SAS SQL #2

JC Wang

Western Michigan University
Department of Statistics
Outline

1. Query a Table, continued
   - Grouping Data
   - Validating Queries
   - Creating Tables from Queries

2. Querying Multiple Tables
   - Joins
   - Inner Joins
   - Outer Joins
   - Specialty Joins
   - Subqueries

3. Combining Queries
   - Combining Queries with Set Operators
Grouping Data Using GROUP BY Clause

to group data by a column or columns

**GROUP BY** *input-column1*

<,*input-column2*, ...>

- usually used with an aggregate function in the SELECT clause or in a HAVING clause about how to summarize the data for each group
- PROC SQL calculates the aggregate function separately for each group
- when a GROUP BY clause is used without an aggregate function, PROC SQL treats the GROUP BY clause as if it were an ORDER BY clause and gives a warning message in the log that informs you that this has happened
Grouping Data Using GROUP BY Clause

to group data by a column or columns

**GROUP BY**  
`input-column1, input-column2, ...`

- usually used with an aggregate function in the SELECT clause or in a HAVING clause about how to summarize the data for each group
- PROC SQL calculates the aggregate function separately for each group
- when a GROUP BY clause is used without an aggregate function, PROC SQL treats the GROUP BY clause as if it were an ORDER BY clause and gives a warning message in the log that informs you that this has happened
Grouping Data Using GROUP BY Clause

to group data by a column or columns

GROUP BY input-column1
<,input-column2, ...>

- usually used with an aggregate function in the SELECT clause or in a HAVING clause about how to summarize the data for each group
- PROC SQL calculates the aggregate function separately for each group
- when a GROUP BY clause is used without an aggregate function, PROC SQL treats the GROUP BY clause as if it were an ORDER BY clause and gives a warning message in the log that informs you that this has happened
Grouping Data Using GROUP BY Clause

to group data by a column or columns

GROUP BY input-column1
    <, input-column2, ...>

- usually used with an aggregate function in the SELECT clause or in a HAVING clause about how to summarize the data for each group
- PROC SQL calculates the aggregate function separately for each group
- when a GROUP BY clause is used without an aggregate function, PROC SQL treats the GROUP BY clause as if it were an ORDER BY clause and gives a warning message in the log that informs you that this has happened
Grouping by One Column

Codes

```
SELECT site, COUNT(*) 'Number of Subjects',
       SUM(visits) 'Total Visits'
FROM mylib.clinic
GROUP BY site;
```

Output

<table>
<thead>
<tr>
<th>site</th>
<th>Number of Subjects</th>
<th>Total Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>61</td>
<td>127</td>
</tr>
<tr>
<td>02</td>
<td>66</td>
<td>130</td>
</tr>
<tr>
<td>03</td>
<td>46</td>
<td>90</td>
</tr>
</tbody>
</table>

... more output lines ...

wang (WMU)
Grouping by One Column

Codes

SELECT site, COUNT(*) 'Number of Subjects',
  SUM(visits) 'Total Visits'
FROM mylib.clinic
GROUP BY site;

Output

<table>
<thead>
<tr>
<th>site</th>
<th>Number of Subjects</th>
<th>Total Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>61</td>
<td>127</td>
</tr>
<tr>
<td>02</td>
<td>66</td>
<td>130</td>
</tr>
<tr>
<td>03</td>
<td>46</td>
<td>90</td>
</tr>
</tbody>
</table>

... more output lines ...

wang (WMU)
Grouping by One Column

Codes

```sql
SELECT site, COUNT(*) 'Number of Subjects',
       SUM(visits) 'Total Visits'
FROM mylib.clinic
GROUP BY site;
```

Output

<table>
<thead>
<tr>
<th>site</th>
<th>Number of Subjects</th>
<th>Total Visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>61</td>
<td>127</td>
</tr>
<tr>
<td>02</td>
<td>66</td>
<td>130</td>
</tr>
<tr>
<td>03</td>
<td>46</td>
<td>90</td>
</tr>
</tbody>
</table>

... more output lines ...
### Grouping and Sorting Results

**Codes**

```sql
SELECT size 'Clinic Size', state 'State',
       COUNT(DISTINCT patient) 'Number Of Patients'
FROM mylib.master
WHERE state IN ('MI','IN')
GROUP BY size, state
ORDER BY size DESC, state;
```

