INTRODUCTION TO PROC SQL®
Rosalind Gusinow, Systems Seminar Consultants, Inc., Madison, WI

ABSTRACT
PROC SQL is a powerful Base SAS® PROC which combines the functionality of the DATA and PROC Steps into a single procedure. PROC SQL in many cases can be a more efficient alternative to traditional SAS code.

PROC SQL can be used to retrieve, update, and report on information from SAS data sets or other database products. This paper will concentrate on SQL's syntax and how to access information from existing SAS data sets. Some of the topics covered include:

Write SQL code using various styles of the SELECT statement. Dynamically create new variables on the SELECT statement. Use SQL options to control the appearance of reports. Create multiple reports with a single PROC SQL step. Create reports containing percentages using PROC SQL. Use CASE/WHEN clauses for conditionally processing the data. Joining data from two or more data sets (like a MERGE!).

WHY LEARN PROC SQL?
PROC SQL can not only retrieve information without having to learn SAS syntax, but it can often do this with fewer and shorter statements than traditional SAS code. Additionally, on average it uses fewer resources than conventional DATA and PROC steps. This means PROC SQL can be a more efficient alternative to traditional SAS code. Further, the knowledge learned is transferrable to other SQL packages.

AN EXAMPLE OF PROC SQL'S SYNTAX
Every PROC SQL must have at least one SELECT statement. The purpose of the SELECT statement is to name the columns that will appear on the report and the order in which they will appear (similar to a VAR statement on PROC PRINT). The FROM clause names the data set from which the information will be extracted (similar to the SET statement). One advantage of SQL is that new variables can be dynamically created on the SELECT statement, which is a feature we do not normally associate with a SAS Procedure:

PROC SQL;
SELECT STATE, SALES,
(SALES * .05) AS TAX
FROM USSALES;
QUIT;
(see output #1 for results)

THE SELECT STATEMENT'S SYNTAX
The purpose of the SELECT statement is to describe how the report will look. It consists of the SELECT statement and several sub-clauses. The sub-clauses name the input dataset, order (or sort) the data, group (or aggregate) the data, and select rows meeting certain conditions (subsetting):

PROC SQL;
SELECT column(s)
FROM table-name | view-name
ORDER BY column(s)
GROUP BY column(s)
WHERE expression;
QUIT;

A SIMPLE PROC SQL
An '*' on the SELECT statement will select all columns from the designated data set. By default a row will wrap when there is too much information to fit across the page. Also by default, column headings will be separated from the data with a line and no observation number will appear:

PROC SQL;
SELECT *
FROM USSALES;
QUIT;
(see output #1 for results)

LIMITING INFORMATION ON THE SELECT
To specify that only certain variables should appear on the report, the variables are listed and separated on the SELECT statement. The SELECT statement does NOT limit the number of variables read. The NUMBER option will print a column on the report labeled 'ROW' which contains the observation number:

PROC SQL NUMBER;
SELECT STATE, SALES
FROM USSALES;
QUIT;
(see output #2 for results)

CREATING NEW VARIABLES
Variables can be dynamically created in PROC SQL. Dynamically created variables can be given a variable name, label, or neither. If a dynamically created variable is not given a name or a label, it will appear on the report as a column with no column heading. Any of the DATA step functions can be used in an expression to create a new variable except LAG, DIF, and SOUND:

PROC SQL;
SELECT SUBSTR(STORENO,1,3)
LABEL='REGION', SALES,
(SALES * .05) AS TAX, (SALES * .05) * .01
FROM USSALES;
QUIT;
(see output #3 for results)

OPTIONS ON THE PROC SQL STATEMENT
There are several useful options that can be used in the PROC SQL statement to help control the appearance of the report. Note that once coded, these options will apply to all SELECT statements within the PROC SQL step unless a RESET statement is used:

PROC SQL INOBS=100 OUTOBS=9 DOUBLE;
SELECT STORE, (SALES * .05) AS TAX
FROM USSALES;
QUIT;
(see output #4 for results)
THE FLOW OPTION AND USING RESET

