Lesson: Subqueries

1. Subquery in the WHERE and HAVING Clauses

2. In-Line Views (Query in the FROM Clause)

3. Subquery in the SELECT Clause

Lesson: Subqueries

1. Subquery in the WHERE and HAVING Clauses

2. In-Line Views (Query in the FROM Clause)

3. Subquery in the SELECT Clause

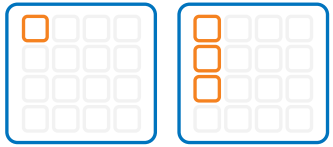
Subquery in the WHERE and HAVING Clauses

Outer Query

SELECT *col-name, col-name*
FROM *input-table*
WHERE *column operator* **(SELECT** *col-name*
FROM *input-table…***);**

Subquery

```
(select avg(PopEstimate1)
     from sq.statepopulation
     where … )…
```



Noncorrelated Subqueries

Outer Query

```
SELECT …
      FROM …
      <WHERE …>
      <GROUP BY …>
      <HAVING …>
      <ORDER BY …>;
```

Independent

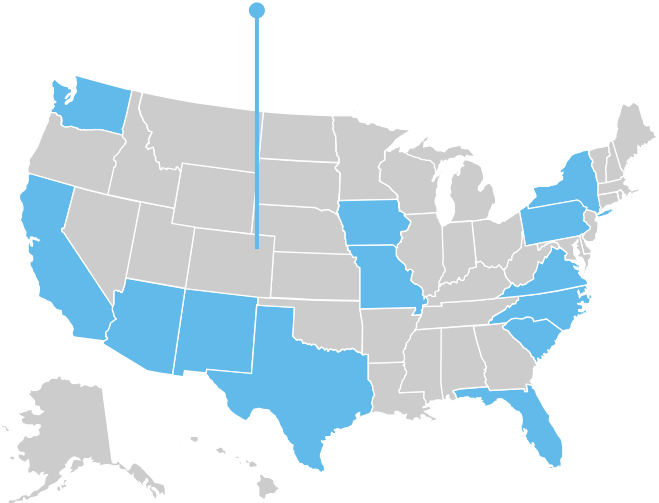
Subquery

```
(select avg(PopEstimate1)
      from sq.statepopulation
      where … )…
```

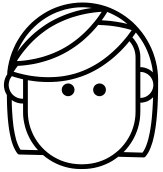
A noncorrelated subquery is a self-contained query. It executes independently of the outer query.

Scenario

State PopEstimate1 > **Total avg(PopEstimate1)**



Create a report that displays states with **PopEstimate1** values greater than the ***average*** **PopEstimate1** value of all states.



Subqueries Steps

1. Calculate the subquery's value.
2. Use the value from the subquery in the outer query.
3. Combine the subquery with the outer query.

statepopulation

Name	PopEstimate1
AL	4864745
AK	741504
AZ	6945452
AR	2990410
CA	39209127
CO	5540921

Solution without a Subquery

1

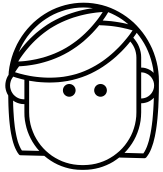
```
proc sql;
select avg(PopEstimate1) as Average
    from sq.statepopulation;
quit;
```

Average
6278420

2

```
proc sql;
select Name, PopEstimate1
    from sq.statepopulation
    where PopEstimate1 > 6278420;
quit;
```

What happens if **PopEstimate1** changes in the data?



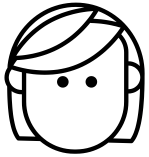
Subquery That Returns a Single Value

3

```
select Name, PopEstimate1
    from sq.statepopulation
    where PopEstimate1 > (select avg(PopEstimate1)
                          from sq.statepopulation);
```

The subquery is evaluated first.

The subquery executes ***independently*** of the outer query.



Subquery That Returns a Single Value

The outer query uses the value returned by the subquery.

3

```
select Name, PopEstimate1
    from sq.statepopulation
    where PopEstimate1 > (6278420);
```

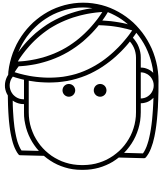
Name	PopEstimate1
AZ	6945452
CA	39209127
FL	20629982
GA	10304763
IL	12826895
IN	6633344

Run the query. What is the syntax error in the log?

```
…
where PopEstimate1 > (select avg(PopEstimate1),
                            'Average Population'
                        from sq.statepopulation);
```

6278420	Average Population

A subquery must return values in a ***single column***.



