Chapter 2 PERFORMING ADVANCED QUERIES

USING PROC SQL

* To display all columns in the order in which they are stored in a table, use an asterisk (\*) in the SELECT statement.

Example. In the *class roster* example, the following SELECT step displays all columns and rows in a table.

data roster;

input name $9. @11 id $ final total grade $;

cards;

Hano 9336 92 98 A

Kelbert 9564 96 95 A

Le 2143 98 93 A

Allen 5686 83 92 A

Chen 1414 67 90 A

Chua 7573 82 89 B

Disbrow 7497 58 88 B

Seo 5666 65 88 B

Hogan 7730 64 87 B

Lacap 6734 83 84 B

Jin 8307 71 84 B

Ciralli 7002 53 78 C

Karni 8793 65 78 C

Martinez 7057 28 77 C

Davis 5392 58 77 C

Franjic 8455 0 53 I

Shimazaki 7439 40 63 D

Uy 4186 0 20 F

;

proc sql;

select \*

from roster;

quit;

The output is

name id final total grade

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

Hano 9336 92 98 A

Kelbert 9564 96 95 A

Le 143 98 93 A

Allen 5686 83 92 A

Chen 1414 67 90 A

Chua 7573 82 89 B

Disbrow 7497 58 88 B

Seo 666 65 88 B

Hogan 7730 64 87 B

Lacap 6734 83 84 B

Jin 8307 71 84 B

Ciralli 7002 53 78 C

Karni 8793 65 78 C

Martinez 7057 28 77 C

Davis 5392 58 77 C

Franjic 8455 0 53 I

Shimazak 7439 40 63 D

Uy 4186 0 20 F

* When using asterisk (\*) in SELECT step, it is advisable to specify the FEEDBACK option in the PROC SQL statement, which writes the expanded list of columns in the SAS log window.

Example. In the above example, type

proc sql feedback;

select \*

from roster;

quit;

The output of the log window contains

proc sql feedback;

select \*

from roster;

NOTE: Statement transforms to:

select ROSTER.name, ROSTER.id, ROSTER.final, ROSTER.total, ROSTER.grade

from WORK.ROSTER;

quit;

* To indicate the maximum number of rows to be displayed, use the OUTOBS= option in the PROC SQL statement.

Example. In the above example, to output only the first 5 rows, type

proc sql outobs=5;

select \*

from roster;

quit;

The output is

name id final total grade

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

Hano 9336 92 98 A

Kelbert 9564 96 95 A

Le 143 98 93 A

Allen 5686 83 92 A

Chen 1414 67 90 A

* To eliminate duplicate rows from the query, use the keyword DISTINCT in the SELECT statement. The DISTINCT keyword applies to all columns that are listed in the SELECT statement.

Example. The data set below contains bank account ID, dates of transactions and type of transaction (deposit or withdrawal).

* The DISTINCT keyword is used to display only the distinct bank account IDs.

data bank;

input @1 accountID $ @6 trans\_date MMDDYY10. @17 trans\_name $10.;

cards;

5367 01/03/2011 withdrawal

5367 01/03/2011 deposit

5367 01/13/2011 deposit

9445 01/01/2011 withdrawal

9445 01/04/2011 withdrawal

9445 01/07/2011 deposit

7427 01/02/2011 deposit

7427 01/03/2011 deposit

7427 01/03/2011 deposit

7427 01/05/2011 withdrawal

;

proc sql;

select distinct accountID

from bank;

quit;

The output is

accountID

ƒƒƒƒƒƒƒƒƒ

5367

7427

9445

* Note that the output is given in increasing order.
* In the following code, the DISTINCT keyword is used to display distinct transaction dates for distinct bank accounts.

proc sql;

select distinct accountID, trans\_date format=date9.

from bank;

quit;

The output is

accountID trans\_date

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

5367 03JAN2011

5367 13JAN2011

7427 02JAN2011

7427 03JAN2011

7427 05JAN2011

9445 01JAN2011

9445 04JAN2011

9445 07JAN2011

* To subset the data that are displayed in the output, the WHERE statement may be used. There are several operators that are allowed in the WHERE statement.
* The logic operators AND and OR

proc sql;

select accountID, trans\_name

from bank

where accountID<'9000' and trans\_name='withdrawal';

quit;