The FLOW option allows text to continue in its column rather than wrapping the text on to the next line. If a value is not specified on the FLOW option, SAS will "flow" the value to the length of the column. The RESET statement changes options within the same step without respesifying the procedure. The option FLOW=30 40 floats the width of the column between the values specified to produce a better layout:

```
PROC SQL FLOW=30;
SELECT STATE, STORENAM, COMMENT
FROM USSALES;
RESET FLOW=30 40 DOUBLE NUMBER
OUTOBS=150;
SELECT STATE, STORENAM, COMMENT
FROM USSALES;
QUIT;
```

(see output #5 for results)

Note: multiple SELECT clauses can be coded under a single PROC SQL. Each SELECT clause will generate a separate report.

THE CALCULATED OPTION ON THE SELECT

Starting with Version 6.07, the CALCULATED component refers to a previously calculated variable so recalculation is not necessary. The CALCULATED component must refer to a variable created within the same SELECT statement:

```
PROC SQL INOBS=9;
SELECT STATE, (SALES * .05) AS TAX,
(SALES * .05) * .01 AS REBATE
FROM USSALES;
- or -
SELECT STATE, (SALES * .05) AS TAX,
CALCULATED TAX * .01 AS REBATE
FROM USSALES;
QUIT;
```

(see output #6 for results)

ASSOCIATING LABELS AND FORMATS

SAS-defined or user-defined formats can be used to improve the appearance of the body of a report. By default, variable names appear as column headings on reports. LABELs give the ability to define up to forty characters to appear as column headings on the report. LABELs and FORMATs do NOT change the way in which a value or variable is stored. They are for appearances ONLY. Be sure when providing formats that their values are adequately large, otherwise the values will not appear fully formatted on the report:

```
TITLE 'REPORT OF THE U.S. SALES';
FOOTNOTE 'PREPARED BY THE MARKETING DEPT.';
PROC SQL;
SELECT STATE, SALES
FORMAT=DOLLAR10.2
LABEL='AMOUNT OF SALES',
(SALES * .05) AS TAX
FORMAT=DOLLAR7.2
LABEL='5% TAX'
FROM USSALES;
QUIT;
```

(see output #7 for results)

THE CASE EXPRESSION ON THE SELECT

The CASE Expression allows conditional processing within PROC SQL:

```
PROC SQL;
SELECT STATE, CASE
WHEN SALES<=10000 THEN 'LOW'
WHEN SALES<=15000 THEN 'AVG'
WHEN SALES<=20000 THEN 'HIGH'
ELSE 'VERY HIGH'
END AS SALESCAT
FROM USSALES;
QUIT;
```

(see results #8 for results)

The END is required when using the CASE. Coding the WHEN in descending order of probability will improve efficiency because SAS will stop checking the CASE conditions as soon as it finds the first true value. You do not have to worry about the length of the newly created variable being assigned like you do with the IF/THEN logic.

The CASE Expression can be coded many different ways; perhaps this WHEN clause looks more familiar to you:

```
PROC SQL;
SELECT STATE,
CASE
WHEN 0     <= SALES <= 10000
THEN 'LOW'
WHEN 10001 <= SALES <= 15000
THEN 'AVG'
WHEN 15001 <= SALES <= 20000
THEN 'HIGH'
ELSE 'VERY HIGH'
END AS SALESCAT
FROM USSALES;
QUIT;
```

(output is the same as output #8)

ANOTHER CASE

The CASE statement has much of the same functionality as an IF statement. Here is yet another variation on the CASE expression:

```
PROC SQL;
SELECT STATE, CASE
WHEN SALES > 20000 AND STORENO
IN ('33281','31983') THEN 'CHECKIT'
ELSE 'OKAY'
END AS SALESCAT
FROM USSALES;
QUIT;
```

(see output #9 for results)

ADDITIONAL SELECT STATEMENT CLAUSES

The GROUP BY clause can be used to summarize or aggregate data. Summary functions (also referred to as aggregate functions) are used on the SELECT statement for each of the analysis variables:

```
PROC SQL;
SELECT STATE, SUM(SALES) AS TOTSALES
FROM USSALES
GROUP BY STATE;
QUIT;
```
Other summary functions available are the AVG/MAN, COUNT/FREQ/N, MAX, MIN, NMISS, STD, SUM, and VAR.

