

Structured Query Language: An Introduction

By Bernard Werner

COPYRIGHT © 2001 Nexgen Software Technologies, Inc. All rights reserved.

How many times have you heard, "The business is changing?" How many times have you dropped your face into your hands in response? As the pace of business continuously speeds up, the rate of developing information systems needs to match it. Programming as we know it must change; there is no option. Using Structured Query Language (SQL) is one such initiative.

SQL was developed by IBM in the mid 1970's, and it is said, was perfected by the likes of Oracle, Sybase, Informix, and others. However, even on the AS/400, it is a valuable and powerful data manipulation tool. SQL is actually a misnomer, because it is more than just a query language. If it were just that, there would be little benefit to it over Query/400.

Unlike Query, SQL can be used to effectively update data as well produce reports. They both use the same low-level query processor available in every AS/400 whether the SQL license is available or not. This means that an AS/400 without the SQL license can still execute stored SQL commands also known as Query Manager Procedures.

SQL on the AS/400 comes in two flavors: SQL/400 and Query Manager/400 (QM/400). It is unfortunate that the name is close enough to Query/400 so as to create confusion. However, they are different and distinct. SQL/400 is a dynamic, interactive version of the language which is executed on the fly. QM/400 is a facility to create, save, and execute stored procedures. They can be run interactively or by means of a CL or other program. In other words, QM/400 can be used instead of RPG to more easily and quickly manipulate data on the AS/400.

Since the mid 1970's, SQL has grown into a more robust and powerful language (read, more complicated). The emphasis here will be on the core group of four data manipulation verbs: SELECT, UPDATE, INSERT, and DELETE. We will also cover two data definition verbs for our examples: CREATE and DROP which have analogues in Data Definition Statements (DDS). Because of space limitations, this article cannot be comprehensive. After introducing the statements in dynamic SQL, we'll build an interactive QM/400 procedure that can be executed from within a CL program.

Dynamic SQL

For the time being, let's get to know what the language can do by using SQL/400 and begin an interactive session with:

```
STRSQL
```

You will be greeted by a screen with a lot of lines on it ready for you to enter an SQL verb. Assuming the PRMS files library is in your library list, try:

```

                                Enter SQL Statements

Type SQL statement, press Enter.
Current connection is to relational database NEXGEN2.
==> select * from mspmp100

                                Bottom

F3=Exit   F4=Prompt   F6=Insert line   F9=Retrieve   F10=Copy line
F12=Cancel F13=Services   F24=More keys

                                (C) COPYRIGHT IBM CORP. 1982, 2000.

```

Before you hit enter, key F4 and you see how the SQL environment can help you build the necessary statements:

```

                                Specify SELECT Statement

Type SELECT statement information. Press F4 for a list.

FROM files . . . . . mspmp100
SELECT fields . . . . . *
WHERE conditions . . . . .
GROUP BY fields . . . . .
HAVING conditions . . . . .
ORDER BY fields . . . . .
FOR UPDATE OF fields . . . . .

                                Bottom

Type choices, press Enter.

DISTINCT records in result file . . . . . N   Y=Yes, N=No
UNION with another SELECT . . . . . N   Y=Yes, N=No
Specify additional options . . . . . N   Y=Yes, N=No

F3=Exit   F4=Prompt   F5=Refresh   F6=Insert line   F9=Specify subquery
F10=Copy line   F12=Cancel   F14=Delete line   F15=Split line   F24=More keys

```

If we concentrate on the upper part of the screen, we see that we are working with the file MSPMP100 (product master). The SELECT fields line contains an asterisk (“*”) which signifies that SQL is to return all fields (also called columns). If the WHERE phrase were filled in, we could specify just those particular records (or rows) that we are interested in. A number of the other phrases will be covered below.

You will note that when the cursor is on the SELECT line and F4 is entered, that a display of field names is retrieved. This can be very useful for specifying which ones to look at. From the above display, when ENTER is hit, the first screen full of data is retrieved. It should be noted that if you are in a multi-company environment, only the first member will be processed. The ability to point to other companies will be covered below as well.

```

Display Data
Data width . . . . . : 10475
Position to line . . . . . Shift to column . . . . .
.....1.....2.....3.....4.....5.....6.....7.....
Active Company Plant Number Product Number Product Description

1      25      015AFD      w/s for CC
1      25      015AFD2     airbag of auto model
1      25      1111111    AIR PUMP FOR BICYCLE TIRES
1      25      2222222    AIR PUMP FOR BICYCLE TIRES
1      25      20722500   AIR PUMP FOR BICYCLE TIRES
1      25      20781000   AIR PUMP FOR BICYCLE TIRES
1      25      010AFD4    AIR PUMP FOR BICYCLE TIRES
1      25      BB         AIR PUMP FOR BICYCLE TIRES
1      25      CC         AIR PUMP FOR BICYCLE TIRES
1      25      DD         AIR PUMP FOR BICYCLE TIRES
1      25      AA1        AIR PUMP FOR BICYCLE TIRES
1      25      AA2        AIR PUMP FOR BICYCLE TIRES
1      25      AIRPUMP10  AIR PUMP FOR BICYCLE TIRES
1      25      ALUMINUM   Aluminum METAL
More...
F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split