ERROR: A subquery cannot select more than one column.

ERROR: A Composite expression (usually a subquery) is used incorrectly in an expression.

Demo 1: Subquery That Returns a Single Value

Display customers who are older than the average customer.

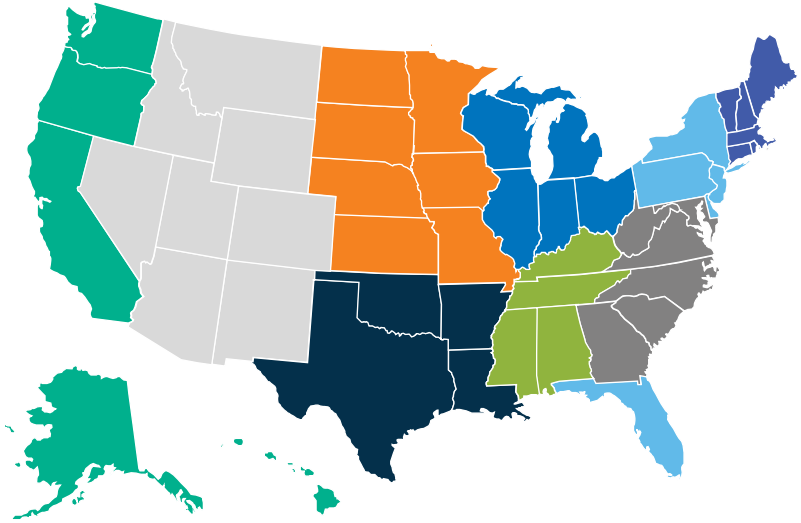
This demonstration illustrates using a noncorrelated subquery that returns a single value.

Subquery in the HAVING Clause

Division avg(PopEstimate1) > **Total avg(PopEstimate1)**

sq.statepopulation

Division	PopEstimate1
6	4864745
9	741504
8	6945452
7	2990410
9	39209127
8	5540921
1	3578674
5	949216
5	686575
5	20639982



Subquery in the HAVING Clause

1

```
proc sql;
select avg(PopEstimate1) as Average
    from sq.statepopulation;
quit;
```

Average
6278420

2

```
proc sql;
select Division, avg(PopEstimate1) as avgDivisionPop
    from sq.statepopulation
    group by Division
    having avgDivisionPop > 6278420;
quit;
```

Demo 2: Subquery in the HAVING Clause

Display countries with average customer age greater than the overall average age.

This demonstration illustrates using a noncorrelated subquery that returns a single value.

s104d02

Subquery That Returns Multiple Values

statepopulation Results

Division	Name
3	IL
3	IN
3	MI
3	OH
3	WI

sq.customer

FirstName	MiddleName	LastName	State
Rodney	Matthew	Joyner	WI
Jeanne	Carol	Ballenger	WA
Brian	Dallas	Harper	WI
Thomas	Eric	Henderson	WA
Becky	Danna	Cheers	WI
Alberto	Daryl	Texter	WI
Peter	Douglas	Schmand	WA
Danielle	Julie	Bell	WI
Robert	Javier	Brousseau	WI
Sharon	Julie	Howell	

Create a table of all customers who reside in a state in Division 3.

Subquery That Returns Multiple Values

1

```
proc sql;
select Name
    from sq.statepopulation
    where Division = '3';
quit;
```

Name
IL
IN
MI
OH
WI

2

```
proc sql;
create table division3 as
select *
    from sq.customer
    where State in ("IL","IN","MI","OH","WI");
quit;
```

Demo 3: Subquery Using Different Tables

Display customers from Asia/Pacific region.

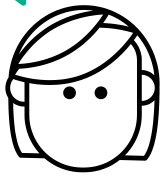
This demonstration illustrates using a noncorrelated subquery that returns multiple values.