REMERGING
Remerging occurs when a summary function is used without a GROUP BY. The result is a grand total shown on every line:

```sql
PROC SQL;
  SELECT STATE, SUM(SALES) AS TOTSALES
  FROM USSALES;
QUIT;
```

(see output #10 for results)

REMERGING FOR TOTALS
Sometimes remerging is good, as in the case when the SELECT statement does not contain any other variables:

```sql
PROC SQL;
  SELECT SUM(SALES) AS TOTSALES
  FROM USSALES;
QUIT;
```

(see output #11 for results)

CALCULATING PERCENTAGE
Remerging can also be used to calculate percentages:

```sql
PROC SQL;
  SELECT STATE, SALES,
       (SALES/SUM(SALES)) AS PCTSALES
  FORMAT=PERCENT7.2
  FROM USSALES;
QUIT;
```

(see output #12 for results)

Check your output carefully when the remerging note appears in your log to determine if the results are what you expect.

SORTING THE DATA IN PROC SQL
The ORDER BY clause will return the data in sorted order: Much like PROC SORT, if the data is already in sorted order, PROC SQL will print a message in the LOG stating the sorting utility was not used. When sorting on an existing column, PROC SQL and PROC SORT are nearly comparable in terms of efficiency. SQL is more efficient when you need to sort on a dynamically created variable:

```sql
PROC SQL;
  SELECT STATE, SALES
  FROM USSALES
ORDER BY STATE, SALES DESC;
QUIT;
```

(see output #14 for results)

SORT ON NEW COLUMN
On the ORDER BY or GROUP BY clauses, columns can be referred to by their name or by their position on the SELECT cause. The option 'ASC' (ascending) on the ORDER BY clause is the default, it does not need to be specified.

```sql
PROC SQL;
  SELECT SUBSTR(STORENO,1,3)
       LABEL='REGION',
       (SALES * .05) AS TAX
  FROM USSALES
GROUP BY STATE, STORE
WHERE TOTSÀLES > 500;
QUIT;
```

(see output #15 for results)

SUBSETTING USING THE WHERE
The WHERE statement will process a subset of data rows before they are read:

```sql
PROC SQL;
  SELECT * 
  FROM USSALES
  WHERE STATE IN ('OH','IN','IL');
```

(see output #16 for results)

INCORRECT WHERE CLAUSE
Be careful of the WHERE clause, it cannot reference a computed variable:

```sql
PROC SQL;
  SELECT STATE, SALES,
       (SALES * .05) AS TAX
  FROM USSALES
WHERE STATE IN ('OH','IL','IN')
  AND (SALES * .05) > 10;
QUIT;
```

(see output #17 for results)

WHERE ON COMPUTED COLUMN
To use computed variables on the WHERE clause they must be recomputed:

```sql
PROC SQL;
  SELECT STATE, SALES,
       (SALES * .05) AS TAX
  FROM USSALES
WHERE STATE IN ('OH','IL','IN')
  AND (SALES * .05) > 10;
QUIT;
```

(see output #18 for results)

SELECTION ON GROUP COLUMN
The WHERE clause cannot be used with the GROUP BY:

```sql
PROC SQL;
  SELECT STATE, STORE,
       SUM(SALES) AS TOTSÀLES
  FROM USSALES
GROUP BY STATE, STORE
WHERE TOTSÀLES > 500;
QUIT;
```
USE HAVING CLAUSE
In order to subset data when grouping is in effect, the HAVING clause must be used:

```
PROC SQL;
  SELECT STATE, STORENO,
       SUM(SALES) AS TOTSALES
  FROM USSALES
  GROUP BY STATE, STORENO
  HAVING SUM(SALES) > 500;
QUIT;
```

