음과 같다.

기호	약어	의미
&	AND	AND 이전, 이후 조건 모두 만족
|	OR	OR 이전, 이후 조건 중 하나라도 만족
^	NOT	따르는 조건이 만족하지 않음

```
proc sql;
       select *
         from sashelp.class
       where  height>60 AND sex='M';
quit;

proc sql;
       select *
         from sashelp.class
       where  height>70 OR weight<110;
```

```
quit;

proc sql;
       select *
         from sashelp.class
       where  NOT (height>70 OR weight<110);
quit;
```

NOTE) WHERE 구에 복잡한 조건이 들어가는 경우 '()'괄호를 이용하여 우선순위를 정하거나 조건을 명확히 알 수 있게 표현하는 것이 좋다.

(좋지 못한 표현) `where` **sex='M' and height>70 OR sex='F' and height>60**

(좋은 표현) `where` **(sex='M' and height>70) OR (sex='F' and height>60)**

BETWEEN-AND 연산자

컬럼값의 범위를 이용하여 행을 선택하고자 할 때 BETWEEN-AND 연산자를 이용하며, 지정된 값을 포함한다.

```
proc sql;
       select *
         from sashelp.class
        where height between 60 and 70;
quit;

proc sql;
       select *
         from sashelp.class
        where height GE 60 and height LE 70;
quit;
```

IS MISSING 연산자

IS MISSING 연산자는 컬럼에 Missing 값의 포함여부를 확인할 수 있게 해준다. IS NULL 연산자와 동일하며, IS NOT MISSING은 반대의 개념이다.

```
proc sql;
       select *
         from sashelp.class
        where height IS NOT MISSING[NULL];
quit;
```

IN 연산자

IN 다음에 오는 목록 내에 비교 대상 값이 존재하는지를 검증하는 기능을 하는 연산자로 OR 연산자와 동일한 기능을 한다. NOT IN 연산자는 반대의 개념이다.

```
proc sql;
    select  *,
            height*2.54 as height_inch format=4.1,
            weight*0.453592 as weight_kg format=4.1,
            case when sex='M' then 22*(calculated  height_inch/100)**2
                 when sex='F' then 21*(calculated  height_inch/100)**2
                 end as weight_sd format=4.1,
            calculated weight_kg- calculated weight_sd as weight_diff,
            case when calculated weight_diff>-5 then 'overweight'
                 when calculated weight_diff between -10 and -5 then 'moderate'
                 else 'underweight' end as result
     from  sashelp.class
     where  calculated result in ('overweight','underweight');
quit;
```

LIKE 연산자와 %(percent), _(underscore) 기호

LIKE 연산자는 주로 문자열 변수에 사용되며 일정한 규칙을 만족하는 행을 선택할 수 있다. %(percent)와 _(underscore) 기호는 규칙을 만드는 데 사용된다. %(percent) 기호는 길이에 관계없이 문자를 대표한다면 _(underscore)는 1byte 길이의 문자를 대표한다. 한글의 경우 1 문자가 2byte에 해당되므로 한글 한 문자는 '__'로 나타내어야 한다.).

```
proc sql;
       select  name, sex, age
         from  sashelp.class
        where  name like 'J%';
quit;
```

```
proc sql;
        select  name, sex, age
          from  sashelp.class
         where  name like '%e';
quit;

proc sql;
        select  name, sex, age
          from  sashelp.class
         where  name like 'Ja__';
quit;

proc sql;
        select  name, sex, age
          from  sashelp.class
         where  name like '___e';
quit;

proc sql;
        select  name, sex, age
          from  sashelp.class
         where  name like '_a%';
quit;

proc sql;
        select  name, sex, age
          from  sashelp.class
         where  name like 'J%e%';
quit;
```

요약통계량 작성

Aggregate 함수(summary 함수)는 Table에 저장된 데이터의 여러 특성을 나타내는 요약통계량을 제공해준다. PROC SQL에서 aggregate 함수의 인수가 하나의 컬럼일 때와 두 개 이상의 컬럼일 때 다르게 작용한다. 하나의 컬럼이 aggregate 함수의 인수로 지정된 경우, 해당 컬럼 내 값들로 연산이 이루어지며 이 때 GROUP BY 구가 PROC SQL 문에 나타나는 것이 일반적이다. 만약 GROUP BY 구가 없으면 aggregate 함수는 전체 데이터에 적용된다. 반면,

두 개 이상의 컬럼이 인수로 지정되었으면 컬럼 간 연산이 이루어지며 PROC SQL 문에 GROUP BY 문에 없는 것이 일반적이다. Aggregate 함수는 SELECT 구와 HAVING 구에서 사용된다.

Aggregate 함수

PROC SQL에서 제공하는 aggregate 함수는 아래와 같다.

함수	정의	함수	정의
AVG, MEAN	평균	RANGE	범위
COUNT, N, FREQ	비결측값의 갯수	STD	표준편차
CSS	수정된 제곱합	STDERR	표준오차
CV	변동계수	SUM	합
MAX	최대값	SUMWGT	가중합
MIN	최소값	T	t값($H_0 : \mu = 0$)
NMISS	결측값의 갯수	USS	수정되지 않은 제곱합
PRT	유의확률	VAR	분산

NOTE) PROC SQL에서 대부분의 SAS 함수를 사용할 수 있지만 모두 aggregate 함수로 취급되는 것은 아니다.

NOTE) 각 aggregate 함수는 상황에 따라 사용여부가 결정된다.

```
proc sql;
       select  country,
               avg(actual) as avg_actual
         from  sashelp.prdsale
        group  by country;
quit;

proc sql outbos=30;
       select  *,
               mean(actual,predict) as avg
         from  sashelp.prdsale
quit;

proc sql;
       select  Product_Category,
               year,
```

```
            count(Product_Group) as N_pg,
            avg(Profit) as avg_prf format=8.1
     from   sashelp.orsales
    group   by Product_Category,year;
quit;
```

NOTE) Aggregate 함수의 인수로 하나의 컬럼을 사용하는 경우 GROUP BY 구에 들어가는 컬럼은 반드시 SELECT 구에 들어가야 원하는 형태의 결과를 얻을 수 있다.

WHERE 구에 이용한 데이터 요약

Aggregate 함수의 인수에 두 개 이상의 컬럼을 지정하는 경우 컬럼 간 연산이 가능하고, PROC SQL 문에 GROUP BY 구는 사용하지 않는 것이 일반적이라고 하였다. 반면, WHRER 구에 aggregate 함수의 결과를 이용하여 조건을 지정할 수 있다.

```
proc sql outobs=30;
       select *,
              mean(ELECTRIC,MASONRY) as m_worker
         from  sashelp.workers
        where calculated m_worker GE 300;
quit;
```

NOTE) calculated *keyword*는 반드시 사용해야 함.

전체 데이터의 요약통계량

Aggregate 함수를 이용해 전체 데이터에 대한 요약통계량을 구할 수 있다. PROC SQL에 GROUP BY 구를 사용하지 않고 SELECT 구에는 전체 데이터에 대해 관심 있는 컬럼 하나를 aggregate 함수의 인수로 지정하면 된다.

```
proc sql;
        select  avg(height) as avg_height format=4.1,
                avg(weight) as avg_weight format=4.1
          from  sashelp.class;
quit;
```

NOTE) 위의 예제처럼 SELECT 구에 aggregate 함수만 포함된 행만 지정하면 여러 개의 행을 가진 데이터에서 하나의 결과값을 얻을 수 있다.

요약통계량 컬럼 만들기

Aggregate 함수의 결과로 새로운 컬럼을 만들거나, 결과를 이용한 새로운 컬럼을 만들 수 있다. 아래의 조건을 만족하는 경우에 해당된다.

- GROUP BY 구에 없는 aggregate 함수가 포함된 컬럼이 SELECT 구에 있는 경우
- GROUP BY 구에 없는 컬럼이 SELECT 구에 있는 경우

```
proc sql;
        select  name,
                height,
                avg(height) as avg_height format=8.1
          from  sashelp.class;
quit;
```

```
proc sql;
        select  name,
                height,
                (height/avg(height))*100 as h_grade format=8.1
          from  sashelp.class
         order  by h_grade desc;
quit;
```

개별값을 이용한 Aggregate 함수

컬럼의 중복값을 제거하고 개별값을 알고자 할 때 DISTINCT Keyword를 사용한다는 것을 앞서 다루었다. 만약 개별값에 대한 요약통계량을 구하고자 한다면 aggregate 함수내에 DISTINCT keyword를 사용하면 된다.

아래의 3가지 예제의 결과를 비교해보자.

```
proc sql;
        select  count(distinct county) as count
          from  sashelp.prdsal3;
quit;

proc sql;
        select  count(COUNTY) as count
          from  sashelp.prdsal3;
quit;

proc sql;
        select  count(*) as count
          from  sashelp.prdsal3;
quit;
```

Aggregate 함수의 결측값 처리

대부분의 aggregate 함수는 결측값을 제외한 결과를 제공한다. 따라서, aggregate 함수의 인수로 결측값이 포함된 컬럼을 사용하는 경우 예상하지 못한 결과를 얻을 수 있으므로 주의해야한다.

```
proc sql;
        select  name,
                case when name='Louise' then .
                else height end as height,
                avg(calculated height) as avg_height
          from  sashelp.class
        where   name in ('Judy','Louise','Philip');
quit;
```

그룹 데이터

GROUP BY 구는 하나 이상 컬럼을 이용해 데이터를 그룹화할 수 있다. PROC SQL 문에 GROUP BY 구를 사용하면, SELECT 구와 HAVING 구에 aggregate 함수를 사용해 각 그룹 데이터에 대한 요약통계량을 구할 수 있다. 두 개 이상의 컬럼을 이용하는 경우 GROUP BY 구에 컬럼 이름을 ,(컴마)로 구분하여 작성한다. 그룹화순서는 GROUP BY에 적힌 순이다.

```
proc sql;
        select  country,
                avg(actual) as avg_actual
          from  sashelp.prdsale
         group  by country;
quit;
```

NOTE) Aggregate 함수는 사용하지 않으면 PROC SQL문은 GROUP BY 구를 ORDER BY 구로 취급하고 log 창에 관련 메시지를 출력한다. 정렬순서는 올림차순이 기본으로 지정되어 있으며 정렬순서를 내림차순으로 변경하고자 하는 경우 ORDER BY 구에서 DESC keyword를 사용해야 한다.

```
proc sql outobs=30;
        select  country,
                region,
                division,
                actual
          from  sashelp.prdsale
         group  by actual
         order  by actual desc;
quit;
```

```
WARNING: A GROUP BY clause has been transformed into an ORDER BY clause because
neither the SELECT clause nor the optional HAVING clause of the associated
table-expression referenced a summary function.
```

NOTE) 컬럼에 결측값이 포함이 되어 있는 경우 PROC SQL은 결측값을 하나의 그룹으로 인식한다. 이로 인한 예기치 못한 결과를 얻을 수 있기 때문에, 결측값이 잇는 경우 WHERE 구에서 결측값을 제외한다는 조건을 넣을 필요가 있다.

