**16. SQL编程**

SQL(Structured Query Language)是一种广泛使用的数据库操作的结构化查询语言，使用SAS中的PROC SQL过程处理数据，比用data步更方便、高效、节省代码。

**（一）基本查询功能**

**一、基本语法**

PROC SQL;

<CREATE TABLE 数据集 as>

SELECT <DISTINCT> 列变量(或其表达式 as 变量名), ...,列变量

FROM 原始数据集

<WHERE 条件表达式>

<GROUP BY 分组变量, ... , 分组变量>

<HAVING 过滤条件>

<ORDER BY 排序变量<DESC>, ..., 排序变量>;

QUIT;

说明：

（1）CREATE TABLE语句

创建新数据集，缺省则自动在结果窗口输出；

（2）SELECT语句

指定所选择的列，或由已知列生成新列；可以用format对列重命名或定义新格式；可以用label=’标签名’对列定义标签；SELECT \*表示选择所有列；DISTINCT可选项用来删掉重复观测（行）；

（3）FROM语句——指定原始数据集；

（4）WHERE语句——指定筛选条件；

（5）GROUP BY语句——将数据集通过分组变量进行分类；

（6）HAVING 语句——相当于WHERE语句，用过滤条件筛选GROUP的分类，得到数据子集；

（7）ORDER BY语句

按排序变量进行排序，默认是升序，可选项DESC指定按降序。

（8）VALIDATE关键词，可以在不运行查询的情况下检查语句的语法错误。

**例1** 代码：

libname sql 'C:\SQLDatas';

**proc** **sql**;

title 'World Population Densities per Square Mile';

select Name, Population format=comma12., Area format=comma8.,

Population/Area as Density format=comma10.

from sql.countries

where Continent = 'Asia' and calculated Density > **1000**

order by Density desc;

**quit**;

运行结果（部分）：

注意：where语句中使用表达式得到的新变量Density，前面加了关键词calculated, 否则会报错。

|  |
| --- |
| World Population Densities per Square Mile |

| **Name** | **Population** | **Area** | **Density** |
| --- | --- | --- | --- |
| Hong Kong | 5,857,414 | 400 | 14,644 |
| Singapore | 2,887,301 | 200 | 14,437 |
| Maldives | 254,495 | 100 | 2,545 |
| Bangladesh | 126,387,850 | 57,300 | 2,206 |
| Bahrain | 591,800 | 300 | 1,973 |
| Taiwan | 21,509,839 | 14,000 | 1,536 |
| Korea, South | 45,529,277 | 38,300 | 1,189 |

注1：常用的关系运算符见**“系列7”**，还有其它运算符：

|  |  |  |
| --- | --- | --- |
| 运算符 | 涵义 | 示例 |
| ANY | 其中的至少一个值 | where Population > any (1000 2000 5000) |
| ALL | 其中的任何一个值 | where Population > all (1000 2000 5000) |
| BETWEEN-AND | 介于两个值之间 | where Population between  1000000 and 5000000 |
| CONTAINS | 判断是否包含某字符串 | where Continent contains 'America'; |
| EXISTS | 判断是否存在 | where exists (select \*from Table2); |
| IN | 判断是否属于 | where Name in ('Africa',  'Asia'); |
| IS NULL 或  IS MISSING | 判断是否缺失值 | where Population is missing; |
| LIKE | 判断是否能匹配某些字符 | where Continent like 'A%';  ‘%’可以通配任意个任意字符，  ‘\_’只能通配一个任意字符 |
| =\* | 类似匹配 | where Name =\* 'Tiland'; |

注2：排序输出时，缺失值（最小）造成错误的排序，为避免缺失值参与排序，可以使用：where Depth < 500 and Depth is not missing

**二、描述统计数据**

|  |  |
| --- | --- |
| 聚合函数 | 描述 |
| AVG, MEAN | 均值 |
| COUNT, FREQ, N | 对非缺失值计数 |
| CSS | 校正平方和 |
| CV | 变异系数（百分数） |
| MAX | 最大值 |
| MIN | 最小值 |
| NMISS | 缺失值个数 |
| PRT | T检验的p值 |
| RANGE | 值的范围 |
| STD | 标准差 |
| STDERR | 均值的标准误差 |
| SUM | 求和 |
| SUMWGT | 加权求和 |
| T | 检验“H0：均值=0”的T值 |
| USS | 未校正平方和 |
| VAR | 方差 |

