Chapter 5 CREATING AND MANAGING TABLES USING

PROC SQL

There are three methods to create a new table with PROC SQL:

* Creating a table from a query result
* Creating an empty table that has the same columns as another table
* Creating an empty table by defining columns

CREATING A TABLE FROM A QUERY RESULT

A new table may be created and populated with data from the result of a PROC SQL query. The syntax is

proc sql;

**create table** *table\_name* **as**

select *column1<*, *column2*, …>

from *table1<*, *table2*, …>

<optional query clauses>

quit;

Example. The following query computes the inner join of tables one and two, creates a table named inner\_join in the Work library, and prints the contents of this table.

data one;

input id varA $ @@;

datalines;

1 a 2 b 4 d

;

data two;

input id varB $ @@;

datalines;

2 x 3 y 5 v

;

options nodate nonumber;

title;

proc sql;

create table inner\_join as

select \*

from one, two

where one.id=two.id;

select \*

from inner\_join;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 b x

CREATING AN EMPTY TABLE THAT IS LIKE ANOTHER TABLE

* To create a new table that has the same columns as an existing table but has no rows, use a CREATE TABLE statement with a LIKE clause. The syntax is

proc sql;

**create table** *table\_name*

**like** *table1*;

quit;

Example. In the previous example, a new table named new is created that has the same columns as inner\_join table but has no data.

proc sql;

create table new

like inner\_join;

quit;

* Note that the table new cannot be printed as it doesn’t contain any data, but its contents can be viewed by clicking on the file in the Work library.
* To create a new table that contains only a specified subset of columns from an existing table, use DROP or KEEP options. The syntax is

proc sql;

create table *table\_name*

**(drop = *column1* <*column2*, …>)**

**/\* or (keep = *column1* <*column2*, …>) \*/**

like *table1*;

quit;

Example. In the previous example, a new table named new is created that has only columns varA and varB and has no data.

proc sql;

create table new

(drop=id) /\* alternatively, (keep=varA varB) \*/

like inner\_join;

quit;

* Note that the table is created with empty rows and can be viewed only by clicking on the file in the Work library.

CREATING AN EMPTY TABLE BY DEFINING COLUMNS

To create a new table that is unlike any existing table, use a CREATE TABLE statement that includes *column specifications* for the columns being created. The syntax is

proc sql;

create table *table\_name*

**(*column1\_specifications*<,**

***column2\_specifications*,**

**…>);**

quit;

where *column\_specifications* include column name, data type (for example, char or num), column format and column label.

Example. The following PROC SQL statement creates a new table named exercise that consists of columns ID, Age, ActLevel, and Gender.

proc sql;

create table exercise

(ID char format=$4.,

Age num format=2.0,

ActLevel char format=$4. label='Activity Level',

Gender char format=$1.);

quit;

INSERTING ROWS OF DATA INTO A TABLE

After an empty table is created, rows of data should be inserted. There are three methods to insert rows with PROC SQL:

* Inserting values by column name using the SET clause
* Inserting list of values using the VALUE clause
* Inserting rows that are copied from another table using a query

INSERTING ROWS BY USING THE SET CLAUSE

Use the SET clause to populate a table with data that are not currently contained in any table. The syntax is

proc sql;

**insert into** *table\_name*

**set** *column1*=*value1*<, *column2*=*value2*,…>

<**set** …>

select \*

from *table\_name*;

quit;

Example. The code below adds two new rows to the inner\_join table.

proc sql;

insert into inner\_join

set id=6, varA='e', varB='w'

set id=7, varA='f', varB='z';

select\*

from inner\_join;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 b x

6 e w

7 f z

INSERTING ROWS BY USING THE VALUES CLAUSE

Use the VALUES clause to insert a list of values into a table. The syntax is

proc sql;

**insert into** *table\_name*

**values** *(value1*<, *value2*,…>)

<**values** …>

select \*

from *table\_name*;

quit;

Example. The code below adds two new rows to the original inner\_join table.

proc sql;

insert into inner\_join

values (6,'e','w')

values (7,'f','z');

select\*

from inner\_join;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 b x

6 e w

7 f z

* To insert a value for only some of the columns in the table, include a list of column names in the INSERT statement. The syntax is

proc sql;