**Output**

<table>
<thead>
<tr>
<th>Clinic Size</th>
<th>State</th>
<th>Number of Patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>IN</td>
<td>105</td>
</tr>
<tr>
<td>Small</td>
<td>MI</td>
<td>89</td>
</tr>
<tr>
<td>Medium</td>
<td>IN</td>
<td>321</td>
</tr>
<tr>
<td>Medium</td>
<td>MI</td>
<td>276</td>
</tr>
<tr>
<td>Large</td>
<td>IN</td>
<td>1123</td>
</tr>
<tr>
<td>Large</td>
<td>MI</td>
<td>756</td>
</tr>
</tbody>
</table>
Grouping and Sorting Results

Codes

SELECT size 'Clinic Size', state 'State',
       COUNT(DISTINCT patient) 'Number Of Patients'
FROM mylib.master
WHERE state IN ('MI','IN')
GROUP BY size, state
ORDER BY size DESC, state;

Output

<table>
<thead>
<tr>
<th>Clinic Size</th>
<th>State</th>
<th>Number Of Patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>IN</td>
<td>105</td>
</tr>
<tr>
<td>Small</td>
<td>MI</td>
<td>89</td>
</tr>
<tr>
<td>Medium</td>
<td>IN</td>
<td>321</td>
</tr>
<tr>
<td>Medium</td>
<td>MI</td>
<td>276</td>
</tr>
<tr>
<td>Large</td>
<td>IN</td>
<td>1123</td>
</tr>
<tr>
<td>Large</td>
<td>MI</td>
<td>756</td>
</tr>
</tbody>
</table>
Grouping and Sorting Results

Codes

```sql
SELECT size 'Clinic Size', state 'State',
       COUNT(DISTINCT patient) 'Number Of Patients'
FROM mylib.master
WHERE state IN ('MI', 'IN')
GROUP BY size, state
ORDER BY size DESC, state;
```

Output

<table>
<thead>
<tr>
<th>Clinic Size</th>
<th>State</th>
<th>Number Of Patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>IN</td>
<td>105</td>
</tr>
<tr>
<td>Small</td>
<td>MI</td>
<td>89</td>
</tr>
<tr>
<td>Medium</td>
<td>IN</td>
<td>321</td>
</tr>
<tr>
<td>Medium</td>
<td>MI</td>
<td>276</td>
</tr>
<tr>
<td>Large</td>
<td>IN</td>
<td>1123</td>
</tr>
<tr>
<td>Large</td>
<td>MI</td>
<td>756</td>
</tr>
</tbody>
</table>
Filtering Results by HAVING Clause

similar to WHERE clause except that HAVING clause filters group return by the GROUP BY clause

**HAVING** \(\text{cond-involving-group by-variables}\)

- usually follows GROUP BY to filter results from GROUP BY and use of aggregate function(s)
- if used without a GROUP BY clause, then it’s treated as a WHERE clause and SAS issues a warning message
- is processed after the GROUP BY clause and any aggregate functions
Filtering Results by HAVING Clause

similar to WHERE clause except that HAVING clause filters group return by the GROUP BY clause

**HAVING** `cond-involving-group by-variables`

- usually follows GROUP BY to filter results from GROUP BY and use of aggregate function(s)
- if used without a GROUP BY clause, then it’s treated as a WHERE clause and SAS issues a warning message
- is processed after the GROUP BY clause and any aggregate functions
Filtering Results by HAVING Clause

similar to WHERE clause except that HAVING clause filters group return by the GROUP BY clause

**HAVING** \( \text{cond-involving-group by-variables} \)

- usually follows GROUP BY to filter results from GROUP BY and use of aggregate function(s)
- if used without a GROUP BY clause, then it’s treated as a WHERE clause and SAS issues a warning message
- is processed after the GROUP BY clause and any aggregate functions
Filtering Results by HAVING Clause

similar to WHERE clause except that HAVING clause filters group return by the GROUP BY clause

**HAVING** \( \text{cond-involving-group by-variables} \)

- usually follows GROUP BY to filter results from GROUP BY and use of aggregate function(s)
- if used without a GROUP BY clause, then it’s treated as a WHERE clause and SAS issues a warning message
- is processed after the GROUP BY clause and any aggregate functions
Filtering Results by HAVING Clause

