INTRODUCTION TO PROC SQL®

ABSTRACT
PROC SQL is a powerful Base SAS® Procedure that combines the functionality of DATA and PROC steps into a single step. 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 in this brief introduction 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.
- 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, SQL often uses fewer resources than conventional DATA and PROC steps. Further, the knowledge learned is transferable to other SQL packages.

AN EXAMPLE OF PROC SQL SYNTAX
Every PROC SQL query must have at least one SELECT statement. The purpose of the SELECT statement is to describe how the report will look. It consists of the SELECT clause and several sub-clauses. The sub-clauses name the input dataset, select rows meeting certain conditions (subsetting), group (or aggregate) the data, and order (or sort) the data:

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

```sas
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:

```sas
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. Notice the commas separating the columns:

```sas
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:

```sas
PROC SQL OUTOBS=4 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 respecifying 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;
  SELECT STATE, STORENAM, COMMENT
  FROM USSALES;
QUIT;
(see output #5 for results)
```

Note: multiple SELECT statements can be coded under a single PROC SQL. Each SELECT statement 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;
  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)
```

USING LABELS AND FORMATS

SAS-defined or user-defined formats can be used to improve the appearance of the body of a report. LABELs give the ability to define longer column headings:

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

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;
(see output #10 for results)
```

Other summary functions available are the AVG/MEDIAN, COUNT/FREQ/N, MAX, MIN, NMISS, STD, SUM, and VAR. This capability is similar to PROC SUMMARY with a CLASS statement.

REMERGING

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

```
PROC SQL;
  SELECT STATE, SUM(SALES) AS TOTSALES
  FROM USSALES;
QUIT;
(see output #11 for results)
```

REMERGING FOR TOTALS

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

```
PROC SQL;
  SELECT SUM(SALES) AS TOTSALES
  FROM USSALES;
QUIT;
(see output #12 for results)
```
CALCULATING PERCENTAGE
Remerging can also be used to calculate percentages:

PROC SQL;
SELECT STATE, SALES,
(SALES/SUM(SALES)) AS PCTSALES
FORM=PERCENT7.2
FROM USSALES;
QUIT;
(see output #13 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 may be more efficient when you need to sort on a dynamically created variable:

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 clause. The option 'ASC' (ascending) on the ORDER BY clause is the default, it does not need to be specified.

PROC SQL;
SELECT SUBSTR(STORENO,1,3)
LABEL='REGION',
(SALES * .05) AS TAX
FROM USSALES
ORDER BY 1 ASC, TAX DESC;
QUIT;
(see output #15 for results)

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

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

SELECT *
FROM USSALES
WHERE NSTATE IN (10-20,30);

SELECT *
FROM USSALES
WHERE STATE IN
('OH','IN','IL')
AND SALES > 500;
QUIT;
(no output shown for this example)

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

PROC SQL;
SELECT STATE, SALES,
(SALES * .05) AS TAX
FROM USSALES
WHERE STATE IN
('OH','IN','IL')
AND TAX > 10;
QUIT;
(see output #16 for results)

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

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)

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

PROC SQL;
SELECT STATE, STORE,
SUM(SALES) AS TOTSALES
FROM USSALES
GROUP BY STATE, STORE
WHERE TOTSALES > 500;
QUIT;
(see output #18 for results)

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;
(see output #19 for results)

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;
(see output #20 for results)
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):

```sql
PROC SQL;
CREATE TABLE TESTA AS
SELECT STATE, SALES
FROM USSALES
WHERE STATE IN ('IL','OH');
SELECT * FROM TESTA;
QUIT;
(see output #21 for results)
```

The name given on the create statement can either be temporary or permanent. Only one table or view can be created by a CREATE statement. The second SELECT statement (without a CREATE) is used to generate the report.

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.

```sql
PROC SQL;
SELECT *
FROM JANSALES, FEBSALES;
QUIT;
(see output #22 for results)
```

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 ONA AND ONB).

```sql
PROC SQL;
SELECT U.STORENO, U.STATE,
      F.SALES AS FEBSALES
FROM USSALES U, FEBSALES F
WHERE U.STORENO=F.STORENO;
QUIT;
(see output #23 for results)
```

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:

```sql
PROC SQL;
SELECT B.FNAME, B.LNAME, CLAIMS,
      E.STORENO F.STORENO AND
      CLAIMS > 1000;
QUIT;
(see output #24 for dataset list and results)
```

CHANGES IN VERSION 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 reference up to 32 views or tables. PROC SQL can 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.

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:

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

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>STORENO</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORENAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 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):

<table>
<thead>
<tr>
<th>ROW</th>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WI</td>
<td>10103.23</td>
</tr>
<tr>
<td>2</td>
<td>WI</td>
<td>9103.23</td>
</tr>
<tr>
<td>3</td>
<td>WI</td>
<td>15032.11</td>
</tr>
</tbody>
</table>

### OUTPUT #3 (PARTIAL):

<table>
<thead>
<tr>
<th>REGION</th>
<th>SALES</th>
<th>TAX</th>
<th>TAX RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>323</td>
<td>10103.23</td>
<td>505.1615</td>
<td>5.051615</td>
</tr>
<tr>
<td>323</td>
<td>9103.23</td>
<td>455.1615</td>
<td>4.551615</td>
</tr>
<tr>
<td>323</td>
<td>15032.11</td>
<td>751.6055</td>
<td>7.516055</td>
</tr>
<tr>
<td>332</td>
<td>33209.23</td>
<td>1660.462</td>
<td>16.60461</td>
</tr>
</tbody>
</table>

### OUTPUT #4 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>505.1615</td>
</tr>
<tr>
<td>WI</td>
<td>455.1615</td>
</tr>
<tr>
<td>WI</td>
<td>751.6055</td>
</tr>
<tr>
<td>MI</td>
<td>1660.462</td>
</tr>
</tbody>
</table>
### OUTPUT #5 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENAM</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: WI  RON’S VALUE RITE STORE
SALES WERE SLOW BECAUSE OF COMPETITORS SALE

Row 2: WI  PRICED SMART GROCERS
SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER

### 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 AMOUNT OF

<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>$15,032.11</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>SALESCAT</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>SALES</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>53341.66</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):

```sas
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: The SAS System stopped processing this step because of errors.
```

### 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>32083.22</td>
</tr>
<tr>
<td>IL</td>
<td>31983</td>
<td>20338.12</td>
</tr>
<tr>
<td>MI</td>
<td>33281</td>
<td>33209.23</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>32083.22</td>
</tr>
<tr>
<td>IL</td>
<td>22223.12</td>
</tr>
</tbody>
</table>

OUTPUT #22 (PARTIAL):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>STORENO</th>
<th>NUMEMP</th>
<th>STATE</th>
<th>SALES</th>
<th>STORENO</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>32320</td>
<td>10</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>32320</td>
<td>10</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
<td>32311</td>
<td>13</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>MI</td>
<td>33209.23</td>
<td>33281</td>
<td>25</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>MI</td>
<td>20132.43</td>
<td>33312</td>
<td>20</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>IL</td>
<td>20338.12</td>
<td>31983</td>
<td>21</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>IL</td>
<td>10332.11</td>
<td>31212</td>
<td>18</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>IL</td>
<td>32083.22</td>
<td>31381</td>
<td>31</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>IL</td>
<td>22223.12</td>
<td>31373</td>
<td>28</td>
<td>IL</td>
<td>30083.22</td>
<td>31381</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>32320</td>
<td>10</td>
<td>IL</td>
<td>26223.12</td>
<td>31373</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>32320</td>
<td>10</td>
<td>IL</td>
<td>26223.12</td>
<td>31373</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
<td>32311</td>
<td>13</td>
<td>IL</td>
<td>26223.12</td>
<td>31373</td>
</tr>
<tr>
<td>MI</td>
<td>33209.23</td>
<td>33281</td>
<td>25</td>
<td>IL</td>
<td>26223.12</td>
<td>31373</td>
</tr>
</tbody>
</table>

OUTPUT #23 (PARTIAL):

<table>
<thead>
<tr>
<th>STORENO</th>
<th>STATE</th>
<th>FEBSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>32320</td>
<td>WI</td>
<td>9103.23</td>
</tr>
<tr>
<td>32331</td>
<td>WI</td>
<td>8103.23</td>
</tr>
<tr>
<td>32320</td>
<td>WI</td>
<td>10103.23</td>
</tr>
<tr>
<td>32311</td>
<td>WI</td>
<td>13032.11</td>
</tr>
<tr>
<td>33281</td>
<td>MI</td>
<td>31209.23</td>
</tr>
<tr>
<td>33312</td>
<td>MI</td>
<td>15132.43</td>
</tr>
<tr>
<td>31983</td>
<td>IL</td>
<td>25338.12</td>
</tr>
<tr>
<td>31212</td>
<td>IL</td>
<td>8332.11</td>
</tr>
</tbody>
</table>

OUTPUT #24:

Hands-on Workshops
<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>FEBSALES</th>
<th>BENEFITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBS</td>
<td>FNAME</td>
<td>LNAME</td>
</tr>
<tr>
<td>1</td>
<td>ANN</td>
<td>BECKER</td>
</tr>
<tr>
<td>2</td>
<td>CHRIS</td>
<td>DOBSON</td>
</tr>
<tr>
<td>3</td>
<td>EARL</td>
<td>FISHER</td>
</tr>
<tr>
<td>4</td>
<td>ALLEN</td>
<td>PARK</td>
</tr>
<tr>
<td>5</td>
<td>BETTY</td>
<td>JOHNSON</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>I</td>
</tr>
</tbody>
</table>