그룹 데이터에 조건달기

GROUP BY 구와 함께 HAVING 구를 이용하면 조건을 만족하는 그룹 데이터만 선택할 수 있다. WHERE 구가 개개의 행에 영향을 미치는 것처럼 HAVING 구는 그룹 데이터에 영향을 미친다. HAVING 구를 사용하면 PROC SQL은 HAVING 구에 표현된 조건을 만족하는 그룹 데이터만 보여준다.

```
proc sql;
        select  country,
                region,
                avg(actual) as avg_actual
          from  sashelp.prdsale
         group  by country,region;
quit;
```

```
proc sql;
        select  country,
                region,
                avg(actual) as avg_actual
          from  sashelp.prdsale
         group  by country,region
        having  avg_actual>500;
quit;
```

HAVING 구와 WHERE 구 선택

HAVING 구와 WHERE 구의 차이점은 아래의 표와 같다. HAVING 구는 그룹 데이터를 다루기 위해서 사용되기 때문에 보통 PROC SQL 문은 GROUP BY 구와 aggregate 함수를 포함한다.

< HAVING 구와 WHERE 구의 차이>

HAVING 구	WHERE 구
행 그룹을 포함시키거나 제외할 때 사용	개별 행을 포함시키거나 제외할 때 사용
GROUP BY 다음에 위치	GROUP BY 앞에 위치
GROUP BY 구에 영향을 받는다. 만약 GROUP BY 구를 사용하지 않은 경우 WHERE 구처럼 인식된다.	GROUP BY 구에 영향을 받지 않는다.
GROUP BY구와 aggregate 함수 다음에 실행	GROUP BY 구와 aggregate 함수 전에 실행

```
proc sql;
       select *
         from sashelp.class
       having height>55;
quit;
```

Query문 작성 검토

VALIDATE 문을 이용하면 PROC SQL을 실행하지 않고 query문의 오작성 여부를 확인할 수 있다. PROC SQL은 log 창에 문법의 오작성 여부를 보여준다.

예제 ?-?

```
proc sql;
       validate
       select name,
              case when name='Louise' then .
              else height end as height,
              avg(calculated height) as avg_height
         from sashelp.class
       where  name in ('Judy','Louise','Philip');
quit;
```

두 개의 테이블 다루기

데이터 분석 및 보고서를 작성을 위한 데이터가 두 개 이상의 Table에 나누어져 있는 경우 Table 결합(join)을 통해 각 Table에 있는 데이터를 가져올 수 있다. 즉, 데이블 결합은 여러 table에 있는 필요한 데이터를 마치 하나의 table에 있는 것과 같이 할 수 있다. 하지만 table 내용을 변경할 순 없다. 가장 간단한 table 결합방식은 PROC SQL 문에 FROM 구에 두 개 table을 ,(comma)로 구분하여 적는 것이다.

다음과 같은 구조를 가지고 있는 테이블 one과 two를 결합하면 테이블 three를 얻을 수 있다. 이러한 결합방법은 테이블 one의 각 행에 테이블 two의 모든 행이 결합되는 Cartesian 곱의 형태이다. 이 때, SAS log창에 다음과 같은 메시지가 출력된다.

```
NOTE: The execution of this query involves performing one or more
Cartesian product joins that can not be optimized.
```

예제 ?-?

Table One

X	Y
1	2
2	3

Table Two

X	Z
2	5
3	6
4	9

```
proc sql;
       select *
         from one, two;
quit;
```

Table Three

X	Y	X	Z
1	2	2	5
1	2	3	6
1	2	4	9
2	3	2	5
2	3	3	6
2	3	4	9

하지만, 테이블 결합 시 Cartesian 곱의 결과가 필요한 경우는 거의 없으며 단지 그 일부의 결과만 필요할 뿐이다. 특히 결합하고자 하는 테이블이 대용량 데이터인 경우 Cartesian 곱의 결과는 엄청나다. 일반적인 테이블 결합은 2가지 형태로 나누어진다.

1. 내적 결합(inner join) : FROM 구에 나오는 모든 테이블 중에서 서로 일치하는 행만 선택되는 형태
2. 외적 결합(outer join) : 내적결합에 다른 테이블과 일치하지 않는 행도 선택가능한 형태로 왼쪽(left), 오른쪽(right), 전체(full) 외적결합이 있다.

내적 결합(inner join)

내적 결합은 첫 번째 table과 두 번째 table의 일치하는 행만 선택하여 결합하는 것을 말한

다. 서로 일치하는 값을 가지고 있는지를 비교하는 컬럼은 WHERE 구에 지정한다.

아래의 예제는 Cartesian 곱의 예제에 WHERE 구만 추가한 것이다. Table one의 X 컬럼과 table two의 X 컬럼 값을 비교하여 같은 값을 가지는 행만 선택하기 위해서 WHERE 구에 X 컬럼을 기재하였다.

예제 ?-?

```
proc sql;
    select *
     from one, two
    where one.x=two.x;
quit;
```

OUTPUT

X	Y	X	Z
2	3	2	5

결과를 보면 하나의 행만 포함한다. 이유는 두 table의 X 컬럼을 비교하면 하나의 값만 일치하기 때문이다. 이렇게 내적 결합(inner join)은 하나의 행만 선택된다. 반면 외적 결합(outer join)은 일치하지 않는 행도 선택할 수 있다. 이 부분은 외적 결합에서 다루도록 하겠다.

NOTE) 위의 예제에서 WHERE 구 컬럼명 앞에 table 명이 기재되어있다. 이것은 여러 table에 같은 컬럼명이 있는 경우 어느 table의 컬럼을 의미하는지를 분명하게 하기 위해서 반드시 필요하다.

Table 별명(aliases) 사용하기

Table 별명은 컬럼의 별명과 마찬가지로 PROC SQL내에서 일시적으로 table명을 대신하는 것으로 FROM 구 테이블명 다음에 지정한다. 컬럼별명 지정 시 사용하는 AS keyword를 반드시 사용할 필요는 없다. 테이블 별명을 사용하는 이유는 테이블 결합 시 SELECT 구에 컬럼은 각 컬럼이 위치한 테이블명과 함께 기재해야 한다(예, 테이블명.컬럼). 이 때 긴 이름의 테이블명 대신 별명을 사용하는 경우 Query 문을 간략하게 하여 가독성을 높일 수 있기 때문이다.

아래의 예제는 학급의 학생들의 발육상태를 알아보기 위해서 몸무게와 키를 비교해보고자 몸무게(weight)와 키(height)의 정보가 있는 두 테이블을 결합한 것이다. 두 테이블에 공통적으로 name 컬럼이 포함되어 있어 이를 기준으로 하였으며, 각 테이블에 별명을 지정하였다. 또한, SELECT 구와 WHERE 구에 컬럼명을 기재할 시 테이블 별명을 같이 적어 컬럼이 어느 테이블에 위치하고 있는지를 명시하였다.

예제 ?-?

```
proc sql;
       select a.name,
              a.height,
              b.weight
         from sql.class1 a, sql.class2 b
        where a.name=b.name;
quit;
```

테이블 결합 시 정렬

테이블 결합 시 ORDER BY 구를 이용하여 테이블을 정렬할 수 있다.

다음 예제는 height 컬럼을 기준으로 올림차순으로 정렬한 예이다. 실제 name 컬럼을 제외한 컬럼들은 모두 한 table에만 있는 컬럼이기 때문에 컬럼명 앞에 테이블명 또는 테이블 별명을 기재할 필요는 없다.

예제 ?-?

```
proc sql;
       select a.name,
              height,
              weight
         from sql.class1 a, sql.class2 b
        where a.name=b.name
        order by height;
quit;
```

INNER JOIN 명령어를 이용한 테이블 결합

테이블의 내적 결합은 INNER JOIN 명령어를 이용해서도 가능하다. INNER JOIN 사용 시 ON 구가 함께 사용되는 데 WHERE 구에 역할을 대신한다. PROC SQL에서는 주요 다른 결합(외적, 왼쪽, 오른쪽 결합)과 호환성을 고려해 이러한 명령어를 제공하고 있다. INNER JOIN을 FROM 구에 기재하고 WHERE 구 대신 ON 구를 사용하면 동일한 결과를 얻을 수 있다. 아래의 예제는 <예제 >를 INNER JOIN 명령어를 이용하여 다시 작성한 것으로 동일한 결과를 얻을 수 있다.

예제 ?-?	

```
proc sql;
       select a.name,
              height,
              weight
         from sql.class1 a inner join sql.class2 b
           on  a.name=b.name
        order by height;
quit;
```

비교연산자를 사용한 테이블 결합

테이블 결합 시 WHERE 구에 '='(equal)과 같은 비교연산자를 이용하여 특정한 조건을 만족하는 행만 선택할 수 있다. 아래의 예제는 class1과 class2 테이블 결합 시 Height가 70이상인 데이터만 선택하고자 하는 경우이다. WHERE 구에 비교연산자 '>'를 사용하였다.

예제 ?-?	

```
proc sql;
       select a.name,
              height,
              weight
         from sql.class1 a inner join sql.class2 b
          on   a.name=b.name
        where height>70
        order by height;
quit;
```

테이블 결합 시 결측값 처리

대부분의 database에서는 NULL 값은 하나의 값으로 처리하지만 테이블 결합 시 제외시킨다. 하지만 PROC SQL에서는 NULL 값은 결측값으로 인식하여 결합 시 동일한 형태(숫자 또는 문자)의 결측값과 결합된다.

만약 테이블 결합 시 NULL 값을 제외하고자 한다면 'IS NOT MISSING' 연산자를 사용하여 제외할수 있다.

다음 예제는 table A와 table B를 컬럼 b를 기준으로 결합한 것이다. 이 때 컬럼 b는 두 테이블 모두에서 NULL 값을 가지고 있다. 테이블 A의 c 행은 NULL 값은 테이블 B의 모든 NULL 값과 일치하여 결합되었다. 이러한 결과는 테이블 결합 시 의도하지 않는 결과이다.

예제 ?-?

Table A

a	b
a	1
b	2
c	.
d	4

Table B

a	b
a	1
b	2
c	.
d	4
e	.
f	.

```
proc sql;
       select one.a, one.b, two.a, two.b
     from    one, two
     where   one.b=two.b;
quit;
```

Table C

a	b	a	b
a	1	a	1
b	2	b	2
c	.	c	.
d	4	d	4
c	.	e	.
c	.	f	.