Codes

```sql
SELECT site,
       COUNT(DISTINCT patient) AS npatients
FROM mytable
GROUP BY site
HAVING npatients GT 14
ORDER BY npatients desc, site;
```

Output

<table>
<thead>
<tr>
<th>site</th>
<th>npatients</th>
</tr>
</thead>
<tbody>
<tr>
<td>061</td>
<td>20</td>
</tr>
<tr>
<td>019</td>
<td>19</td>
</tr>
<tr>
<td>013</td>
<td>15</td>
</tr>
<tr>
<td>018</td>
<td>15</td>
</tr>
<tr>
<td>032</td>
<td>15</td>
</tr>
</tbody>
</table>

... more output...
Filtering Results by HAVING Clause

Codes

```sql
SELECT site,
       COUNT(DISTINCT patient) AS npatients
FROM mytable
GROUP BY site
HAVING npatients GT 14
ORDER BY npatients desc, site;
```

Output

<table>
<thead>
<tr>
<th>site</th>
<th>npatients</th>
</tr>
</thead>
<tbody>
<tr>
<td>061</td>
<td>20</td>
</tr>
<tr>
<td>019</td>
<td>19</td>
</tr>
<tr>
<td>013</td>
<td>15</td>
</tr>
<tr>
<td>018</td>
<td>15</td>
</tr>
<tr>
<td>032</td>
<td>15</td>
</tr>
</tbody>
</table>

... more output...
Filtering Results by HAVING Clause

Codes

SELECT site,
       COUNT(DISTINCT patient) AS npatients
FROM mytable
GROUP BY site
HAVING npatients GT 14
ORDER BY npatients desc, site;

Output

<table>
<thead>
<tr>
<th>site</th>
<th>npatients</th>
</tr>
</thead>
<tbody>
<tr>
<td>061</td>
<td>20</td>
</tr>
<tr>
<td>019</td>
<td>19</td>
</tr>
<tr>
<td>013</td>
<td>15</td>
</tr>
<tr>
<td>018</td>
<td>15</td>
</tr>
<tr>
<td>032</td>
<td>15</td>
</tr>
</tbody>
</table>

... more output...

wang (WMU)
Validating a Query

**VALIDATE**

`query;`

Upon submitting the query, PROC SQL print a message to SAS log indicating whether the syntax is correct.
Creating a Table from a Query

CREATE TABLE table AS
query;
Note that data set options are allowed in the table the query works on.
Outline

1. **Query a Table, continued**
   - Grouping Data
   - Validating Queries
   - Creating Tables from Queries

2. **Querying Multiple Tables**
   - Joins
   - Inner Joins
   - Outer Joins
   - Specialty Joins
   - Subqueries

3. **Combining Queries**
   - Combining Queries with Set Operators
Joins

A join is a query that selects data from two or more tables/views, including self-join in which an alias can be considered as different table.

- Cartesian product
- Inner join
- Self join
- Left (outer) join
- Right (outer) join
- Full (outer) join
- Specialty joins: cross join (=cartesian product), union join, natural join
Joins

A join is a query that select data from two or more tables/views, including self-join in which an alias can be considered as different table.

- Cartesian product
- Inner join
- Self join
- Left (outer) join
- Right (outer) join
- Full (outer) join
- Specialty joins: cross join (=cartesian product), union join, natural join
Cartesian Product

\begin{verbatim}
SELECT * FROM table-1, table-2;
\end{verbatim}

- resulted table size = product of input table sizes
- subset of the cartesian product is usually desired (two types: inner joins and outer joins)

<table>
<thead>
<tr>
<th>table_1</th>
<th>table_2</th>
<th>Output table</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>x11</td>
<td>y11</td>
<td>x11</td>
</tr>
<tr>
<td>x21</td>
<td>y21</td>
<td>x12</td>
</tr>
<tr>
<td>x31</td>
<td>y31</td>
<td>x22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x31</td>
</tr>
</tbody>
</table>
**Cartesian Product**

**SELECT * FROM** `table-1, table-2;`

- resulted table size = product of input table sizes
- subset of the cartesian product is usually desired (two types: inner joins and outer joins)