**例2** 代码：

libname sql 'C:\SQLDatas';

**proc** **sql** outobs=**12**;

title 'Percentage of World Population in Countries';

select Name, Population format=comma14.,

(Population / sum(Population) \* **100**) as Percentage

format=comma8.2

from sql.countries

order by Percentage desc;

**quit**;

运行结果（部分）：

|  |
| --- |
| Percentage of World Population in Countries |

| **Name** | **Population** | **Percentage** |
| --- | --- | --- |
| China | 1,202,215,077 | 21.10 |
| India | 929,009,120 | 16.30 |
| United States | 263,294,808 | 4.62 |
| Indonesia | 202,393,859 | 3.55 |
| Brazil | 160,310,357 | 2.81 |
| Russia | 151,089,979 | 2.65 |
| Bangladesh | 126,387,850 | 2.22 |

注：“outobs=12”，只是测试代码需要，规定只输出12条观测。

**例3** 代码：

libname sql 'C:\SQLDatas';

**proc** **sql**;

title 'Number of Continents in the COUNTRIES Table';

select count(distinct Continent) as Count

/\*使用关键词distinc, 只对不同的continent计数;\*/

from sql.countries;

**proc** **sql**;

title 'Countries for Which a Continent is Listed';

select count(Continent) as Count

/\*只对非缺省的continent计数;\*/

from sql.countries;

**proc** **sql**;

title 'Number of Countries in the SQL.COUNTRIES Table';

select count(\*) as Number

/\*对所有行计数（行数）;\*/

from sql.countries;

**quit**;

运行结果：

注：SLECT语句没有选择其它列，只输出一个单独行。

|  |
| --- |
| Number of Continents in the COUNTRIES Table |

| **Count** |
| --- |
| 8 |

|  |
| --- |
| Countries for Which a Continent is Listed |

| **Count** |
| --- |
| 205 |

|  |
| --- |
| Number of Countries in the SQL.COUNTRIES Table |

| **Number** |
| --- |
| 208 |

**例4** 缺失值处理。

代码：

libname sql 'C:\SQLDatas';

**proc** **sql**;

title 'Average Length of Angel Falls, Amazon and Nile Rivers';

select Name, Length, avg(Length) as AvgLength

from sql.features

where Name in ('Angel Falls', 'Amazon', 'Nile');

**proc** **sql**;

title 'Average Length of Angel Falls, Amazon and Nile Rivers';

select Name, Length, coalesce(Length, **0**) as NewLength,

avg(calculated NewLength) as AvgLength

from sql.features

where Name in ('Angel Falls', 'Amazon', 'Nile');

**quit**;

运行结果：

注：用coalesce语句处理缺失值。

|  |
| --- |
| Average Length of Angel Falls, Amazon and Nile Rivers |

| **Name** | **Length** | **AvgLength** |
| --- | --- | --- |
| Amazon | 4000 | 4072.5 |
| Angel Falls | . | 4072.5 |
| Nile | 4145 | 4072.5 |

|  |
| --- |
| Average Length of Angel Falls, Amazon and Nile Rivers |

| **Name** | **Length** | **NewLength** | **AvgLength** |
| --- | --- | --- | --- |
| Amazon | 4000 | 4000 | 2715 |
| Angel Falls | . | 0 | 2715 |
| Nile | 4145 | 4145 | 2715 |

**三、分组数据**

1. 用GROUP BY语句对一列或多列的不同取值进行分组，从而按组用聚合函数分别描述统计数据**。**

**例5** 按洲统计人口数。代码：

libname sql 'C:\SQLDatas';

**proc** **sql**;

title 'Total Populations of World Continents';

select Continent, sum(Population) format=comma14. as TotalPopulation

from sql.countries

where Continent is not missing

group by Continent

order by Continent;