HAVING WITHOUT A COMPUTED COLUMN
The HAVING clause is needed even if it is not referring to a computed variable:

```
PROC SQL;
  SELECT STATE,
       SUM(SALES) AS TOTSALES
  FROM USSALES
  GROUP BY STATE
  HAVING STATE IN ('IL', 'WI');
QUIT;
```

CREATING NEW TABLES OR VIEWS
The CREATE statement provides the ability to create a new data set as output in lieu of a report (which is what happens when a SELECT is present without a CREATE statement). The CREATE statement can either build a TABLE (a traditional SAS dataset, like what is built on a SAS DATA statement) or a VIEW (not covered in this paper):

```
PROC SQL;
  CREATE TABLE TESTA AS
    SELECT STATE, SALES
    FROM USSALES
    WHERE STATE IN ('IL', 'OH');
    SELECT * FROM TESTA;
QUIT;
```

JOINING DATASETS USING PROC SQL
A join is used to combine information from multiple files. One advantage of using PROC SQL to join files is that it does not require sorting the datasets prior to joining as is required with a DATA step merge.

A Cartesian Join combines all rows from one file with all rows from another file. This type of join is difficult to perform using traditional SAS code.

```
PROC SQL;
  SELECT *
  FROM DATA1, DATA2;
QUIT;
```

INNER JOIN
A Conventional or Inner Join combines datasets only if an observation is in both datasets. This type of join is similar to a DATA step merge using the IN Data Set Option and IF logic requiring that the observation is on both data sets (IF ON A AND ON B).

```
PROC SQL;
  SELECT *
  FROM DATA1, DATA2;
  WHERE DATA1.VAR1=DATA2.VAR1;
QUIT;
```

JOINING THREE OR MORE TABLES
An Associative Join combines information from three or more tables. Performing this operation using traditional SAS code would require several PROC SORTs and several DATA step merges. The same result can be achieved with one PROC SQL:

```
PROC SQL;
  SELECT B.FNAME, B.LNAME, CLAIMS,
       E.STORENO, STATE
  FROM BENEFITS B, EMPLOYEE E,
       FEBSALES F;
  WHERE B.FNAME=E.FNAME AND
        B.LNAME=E.LNAME AND
        E.STORENO=F.STORENO AND
        CLAIMS > 1000;
QUIT;
```

CHANGES IN VERSION 7/8
1. Some PROC SQL views are now updateable. The view must be based on a single DBMS table or SAS data file and must not contain a join, an ORDER BY clause, or a subquery.
2. Whenever possible, PROC SQL passes joins to the DBMS rather than doing the joins itself. This enhances performance.
3. You can now store DBMS connection information in a view with the USING LIBNAME clause.
4. A new option, DQUOTE=ANSI, enables you to non-SAS names in PROC-SQL.
5. A PROC SQL query can now refer ence up to 32 views or tables. PROC SQL can now perform joins on up to 32 tables.
6. PROC SQL can now create and update tables that contain integrity constraints.

IN SUMMARY
PROC SQL is a powerful data analysis tool. It can perform many of the same operations as found in traditional SAS code, but can often be more efficient because of its dense language structure.

PROC SQL can be an effective tool for joining data, particularly when doing associative, or three-way joins. For more information regarding SQL joins, reference the papers noted in the bibliography.

TRADEMARK NOTICE
SAS and PROC SQL are registered trademarks of the SAS Institute Inc., Cary, NC, USA and other countries. * indicates USA registration.
USEFUL PUBLICATIONS
SAS Institute Inc., Getting Started with the SQL Procedure, Version 6, First Edition


Kolbe Ritzow, Kim, “Joining Data with SQL”, Proceedings of the 6th Annual MidWest SAS® Users Group Conference


CONTACT INFORMATION
Any questions or comments regarding the paper may be directed to the author:

Rosalind K. Gusinow
Systems Seminar Consultant, Inc.
2997 Yarmouth Greenway Drive
Madison, WI 53711
Phone: (608) 278-9964
Fax: (608) 278-0065
Email: rgusinow@sys-seminar.com
OUTPUT #1 (PARTIAL):

