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

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

Output

<table>
<thead>
<tr>
<th>Number</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site</td>
<td>Subjects</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>01</td>
<td>61</td>
</tr>
<tr>
<td>02</td>
<td>66</td>
</tr>
<tr>
<td>03</td>
<td>46</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</th>
<th>Number Of Patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>State</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Small</td>
<td>IN</td>
</tr>
<tr>
<td>Small</td>
<td>MI</td>
</tr>
<tr>
<td>Medium</td>
<td>IN</td>
</tr>
<tr>
<td>Medium</td>
<td>MI</td>
</tr>
<tr>
<td>Large</td>
<td>IN</td>
</tr>
<tr>
<td>Large</td>
<td>MI</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

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...
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.
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**

```sql
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>y</td>
<td>x y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x z</td>
</tr>
<tr>
<td>x11</td>
<td>y11</td>
<td>x11 y11 x12 z12</td>
</tr>
<tr>
<td>x21</td>
<td>y21</td>
<td>x21 y21 x12 z12</td>
</tr>
<tr>
<td>x31</td>
<td>y31</td>
<td>x31 y31 x12 z12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x11 y11 x22 z22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x21 y21 x22 z22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x31 y31 x22 z22</td>
</tr>
</tbody>
</table>
Syntax of Inner Join

\[
\text{SELECT } * \text{ FROM } \text{table-1, table-2} \\
\text{WHERE } <\text{table-1.}>\text{col-1}=<\text{table-2.}>\text{col-2}; \\
\]

or

\[
\text{SELECT } * \text{ FROM } \text{table-1 INNER JOIN table-2} \\
\text{ON } <\text{table-1.}>\text{col-1}=<\text{table-2.}>\text{col-2}; \\
\]

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

Using Table Alias

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

▶ 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 \text{alias.col} (when identical column name occurs from distinct tables)
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
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

```sql
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

```sql
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.
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
Set Symmetric Difference

\[(\text{query1 EXCEPT query2})\]
\[\text{UNION}\]
\[(\text{query2 EXCEPT query1});\]
as a single statement