테이블 결합시 결측값을 제외하기 위해서는 'IS NOT MISSING' 연산자를 사용하면 다음과 같다.

예제 ?-?

```
proc sql;
       select one.a,
              one.b,
              two.a,
              two.b
     from     one, two
     where    one.b=two.b and
              one.b is not missing;
quit;
```

Table C

a	b	a	b
a	1	a	1
b	2	b	2
d	4	d	4

여러 개의 컬럼을 이용한 테이블 결합

행이 여러 개의 컬럼 값의 조합으로 특징지어 나타난다면, 테이블 결합 시 필요한 컬럼을 모두 이용해야한다. 예를 들어, 도시의 특성을 나타내는 컬럼이 여러 테이블에 나누어져 있는 경우, 단순히 도시명을 나타내는 컬럼만을 사용하여 테이블을 결합하게 되면 심각한 오류를 범할 수 있다. 왜냐하면 여러 나라에 동일한 도시명이 있을 수 있기 때문이다. 이런 경우 정확히 해당 도시명에 맞게 테이블 결합하기 위해서는 국가와 도시를 나타내는 컬럼을 모두 이용해야만 원하는 결과를 얻을 수 있다. 즉, WHERE 구에 나라명과 도시명을 같이 기재해야 한다.

다음 예제를 보자. sql.sinpan 테이블과 sql.ca 테이블을 결합하여 신판사용액과 CA사용액을 합한 전체사용액을 알고자하는 경우이다. 각 테이블은 기준년월과 리스크등급별 사용액으로 정리되어있다. 따라서, 기준년월과 리스크등급에 맞춰 테이블을 결합해야하므로 WHERE 구에 기준년월과 리스크등급을 사용하였다. 만약 WHERE 구에 기준년월과 리스크등급 중 하나만 사용하는 경우에는 원하지 않는 결과가 나타난다.

예제 ?-?

```
proc sql;
       select a.기준년월,
              a.리스크등급,
              a.전체신판액,
              b.전체CA,
              a.전체신판액+b.전체CA as 전체사용액
         from sql.sinpan a, sql.ca b
       where  a.기준년월=b.기준년월
         and  a.리스크등급=b.리스크등급;
quit;
```

여러 개의 테이블을 이용한 결합

필요한 데이터가 여러 개의 테이블에 분산되어있을 수 있다. 이런 경우, 두 개의 테이블을 결합하는 방법을 이용하면 쉽게 세 개의 테이블 결합도 가능하다. 위의 예를 약간 변형해보자. sql.sinpan 테이블과 sql.ca 테이블에 있는 리스크등급을 나타내는 컬럼이 있다. 하지만, sql.sinpan 테이블에는 지금처럼 ('A','B','C','D') 형태로 표현되어 있고 sql.ca 테이블에는 ('01','02','03','04')로 표현되어있다. 단 두 정보는 표현방식만 다를 뿐 의미하는 바는 동일하다고 가정하자. 두 테이블을 결합하기 위해서는 기준년월과 리스크등급 컬럼을 사용해야 하지만 리스크등급이 테이블에 따라 다르게 표현되어 있어 직접적인 결합은 불가능하다. 이런 경우 또 다른 테이블에 이 두 컬럼을 연결시켜주는 정보가 있다면 이를 이용해야한다. 이런 경우 3개의 테이블 결합이 필요하다.

예제 ?-?

```
proc sql;
       select a.기준년월,
              a.리스크등급,
              a.전체신판액,
              b.전체CA,
              a.전체신판액+b.전체CA as 전체사용액
         from sql.sinpan a, sql.ca1 b, sql.risk c
        where a.기준년월=b.기준년월
          and a.리스크등급=c.리스크등급1
          and b.리스크등급=c.리스크등급2;
quit;
```

자기 결합(self-join)

테이블 결합에는 동일한 테이블 내의 컬럼을 결합하는 자기 결합(self-join)의 형태가 있다. 이러한 결합은 aggregate 함수의 결과를 동일한 테이블내의 다른 컬럼과 결합하거나 한 컬럼 일부의 값을 다른 컬럼과 결합할 때 사용된다. 자기 결합은 PROC SQL이 한 테이블내의 정보를 복사하여 그 테이블에 다시 결합시킨다고 생각하면 된다.

다음 예제는 위의 예제에서 2009년 5월을 기준으로 기준년월별 신판증가률을 구하고자 한다. 자기 결합방법을 사용하면 2009년 5월 리스크등급별 신판사용액을 sql.sinpan 테이플에서 복사하여 다시 sql.sinpan 테이블에 리스크등급별로 결합시켜면 신판증가률을 구할 수 있다.

예제 ?-?

```
proc sql
       select a.기준년월,
              a.리스크등급,
              a.전체신판액,
              b.전체신판액 as 전체신판액_0905,
              100*a.전체신판액/b.전체신판액 as 신판증가율 format=8.1
         from sql.sinpan a, sql.sinpan b
        where a.리스크등급=b.리스크등급
          and b.기준년월=200905
quit
```

외적 결합(OUTER JOIN)을 이용한 테이블 결합

외적 결합은 하나의 기준 테이블이 있고 하나 이상의 비교 테이블이 있다고 한다면 기준 테이블의 모든 행이 결합의 결과로 나타나지만 비교 테이블과 일치하는 행과 일치하지 않는 행으로 구분할 수 있다. 즉, 외적 결합(OUTER JOIN)은 내적 결합(INNER JOIN)에 다른 테이블과 일치하지 않는 행이 추가된 것이라고 할 수 있다. 여기서 일치하지 않는 행은 NULL 값을 가진다. 테이블 결합에 사용되는 컬럼 지정은 WHERE 구 대신에 ON 구가 사용된다. 물론, WHERE 구에는 결합 시 필요한 조건을 지정할 수 있다.

왼쪽 외적결합(Left Outer Join)

왼쪽 외적결합은 가장 왼쪽 테이블(FROM 구의 첫 번째 테이블)을 기준 테이블로 하고 오른쪽 테이블과 일치하는 행과 일치하지 않는 행을 포함하며 LEFT OUTER JOIN과 ON 명령어를 사용한다.

아래 예제는 sql.class1 테이블에 있는 학생들을 기준으로 sql.class2에 있는 학생들의 키 컬럼을 왼쪽 외적결합을 통해 결합하고자 한다.

예제 ?-?

```
proc sql;
       select a.name,
              b.height
         from sql.class1 a left join sql.class2 b
           on  a.name=b.name
        order by name;
quit;
```

OUTPUT

Name	Height
Alfred	112.5
Alice	84
Barbara	98
Bayoba	.
Carol	102.5
Henry	102.5
James	83
Jane	84.5
Janet	112.5
Jeffrey	84
John	99.5
Joyce	50.5
Judy	90
Louise	77
Mary	112
Philip	150
Robert	128
Ronald	133
Sindy	.
Thomas	85
William	112

왼쪽 외적결합의 결과 sql.class1 테이블의 있는 행은 sql.class2 테이블에 키 컬럼과의 일치 여부와 관계없이 모든 행이 나타난다. 일치되는 경우 키 컬럼의 값이 일치되지 않는 경우 NULL 값이 들어간다.

오른쪽 외적결합(Right Outer Join)

오른쪽 외적결합은 왼쪽 외적결합의 반대로 가장 오른쪽 테이블을 기준 테이블로 하고 왼쪽 테이블과 일치하는 행과 일치하지 않는 행을 포함하며 RIGHT OUTER JOIN과 ON 명령어를 사용한다. 왼쪽 외적결합과의 차이점은 기준이 되는 테이블이 FROM 구 첫 번째로 나오는 테이블이면 왼쪽 외적결합이고 두 번째로 나오는 테이블이면 오른쪽 외적결합이다.

아래의 예제는 sql.class1 테이블을 FROM 구 두 번째에 기재하는 경우로 왼쪽 외적결합의 예제와 동일한 결과를 얻을 수 있다.

예제 ?-?	

```
proc sql;
       select

         from
           on
        order by ;
quit;
```

전체 외적결합(Full Outer Join)

전체 외적결합은 기준이 되는 테이블이 없이 FROM 구에 나오는 모든 테이블의 일치하는 행과 일치하는 않는 행이 포함되며 FULL JOIN과 ON 명령어를 사용한다.

다음 예제는 sql.class1 테이블과 sql.class2 테이블를 전체 외적결합을 하고자 한다.

예제 ?-?	

```
proc sql;
       select a.name as name1 label='Name 1',
              b.name as name2 label='Name 2',
              a.weight,
              b.height
         from sql.class1 a full join sql.class2 b
           on  a.name=b.name
        order by name1, name2;
quit;
```

특수형태 결합

PROC SQL은 테이블 결합방법으로 내적결합과 외적결합 이외에도 Cross 결합, union 결합, natural 결합을 지원한다. 각 결합방법에 대해서 알아보기로 하자.

CROSS 결합(Join)

Cross 결합은 Cartesian 곱으로 두 테이블의 곱을 얻을 수 있다. Cross 결합도 결합 시 Cartesian 곱처럼 WHERE 구를 통해 조건을 지정할 수 있다.

앞에서 Cartesian 곱에서 사용된 예제를 그대로 CROSS 결합을 사용해 결합하면 같은 결과를 얻을 수 있다. 확인해보자.

예제 ?-?

```
proc sql;
       select *
         from one cross join two;
quit;
```

Cross 결합 시 Cartesian 곱에서 처럼 SAS log 창에 다음과 같은 메시지가 출력된다.

```
NOTE: The execution of this query involves performing one or more
Cartesian product joins that can not be optimized.
```

UNION 결합(Join)

Union 결합은 행의 일치여부와 관계없이 두 테이블의 모든 컬럼과 행이 포함하는 결합형태이다. Union 결합을 이용한 테이블 조합방법은 OUTER UNION 집합 연산자(이후 설명)를 이용한 결합과 유사하다. Union 결합도 WHERE 구를 통해 결합 시 조건을 지정할 수 있다.

다음 예제는 cross 결합의 예제를 이용해 union 결합을 한 결과이다. 차이점을 살펴보자.