ANY Keyword

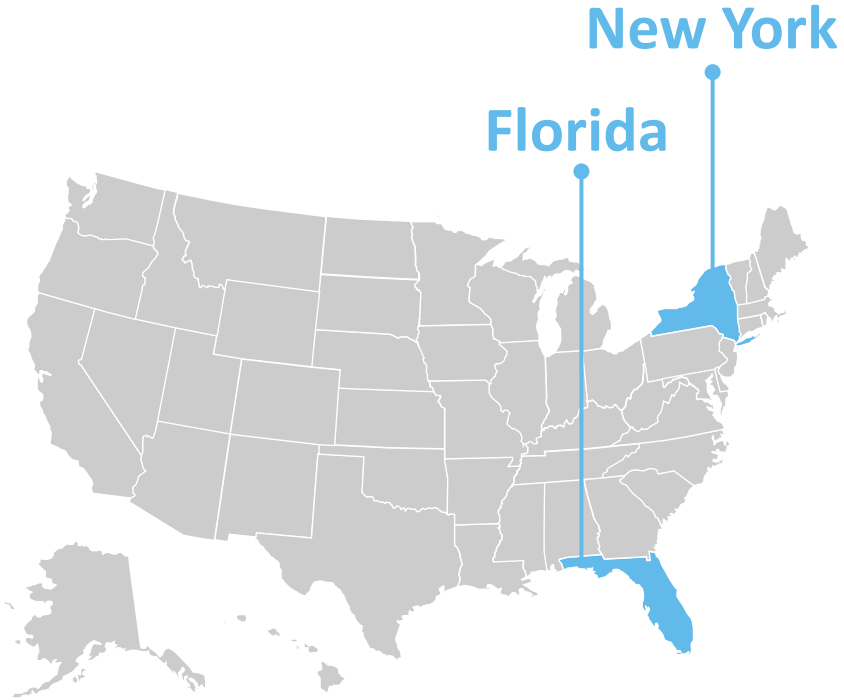
WHERE *column* = **ANY**(*subquery values*)

```
proc sql;
create table Division3 as
select *
    from sq.customer
    where State = any(select Name
                      from sq.statepopulation
                      where Division = '3');
quit;
```

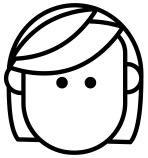
Equivalent to using the **IN** operator



Scenario



Which states have a **PopEstimate1** value that is ***greater*** than New York or Florida?



ANY Keyword

```
select Name, PopEstimate1
    from sq.statepopulation
    where PopEstimate1 > any(select PopEstimate1
                             from sq.statepopulation
                             where Name in ("NY","FL"));
```

Name	PopEstimate1
CA	39209127
FL	20629982
TX	27937492

```
20629982,19641589
```

The ANY keyword is true when the value of the specified column is greater than ***any*** of the values returned by the subquery.

MIN Function

```
select Name, PopEstimate1
    from sq.statepopulation
    where PopEstimate1 > (select min(PopEstimate1)
                          from sq.statepopulation
                          where Name in ("NY","FL"));
```

Name	PopEstimate1
CA	39209127
FL	20629982
TX	27937492

19641589

You can also use the MIN function inside the subquery to return the minimum **PopEstimate1**.

Demo 4: Subquery That Returns Multiple Values

How many employees earn more than a manager or a specialist? How many employees earn less than a sales rep, but they're not a sales rep?

This demonstration illustrates using a noncorrelated subquery that returns multiple values.

Correlated Subqueries

Outer Query

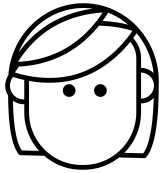
```
SELECT …
      FROM …
      <WHERE …>
      <GROUP BY …>
      <HAVING …>
      <ORDER BY …>;
```

Dependent

Correlated

```
(SELECT …
      FROM …
      <WHERE …>)
```

Correlated subqueries are ***resource intensive***.



Correlated Subqueries

```
select count(*) as TotalCustomer
    from sq.customer as c
    where '1' = (select Division
                 from sq.statepopulation as a
                 where a.Name = c.State);
```

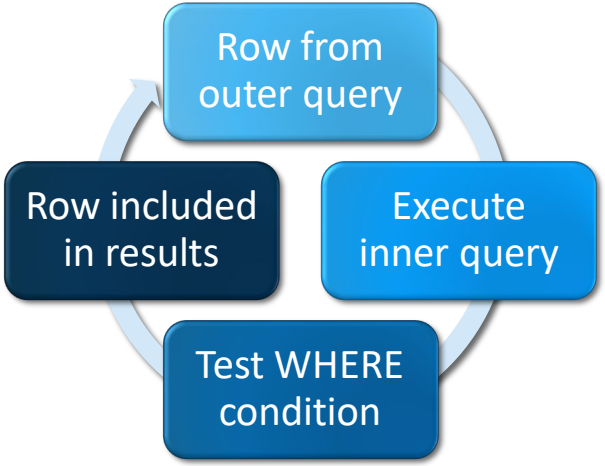
The inner query needs information from the ***outer query***.