**quit**;

运行结果（部分）：

|  |
| --- |
| Total Populations of World Continents |

| **Continent** | **TotalPopulation** |
| --- | --- |
| Africa | 710,529,592 |
| Asia | 3,381,858,879 |
| Australia | 18,255,944 |
| Central America and Caribbean | 66,815,930 |
| Europe | 813,481,724 |
| North America | 384,801,818 |
| Oceania | 5,342,368 |
| South America | 317,568,801 |

注：（1）若没有“聚合函数”，GROUP BY语句的效果同ORDER BY语句（排序）；

（2）语句where Continent is not missing，筛选非缺失值的Continent进行分组，否则可能得到不期望的结果。

2. HAVING语句

HAVING语句跟在GROUP BY语句后面，用来筛选满足某条件的行，相当于WHERE语句。

二者区别：

WHERE语句必须用在GROUP BY前面，用来筛选分组前的各个“行”，不能使用聚合函数；HAVING语句用在GROUP BY后面，用来筛选分组后的“组”（整体属性），可以使用聚合函数；

若不使用GROUP BY只用HAVING语句则默认表整体为一组。

**例6** 统计国家数大于15的各洲的总人数。代码：

libname sql 'C:\SQLDatas';

**proc** **sql**;

title 'Total Populations of Continents with More than 15 Countries';

select Continent,

sum(Population) as TotalPopulation format=comma16.,

count(\*) as Count

from sql.countries

group by Continent

having count(\*) gt **15**

order by Continent;

**quit**;

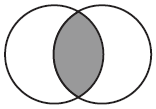
运行结果：

|  |
| --- |
| Total Populations of Continents with More than 15 Countries |

| **Continent** | **TotalPopulation** | **Count** |
| --- | --- | --- |
| Africa | 710,529,592 | 53 |
| Asia | 3,381,858,879 | 48 |
| Central America and Caribbean | 66,815,930 | 25 |
| Europe | 813,481,724 | 51 |

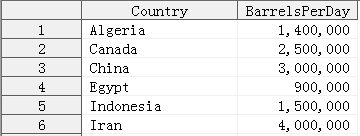
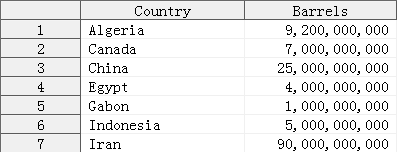
**（二）数据合并（拼接表）**

**一、内连接**

仅返回两个表匹配的数据。

语法：… where … 或 … inner join … on …

**例7** 将两个表按Country匹配合并成新表，并按BarrelsPerDay递降排列：（只截取部分表）

**代码**：

libname sql 'C:\SQLDatas';

**proc** **sql**;

create table sql.temp as

select \* from sql.oilprod as p, sql.oilrsrvs as r

where p.country = r.country and p.country is not missing

/\* 或者用下列语句:

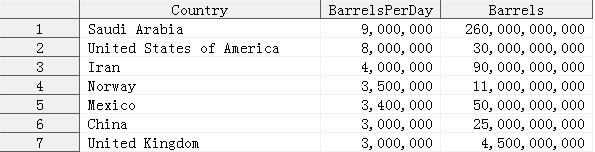
select \* from sql.oilprod as p inner join sql.oilrsrvs as r

on p.country = r.country and p.country is not missing \*/

order by barrelsperday desc;

**quit**;

运行结果（部分）：



注：（1）“select \*”，表示选择所有列，也可以指定选择的列变量；

（2）“as p”对表取个新的短名，简化代码；

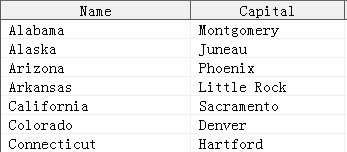
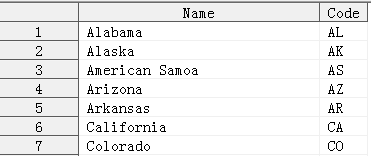
（3）“and p.country is not missing”，是为了避免如果Country是缺失值，可能出现不期望的拼接结果；

（4）一个表也可以和自身做拼接：from sql.oilprod as p, sql. oilprod as r即可，这里的as可以省略。

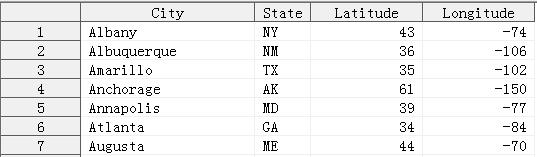
**例8（多个表拼接）**新表想要的列：Capital(首府), State(州全名), Code(周代码), Latitude(纬度), Longitude(经度).