예제 ?-?

```
proc sql;
       select *
         from one union join two;
quit;
```

NATURAL 결합(Join)

Natural 결합은 테이블 결합 시 기준이 되는 컬럼을 사용자가 지정하는 것이 아니라 자동적으로 선택한다. PROC SQL은 natural 결합 시 각 테이블에서 동일한 이름과 형태를 가진 컬럼들을 찾아 해당 컬럼을 결합의 기준으로 삼으며 동일한 값을 가지는 행만 가져온다. 따라서, 기준 컬럼을 지정할 필요가 없기 때문에 natural 결합에서는 ON 구는 사용할 필요가 없어진다. Natural 결합의 장점은 코딩이 간결하다는 것이다. ON 구를 사용할 필요가 없고, PROC SQL이 자동적으로 컬럼을 결정하기 때문에 테이블명이나 테이블별명을 컬럼명 앞에 지정할 필요가 없다. 만약 natural 결합 시 공통된 컬럼명과 형태가 없는 경우 Cartesian 곱을 결과로 얻을 수 있다.

Natural 결합 시 중요한 것은 사전에 테이블 결합 후의 테이블 형태가 구체적으로 그려져 있어야 한다는 것이다. 이는 사용자가 사전에 데이터 전체를 충분히 알고 있다는 가정을 하는 것과 동일하다. 예를 들어, natural 결합 이전에 두 테이블에 공통된 컬럼은 하나만 있다고 생각했는데 실제 두 개의 공통된 컬럼이 있는 경우 예기치 않는 결과를 얻을 수 있다(PROC SQL이 Query를 어떻게 실행하는지 정확히 알고 싶다면 FEEDBACK 옵션을 통해 알 수 있다.). 또한, 연속된 Query 작성 시 중간에 한 테이블에만 컬럼이 삭제되거나 다른 테이블에 있는 동일한 형태의 컬럼이 추가되는 경우 그 결과가 달라질 수 있다는 점도 유의해야 한다. 추가적으로, natural 결합은 두 테이블의 공통 컬럼의 같은 값에 기반을 둔 결합이라는 점이다. 만약 부등식이나 다른 비교연산자에 기본한 결합을 하고자 한다면 내적 또는 외적 결합을 사용하는 것이 바람직하다.

다음 예제는 sql.class1 테이블과 sql.class2 테이블을 natural 결합방법으로 결합한 것이다. 결과는 어떠한 결합형태와 동일한가?

예제 ?-?

```
proc sql;
      select *
        from sql.class1 natural join sql.class2;
quit;
```

결합 시 COALESCE 함수 이용

외적결합 시 일치하지 않는 행은 결측값을 가진다. 결측값 대신 유사한 성격의 컬럼의 값으로 변경하고자 하는 경우 COALESCE 함수를 이용하면 간단히 해결할 수 있다. COALESCE 함수는 앞에서도 다루었듯이 인수로 컬럼들이 사용되고, 결측값이 있는 경우 첫 번째로 결측값이 아닌 컬럼의 값으로 대체된다. COALESCE 함수는 내적결합과 외적결합 둘 다에서 사용가능하다.

다음 예제는 전체 외적결합(Full Outer join) 예제에서 sql.class1 테이블과 sql.class2 테이블을 전체 외적결합 시 공통적으로 들어가 있는 name 컬럼을 하나의 컬럼으로 만들기 위해 sql.class1에 해당 컬럼이 없어 NULL값으로 표시된 컬럼의 값을 sql.class2의 name 컬럼의 값으로 대체하기 위한 예제이다.

예제 ?-?

```
proc sql;
       select COALESCE(a.name,b.name) as name label='Name',
              a.weight,
              b.height
         from sql.class1 a full join sql.class2 b
           on a.name=b.name
        order by name;
quit;
```

서브쿼리를 이용한 데이터 선택

데이트 결합을 통해 다양한 작업이 가능하지만 DB에 하나 이상의 질문을 해야하는 경우가 있다. 쿼리의 결과를 가져와 다른 쿼리에 사용해야 할 경우도 있다. 이런 경우에 필요한 것이 바로 서브쿼리(subquery, 보통 괄호안에 포함)이다. 서브쿼리를 사용하면 데이터의 중복을 피할 수 있고, 쿼리를 좀 더 동적으로 작성할 수 있다. 또한, 테이블 결합 시 이용하면 효과는 배가 된다. 서브쿼리는 다른 query에 포함된 query로 내부쿼리(inner query)라고도 한다. 서브쿼리를 포함하고 있는 쿼리는 외적쿼리(outer query)라 한다. 서브쿼리는 어느 구에 포함되어 있느냐에 따라 하나의 값 또는 여러 개의 값을 가질 수 있다. 서브쿼리는 특히 WHERE 구나 HAVING 구에 가장 많이 사용된다.

단일값 서브쿼리

단일값 서브쿼리는 서브쿼리 실행결과 하나의 컬럼과 행을 가져오는 경우를 말한다. WHERE 구나 HAVING 구에 비교연산자를 함께 사용해 얻을 수 있다. 이 서브쿼리는 반드시 하나의 값만 나타나야하며 그렇지 않을 경우 SAS log창에 에러 메시지가 출력된다.

아래의 예제는 WHERE 구에 벨기에보다 인구가 많은 미국이 선택되는 서브쿼리를 사용했다. 이 서브쿼리는 먼저 계산을 한 후 벨기에 인구를 주 쿼리(outer query)문으로 넘긴 것이다.

예제 ?-?

```
proc sql;
       select Name 'State',
              population format=comma10.
         from sql.unitedstates
       where  population gt (
              select population
                from sql.countries
               where name = "Belgium");
quit;
```

내부적으로 서브쿼리를 실행한 후 아래와 같은 쿼리와 동일하다.

예제 ?-?

```
proc sql;
       select Name 'State',
              population format=comma10.
         from sql.unitedstates
       where  population gt 10162614;
quit;
```

다중값 서브쿼리

다중값 서브쿼리는 서브쿼리 실행결과 한 개의 컬럼에 여러 개의 값이 나타나는 경우를 말한다. WHERE 구나 HAVING 구를 IN 연산자나 ANY, ALL과 같은 비교연산자와 함께 사용하여 얻을 수 있다. 아래의 예제는 오일 생산국의 인구를 보여준다. 먼저, 서브쿼리로 OILPROD 테이블에 있는 모든 국가를 가져온다. 그런 다음 주 쿼리에서는 COUNTRIES 테이블에 있는 국가 중 서브쿼리의 결과와 일치하는 국가를 가져온다.

예제 ?-?

```
proc sql outobs=5;
       select name 'Country',
              Population format=comma15.
         from sql.countries
        where Name in (
              select Country
                from sql.oilprod);
quit;
```

만약 NOT IN 연산자를 사용한다면, query 결과는 OILPROD 테이블에 포함되지 않는 모든 국가를 얻을 수 있다.

예제 ?-?

```
proc sql outobs=5;
       select name 'Country',
              Population format=comma15.
         from sql.countries
        where Name not in (
              select Country
                from sql.oilprod);
quit;
```

상호연관 서브쿼리

앞의 서브쿼리는 외부쿼리와 독립적으로 실행되는 간단한 쿼리이다. 하지만 서브쿼리와 주 쿼리가 독립적으로 실행되는 것이 아니라 각 쿼리의 값을 서로 참조하는 방식으로 쿼리를 작성할 수 있다. 이러한 쿼리를 상호연관 서브쿼리(correlated subquery)라 한다. 상호연관 서브쿼리는 단일값과 다중값을 가질 수 있다.

다음 예제는 아프리카의 주요 오일 보유국을 선택하는 것이다.

예제 ?-?

```
proc sql;
       select *
         from sql.oilrsrvs o
        where 'Africa' =
              (select   Continent
                 from  sql.countries c
                where  c.Name = o.country);
quit;
```

간단히 설명하면, 주 쿼리가 OILRSRVS 테이블의 첫 번째 행을 가져와 서브쿼리에 그 값(여기에서 'Algeria')을 넘긴다. 이 시점에서의 서브쿼리 형태는 아래와 같다.

(select Continent from sql.countries c
where c.Name = 'Algeria')'

서브쿼리는 COUNTRIES 테이블에서 해당 국가를 선택해서 다시 주 쿼리의 WHERE 구에 해당국가의 대륙명을 넘겨준다. 만약 그 대륙이 'Africa'면 선택되어 출력된다. 즉, 주 쿼리는

OILRSRVS 테이블에서 각 행을 가져와 서브쿼리로 넘겨주고, 서브쿼리는 해당 국가의 대륙을 주 쿼리 WHERE 구에 넘겨주는 방식이다.

NOTE) 위의 예제에서 WHERE 구에 =(equal) 연산자를 사용하였다. 만약 서브쿼리에 단일값을 가지려고 한다면 '='을 사용한다. 그러나 서브쿼리의 결과 다중값을 가지고 싶다면 IN 연산자나 ANY나 ALL 같은 비교연산자를 사용해야한다. 연산자에 대한 자세한 사항은 다음 장에서 자세히 다루고자 한다.

그룹값의 존재성 검증

EXISTS 조건은 값들이 있는지를 시험해보는 것이다. 서브쿼리에 의해 하나의 행이라도 존재하면 EXISTS 조건은 참이 되고 하나도 존재하지 않으면 거짓이 된다. 역으로 서브쿼리에 의해 하나의 행도 존재하지 않는다면 NOT EXISTS 조건은 참이 되고 하나의 행이라도 존재하면 NOT EXISTS 조건은 거짓이 된다.

다음 예제는 상호연관 서브쿼리와 동일한 결과를 주는 것으로 아프리카 대륙에 오일 보유국이 있는 지를 확인하는 데 EXISTS를 사용해보았다.

예제 ?-?

```
proc sql;
       select * from sql.oilrsrvs o
       where  exists
              (select Continent from sql.countries c
                where c.Name = o.country
                  and Continent = 'Africa');
quit;
```

차이점은 Continent = 'Africa' 조건이 앞의 쿼리에서는 주 쿼리에 있었지만 현재는 서브쿼리의 WHERE구에 있는 것이다.

서브쿼리의 단계적 실행구조

서브쿼리의 실행순서를 보면 가장 안쪽에 위치하고 있는 서브쿼리가 실행되고 그 값이 다음 외부쿼리에 이용되는 구조를 가지고 있다. 즉, 서브쿼리의 값이 외부쿼리에 이용되는 구조이다. 따라서 계산은 언제나 가장 안쪽의 쿼리로 시작해서 바깥 방향으로 작용한다.

다음 예제는 아프리카에 있는 주요 산유국을 나타내는 것이다. 이 예제를 이용해 서브쿼리

구조의 실행순서를 살펴보도록 하자.

1. 가장 안쪽의 서브쿼리가 먼저 실행되어 아프리카에 있는 국가명을 가져온다.
2. 서브쿼리에서 가져온 국가명과 OILRSRVS 테이블에 있는 국가명을 비교해 일치된 국가명만 가져온다.
3. 마지막으로, 주 쿼리에서는 WHERE 구를 통해 서브쿼리의 결과와 WORLDCITYCOORDS 테이블에 자료와 비교해 일치하는 국가의 도시와 좌표를 가져온다.