How many customers are from a ***state*** in ***Division 1***?

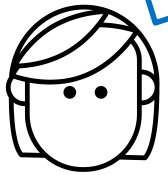
Correlated Subqueries

Executing a correlated subquery can be resource intensive.

```
select count(*) as TotalCustomer
    from sq.customer as c
    where '1' = (select Division
                 from sq.statepopulation as s
                 where s.Name = c.State);
```



Each subquery is executed once for ***every row*** of the outer query.



Demo 5: Correlated Subqueries

Display the employees earning more than their department's average.

This demonstration illustrates using a correlated subquery.

Using Joins

```
select count(*) as TotalCustomer
    from sq.customer as c
    where '1' = (select Division
                 from sq.statepopulation as s
                 where s.Name = c.State);
```

TotalCustomer
3292

A more concise method is to use a join.

```
select count(*) as TotalCustomer
    from sq.customer as c inner join
         sq.statepopulation as p
       on p.Name = c.State
    where Division = '1';
```

TotalCustomer
3292

Syntax Summary

```
SELECT col-name, col-name
      FROM input-table
      WHERE column operator (SELECT col-name
                                   FROM input-table…)
      GROUP BY col-name
      HAVING column operator (SELECT col-name
                                   FROM input-table…);
```

WHERE and HAVING Subqueries

```
WHERE column IN (subquery values)
WHERE column = ANY(subquery values)
WHERE column > ANY(subquery values)
```

Multiple Values Returned by a Subquery

Lesson 4: Subqueries

4.1 Subquery in the WHERE and HAVING Clauses

4.2 In-Line Views (Query in the FROM Clause)

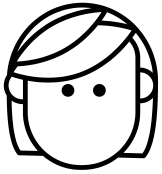
4.3 Subquery in the SELECT Clause

In-Line View

Outer Query

```
SELECT …
      FROM (SELECT col-name
                   FROM …
                   <WHERE …>)
      <WHERE …>
      <GROUP BY …>
      <HAVING …>
      <ORDER BY …>;
```

An in-line view acts as a ***virtual table***.

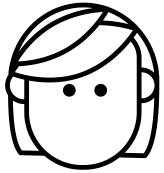


In-Line View

```
proc sql;
select *
    from (select CustomerID, State, Income
              from sq.customer
              where Income > 100000);
quit;
```

Creates a virtual table to use in the outer query.

You ***cannot*** use an ORDER BY clause in an in-line view.



Scenario

State	TotalCustomer	EstimateBase	PctCustomer
VT	47	625744	.008%
WV	217	1853001	.012%
ME	159	1328369	.012%
DE	110	897934	.012%
MD	768	5773798	.013%
SC	745	4625381	.016%
NH	234	1316464	.018%
PA	2405	12702873	.019%
HI	263	1360307	.019%
GA	1912	9688709	.020%

Create a report that shows the percentage of customers in each state based on each state's estimated population.

Using Temporary Tables

1

```
create table totalcustomer as
select State,count(*) as TotalCustomer
     from sq.customer
     group by State;
```

Count the number of customers in each state.

State	TotalCustomer
AK	289
AL	1337
AR	760
AZ	3185
CA	18134
CO	1920
CT	755

Using Temporary Tables

2

```
select c.State,
       c.TotalCustomer,
       s.EstimateBase,
       c.TotalCustomer/s.EstimateBase as
           PctCustomer format=percent7.3
    from totalcustomer as c inner join
         sq.statepopulation as s
    on c.State = s.Name
    order by PctCustomer;
```

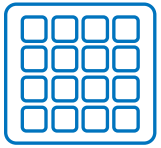
Join with the **sq.statepopulation** table.

State	TotalCustomer
AK	289
AL	1337
AR	760
AZ	3185
CA	18134
CO	1920
CT	755

Name	EstimateBase
AL	4780138
AK	710249
AZ	6392288
AR	2916028
CA	37254523
CO	5029316
CT	3574147
DE	897934
DC	601766

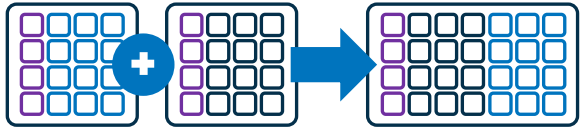
Using Temporary Tables

1



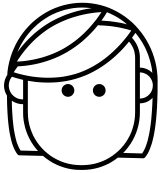
Create the **totalcustomer** table.

2



Join **totalcustomer** with **sq.statepopulation**.