STATE   SALES   STORENO

COMMENT
STORENAM

----------------------------------------
WI  10103.23  32331
SALES WERE SLOW BECAUSE OF COMPETITORS SALE
RON'S VALUE RITE STORE

WI  9103.23  32320
SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER
PRICED SMART GROCERS

WI  15032.11  32311
AVERAGE SALES ACTIVITY REPORTED
VALUE CITY

OUTPUT #2 (PARTIAL):

ROWS   STATE   SALES

---------------------
1  WI  10103.23
2  WI  9103.23
3  WI  15032.11

OUTPUT #3 (PARTIAL):

REGION   SALES   TAX

---------------------
323  10103.23  505.1615  5.051615
323  9103.23   455.1615  4.551615
323  15032.11  751.6055  7.516055
332  33209.23  1660.462  16.60461

OUTPUT #4 (PARTIAL):

STATE   TAX

---------
WI  505.1615
WI  455.1615
WI  751.6055
MI  1660.462
**OUTPUT #5 (PARTIAL):**

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENAME</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>RON'S VALUE RITE STORE</td>
<td>SALES WERE SLOW BECAUSE OF COMPETITORS SALE</td>
</tr>
<tr>
<td>WI</td>
<td>PRICED SMART GROCERS</td>
<td>SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER</td>
</tr>
</tbody>
</table>

**Row 1**

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENAME</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>RON'S VALUE RITE STORE</td>
<td>SALES WERE SLOW BECAUSE Of COMPETITORS SALE</td>
</tr>
</tbody>
</table>

**Row 2**

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENAME</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>PRICED SMART GROCERS</td>
<td>SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER</td>
</tr>
</tbody>
</table>

**OUTPUT #6 (PARTIAL):**

<table>
<thead>
<tr>
<th>STATE</th>
<th>TAX</th>
<th>REBATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>505.1615</td>
<td>5.051615</td>
</tr>
<tr>
<td>WI</td>
<td>455.1615</td>
<td>4.551615</td>
</tr>
<tr>
<td>WI</td>
<td>751.6055</td>
<td>7.516055</td>
</tr>
<tr>
<td>MI</td>
<td>1660.462</td>
<td>16.60461</td>
</tr>
</tbody>
</table>

**OUTPUT #7 (PARTIAL):**

**REPORT OF THE U.S. SALES**

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>5% TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>$10,103.23</td>
<td>$505.16</td>
</tr>
<tr>
<td>WI</td>
<td>$9,103.23</td>
<td>$455.16</td>
</tr>
<tr>
<td>WI</td>
<td>$15,092.11</td>
<td>$751.61</td>
</tr>
<tr>
<td>MI</td>
<td>$33,209.23</td>
<td>1660.46</td>
</tr>
</tbody>
</table>

PREPARED BY THE MARKETING DEPT.

**OUTPUT #8 (PARTIAL):**

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALESCLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>AVG</td>
</tr>
<tr>
<td>WI</td>
<td>LOW</td>
</tr>
<tr>
<td>WI</td>
<td>HIGH</td>
</tr>
<tr>
<td>MI</td>
<td>VERY HIGH</td>
</tr>
</tbody>
</table>
OUTPUT #9 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALESCAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>OKAY</td>
</tr>
<tr>
<td>WI</td>
<td>OKAY</td>
</tr>
<tr>
<td>WI</td>
<td>OKAY</td>
</tr>
<tr>
<td>MI</td>
<td>CHECKIT</td>
</tr>
</tbody>
</table>

OUTPUT #10:

<table>
<thead>
<tr>
<th>STATE</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>84976.57</td>
</tr>
<tr>
<td>MI</td>
<td>53941.66</td>
</tr>
<tr>
<td>WI</td>
<td>34238.57</td>
</tr>
</tbody>
</table>

OUTPUT #11 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>172556.8</td>
</tr>
<tr>
<td>WI</td>
<td>172556.8</td>
</tr>
<tr>
<td>WI</td>
<td>172556.8</td>
</tr>
<tr>
<td>MI</td>
<td>172556.8</td>
</tr>
</tbody>
</table>

OUTPUT #12:

<table>
<thead>
<tr>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>172556.8</td>
</tr>
</tbody>
</table>

OUTPUT #13 (PARTIAL):

(log message shown)

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>PCTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>10103.23</td>
<td>5.86%</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>5.28%</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
<td>8.71%</td>
</tr>
<tr>
<td>MI</td>
<td>33209.23</td>
<td>19.2%</td>
</tr>
</tbody>
</table>

NOTE: The query requires remerging summary Statistics back with the original data.