<table>
<thead>
<tr>
<th><code>table_1</code></th>
<th><code>table_2</code></th>
<th><code>Output table</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td><code>x</code></td>
<td><code>x</code></td>
</tr>
<tr>
<td><code>y</code></td>
<td><code>z</code></td>
<td><code>y</code></td>
</tr>
<tr>
<td><code>-------------</code></td>
<td><code>-------------</code></td>
<td><code>-------------</code></td>
</tr>
<tr>
<td>x11 y11</td>
<td>x12 z12</td>
<td>x11 y11</td>
</tr>
<tr>
<td>x21 y21</td>
<td></td>
<td>x12 z12</td>
</tr>
<tr>
<td>x31 y31</td>
<td></td>
<td>x22 z22</td>
</tr>
<tr>
<td></td>
<td>x12 z12</td>
<td>x22 z22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x22 z22</td>
</tr>
<tr>
<td></td>
<td>x12 z12</td>
<td>x22 z22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x22 z22</td>
</tr>
</tbody>
</table>
Cartesian Product

**SELECT * FROM** table-1, table-2;

- resulted table size = product of input table sizes
- subset of the castesian product is usually desired (two types: inner joins and outer joins)

<table>
<thead>
<tr>
<th>table_1</th>
<th>table_2</th>
<th>Output table</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>x11</td>
<td>y11</td>
<td>x11</td>
</tr>
<tr>
<td>x21</td>
<td>y21</td>
<td>x12</td>
</tr>
<tr>
<td>x31</td>
<td>y31</td>
<td>x12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z22</td>
</tr>
</tbody>
</table>
Cartesian Product

**SELECT** * FROM table-1, table-2;

- resulted table size = product of input table sizes
- subset of the cartesian product is usually desired (two types: inner joins and outer joins)

<table>
<thead>
<tr>
<th>table_1</th>
<th>table_2</th>
<th>Output table</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>y</td>
<td>z</td>
<td>y</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>--------------</td>
</tr>
<tr>
<td>x11</td>
<td>x12</td>
<td>x11 y11 x12 z12</td>
</tr>
<tr>
<td>y11</td>
<td>z12</td>
<td>x12 z12</td>
</tr>
<tr>
<td>x21</td>
<td>x12</td>
<td>x12 z12</td>
</tr>
<tr>
<td>y21</td>
<td>z12</td>
<td>x12 z12</td>
</tr>
<tr>
<td>x31</td>
<td>x22</td>
<td>x12 z12</td>
</tr>
<tr>
<td>y31</td>
<td>z22</td>
<td>x12 z12</td>
</tr>
</tbody>
</table>

wang (WMU)
Syntax of Inner Join

```
SELECT * FROM table-1, table-2
    WHERE <table-1.>col-1=<table-2.>col-2;
```

or

```
SELECT * FROM table-1 INNER JOIN table-2
    ON <table-1.>col-1=<table-2.>col-2;
```

Note: the condition in the WHERE clause above is only an example.
Using Table Alias

FROM `table-1` <AS> `alias-1`, `table-2` <AS> `alias-2`

- to avoid referencing a table with (one- or two- word) name all the time, a shorter (one-word) table alias can be used
- when referencing table using alias in any part of the query (SELECT, WHERE, ...), use `alias.col` (when identical column name occurs from distinct tables)
Using Table Alias

```
FROM table-1 <AS> alias-1, table-2 <AS> alias-2
```

- to avoid referencing a table with (one- or two-word) name all the time, a shorter (one-word) table alias can be used
- when referencing table using alias in any part of the query (SELECT, WHERE, ...), use `alias.col` (when identical column name occurs from distinct tables)
Using Table Alias

FROM table-1 <AS> alias-1, table-2 <AS> alias-2

- to avoid referencing a table with (one- or two- word) name all the time, a shorter (one-word) table alias can be used
- when referencing table using alias in any part of the query (SELECT, WHERE, ...), use alias.col (when identical column name occurs from distinct tables)
Sorting Results from Joins

```
SELECT * FROM table-1, table-2
  WHERE <table-1.>col-1=<table-2.>col-2
  ORDER BY column(s);
```

where

- `column(s)`: comma-separated list of columns
- keyword `DESC` can follow a column if needed
Sorting Results from Joins

```
SELECT * FROM table-1, table-2
 WHERE <table-1.>col-1=<table-2.>col-2
 ORDER BY column(s);
```

where

- `column(s)`: comma-separated list of columns
- keyword `DESC` can follow a column if needed
Self-Joins