예제 ?-?

```
proc sql;
       select * from sql.worldcitycoords
        where country in
              (select country from sql.oilrsrvs o
               where o.country in
                     (select Name from sql.countries c
                      where c.continent='Africa'));
quit;
```

테이블 결합과 서브쿼리 이용

하나의 쿼리안에 서브쿼리와 테이블 결합을 동시에 할 수 있다. USCITYCOORDS 테이블에서 각 도시에서 가장 가까운 도시를 찾는다고 가정해보자. 쿼리는 먼저 선택된 도시간의 거리를 계산하고, 각 도시에서 가장 가까운 도시를 선택할 것이다. 이러한 쿼리를 작성하기 위해서는 USCITYCOORDS 테이블을 자기 결합(self-join) 해야하고, 다시 서브쿼리에서 자기 결합을 이용해 가장 가까운 도시를 결정해야 한다.

두 좌표간 거리를 구하는 공식은 다음과 같다.

SQRT(((Latitude2 - Latitude1)**2) + ((Longitude2 - Longitude2)**2))

위의 거리를 구하는 공식이 정확하지는 않지만 가장 가까운 도시를 결정하는 데는 충분하다.

예제 ?-?

```
proc sql;
       select * from sql.worldcitycoords
        where country in
              (select country from sql.oilrsrvs o
               where o.country in
                     (select Name from sql.countries c
                      where c.continent='Africa'));
quit;
```

외적쿼리는 테이블 A의 도시 A1과 테이블 B의 도시 B1간의 거리를 구하기 위해서 자기결합(self-join)을 한다. 단, 도시 A1과 도시 B2는 동일한 도시는 아니다. PROC SQL는 다시 서브커리를 실행하는 데 다시 자기결합(self-join)을 해서 도시 A1과 테이블에 있는 다른 도시들(도시 A1는 제외)간의 최소거리를 계산한다. 외적 쿼리는 서브쿼리에서 계산된 도시 A1과 도시 B2간 최소거리가 도시 A1과 도시 B2간의 거리와 동일한지 테스트한다. 동일하다면 도시 A1과 도시 B1의 좌표와 거리가 포함된 행을 불러온다.

예제 ?-?

```
proc sql outobs=10;
       select a.city format=$10., a.state,
              a.latitude 'Lat', a.longitude 'Long',
              b.city format=$10., b.state,
              b.latitude 'Lat', b.longitude 'Long',
              sqrt(((b.latitude - a.latitude)**2) +
             ((b.longitude - a.longitude)**2)) as dist format=6.1
         from sql.uscitycoords a, sql.uscitycoords b
        where a.city ne b.city
          and calculated dist =
              (select min(sqrt(((b.latitude - a.latitude)**2) +
                        ((b.longitude - a.longitude)**2)))
                 from sql.uscitycoords c, sql.uscitycoords d
                where c.city=a.city
                  and c.state=a.state
                  and d.city ne c.city)
       order  by a.city;
quit;
```

테이블 결합과 서브쿼리 이용 시 주의사항

여러 개 테이블에 있는 정보를 참조할 때 테이블 결합이나 서브쿼리를 이용한다. 테이블 결합과 서브쿼리는 종종 같은 쿼리내에서 같이 사용된다. 이렇듯 많은 경우에 데이터 검색 문제는 테이블 결합, 서브쿼리 또는 이 둘을 함께 사용하여 해결할 수 있다. 아래는 테이블 결합과 쿼리를 위한 몇 가지 지침이다.

- 만약 하나 이상의 테이블에 있는 데이터가 필요하다면 테이블 결합을 실행해야한다. FROM 구에 있는 테이블(또는 뷰)이 결합된다.
- 만약 같은 테이블에 있는 다른 행들이 관련된 정보가 필요하다면 자기결합(self-join)을 이용할 수 있다.
- 원하는 결과가 한 쿼리로 얻을 수 있는 것보다 많은 경우 서브쿼리를 이용해야 하고 각 서브쿼리는 한 쿼리내에서 관련된 테이블의 일부분만 제공한다.
- 만약 존재여부를 알고자 할 경우 서브쿼리가 많이 사용된다. 만약 쿼리가 NOT EXISTS

조건이 필요하다면 반드시 서브쿼리를 이용해야 한다. 왜냐하면 NOT EXISTS은 서브쿼리에서만 사용할 수 있기 때문이다. 똑같은 원칙이 EXISTS 조건에도 작용한다.

- 많은 쿼리들은 테이블 결합 또는 서브쿼리로 표현가능하다. 비록 PROC SQL 쿼리가 서브쿼리를 테이블 결합을 위해 좀 더 최적하게 바꿀 수 있지만, 일반적으로 테이블 결합이 처리에 더 효율적이다.

집합연산자를 이용한 쿼리문 조합

두 개 이상 쿼리를 이용한 결과

PROC SQL은 아래의 집합 연산자를 이용하여 다양한 방법으로 두 개 이상의 쿼리 결과를 조합할 수 있다.

UNION	두 쿼리에 의해 생성된 모든 행이 참조된다. 단, 중복행 중 하나는 제거된다.
EXCEPT	첫 번째 쿼리에 의해 생성된 부분만 참조된다.
INTERCEPT	두 개 쿼리에 의해 공통적으로 생성된 행만 참조된다.
OUTER UNION	쿼리의 결과들을 연관시킨다.

연산자는 두 쿼리 사이에 사용된다. 예를 들면 아래와 같다.

```
select columns from table
set-operator
select columns from table;
```

세미콜론(;)은 마지막 쿼리문 뒤에만 위치하며, 집합 연산자는 개개의 컬럼명과 관계없이 참조 테이블의 위치에 기초해 두 쿼리의 컬럼을 조합한다. 따라서, 두 쿼리의 상대적 위치가 동일한 컬럼은 반드시 같은 타입이어야 한다. 첫 번째 컬럼의 테이블에 있는 컬럼명이 최종 테이블의 컬럼명이 된다. 아래의 옵션(option)은 집합 연산자을 수월하게 해준다.

ALL

중복된 행을 없애지 않는다. ALL 명령어를 지정하면, PROC SQL은 중복된 행을 제거하기 위해 데이터를 두 번 거쳐가지 않는다. 따라서, ALL 사용하는 것은 사용하지 않을 때보다 효율적이다. OUTER UNION에서는 ALL은 필요없다.

CORRESPONDING(CORR)

두 테이블에 같은 이름의 컬럼이 있는 경우 덮어쓴다. EXCEPT, INTERSECT 그리고 UNION과 함께 사용하면 CORR 명령어는 동일한 컬럼이 두 테이블에 사용하지 못하게 한다.

집합 연산자에 대한 설명과 사용법은 아래의 두 테이블을 이용하여 설명하도록 하겠다.

예제 ?-?

Table A

x	y
1	one
2	two
2	two
3	three

Table B

x	z
1	one
2	two
4	four

테이블 결합은 테이블을 수평적으로 결합한다면, 집합 연산자는 테이블을 수직적으로 결합한다. 그러므로, 밴다이어그램을 그려보면서 쉽게 이해할 수 있을 것이다.

UNION

UNION 연산자는 두 쿼리의 결과를 조합할 때, 중복되지 않는 유일한 행만을 모두 가져온다. 집합에서 합집합과 동일하다고 생각하면 된다. 즉, 첫 번째, 두 번째 아니면 둘 다 있는 경우 하나의 행만 가져온다. UNION은 중복된 행은 가져오지 않는다. 하나의 행이 여러번 발생했다면 하나의 행만 가져오는 것이다.

예제 ?-?

```
proc sql;
       select * from sql.a
       union
       select * from sql.b;
quit;
```

ALL 명령어를 사용하면 결과에 중복 행을 유지한다.

예제 ?-?	

```
proc sql;
       select * from sql.a
       union all
       select * from sql.b;
quit;
```

EXCEPT

EXCEPT 명령어는 첫 번째 쿼리 결과에서 두 번째 쿼리 결과는 제외한 행을 가져온다. 위의 테이블에서 EXCEPT 명령어를 사용하면 3과 'three'값만 포함한다.

예제 ?-?	

```
proc sql;
       select * from sql.a
       except
       select * from sql.b;
quit;
```

테이블 A에 중복으로 나타난 2와 'two'값이 나타나지 않았다. EXCEPT는 두 번째 쿼리와 일치되는 않는 중복 행은 가져오지 않는다. ALL 명령어를 추가하면 두 번째 쿼리와 중복적으로 매칭되지 않는 행은 유지시킬 수 있다.

예제 ?-?	

```
proc sql;
       select * from sql.a
       except all
       select * from sql.b;
quit;
```

INTERSECT

INTERSECT 연산자는 첫 번째 쿼리와 두 번재 쿼리 결과 둘다에서 발생한 행만 가져온다.

예제 ?-?	

```
proc sql;
       select * from sql.a
       intersect
       select * from sql.b;
quit;
```

INTERSECT ALL 연산의 결과는 첫 번째 쿼리에 생성된 행과 두 번째 쿼리에 생성된 행을 일대일로 일치시킨 행을 포함한다. 이 예제에서는 INTERSECT과 INTERSECT ALL 결과는 동일하다.

OUTER UNION

OUTER UNION 연산자는 쿼리들의 결과를 연관시킨다.

예제 ?-?	

```
proc sql;
       select * from sql.a
       outer union
       select * from sql.b;
quit;
```

OUTER UNION은 두 테이블의 컬럼을 덮어쓰지 않는다. 같은 위치에 있는 컬럼을 덮어쓰기 위해서는 CORRESPONDING 명령어를 사용한다.

예제 ?-?	

```
proc sql;
       select * from sql.a
       outer union corr
       select * from sql.b;
quit;
```

PROC SQL문에서 첫 번째 테이블과 두 번째 테이블 모두에 있는 행은 제외하고 중복이 제

거한 행을 가져오는 명령어는 없다. 여기서는 여러 연산자를 사용해서 가능한 방법을 소개하겠다.

(query1 except query2)
union
(query2 except query1)

다음 예제는 이러한 연산자를 어떻게 사용하는지를 보여준다.

예제 ?-?

```
proc sql;
       (select * from sql.a
              except
              select * from sql.b)
       union
       (select * from sql.b
              except
              select * from sql.a);
quit;
```

첫 번째 EXCEPT는 첫 번째 테이블에서 중복을 제거한 행만 가져온다. 두 번째 EXCEPT는 두 번째 테이블에서 중복을 제거한 행만 가져온다. 중간에 UNION은 두 결과를 조합한다. 따라서 이 쿼리는 첫 번째 테이블에만 있는 행과 두 번째 테이블에만 있는 행을 가져온다.