OUTPUT #14 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>32083.22</td>
</tr>
<tr>
<td>IL</td>
<td>22223.12</td>
</tr>
<tr>
<td>IL</td>
<td>20338.12</td>
</tr>
<tr>
<td>IL</td>
<td>10332.11</td>
</tr>
<tr>
<td>MI</td>
<td>33209.23</td>
</tr>
</tbody>
</table>
OUTPUT #15 (PARTIAL):

<table>
<thead>
<tr>
<th>REGION</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>516.6055</td>
</tr>
<tr>
<td>313</td>
<td>1604.161</td>
</tr>
<tr>
<td>313</td>
<td>1111.156</td>
</tr>
<tr>
<td>319</td>
<td>1016.906</td>
</tr>
</tbody>
</table>

OUTPUT #16 (THE RESULTING SAS LOG- PARTIAL):

```sql
27   PROC SQL;
28   SELECT STATE,SALES, (SALES * .05) AS TAX
29   FROM USSALES
30   WHERE STATE IN ('OH','IN','IL') AND TAX > 10;
ERROR: THE FOLLOWING COLUMNS WERE NOT FOUND IN THE
CONTRIBUTING TABLES: TAX.
NOTE: PROC SQL SET OPTION NOEXEC AND WILL CONTINUE TO
CHECK THE SYNTAX OF STATEMENTS.
```

OUTPUT #17 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>10103.23</td>
<td>505.1615</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>455.1615</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
<td>751.6055</td>
</tr>
<tr>
<td>IL</td>
<td>20338.12</td>
<td>1016.906</td>
</tr>
</tbody>
</table>

OUTPUT #18 (THE RESULTING SAS LOG- PARTIAL):

```
167   GROUP BY STATE, STORE
168   WHERE TOTSALES > 500;
......
22
202
ERROR 22.322: Expecting one of the following: (, **, *, /, +, -, !, ||, <, <=, <>, ==, >=, EQ, GE, GT, LE, LT, NE, -=, -=, &), AND, !, OR, |, ', HAVING, ORDER,
The statement is being ignored.

ERROR 202.322: The option or parameter is not recognized.
```
OUTPUT #19 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENO</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>31212</td>
<td>10332.11</td>
</tr>
<tr>
<td>IL</td>
<td>31373</td>
<td>22223.12</td>
</tr>
<tr>
<td>IL</td>
<td>31381</td>
<td>32633.22</td>
</tr>
<tr>
<td>IL</td>
<td>31983</td>
<td>20338.12</td>
</tr>
<tr>
<td>MI</td>
<td>33281</td>
<td>32092.23</td>
</tr>
</tbody>
</table>

OUTPUT #20:

<table>
<thead>
<tr>
<th>STATE</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>84976.57</td>
</tr>
<tr>
<td>WI</td>
<td>34238.57</td>
</tr>
</tbody>
</table>

OUTPUT #21:

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>20338.12</td>
</tr>
<tr>
<td>IL</td>
<td>10332.11</td>
</tr>
<tr>
<td>IL</td>
<td>32633.22</td>
</tr>
<tr>
<td>IL</td>
<td>22223.12</td>
</tr>
</tbody>
</table>

OUTPUT #22:

<table>
<thead>
<tr>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR1</th>
<th>VAR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>10</td>
<td>ABC</td>
<td>20</td>
</tr>
<tr>
<td>ABC</td>
<td>10</td>
<td>JKL</td>
<td>25</td>
</tr>
<tr>
<td>ABC</td>
<td>10</td>
<td>PQR</td>
<td>30</td>
</tr>
<tr>
<td>GHI</td>
<td>15</td>
<td>ABC</td>
<td>20</td>
</tr>
<tr>
<td>GHI</td>
<td>15</td>
<td>JKL</td>
<td>25</td>
</tr>
<tr>
<td>GHI</td>
<td>15</td>
<td>PQR</td>
<td>30</td>
</tr>
<tr>
<td>MNO</td>
<td>20</td>
<td>ABC</td>
<td>20</td>
</tr>
<tr>
<td>MNO</td>
<td>20</td>
<td>JKL</td>
<td>25</td>
</tr>
<tr>
<td>MNO</td>
<td>20</td>
<td>PQR</td>
<td>30</td>
</tr>
<tr>
<td>GHI</td>
<td>15</td>
<td>ABC</td>
<td>20</td>
</tr>
<tr>
<td>GHI</td>
<td>15</td>
<td>JKL</td>
<td>25</td>
</tr>
<tr>
<td>GHI</td>
<td>15</td>
<td>PQR</td>
<td>30</td>
</tr>
<tr>
<td>MNO</td>
<td>20</td>
<td>ABC</td>
<td>20</td>
</tr>
<tr>
<td>MNO</td>
<td>20</td>
<td>JKL</td>
<td>25</td>
</tr>
<tr>
<td>MNO</td>
<td>20</td>
<td>PQR</td>
<td>30</td>
</tr>
</tbody>
</table>

OUTPUT #23:

<table>
<thead>
<tr>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR1</th>
<th>VAR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>10</td>
<td>ABC</td>
<td>20</td>
</tr>
</tbody>
</table>
OUTPUT #24:

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>FNAME</th>
<th>LNAME</th>
<th>STORENO</th>
<th>OBS</th>
<th>FNAME</th>
<th>LNAME</th>
<th>STATE</th>
<th>SALES</th>
<th>STORENO</th>
<th>OBS</th>
<th>FNAME</th>
<th>LNAME</th>
<th>CLAIMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ANN</td>
<td>BECKER</td>
<td>33281</td>
<td>1</td>
<td>ANN</td>
<td>BECKER</td>
<td>MI</td>
<td>31209.23</td>
<td>33281</td>
<td>1</td>
<td>ANN</td>
<td>BECKER</td>
<td>2003</td>
</tr>
<tr>
<td>2</td>
<td>CHRIS</td>
<td>DOBSON</td>
<td>33281</td>
<td>2</td>
<td>CHRIS</td>
<td>DOBSON</td>
<td>MI</td>
<td>15132.43</td>
<td>33312</td>
<td>2</td>
<td>CHRIS</td>
<td>DOBSON</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>EARL</td>
<td>FISHER</td>
<td>33281</td>
<td>3</td>
<td>ALLEN</td>
<td>PARK</td>
<td>IL</td>
<td>25336.12</td>
<td>31983</td>
<td>3</td>
<td>ALLEN</td>
<td>PARK</td>
<td>10392</td>
</tr>
<tr>
<td>4</td>
<td>ALLEN</td>
<td>PARK</td>
<td>31373</td>
<td>4</td>
<td>BETTY</td>
<td>JOHNSON</td>
<td>IL</td>
<td>26223.12</td>
<td>31373</td>
<td>4</td>
<td>BETTY</td>
<td>JOHNSON</td>
<td>3832</td>
</tr>
<tr>
<td>5</td>
<td>BETTY</td>
<td>JOHNSON</td>
<td>31373</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>KAREN</td>
<td>ADAMS</td>
<td>31373</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>CLAIMS</th>
<th>STORENO</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>BECKER</td>
<td>2003</td>
<td>33281</td>
<td>MI</td>
</tr>
<tr>
<td>ALLEN</td>
<td>PARK</td>
<td>10392</td>
<td>31373</td>
<td>IL</td>
</tr>
<tr>
<td>BETTY</td>
<td>JOHNSON</td>
<td>3832</td>
<td>31373</td>
<td>IL</td>
</tr>
</tbody>
</table>