What if the **customer** table is updated daily and we always want the most ***recent results***?



Demo 6: Using an In-Line View

Display the shoe line-group-category combinations that return negative profits on average.

This demonstration illustrates using an In-Line View to create a virtual table.

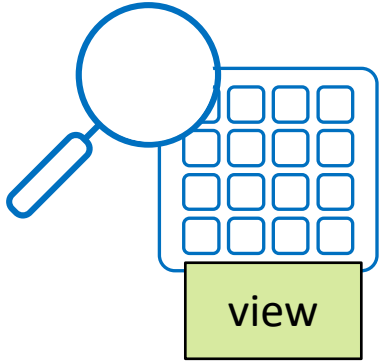
Storing an In-Line View

```
…
from (select State,count(*) as TotalCustomer
          from sq.customer
          group by State)
…;
```

How can you ***store*** the in-line view to use in other queries?

Creating a View

```
select State,count(*) as TotalCustomer
          from sq.customer
          group by State);
```



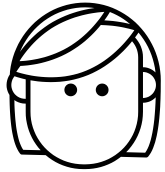
- is a stored query and contains no actual data
- can be derived from one or more tables, PROC SQL views, DATA step views, or SAS/ACCESS views
- accesses the most current data
- can be referenced in SAS programs
- cannot have the same name as a data table stored in the same SAS library

CREATE VIEW Statement

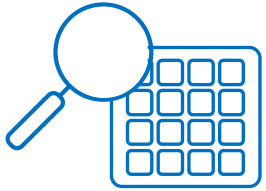
CREATE VIEW *view-name* **AS**
SELECT …;

```
proc sql;
create view sq.totalcustomer as
select State,count(*) as TotalCustomer
    from sq.customer
    group by State;
quit;
```

The ***query*** is stored as a permanent view in the **sq** library.



sq.totalcustomer



Using a VIEW

sq.totalcustomer

State	TotalCustomer
AK	289
AL	1337
AR	760
AZ	3185

```
select c.State, c.TotalCustomer, s.EstimateBase,
       c.TotalCustomer/s.EstimateBase as
            PctCustomer format=percent7.3
    from (select State,count(*) as TotalCustomer
            from sq.customer
            group by State) as c inner join
         sq.statepopulation as s
    on c.State = s.Name
    order by PctCustomer;
```

You can use a view ***in place*** of the in-line view.

Using a VIEW

sq.totalcustomer

State	TotalCustomer
AK	289
AL	1337
AR	60
AZ	3

The view ***executes*** the stored query and extracts the ***most current data***.

```
select c.State, c.TotalCustomer, s.EstimateBase,
       c.TotalCustomer/s.EstimateBase as
            PctCustomer format=percent7.3
    from sq.totalcustomer as c inner join
         sq.statepopulation as s
    on c.State = s.Name
    order by PctCustomer;
```

Demo 7: Creating and Using Views

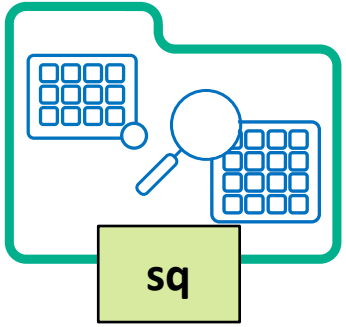
Create a View to display the total number of customers in each country.

This demonstration illustrates creating and using Views to create a virtual table.