테이블과 뷰의 생성과 업데이트

테이블 생성

CREATE TABLE 문은 컬럼을 정의하여 행없이 테이블을 만들거나 쿼리 결과로 테이블을 만들 수 있다. 또한, 기존의 테이블을 복사하는 데에도 사용할 수 있다.

컬럼 정의를 통한 테이블 생성

CREATE TABLE 문은 컬럼과 컬럼의 속성을 정의하여 행없이 새로운 테이블을 만들 수 있다. 이때, 컬럼의 이름, 속성, 길이, 입출력 형태 그리고 라벨등을 지정할 수 있다.

아래의 CREATE TABLE 문은 NEWSTATES 테이블을 생성한다.

예제 ?-?

```
proc sql;
       create table sql.newstates
              (state char(2),
               date num
                     informat=date9.
                     format=date9.,
               population num);
quit;
```

테이블 NEWSTATES는 3개의 컬럼과 0개의 행을 가진다. char(2)는 State의 길이를 변경하는 수정하는 데 사용된다.

DESCRIBE TABLE 문을 사용하면 테이블의 존재와 컬럼의 속성을 확인할 수 있다. 아래의 DESCRIBE TABLE 문은 CREATE TABLE 문에 의해 생성된 테이블에 관한 내용을 SAS log에 출력한다.

예제 ?-?

```
proc sql;
       describe table sql.newstates;
quit;
```

SAS log--

```
1   proc sql;
2       describe table sql.newstates;
NOTE: SQL table SQL.NEWSTATES was created like:

create table SQL.NEWSTATES( bufsize=4096 )
  (
   state char(12),
   date num format=DATE9. informat=DATE9.,
   population num
  );
```

DESCRIBE TABLE 문은 CREATE TABLE 문을 가지고 테이블을 생성하지 않아도 CREATE TABLE 문으로 테이블을 생성한 것처럼 SAS log창에 출력한다. 또한, PROC CONTENTS 문을 사용해도 NEWSTATES 테이블에 대한 정보를 얻을 수 있다.

PROC SQL가 쿼리의 결과로부터 테이블을 생성하기 위해서는 SELECT 문 이전 위치에 CREATE TABLE 문을 사용해야 한다. 이러한 방식으로 테이블을 생성하면, 쿼리의 FROM 구에 있는 테이블과 뷰를 참조해서 데이터를 불러온다. 새로운 테이블의 컬럼명은 쿼리의 SELECT 구에 지정한 대로 된다. 컬럼 속성(형태, 길이 입출력 형태)는 기존의 테이블의 속성을 그대로 따라간다.

아래의 CREATE TABLE 문은 COUNTRIES 테이블로부터 DENSITIES 테이블을 생성한다. 새롭게 생성된 테이블은 쿼리를 실행했지만 테이블이 출력창에 보이지 않는다.

예제 ?-?

```
proc sql outobs=10;
       create table sql.densities as
       select Name 'Country' format $15.,
              Population format=comma10.0,
              Area as SquareMiles,
              Population/Area format=6.2 as Desity
         from sql.countries;
quit;
```

NOTE) Outobs 옵션을 사용했기 때문에 DENSITIES 테이블은 10개의 행만 가지고 있음.

아래의 DESCRIBE TABLE 문은 CREATE TABLE 문을 SAS log창에 출력한다.

예제 ?-?

```
proc sql;
       describe table sql.densities;
quit;
```

```
1  proc sql;
2      describe table sql.densities;
NOTE: SQL table SQL.DENSITIES was created like:

create table SQL.DENSITIES( bufsize=8192 )
  (
   Name char(35) format=$15. informat=$35. label='Country',
   Population num format=COMMA10. informat=BEST8. label='Population',
   SquareMiles num format=BEST8. informat=BEST8.,
   Desity num format=6.2
  );
```

위의 CREATE TABLE에서는 컬럼명을 컬럼의 별명으로 사용했지만, 라벨은 사용하지 않았다. 하지만 DESCRIBE TABLE에서는 Area 컬럼은 SquareMiles로 컬럼명을 변경했고, 계산으로 생성된 컬럼은 Desities로 명명했다. 그러나 Name 컬럼은 이름을 그대로 유지했고 단지 라벨만 country로 보여진다.

기존의 테이블을 이용한 테이블 생성

기존의 테이블이나 뷰와 같은 컬럼이나 속성을 가진 비어있는 테이블을 생성하기 위해서 CREATE TABLE 문에 LIKE 구를 사용한다. 아래의 예제에서 CREATE TABLE 문은 COUNTRIES 테이블에 있는 컬럼명과 컬럼속성을 가진 6개의 컬럼과 0개의 행을 가진 NEWCOUNTRIES 테이블을 생성한다. DESCRIBE TABLE 문은 CREATE TABLE 문을 SAS log에 다음과 같이 나타내었다.

예제 ?-?

```
proc sql;
       create       table sql.newcountries
           like     sql.countries;
       describe     table sql.newcountries;
quit;
```

```
1  proc sql;
2      create      table sql.newcountries
3        like       sql.countries;
NOTE: Table SQL.NEWCOUNTRIES created, with 0 rows and 6 columns.
115      describe    table sql.newcountries;
NOTE: SQL table SQL.NEWCOUNTRIES was created like:

create table SQL.NEWCOUNTRIES( bufsize=12288 )
  (
   Name char(35) format=$35. informat=$35.,
   Capital char(35) format=$35. informat=$35. label='Capital',
   Population num format=BEST8. informat=BEST8. label='Population',
   Area num format=BEST8. informat=BEST8.,
   Continent char(30) format=$30. informat=$30. label='Continent',
   UNDate num format=YEAR4.
  );
```

기존의 테이블 복사하기

PROC SQL을 이용하여 테이블을 복사하는 가장 빠른 방법은 쿼리에 CREATE TABLE 문을 이용해 테이블 전체를 가져오는 것이다. 아래 예제는 COUNTRIES에 있는 모든 컬럼과 행을 복사해 COUNTRIES1에 포함시키는 것이다.

예제 ?-?

```
proc sql;
       create table countries1 as
       select * from sql.countries;
quit;
```

Data Set 옵션 사용하기

Data Set 옵션을 CREATE TABLE 문에서 사용할 수 있다. 아래의 CREATE TABLE 문은 COUNTRIES 테이블에서 COUNTRIES2를 생성하는 것이다. DROP= 옵션은 UNDate 컬럼을 삭제한다. 따라서, COUNTRIES2 테이블에는 UNDate 컬럼이 존재하지 않는다.

예제 ?-?

```
proc sql;
       create table countries2 as
       select * from sql.countries(drop=UNDate);
quit;
```

테이블에 행 삽입하기

INSERT 문을 이용하여 테이블에 테이터 값을 삽입할 수 있다. INSERT 문은 기존의 테이블에 새로운 행을 추가하고, 그 행에 특정한 값을 삽입하는 것이다. 특정한 값은 SET 구나 VALUES 구를 이용하여 지정할 수 있다. 또한 쿼리의 결과로 나타난 행을 삽입할 수도 있다.

SET 구를 이용한 행 삽입

SET 구를 이용하면, 컬럼명으로 새로운 데이터값을 할당한다. 컬럼은 SET 구에 순서에 관계없이 나타날 수 있다. 아래의 INSERT 구는 다중 SET 구를 이용해 NEWCOUNTRIES 테이블에 두 개의 행을 삽입하였다.

예제 ?-?

```
proc sql;
       insert into sql.newcountries
          set name='Bangladesh',
              capital='Dhaka',
              population=126391060
          set name='Japan',
              capital='Tokyo',
              population=126352003;
       select name format=$20.,
              capital format=$15.,
              population format=comma15.0
         from sql.countries;
quit;
```

SET 구의 특징은 다음과 같다.

다른 SQL 구를 사용하는 것처럼 ,(comma)로 컬럼을 구분한다. 추가적으로 마지막 SET 구 다음에 세미콜론(;)을 반드시 사용해야 한다.

만약 컬럼에 대한 데이터를 생략했다면, 해당 컬럼의 값은 결측값으로 처리된다.

컬럼의 값으로 결측값을 지정하고 싶다면 문자의 경우 ' '(blank), 숫자는 .(period) 값을 사용한다.

VALUES 구를 이용한 행 삽입

VALUES 구를 이용하면 컬럼의 위치에 해당 값을 할당한다. 아래의 INSERT 문은 NEWCOUNTRIES 테이블에 행을 추가하기 위해서 여러 개의 VALUES 구를 이용한 것이다. NEWCOUNTRIES 테이블은 6개의 컬럼을 가지고 있어 6개의 모든 컬럼에 특정한 값을 지정하거나 결측값을 부여할 필요가 있다.

예제 ?-?

```
proc sql;
       insert into sql.newcountries
              values ('Pakistan', 'Islamabad', 123060000, .,' ', .)
              values ('Nigeria', 'Lagos', 99062000, ., ' ', .);

       select name format=$20.,
              capital      format=$15.,
              population format=comma15.0
         from sql.newcountries;
quit;
```

VALUES 구의 특징은 다음과 같다.

다른 SQL 구를 사용하는 것처럼 ,(comma)로 컬럼을 구분한다. 추가적으로 마지막 VALUES 구 다음에 세미콜론(;)을 반드시 사용해야 한다.

만약 컬럼에 결측값이라는 표시하지 않고 데이터를 생략했다면, 에러 메시지가 출력되면서 해당 행은 삽입되지 않는다.

컬럼의 값으로 결측값을 지정하고 싶다면 문자의 경우 ' '(blank), 숫자는 .(period) 값을 사용한다.

쿼리 결과 삽입하기

쿼리 결과로 나온 행을 테이블에 삽입할 수 있다. 아래의 쿼리는 COUNTRIES 테이블에서 인구가 1억3천이상인 국가를 가져온다. INSERT 문을 이용해 비어있는 NEWCOUNTRIES 테이블에 해당 국가를 추가한 것이다.

예제 ?-?

```
proc sql;
       create table sql.newcountries
         like sql.countries;
quit;

proc sql;
       insert into sql.newcountries
       select * from sql.countries
       where  population ge 130000000;

       select name format=$20.,
              capital format=$15.,
              population format=comma15.0
         from sql.newcountries;
quit;
```

만약 쿼리가 모든 컬럼을 가져오지 않는다면 에러 메시지가 출력되고 행이 삽입되지 않는다.

데이터값 업데이트

UPDATE 문을 이용하여 테이블에 있는 데이터값을 수정할 수 있다. UPDATE 문은 새로운 컬럼을 만드는 것이 아니라 기존 컬럼의 데이터를 업데이트하는 것이다. 새로운 컬럼을 추가하기 위해서는 Altering Columns와 Creating New Column을 참고하기 바란다. 여기에서 사용하는 예제는 NEWCOUNTRIES 테이블이다.