FROM table-1 <AS> alias-1, table-1 <AS> alias-2

A self-join

- is also called a reflexive join
- is used to show comparative relationships between values in a table
Self-Joins

\[
\text{FROM } \text{table-1 <AS> alias-1, table-1 <AS> alias-2}
\]

A self-join

- is also called a reflexive join
- is used to show comparative relationships between values in a table
Self-Joins

`FROM table-1 <AS> alias-1, table-1 <AS> alias-2`

A self-join
- is also called a reflexive join
- is used to show comparative relationships between values in a table
Types of Outer Joins

- left join
- right join
- full join
Types of Outer Joins

- left join
- right join
- full join
Types of Outer Joins

- left join
- right join
- full join
Types of Outer Joins

- left join
- right join
- full join
Left Joins

SELECT input-columns
    FROM table-1 LEFT JOIN table-2;
    ON matching-condition(s);

list matched rows and non-matched rows from left table (table-1)
Right Joins

```
SELECT input-columns
FROM table-1 RIGHT JOIN table-2;
ON matching-condition(s);
```

list matched rows and non-matched rows from right table (table-2)
Full Joins

SELECT input-columns
    FROM table-1 FULL JOIN table-2;
    ON matching-condition(s);

list matched rows and non-matched rows from both tables
Specialty Joins

The specialty joins are special cases of the standard joins:

- CROSS JOIN = cartesian product
- UNION JOIN = queries (tables) combined with OUTER UNION operator, does not attempt to match rows
- NATURAL JOIN = cartesian product if no column with common name and type from two tables; NATURAL JOIN = inner join if at least one column with a common name and type from two tables (ON clause implied and not specified).
- a WHERE clause can be used to limit output
Subqueries

A subquery, also known as inner query (enclosed in parentheses)

- selects rows from one table based on values in another;
- is a query-expression that is nested as part of another query-expression;
- depending on the clause that contains it, can return a single value or multiple values;
- is most often used in the WHERE and the HAVING expressions.
Outline

1. Query a Table, continued
   - Grouping Data
   - Validating Queries
   - Creating Tables from Queries

2. Querying Multiple Tables
   - Joins
   - Inner Joins
   - Outer Joins
   - Specialty Joins
   - Subqueries

3. Combining Queries
   - Combining Queries with Set Operators
Combining Queries with set operators

query-1
set-operator <keyword>
query-2;
* as a single statement;
where set-operator is one of

- UNION (all unique rows)
- EXCEPT (rows from first query only)
- INTERSECT (common rows from both queries)
- OUTER UNION (concatenates query outputs)

keyword is one of

- ALL (not suppressing duplicated rows, making one pass only)
- CORR | CORRESPONDING (overlapping columns with common names)
Combining Queries with set operators

query-1

set-operator <keyword>

query-2; * as a single statement;

where set-operator is one of

- UNION (all unique rows)
- EXCEPT (rows from first query only)
- INTERSECT (common rows from both queries)
- OUTER UNION (concatenates query outputs)

keyword is one of

- ALL (not suppressing duplicated rows, making one pass only)
- CORR | CORRESPONDING (overlaying columns with common names)
Combining Queries with set operators

query-1
set-operator <keyword>
query-2; * as a single statement;
where set-operator is one of

- UNION (all unique rows)
- EXCEPT (rows from first query only)
- INTERSECT (common rows from both queries)
- OUTER UNION (concatenates query outputs)

keyword is one of

- ALL (not suppressing duplicated rows, making one pass only)
- CORR | CORRESPONDING (overlaying columns with common names)
Combining Queries with set operators

query-1
set-operator <keyword>
query-2;  * as a single statement;
where set-operator is one of
  - UNION (all unique rows)
  - EXCEPT (rows from first query only)
  - INTERSECT (common rows from both queries)
  - OUTER UNION (concatenates query outputs)

keyword is one of
  - ALL (not suppressing duplicated rows, making one pass only)
  - CORR | CORRESPONDING (overlaying columns with common names)
Schematic View of Set Operators

- UNION
- EXCEPT
- INTERSECT
- OUTER UNION
Set Symmetric Difference

\[(query_1 \ EXCEPT \ query_2)\]
\[\text{UNION}\]
\[(query_2 \ EXCEPT \ query_1)\];
as a single statement