The output is

accountID trans\_name

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

5367 withdrawal

7427 withdrawal

* The conditional operators

|  |  |  |
| --- | --- | --- |
| Operator | Tests for | Example |
| BETWEEN -AND | values that occur within the specified range | where salary between 70000 and 80000 |
| CONTAINS or ? | values that contain a specified string | where name contains ‘ER’  where name ? ‘ER’ |
| IN | values that match one on a list of values | where code in(‘PT’, ‘NA’, ‘FA’) |
| IS MISSING or IS NULL | missing values | where dateofbirth is missing  where dateofbirth is null |
| LIKE (%,\_\_ ) | values that match a specified pattern (‘\_’ replaces any single character, ‘%’ replaces any sequence of zero or more characters) | where address like ‘%P\_PLACE’ |
| =\* | values that sound like a specified value | where lastname=\*’Smith’ |
| ANY | values that meet a specified condition with respect to any one of the values returned by a subquery | where dateofbirth<any (select dateofbirth from payroll where jobcode=’FA3’) |
| ALL | values that meet a specified condition with respect to all the values returned by a subquery | where dateofbirth<all (select dateofbirth from payroll where jobcode=’FA3’) |
| EXISTS | the existence of values returned by a subquery | where exists (select \* from bank where accountID<'9000') |

* To change column format and label, use FORMAT and LABEL options after column name in the select statement.

proc sql;

select distinct accountID, trans\_date format date9. label 'Date of Transaction', trans\_name label 'Transaction type'

from bank;

quit;

Date of Transaction

accountID Transaction type

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

5367 03JAN2011 deposit

5367 03JAN2011 withdrawal

5367 13JAN2011 deposit

7427 02JAN2011 deposit

7427 03JAN2011 deposit

7427 05JAN2011 withdrawal

9445 01JAN2011 withdrawal

9445 04JAN2011 withdrawal

9445 07JAN2011 deposit

* To specify titles and footnotes with PROC SQL query, place the TITLE and FOOTNOTE statements either before PROC SQL statement or between the PROC SQL statement and the SELECT statement.

title 'Distinct account IDs';

footnote 'Bank dataset';

proc sql;

select distinct accountID

from bank;

quit;

OR

proc sql;

title 'Distinct account IDs';

footnote 'Bank dataset';

select distinct accountID

from bank;

quit;

The output is

Distinct account IDs

accountID

ƒƒƒƒƒƒƒƒƒ

5367

7427

9445

Bank dataset

* To define a column that contains a character constant, include a text string in single quotation marks in the SELECT statement.

proc sql;

select distinct accountID, 'date of transaction', trans\_date format date9.

from bank;

quit;

The output is

accountID trans\_date

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

5367 date of transaction 03JAN2011

5367 date of transaction 13JAN2011

7427 date of transaction 02JAN2011

7427 date of transaction 03JAN2011

7427 date of transaction 05JAN2011

9445 date of transaction 01JAN2011

9445 date of transaction 04JAN2011

9545 date of transaction 07JAN2011

* To compute a summary function (e.g., mean or sum) for several columns, define the calculated variable in the SELECT statement. List columns using commas.

To calculate mean, use function mean().

data homeworks;

input id $ hw1 hw2;

cards;

2157 93 85

8166 86 80

4499 100 90

2852 100 60

8820 65 60

6454 93 70

931 65 25

3158 70 0

2839 93 80

;

proc sql;

select id, mean(hw1, hw2) as hw\_avg

from homeworks;

quit;

The output is

id hw\_avg

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2157 89

8166 83

4499 95

2852 80

8820 62.5

6454 81.5

931 45

3158 35

2839 86.5

* To compute a summary function (e.g., mean or sum) for all observations in a single column, define the appropriate calculated variable in the SELECT statement. To calculate mean, use either function mean() or avg().

proc sql;

select id, avg(hw1) as hw1\_avg, mean(hw2) as hw2\_avg

from homeworks;

quit;

The output is

id hw1\_avg hw2\_avg

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2157 85 61.11111

8166 85 61.11111

4499 85 61.11111

2852 85 61.11111

8820 85 61.11111

6454 85 61.11111

931 85 61.11111

3158 85 61.11111

2839 85 61.11111

* If a variable is calculated in the SELECT statement, it cannot be used in the WHERE statement unless the keyword CALCULATED is specified in front of it in the WHERE statement.