```

The user can shift right and left by using F20 and F19. A particular line can be positioned to by specifying a number or "B" for bottom or "T" for the top. Your screen may not look exactly like this.

I suggest to those working on RPG programs to use SQL to scope out the data one is working with. Too often a programmer goes to create a program not knowing what the file looks like and consequently creates something useless or not able to handle all the required situations. For example, should there be a consideration to handle combined (MSFLG='C') items? Or other codes? Entering the following gives the analyst a window into the data's behavior:

```

SELECT msflg, count(*)
FROM mspmp100
GROUP BY msflg

```

This statement will return all the demand types in MSFLG, counting them and grouping each by demand type. "Count(*)" is known as a group function:

```

Display Data
Data width . . . . . : 24
Position to line . . . . . Shift to column . . . . .
.....1.....2.....
Dmd Type      COUNT ( * )
      C              195
      D              321
      I              178
***** End of data *****
Bottom
F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split

```

The "GROUP BY" phrase essentially identifies on which field to break when there is a change. An internal sort is implicit when using grouping. If the analyst did not do this, the blank demand type with 8 rows may not have been noticed and an erroneous program could have resulted.

Other group functions are SUM, AVG, MAX, and MIN. If "GROUP BY" is not used, it refers to the entire file (or table) or the set of rows as specified by the "WHERE" phrase. For example, this statement will return the average standard cost of all dependent demand items:

```
SELECT avg(stdc1)
FROM mspmp100
WHERE msflg='D'
```

Simply replacing the AVG group function with MAX will return the dependent demand item with the greatest standard cost:

```
SELECT max(stdc1)
FROM mspmp100
WHERE msflg='D'
```

In preparation for updating data, let's create a table (file) called BLANK_FLAG. Notice that when creating a table, one should specify library name. Otherwise it is created in QGPL. It has 2 columns, the product and standard cost; character of length 15 and decimal with 15 positions and 2 to the right of the decimal point:

```
CREATE table werner/blank_flag
(prdno char (15), stdc1 dec (15,2))
```

It is populated by using the INSERT statement. Pulling data from another table (MSPMP100) is done using the SELECT statement:

```
INSERT into blank_flag (prdno, stdc1)
SELECT prdno, stdc1
FROM mspmp100
WHERE msflg=' '
```

The table BLANK_FLAG now contains 8 rows with the products and standard costs from where the demand type is blank. Note the columns from MSPMP100 were specified in the same order as they exist in BLANK_FLAG, eliminating the need to specify how the data was to flow from one to the other. The variables following BLANK_FLAG are therefore not required in this example.

To look at this data meaningfully requires data from 2 tables. We can see the product and description by performing a JOIN:

```
SELECT a.prdno, b.descp,a.stdc1
FROM blank_flag a,mspmp100 b
WHERE a.prdno=b.prdno
```

This statement contains a number of new concepts. First is the use of what is called a correlation name, the "a" and "b" following the table names. This is necessary because both tables contain the same column names of PRDNO and STDC1. One must tell SQL from which table to pull the data. It also allows the 2 tables to be synchronized by means of the "WHERE a.prdno=b.prdno". Only the 8 rows that exist in both tables will be retrieved. The product and standard cost are from BLANK_FLAG (A.PRDNO and A.STDC1) and the description is from MSPMP100 (B.DESCP):

Display Data		Data width :	70
Position to line		Shift to column	
.....1.....+.....2.....+.....3.....+.....4.....+.....5.....+.....6.....+.....7			
PRDNO	Product Description		STDC1
BIKES	Bicycle Planner Item		.00
BIKES01	Bicycle Planner Item		.00
FRAMECOLOR	Frame Color Feature		18.37
SEATS	feature		.00
SOUNDSYSTEM	feature		.00
TRANSMISSION	feature		.00
DECALFEATURE	Decal Feature		.19
GEARFEATURE	Gear Feature		15.27
*****	End of data	*****	