Location of a PROC SQL View

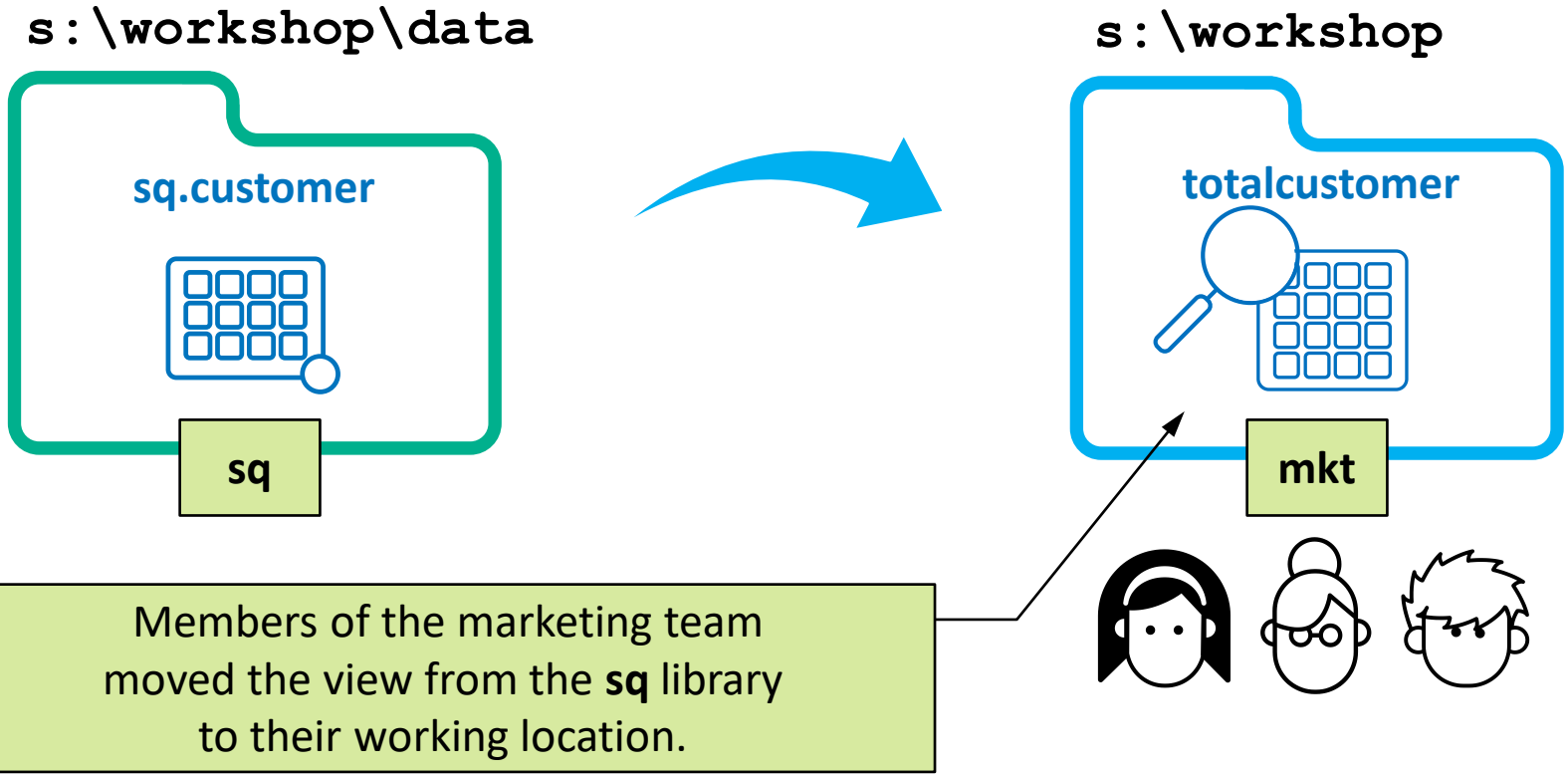
```
proc sql;
create view sq.totalcustomer as
select State,count(*) as TotalCustomer
    from customer
    group by State;
quit;
```

PROC SQL expects the view to reside in the ***same*** SAS library as the contributing table or tables.



S:\workshop\data

Scenario

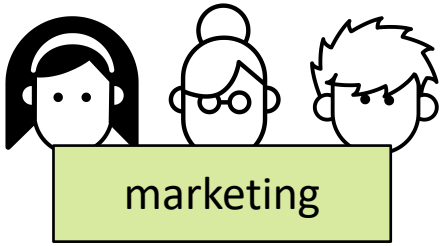


Scenario

```
libname mkt "s:/workshop";

proc sql;
select *
    from mkt.totalcustomer
quit;
```

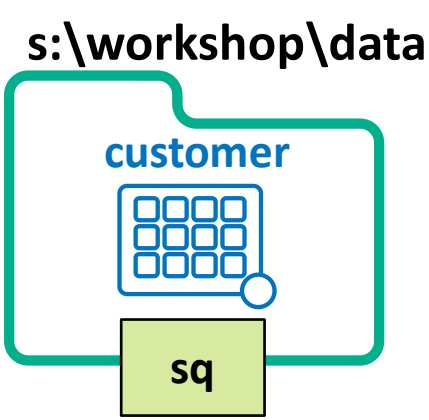
ERROR: File MKT.CUSTOMER.DATA does not exist.