동일한 형태로 컬럼의 모든 행 업데이트

아래의 UPDATE 문은 NEWCOUNTRIES 테이블의 population 컬럼을 5% 증가하도록 한 것이다.

예제 ?-?

```
proc sql;
       update sql.newcountries
          set population=population*1.05;
       select name format=$20.,
              capital format=$15.,
              population format=comma15.0
         from sql.newcountries;
quit;
```

다른 방식으로 컬럼의 행 업데이트

만약 컬럼의 일부만 업데이트하고자 한다면 UPDATE 문에 WHERE 구를 사용하면 가능하다. 또한 다른 표현식을 가진 여러개의 UPDATE 문을 사용할 수 있다. 그러나 각 UPDATE 문에는 하나의 WHERE 구만 가질 수 있다. 아래의 UPDATE 문은 NEWCOUNTRIES 테이블의 서로 다른 나라에 다른 방식으로 인구를 증가시킨 결과를 얻을 수 있다.

예제 ?-?

```
proc sql;
       update sql.newcountries
          set population=population*1.05
        where name like 'B%';

       update sql.newcountries
          set population=population*1.07
        where name in ('China','Russia');

       select name format=$20.,
              capital format=$15.,
              population format=comma15.0
         from sql.newcountries;
quit;
```

CASE WHEN 문을 사용하면 동일한 결과를 얻을 수 있다. 아래는 CASE WHEN 문을 사용한 것이다.

예제 ?-?

```

```

만약 WHEN 구가 참이라면, 따르는 THEN 구에 있는 값이 SET 구에 반환되어 표현식이 완

성된다. 위의 예제에서는 국가명이 'B'로 시작되는 경우 1.05값이 SET에 반환되어 표현식 population=population*1.05가 완성된다.

주의

ELSE 구는 반드시 작성해야 한다.

만약 ELSE 구를 생략한다면 WHEN 구에서 지정되지 않는 행은 결측값으로 업데이트된다. 이런 경우가 발생하는 이유는 CASE WHEN 표현식이 SET 구에 결측값을 반환하기 때문에 population 컬럼은 결측값과 곱하여져 결측값으로 나타난다.

업데이트 에러 다루기

테이블에 행을 업데이트하거나 삽입하는 동안 업데이트나 삽입이 실행되지 않는다는 에러 메시지를 받을 수 있다. UNDO_POLICY 옵션을 사용하면, 이미 실행된 사항을 영구적으로 할지 여부를 조정할 수 있다.

PROC SQL과 RESET 문에서 UNDO_POLICY 옵션은 PROC SQL이 에러가 나타날 때 까지 현재의 INSERT와 UPDATE 문에 의해 삽입되고 업데이트된 행을 어떻게 다룰 건지를 결정한다.

UNDO_POLICY=REQUIRED

기본으로 설정되어 있으며, 에러가 나타나면 모든 업데이트나 삽입을 원상태로 한다.

UNDO_POLICY=NONE

어떤 업데이트나 삽입도 원상태로 돌리지 않는다.

UNDO_POLICY=OPTIONAL

신뢰할 수 있게 원상태로 돌릴 수 있는 업데이트나 삽입을 원상태로 한다.

행 삭제

DELETE 문은 테이블에 있는 하나 이상의 행을 삭제한다. 아래의 DELETE 문은 'R'로 시작되는 국가명을 삭제한다.

예제 ?-?	

```
proc sql;
       delete
         from sql.newcountries
        where name like 'R%';
quit;
```

SAS log창에서는 얼마만큼의 행이 삭제되었는지 알려준다.

```
NOTE: 1 row was deleted from SQL.NEWCOUNTRIES.
```

NOTE) PROC SQL 테이블에서 SAS는 행의 데이터를 삭제하지만 테이블의 공간은 남아있다.

주의:

만약 WHRER 구 없이 DELETE 문을 사용하면 모든 행이 삭제된다.

컬럼 수정

ALTER TABLE 문은 기존 테이블에 컬럼을 추가, 수정 그리고 삭제한다. ALTER TABLE 문은 테이블에만 사용가능하며 VIEW에는 사용할 수 없다. SAS log창에 어떻게 테이블이 수정되었는지 알려준다.

컬럼 추가

ADD 구는 기존 테이블에 새로운 컬럼을 추가한다. 이 때 반드시 컬럼의 이름과 형태를 지정해줘야 한다. 또한 길이, 입출력형태, 라벨을 지정할 수 있다. 아래의 예제는 ALTER TABLE 문은 NEWCOUNTREIS 테이블에 숫자 형태의 DENSITY라는 컬럼을 추가하는 것이다.

예제 ?-?

```
proc sql;
       alter  table sql.newcountries
              add density num label='Population Density' format=6.2;

       select name format=$20.,
              capital format=$15.,
              population format=comma15.0,
              density
         from sql.newcountries;
quit;
```

NEWCOUNTRIES 테이블에 새로운 컬럼이 추가되었지만 데이터 값을 가지고 있는 것은 아니다. 아래의 UPDATE 문을 이용해 결측값을 나라별 적절한 인구밀도 값으로 바꾼다.

예제 ?-?

```
proc sql;
       update sql.newcountries
              set density=population/area;

       select name format=$20.,
              capital format=$15.,
              population format=comma15.0,
              density
         from sql.newcountries;
quit;
```

위와 같은 업데이트는 새로운 테이블 생성시 수식을 이용해 인구밀도(population density) 컬럼을 만드는 것과 동일하게 할 수 있다.

예제 ?-?

```
proc sql;
       create table sql.newcountries as
       select *,
              population/area as density
              label='Population Density'
              format=6.2
         from sql.newcountries;
quit;
```

컬럼 수정

MODIFY 구를 사용해 컬럼의 길이, 입출력형태 그리고 라벨을 변경할 수 있다. 컬럼 이름을 변경하기 위해서는 RENAME= 옵션을 사용해야 한다. 그러나 컬럼의 형식를 변경할 수는 없다.

아래의 MODIFY 구는 Population 컬럼의 출력형태를 영구적으로 변경하는 것이다.

예제 ?-?

```
proc sql;
        alter table sql.newcountries
              modify population format=comma15.
       select name, population from sql.newcountries;
quit;
```

컬럼을 업데이트 하기전에 해당 컬럼의 길이도 변경할 수 있다. 예를 들어, 이름 앞에 긴 문자열을 붙이고 싶다면 이름에 해당되는 컬럼의 문자 길이를 35에서 60으로 변경해야된다. 아래의 예제는 이름 컬럼을 수정하고 업데이트한 것이다.

예제 ?-?

```
proc sql;
        alter table sql.newcountries
              modify name char(60) format=$60.;
       update sql.newcountries
              set name='The United Nations member country is '||name;
       select name from sql.newcountries;
quit;
```

컬럼 삭제

DROP 구는 테이블에 있는 컬럼을 삭제한다. 아래는 DROP 구를 이용하여 NEWCOUNTRIES 테이블에 있는 undate 컬럼을 삭제한 것이다.

예제 ?-?

```
proc sql;
       alter  table sql.newcountries
              drop undate;
quit;
```

참조키(INDEX) 생성

참조키는 테이블과 연관된 파일로 참조키로 테이블의 행에 접근가능하다. 참조키는 데이터의 작은 일부에 빠르게 접근가능하고 테이블 결합을 효과적으로 할 수 있다. 이러한 참조키는 생성할 수 있지만 PROC SQL이 참조키를 사용하게 할 순 없다. 단지 참조키 사용하는데 효과적인지 아닌지만 결정할 뿐이다.

PROC SQL을 이용한 참조키 생성

하나의 컬럼을 적용한 간단한 참조키 생성할 수 있다. 참조키의 이름은 참조키를 만들고자 하는 컬럼의 이름과 일치해야 한다. 컬럼은 테이블 다음 괄호안에 표기한다. 아래의 CREATE INDEX 문은 NEWCOUNTRIES 테이블의 'Area' 컬럼의 참조키를 생성한다.

예제 ?-?

```
proc sql;
       create index area
            on sql.newcountries(area);
quit;
```

또한 두 개 이상의 컬럼에 적용한 합성 참조키도 생성가능하다. 아래의 CREATE INDEX 문은 NEWCOUNTRIES 테이블에 있는 'Name'과 'Continent' 컬럼을 이용한 Places 참조키를 생성하는 예제이다.

예제 ?-?

```
proc sql;
       create index places
            on sql.newcountries(name, continent);
quit;
```

참조키의 개개값(또는 합성 참조키에서 컬럼 값들의 조합)이 유일한 값을 가지게 하기 위해서 UNIQUE 키워드를 사용한다.

예제 ?-?

```
proc sql;
      create unique index places
          on sql.newcountries(name, continent);
quit;
```

UNIQUE 키워드를 사용하면 SAS는 하나 이상의 행이 같은 참조키를 가지지 못하게 한다.

참조키 생성 시 조언

합성 참조키의 이름이 테이블에 있는 특정한 컬럼명과 동일하게 할 수 없다.

만약 두 개의 컬럼을 이용하여 정기적으로 데이터에 접근해야 한다면, 합성 참조키를 사용하는 것이 좋다.

디스크 용량과 업데이트 비용을 줄이기 위해서는 참조키 수를 최소화하는 것이 좋다.

상대적으로 적은 수의 행을 검색하는 쿼리에 참조키가 사용된다.(15% 아래)

일반적으로, 적은 용량의 테이블에서 참조키를 사용할 때 얻을 수 있는 이득은 많지 않다.

일반적으로, 개별값이 적은 컬럼에 참조키를 사용하는 경우 얻을 수 있는 이득은 많지 않다.

단순 참조키나 합성 참조키에서 동일한 컬럼을 사용할 수 있으나 주요키가 완전 제한된 테이블에서는 주요키와 동일한 컬럼에 기초한 참조키가 하나 이상은 생성되지 않는다.

참조키 삭제

테이블에 있는 참조키를 삭제하기 위해서는 DROP INDEX 문을 사용한다. 아래의 DROP INDEX 문은 NEWCOUNTRIES 테이블에 참조키 'Places' 삭제하는 것이다.

예제 ?-?

```
proc sql;
      drop   index places from sql.newcountries;
quit;
```

테이블 삭제

PROC SQL 테이블을 삭제하기 위해서 DROP TABLE 문을 이용한다.

예제 ?-?

```
proc sql;
       drop   table sql.newcountries;
quit;
```

PROC SQL에서 SAS 매크로기능 사용하기

매크로기능은 SAS 소프트웨어를 폭 넓고 사용자 요구에 맞게 사용할 수 있게 해주는 프로그램 툴이다. 이 기능을 사용하면 일상적이고 반복적으로 실행해야하는 프로그램를 단축할 수 있고 SQL 프로그램의 효율성과 유용성을 개선할 수 있다.