Bottom

F3=Exit F12=Cancel F19=Left F20=Right F21=Split

Previously, we saw how INSERT can work with a SELECT. Here's another, simpler form for adding another row consisting of an AIRPUMP10 with a cost of \$100:

```
INSERT into blank_flag
VALUES('AIRPUMP10', 100)
```

Assuming we miskeyed the \$100 and it should have been \$10, we can change the cost specifically for AIRPUMP10:

```
UPDATE blank_flag
SET stdc1 = 10
WHERE prdno='AIRPUMP10'
```

Note that now when the standard cost is pulled from both of the tables, there is a difference for AIRPUMP10 in the standard costs. BLANK_FLAG contains the \$10 standard cost in A.STDC1 while the product master has a \$0 cost in B.STDC1:

```
SELECT a.prdno,b.descp,a.stdc1,b.stdc1
FROM blank_flag a,mspmp100 b
WHERE a.prdno=b.prdno
```

Using the split screen to get all the pertinent data together:

```

                                Display Data
                                Data width . . . . . :      87
Position to line . . . . .      Shift to column . . . . .
.....1.....2.....3.....4... | +.....6.....7.....8.....+..
PRDNO          Product Description      STDC1   Total Std Cost
BIKES          Bicycle Planner Item      .00      .000000
BIKES01        Bicycle Planner Item      .00      .000000
FRAMECOLOR     Frame Color Feature      18.37    18.373685
SEATS          feature                   .00      .000000
SOUNDSYSTEM    feature                   .00      .000000
TRANSMISSION   feature                   .00      .000000
DECALFEATURE   Decal Feature              .19      .190000
GEARFEATURE    Gear Feature              15.27    15.272000
AIRPUMP10      AIR PUMP FOR BICYCLE TIRES      10.00    .000000
***** End of data *****

                                Bottom

F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=No split
Last column of data.
    
```

If we later realize that we no longer need AIRPUMP10, we can remove that specific row from BLANK_FLAG:

```

DELETE FROM blank_flag
WHERE prdno='AIRPUMP10'
    
```

Let's say now we only want to work with those products that have a zero standard cost. This means deleting everything greater than 0:

```

DELETE from blank_flag
WHERE stdc1>0
    
```

Using the following statement to show what the table currently contains:

```

SELECT *
FROM blank_flag
    
```

This now leaves:

```

                                Display Data
                                Data width . . . . . :      38
Position to line . . . . .      Shift to column . . . . .
.....1.....2.....3.....+...
PRDNO          STDC1
BIKES          .00
BIKES01        .00
SEATS          .00
SOUNDSYSTEM    .00
TRANSMISSION   .00
***** End of data *****

                                Bottom

F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split
    
```

Introducing a number of concepts, one can easily build more complex statements. Here we're only dealing with one table, MSPMP100, but we want the third column to be current inventory. PRMS does not keep a running balance, just the 4 buckets which we must calculate together. For the temporary

purpose of this “report” the resulting column is called “Inventory”. The WHERE clause is used to make the resulting list smaller and more manageable, only returning rows that have something other than zero in the opening balance and issues columns:

```
SELECT prdno, descp, OPBAL - ISSUE + RECPT + ADJST Inventory
FROM mspmp100
WHERE opbal<>0 and issue<>0
```

Display Data		
Position to line	Data width	: 71
.....1.....2.....3.....4.....5.....6.....7.	Shift to column	
Product Number	Product Description	INVENTORY
BMX100	Standard Bicycle	87,390.500
CALIPER	Brake Caliper	4,956.000-
HANDLE	Brake Handle	4,958.000-
PAD	Brake Pad	9,430.000-
CAKE	cake with frosting and nuts yy	4,756.000
DW1	DW1, buy	36.000-
DW2	DW2, make	1.000
ALS	als product	2.000
*****	End of data	*****
		Bottom
F3=Exit	F12=Cancel	F19=Left
		F20=Right
		F21=Split

There are tools called Scalar Functions which are extremely useful. Some are also available in a somewhat different form in Query/400. The most common are SUBSTR (substring), || (catenate, shift - \), and STRIP (remove unwanted characters). The following takes the product number and tacks it onto the first 20 characters of the description. Unwanted trailing spaces are removed from the product number to save space and a colon is inserted for readability. The result is temporarily called “Identification” just for this report:

```
SELECT strip(prdno,T,' ') || ':' || substr(descp, 1,20) Identification ,
OPBAL - ISSUE + RECPT + ADJST Inventory
FROM mspmp100
WHERE opbal<>0 and issue<>0
```