proc sql;

select id, mean(hw1, hw2) as hw\_avg

from homeworks

where calculated hw\_avg >70;

quit;

The output is

id hw\_avg

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2157 89

8166 83

4499 95

2852 80

6454 81.5

2839 86.5

* To select a subset of groups for the query output, use a HAVING clause following a GROUP BY clause.

proc sql;

select name, grade, avg(final) as avg\_final

from roster

group by grade

having avg\_final>70;

quit;

The output is

name grade avg\_final

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

Kelbert A 87.2

Chen A 87.2

Allen A 87.2

Le A 87.2

Hano A 87.2

Disbrow B 70.5

Chua B 70.5

Jin B 70.5

Lacap B 70.5

Hogan B 70.5

Seo B 70.5

* PROC SQL required **remerging** if the values returned by a summary function are used in a calculation. During remerging, PROC SQL makes two passes through the table.

proc sql;

select name, grade, (total/sum(total)\*100) as percent\_of\_total

from roster

where grade< 'C';

quit;

The output is

percent\_

name grade of\_total

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

Hano A 6.834031

Kelbert A 6.624826

Le A 6.485356

Allen A 6.415621

Chen A 6.276151

Chua B 6.206416

Disbrow B 6.136681

Seo B 6.136681

Hogan B 6.066946

Lacap B 5.857741

Jin B 5.857741

* A subquery may be used in PROC SQL. A subquery is a query nested in another query. A subquery is also called a nested or inner query. There are two types of subqueries:
* Noncorrelated (a self-contained subquery that executes independently of the outer query);

proc sql;

select name, grade, avg(final) as avg\_final

from roster

group by grade

having avg(final)> (select avg(final) from roster);

quit;

The output is

name grade avg\_final

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

Kelbert A 87.2

Chen A 87.2

Allen A 87.2

Le A 87.2

Hano A 87.2

Disbrow B 70.5

Chua B 70.5

Jin B 70.5

Lacap B 70.5

Hogan B 70.5

Seo B 70.5

* Correlated (a dependent query that requires one or more variables to be passed to it by the outer query before the subquery can return a value to the outer query);

data homeworks;

input id $ hw1 hw2;

cards;

2157 93 85

8166 86 80

4499 100 90

2852 100 60

8820 65 60

6454 93 70

931 65 25

3158 70 0

2839 93 80

;

data homework3;

input id $ hw3;

cards;

2157 78

8166 91

4499 84

2852 88

6454 62

2839 73

;

proc sql;

select homeworks.id, hw1, hw2, hw3

from homeworks, homework3

where hw3> (select avg(hw3) from homework3 where homeworks.id=homework3.id);

quit;

* Note that the outer query must pass the column homeworks.id to the nested query before it executes.
* Operators EXIST and NOT EXISTS may be used in the WHERE clause or in the HAVING clause of the outer query to test for existence or non-existence of a set of values returned b by an inner query.

proc sql;

select id, hw1, hw2

from homeworks

where not exists (select \* from homework3 where homeworks.id = homework3.id);

quit;

The output is

id hw1 hw2

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

8820 65 60

931 65 25

3158 70 0

* To verify PROC SQL syntax it is recommended to check the query without actually executing it. To this end, do either of the two:
* Use the NOEXEC option in the PROC SQL statement

proc sql noexec;

select id, hw1, hw2

from homeworks

where not exists (select \* from homework3 where homeworks.id = homework3.id);

quit;

In the log window

NOTE: Statement not executed due to NOEXEC option.

* Use the VALIDATE keyword before the SELECT statement

proc sql;

validate

select id, hw1, hw2

from homeworks

where not exists (select \* from homework3 where homeworks.id = homework3.id);

quit;

In the log window

NOTE: PROC SQL statement has valid syntax.

* The difference between the two cases is that the NOEXEC option applies to all lines in the PROC SQL statement, whereas the VALIDATE keyword affects only the SELECT statement that follows it. If more than one SELECT statement or other statements are used, the VALIDATE keyword must be specified before each new statement.