매크로 변수는 SAS 코드에서 문자열을 대체하는 유용한 방법을 제공한다. 사용자에 의해 생성되고 명명된 매크로 변수를 사용자 정의 매크로 변수(user-defined macro variable)이라고 하고, SAS에 의해 정의된 매크로 변수를 자동 정의 매크로 변수(automatic macro variable)이라고 한다. PROC SQL에서는 프로그램의 문제를 처리하는데 도움이 되는 3가지 종류(SQLOBS, SQLRC, SQLOOPS)의 자동 정의 매크로 변수를 제공한다.

첫 번째 쿼리의 결과로 매크로 변수 생성하기

만약 INTO 구에 하나의 매크로 변수를 지정하면 PROC SQL은 SELECT에 있는 변수의 첫 번째 행의 값을 매크로 변수에 할당한다. 예를 들어, &country1은 Country 컬럼의 첫 번째 행의 값이 할당되고, &barrels1은 Barrels 컬럼의 첫 번째 행의 값이 할당된다. 여기서, NOPRINT옵션은 PROC SQL 결과가 output 창에 나타나지 않게 하는 것이고 대신에 %PUT 문을 사용하여 매크로 변수의 내용을 SAS 로그에 기록하였다.

예제 ?-?

```
proc sql noprint;
       select country, barrels
       into          :country1, :barrels1
       from          sql.oilrsrvs;
quit;

%put &country1 &barrels1;
```

```
16   proc sql noprint;
17        select  country, barrels
18        into        :country1, :barrels1
19        from        sql.oilrsrvs;
20   quit;
NOTE: 프로시저 SQL 실행:
      실행 시간           0.01 초
      cpu 시간            0.01 초

21
22   %put &country1 &barrels1;
Algeria                          9,200,000,000
```

Aggregate 함수의 결과로 매크로 변수 생성하기

매크로 변수는 SAS title에 자료 값을 보이게 할 수 있는 유용한 기능을 가지고 있다. 다음의 예는 WORLDTEMPS의 일부분으로 캐나다의 가장 높은 기온을 나타내는 것이다.

예제 ?-?

```
proc sql outobs=12;
         reset       noprint
       select        max(AvgHigh)
          into       :maxtemp
          from       sql.worldtemps
         where       country='Canada'

         reset       print
         title       "The Highest Temperature in Canada: &maxtemp"
        select       city,AvgHigh format 4.1
          from       sql.worldtemps
         where       country='Canada'
quit;

<OUTPUT>
              The Highest Temperature in Canada:       80
                                          Avg
                      City                High
                      ------------------------
                      Montreal            77.0
                      Quebec              76.0
                      Toronto             80.0
```

NOTE) TITLE 문에서는 매크로변수의 참조 문제를 해결하기 위해서는 반드시 큰 따음표를 사용해야한다.

NOTE) RESET 문은 프로시저를 다시 시작할 필요없이 PROC SQL 옵션을 추가, 삭제, 변경이 가능하다.

여러 개의 매크로 변수 생성하기

SELECT 문의 결과로부터 각 행마다 하나의 새로운 매크로 변수를 생성할 수 있다. INTO 문에 THROUGH, THRU 또는 하이픈(-) 키워드를 사용하면 매크로 변수의 범위를 생성할 수 있다. 다음 예제는 Name 컬럼 처음 4개 행의 값과 Population 컬럼 처음 3개 행의 값을 매크로 변수에 할당하는 것이다. %PUT 문을 사용하여 SAS 로그에 출력하였다.

예제 ?-?

```
proc sql noprint
       select name, population
          into:country1 - :country4, :pop1 - :pop3
          fromsql.countries;
quit

%put &country1 &pop1;
%put &country2 &pop2;
%put &country3 &pop3;
%put &country4;
```

<LOG>

```
38   proc sql noprint;
39       select  name, population
40          into :country1 - :country4, :pop1 - :pop3
41          from sql.countries;
42   quit;
NOTE: 프로시저 SQL 실행:
      실행 시간           0.06 초
      cpu 시간            0.00 초


43
44   %put &country1 &pop1;
Afghanistan 17070323
45   %put &country2 &pop2;
Albania 3407400
46   %put &country3 &pop3;
Algeria 28171132
47   %put &country4;
Andorra
```

매크로 변수의 값 연결시키기

한 컬럼의 값을 한 매크로 변수에 연결시킬 수 있다. 이 형태는 변수나 상수 리스트를 만드는 데 유용하다. SEPARATED BY 키워드는 매크로 변수에서 값의 구분자를 지정해서 사용한다.

다음 예제는 COUNTRIES 테이블의 Name 컬럼의 처음 5개의 값을 %countries 매크로 변수에 할당하는 것이다. INOBS 옵션은 PROC SQL이 COUNTRIES 테이블의 처음 4개의 행만 사

용하도록 제한하는 것이다. 컬럼와 공백이 매크로 변수에서 값을 구분짓는 구분자로 사용되었다.

예제 ?-?

```
proc sql noprint inobs=5
       select Name
         into :countries separated by ', '
         from sql.countries;
quit

%put &countries;
```

```
<LOG>
49   proc sql noprint inobs=5;
50       select  Name
51           into    :countries separated by ', '
52         from  sql.countries;
WARNING: Only 5 records were read from SQL.COUNTRIES due to INOBS= option.
53   quit;
NOTE: 프로시저 SQL 실행:
      실행 시간           0.01 초
      cpu 시간            0.01 초



54
55   %put &countries;
Afghanistan, Albania, Algeria, Andorra, Angola
```

위의 예제에서는 매크로 변수가 생성되기 전에 값의 앞과 뒤의 공백을 잘라내었다. 만약 공백을 잘라내고 싶지 않으면 INTO 구에 'NOTRIM'을 추가하면 된다. 아래는 앞의 예제에서 NOTRIM을 추가한 것이다.

예제 ?-?

```
proc sql noprint inobs=5;
       select Name
         into :countries separated by ',' NOTRIM
         from sql.countries;
quit;

%put &countries;
```

```
<LOG>
57   proc sql noprint inobs=5;
58       select  Name
59           into    :countries separated by ',' NOTRIM
60         from  sql.countries;
WARNING: Only 5 records were read from SQL.COUNTRIES due to INOBS= option.
61   quit;
NOTE: 프로시저 SQL 실행:
      실행 시간           0.00 초
      cpu 시간            0.00 초



62
63   %put &countries;
Afghanistan                         ,Albania                          ,Algeria
,Andorra                             ,Angola
```

테이블 생성을 위한 매크로 정의

매크로는 테이블 생성을 위한 인터페이스로 유용하다. 새로운 테이블을 생성하고 기존의 테이블에 새로운 행을 추가하는 데 SAS 매크로 기능을 사용하면 도움이 된다.

아래의 예제는 학술 논문을 심의하는 데 도움을 준 사람들의 목록을 테이블로 생성하는 것이다. 한 테이블에 주제당 세 사람이상을 둘 순 없다. 매크로는 테이블에 새로운 심사자의 이름을 추가하기 전에 심사자의 수를 점검한다. 매크로는 두 개의 파라미터를 가진다. 하나는 심사자의 이름이고 나머지는 학술 논문의 주제이다.

예제 ?-?

```
proc sql;
       create table sql.referee
              (Name         char(15),
               Subject      char(15));
quit;

/* define the macro */
%macro addref(name,subject);
%local count;

proc sql;
       reset  noprint;
       select count(*)
          into      :count
       from   sql.referee
       where subject="&subject";

%if &count ge 3 %then %do;
proc sql;
       reset  print;
       title "ERROR: &name not inserted for subject - &subject..";
       title2 "            There are 3 referee already.";
       select * from sql.referee where subject="&subject";
       reset noprint;
       %end

%else %do;
       insert into sql.referee(name,subject)
values("&name","&subject");
       %put NOTE: &name has been added for subject - &subject..;
       %end;
quit;
%mend;
```

두 개의 파라미터를 가진 %ADDREF() 매크로를 실행하면 테이블에 심사자의 이름이 추가된다. 매크로를 실행할 때마다 SAS 로그에 메시지가 출력된다.

예제 ?-?

```
%addref(Conner,sailing);
%addref(Fay,sailing);
%addref(Einstein,relatively);
%addref(smythe,sailing);
%addref(Naish,sailing);

ERROR: Naish not inserted for subject - sailing.                    10
                        There are 3 referee already.
                             Name            Subject
                             ---------------------------------
                             Conner          sailing
                             Conner          sailing
                             Fay             sailing
                             smythe          sailing
                             Naish           sailing
```

Ouput은 %ADDREF() 매크로가 실행될 때마다 행이 나타난다. 테이블이 같은 주제에 3명의 이름을 가지면 SAS output은 더 이상의 심사자를 포함할 수 없다는 메시지가 나타난다.

PROC SQL 자동 매크로 변수 이용하기

PROC SQL은 한번 실행한 후 자동 매크로 변수에 값을 할당한다. 이러한 매크로 변수를 이용하면 SQL 프로그램을 테스트하고 계속 진행할지 여부를 결정할 수 있다.

SQLOBS

SQL procedure문에 의해 실행된 행의 개수를 가진다. 예를 들면, SELECT 문에 의해 SAS output에 나타나거나 형성된 행의 개수나 DELETE 문에 의해 삭제된 행의 개수를 가진다.

SQLOOPS

PROC SQL 과정의 내부 루프의 반복수를 가진다. 반복수는 쿼리의 복잡할수록 증가한다.

SQLRC

PROC SQL 문의 성공을 가리키는 상태 값을 가진다.

아래의 예제는 COUNTRIES 테이블로부터 데이터를 검색하는 것으로 PROC SQL에 NOPRINT 옵션을 사용했기 때문에 출력창에 나타나지는 않는다. %PUT 매크로 언어는 SAS 로그에 세

가지 자동 매크로 변수 값을 보이게 한다.

예제 ?-?

```
proc sql noprint;
       select * from sql.countries;
quit;

%put SQLOBS=*&sqlobs* SQLOOPS=*&sqloops* SQLRC=*&sqlrc*;

<LOG>
304  %put SQLOBS=*&sqlobs* SQLOOPS=*&sqloops* SQLRC=*&sqlrc*;
SQLOBS=*1* SQLOOPS=*11* SQLRC=*0*
```

NOTE) NOPRINT 옵션을 사용하여 테이블이나 매크로 변수를 생성하지 않았다. 단지 하나의 행만실행되었기 때문에 SQLOBS 값은 1을 받는다.