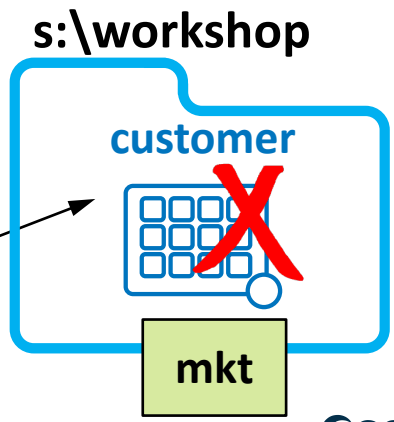
Marketing executes a query using the view but receives this error. Why?

Exploring the Problem

```
proc sql;
create view sq.totalcustomer as
select State,count(*) as TotalCustomer
    from customer
    group by State;
quit;
```



The view assumes that the **customer** data is in the **s:\workshop** library.



Making a View Portable

CREATE VIEW *view-name* **AS SELECT** …
USING *LIBNAME-clause<, …LIBNAME-clause>*;

```
libname mkt "s:/workshop";
proc sql;
create view mkt.totalcustomer as
select State,count(*) as TotalCustomer
    from sq.customer
    group by State
    using libname sq 's:/workshop/data';
quit;
```

The scope of the libref is local to the view and does not conflict with any identically named librefs in the SAS session.

Views

Advantages

- avoid storing copies of large tables
- avoid a frequent refresh of table copies; when the underlying data changes, a view surfaces the most current data
- combine data from multiple database tables and multiple libraries or databases
- simplify complex queries
- prevent other users from inadvertently altering the query code

Views

Disadvantages

- Views might produce different results each time they are accessed if the data in the underlying data sources changes.
- Views can require significant resources each time that they execute. With a view, you save disk storage space at the cost of extra CPU and memory usage.

Syntax Summary

```
SELECT …
      FROM (SELECT col-name
                   FROM …
                   <WHERE …>) ;
```

In-Line View

```
CREATE VIEW table-name AS query
```

CREATE VIEW

```
CREATE VIEW …
USING LIBNAME libref engine "path";
```

USING Clause

Lesson 4: Subqueries

4.1 Subquery in the WHERE and HAVING Clauses

4.2 In-Line Views (Query in the FROM Clause)

4.3 Subquery in the SELECT Clause

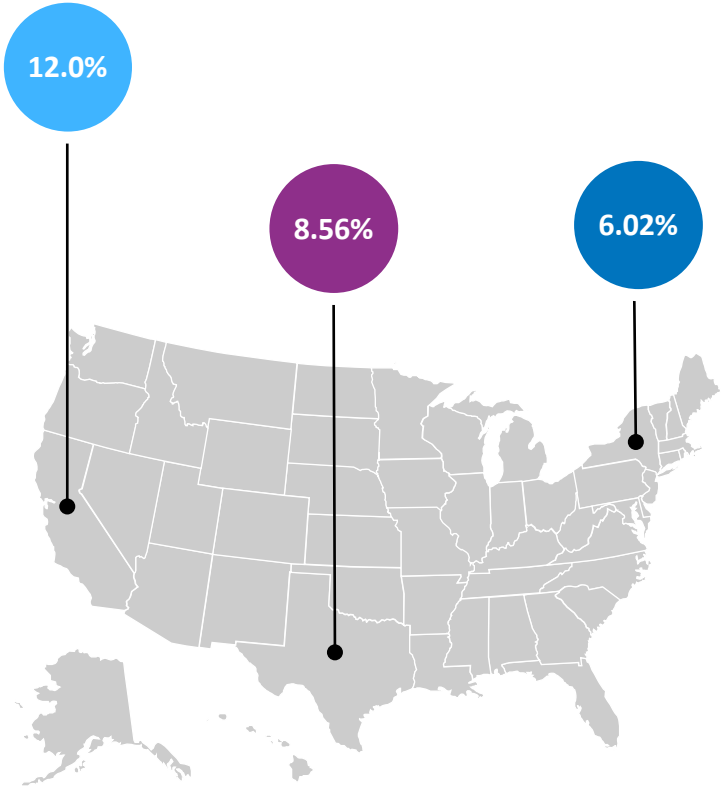
Subquery in the SELECT Clause

Outer Query

```
SELECT col-name, (SELECT col-name
                  FROM ...
                  <WHERE ...>)
    FROM ...
    <WHERE ...>
    <GROUP BY ...>
    <HAVING ...>
    <ORDER BY ...>;
```

A subquery in the SELECT clause can return a ***single value*** to the outer query.