注：c.city中不止包含capital

us pc

c



**代码：**

libname sql 'C:\SQLDatas';

title 'Coordinates of State Capitals';

**proc** **sql** outobs=**6**;

select us.Capital format=$15., us.Name 'State' format=$15., pc.Code, c.Latitude, c.Longitude

from sql.unitedstates us, sql.postalcodes pc, sql.uscitycoords c

where us.Capital = c.City and us.Name = pc.Name and pc.Code = c.State;

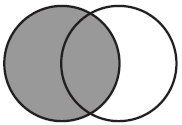
**quit**;

运行结果：

|  |
| --- |
| Coordinates of State Capitals |

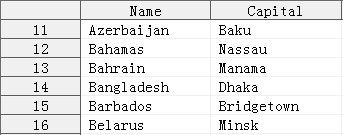
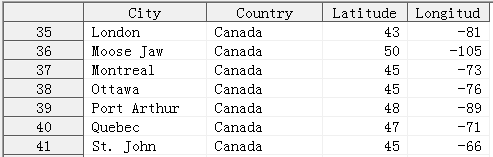
| **Capital** | **State** | **Code** | **Latitude** | **Longitude** |
| --- | --- | --- | --- | --- |
| Montgomery | Alabama | AL | 32 | -86 |
| Juneau | Alaska | AK | 58 | -134 |
| Phoenix | Arizona | AZ | 33 | -113 |
| Little Rock | Arkansas | AR | 35 | -92 |
| Sacramento | California | CA | 38 | -121 |
| Denver | Colorado | CO | 40 | -105 |

**二、外连接**

左外连接：返回左表的数据，以及匹配的数据。

语法：… left join … on …

**例9** 左表数据更多，将城市的经纬度（若有的话）合并进来。

代码：

libname sql 'C:\SQLDatas';

**proc** **sql**;

title 'Coordinates of Capital Cities';

select Capital format=$20., Name 'Country' format=$20.,Latitude, Longitude

from sql.countries a left join sql.worldcitycoords b

on a.Capital = b.City and a.Name = b.Country

order by Capital;

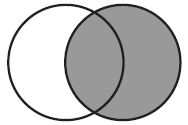
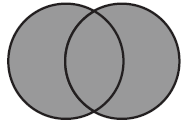
**quit**;

运行结果（部分）：

|  |
| --- |
| Coordinates of Capital Cities |

| **Capital** | **Country** | **Latitude** | **Longitude** |
| --- | --- | --- | --- |
|  | Channel Islands | . | . |
| Abu Dhabi | United Arab Emirates | . | . |
| Abuja | Nigeria | . | . |
| Accra | Ghana | 5 | 0 |
| Addis Ababa | Ethiopia | 9 | 39 |

类似的，还有右连接（… right join … on…）和全连接（… full join … on …）：

**三、笛卡尔积连接（… cross join …）**

左表的行和右表的行，所有笛卡尔积组合。

**四、其它连接**

1. …union join … ——只拼接，不匹配。

2. … natural join … ——自动匹配拼接，节省代码。

**PROC SQL和DATA STEP拼接表的区别：**

（1）PROC SQL不用事先按变量排序；

（2）BY排序的值都匹配，且没有重复值；DATA STEP拼接和PROC SQL内连接相同；

（3）BY排序的值只有部分匹配，PROC SQL（全）外连接可以实现相同的DATA STEP拼接；

（4）PROC SQL实际操作上是创建所有的笛卡尔连接，再通过WHERE条件做筛选（对位置敏感的拼接，适合用DATA STEP而不是PROC SQL）。

**（三）子查询**