**insert into** *table\_name (column1, column3<, …>)*

**values** *(value1*, *value3<*,…>)

<**values** …>

select \*

from *table\_name*;

quit;

Example. The code below adds two new rows to the original inner\_join table with nonmissing values for id and varA.

proc sql;

insert into inner\_join (id, varA)

values (6,'e')

values (7,'f');

select\*

from inner\_join;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 b x

6 e

7 f

INSERTING ROWS FROM A QUERY RESULT

The fastest way to insert rows of data into a table is to use a query to select existing rows from one or more tables and to insert the rows into another table. The syntax is

proc sql;

**insert into** *table\_name*

**select** *column1<, column2,…>*

**from** *table1* < table2 …>

<optional clauses>;

select \*

from *table\_name*;

quit;

Example. The code below inserts three rows from the inner\_join data set into a data set new.

proc sql;

create table new

like inner\_join;

quit;

proc sql;

insert into new

select \*

from inner\_join;

select\*

from new;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 b x

6 e

7 f

UPDATING VALUES IN EXISTING TABLE ROWS

To modify data values in some or all of the existing rows in a table, use the UPDATE statement.

* To update rows (or a subset of rows) using the same expression, use an UPDATE statement that contains a SET clause and an optional WHERE clause. The syntax is

proc sql;

**update** *table\_name*

**set** *column1=expression<, column2=expression,…>*

**<where expression>;**

select \*

from *table\_name*;

quit;

Example. The code below makes a single update to the new data set.

proc sql;

update new

set varA='u'

where id=2;

select\*

from new;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 u x

6 e

7 f

* There are two possible ways to update rows (or subsets of rows) using different expressions:
* Using multiple UPDATE statements, one for each subset of rows

The syntax is

proc sql;

**update** *table\_name*

**set** *column1=expression<, column2=expression,…>*

**<where expression>;**

**update …;**

**<update …>;**

select \*

from *table\_name*;

quit;

Example. The code below updates two rows of the new data set using different expressions in two UPDATE statements.

proc sql;

update new

set varA='u'

where id=2;

update new

set varB='oops!'

where id=7;

select\*

from new;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 u x

6 e

7 f

* Using a single UPDATE statement that contains a CASE expression

The syntax is

proc sql;

**update** *table\_name*

**set** *column1=*

**case**

**when** condition **then** expression

**<when** condition **then** expression**>**

**<else** expression**>**

**end;**

select \*

from *table\_name*;

quit;

Example. The code below updates two rows of the new data set using different expressions in a single UPDATE statement using a CASE clause.

proc sql;

update new

set varA=

case

when id=2 then 'u'

when id=7 then 'oops!'

end;

select\*

from new;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 u x

6

7 oops!

DELETING ROWS IN A TABLE

To delete some or all of the rows in a table, use the DELETE statement. The syntax is

proc sql;

**delete from** *table\_name*

**<where** expression**>;**

select \*

from *table\_name*;

quit;

Example. The code below deletes a row in the new data set.

proc sql;

delete from new

where id=6;

select\*

from new;

quit;

The output is

id varA varB

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 u x

7 oops!

ALTERING COLUMNS IN A TABLE

To add, drop (delete), or modify columns in a table, use the ALTER TABLE statement. The syntax is

proc sql;

**alter table** *table\_name*

<**add** *column1*-definition<, *column2*-definition, …>>

<**drop** *column1* <, *column2*, …>>

<**modify** *column1*-definition<, *column2*-definition, …>>

select

from *table\_name*;

quit;

Example. The code below adds, drops, and modifies columns in the new data set.

proc sql;

alter table new

add varC num format=comma10.2, varD char format=$3.

drop varB

modify varA format=$3. label='Modified Variable';

select\*

from new;

quit;

The output is

Modified var

id Variable varC D

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 u .

7 oop .

DROPPING TABLES

To delete one or more entire tables, use the DROP TABLE statement. The syntax is

proc sql;

**drop table** *table\_name*

quit;