Scenario



State
PopEstimate1

/

Total
sum(PopEstimate1)

Create results that show next year's estimated percentage of population of each state.

Subquery in the SELECT Clause

```
select Name, PopEstimate1/(select sum(PopEstimate1)
                           from sq.statepopulation)
             as PctPop format=percent7.2
    from sq.statepopulation
    order by PctPop desc;
```

Name	PctPop
CA	12.0%
TX	8.56%
FL	6.32%
NY	6.02%
IL	3.03%

State
PopEstimate1

/

326477837

PopEstimate1 from each state

total estimated population for the US

Remerging Summary Statistics

selects each name and **PopEstimate1** value from each row

summarizes the ***entire*** column

The remerge feature of SAS makes ***two passes*** through a table.

```
select Name, PopEstimate1 / sum(PopEstimate1)
             as PctPop format=percent7.2
    from sq.statepopulation
    order by PctPop desc;
```

Name	PctPop
CA	12.0%
TX	8.56%
FL	6.32%
NY	6.02%
IL	2.03%

Remerging Summary Statistics

```
select Region,
       sum(PopEstimate1) as TotalRegion format=comma14.
    from sq.statepopulation;
```

Specifying a column and a summary function remerges the data and does ***not*** return the desired result.

Region	TotalRegion
3	326,477,837
4	326,477,837
4	326,477,837
3	326,477,837
4	326,477,837
4	326,477,83

Disabling the Remerging of Summary Statistics

PROC SQL NOREMERGE;

```
proc sql noremerge;
select Region,
       sum(PopEstimate1) as TotalRegion format=comma14.
    from sq.statepopulation;
quit;
```

ERROR: The query requires remerging summary statistics back with the original data. This is disallowed due to the NOREMERGE proc option or NOSQLREMERGE system option.

Scenario

Name	PopEstimate1	TotalRegionEst	PctRegion	Region
NY	19641589	56,058,789	35.0%	1
PA	12783538	56,058,789	22.8%	1
NJ	8874516	56,058,789	15.8%	1
MA	6826022	56,058,789	12.2%	1
CT	3578674	56,058,789	6.4%	1
NH	1342373	56,058,789	2.4%	1
ME	1331370	56,058,789	2.4%	1
RI	1057063	56,058,789	1.9%	1
VT	623644	56,058,789	1.1%	1
IL	12826895	67,996,917	18.9%	2
OH	11635003	67,996,917	17.1%	2
MI	9951890	67,996,917	14.6%	2
IN	6633344	67,996,917	9.8%	2
MO	6087203	67,996,917	9.0%	2

How can we calculate the percentage of population in each ***region***?

Remerging GROUP BY Summary Statistics

```
proc sql;
select Name, PopEstimate1,
       sum(PopEstimate1) as TotalRegionEst format=comma14.,
       PopEstimate1/calculated TotalRegionEst as
                    PctRegion format=percent7.1,
     Region
  from sq.statepopulation
  group by Region
  order by Region, PctRegion desc;
quit;
```

SAS remerges the sum of each region.

Name	PopEstimate1	TotalRegionEst	PctRegion	Region
NY	19641589	56,058,789	35.0%	1
PA	12783538	56,058,789	22.8%	1
NJ	8874516	56,058,789	15.8%	1
MA	6826022	56,058,789	12.2%	1
CT	3578674	56,058,789	6.4%	1
NH	1342273	56,058,789	2.4%	1

Remerging Summary Statistics

Display the percent profit contributed by each product category.

This demonstration illustrates remerging summary statistics in SAS to find the percentage population of each state.

Syntax Summary

```
SELECT col-name, (SELECT col-name
                  FROM …
                  <WHERE …>)
    FROM table1;
```

Subquery in the SELECT Clause

```
PROC SQL NOREMERGE;
```

PROC SQL Option

```
SELECT col-name, summary function(column)
    FROM table1;
```

Remerging Summary Statistics

Beyond SQL Essentials

What if you want to ...

. . . learn about using a reflexive join with a subquery?

- View the SAS paper Nifty Uses of SQL Reflexive Join and Sub-query in SAS.

. . . learn about building subqueries in SAS Enterprise Guide?

- Read the SAS blog Building an SQL subquery in SAS Enterprise Guide.

. . . review examples using subqueries to select data?

- Visit Using Subqueries to Select Data in the SAS documentation.