Compare this screen to the preceding one above:

Display Data	
Position to line	Data width : 60
.....1.....2.....3.....4.....5.....6	Shift to column
IDENTIFICATION	INVENTORY
BMX100:Standard Bicycle	87,390.500
CALIPER:Brake Caliper	4,956.000-
HANDLE:Brake Handle	4,958.000-
PAD:Brake Pad	9,430.000-
CAKE:cake with frosting a	4,756.000
DW1:DW1, buy	36.000-
DW2:DW2, make	1.000
ALS:als product	2.000
***** End of data *****	
Bottom	
F3=Exit	F12=Cancel
F19=Left	F20=Right
	F21=Split

In the area of date manipulation are the functions YEAR, MONTH, DATE, and reserved words CURRENT DATE, DAYS, MONTHS, and YEARS. Here, the system maintained last activity date in MSPMP100, PMADT is printed using the system default format. It is then deconstructed using the YEAR, MONTH, and DAY functions into CCYMMDD format. DAYS, MONTHS, and YEARS is used to perform date arithmetic as in adding 3 days to a given date. Today's date is printed in the fourth column, followed by the difference between today and the last activity date:

```

SELECT prdno, pmadt,
       year(pmadt) * 10000 + month(pmadt) * 100 + day(pmadt) CCYMMDD,
       current date,
       current date - pmadt Months_Days_Ago
FROM mspmp100
WHERE opbal<>0 and issue<>0
    
```

Note that the temporary field MONTHS_DAYS_AGO shows the number of months and days in YYMMDD format. In this case, only months and days are shown:

Display Data				
Position to line	Data width : 72			
.....1.....2.....3.....4.....5.....6.....7..	Shift to column			
Product Number	Date	CCYMMDD	CURRENT DATE	MONTHS_DAYS_AGO
BMX100	04/24/01	20,010,424	04/26/01	2
CALIPER	03/06/01	20,010,306	04/26/01	120
HANDLE	03/06/01	20,010,306	04/26/01	120
PAD	03/06/01	20,010,306	04/26/01	120
CAKE	04/16/01	20,010,416	04/26/01	10
DW1	04/02/01	20,010,402	04/26/01	24
DW2	03/20/01	20,010,320	04/26/01	106
ALS	04/12/01	20,010,412	04/26/01	14
***** End of data *****				
Bottom				
F3=Exit	F12=Cancel	F19=Left	F20=Right	F21=Split

Often times, when joining 2 or more tables, it is necessary to have the rows from the first or primary table represented in the end result even though there is no corresponding data in the others. The mechanism to do this is called an Outer Join. In SQL/400 it is specifically a Left Outer Join in that the table on the left is the primary one. This is the same as using option 2 to join tables in Query/400 for those familiar with that product. This can be demonstrated by adding an invalid part to the table BLANK_FLAG:

```
INSERT into blank_flag
VALUES('WRONG PART', 0)
```

It now looks like this:

```

                                Display Data
                                Data width . . . . . :      38
Position to line . . . . .      Shift to column . . . . .
.....+.....1.....+.....2.....+.....3.....+....
PRDNO                               STDC1
BIKES                               .00
BIKES01                             .00
WRONG PART                          .00
SEATS                               .00
SOUNDSYSTEM                         .00
TRANSMISSION                        .00
***** End of data *****

F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split      Bottom
```

When we use the outer join, it is easy to spot the invalid part with the statement:

```
SELECT a.prdno,b.descp
FROM blank_flag a left outer join mspmp100 b on a.prdno=b.prdno
ORDER BY 1
```

The FROM phrase contains the tables needed to perform the join as well as the joining variables. The ORDER BY sorts the output by the first column so it appears in alphabetical order. The missing data is denoted visually by a dash:

```

                                Display Data
                                Data width . . . . . :      47
Position to line . . . . .      Shift to column . . . . .
.....+.....1.....+.....2.....+.....3.....+.....4.....+..
PRDNO          Product Description
BIKES          Bicycle Planner Item
BIKES01        Bicycle Planner Item
SEATS          feature
SOUNDSYSTEM    feature
TRANSMISSION   feature
WRONG PART     -
***** End of data *****

F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split      Bottom
```

The offending row can also be specifically selected with the below statement by searching for a null value:

```
SELECT a.prdno,b.descp
FROM blank_flag a left outer join mspmp100 b on a.prdno=b.prdno
WHERE b.descp is null
```

Note that a null value is not a zero or blank, but an undefined data value.

When we're done with BLANK_FLAG, it is removed from the system by the DROP TABLE statement:

```

                                Enter SQL Statements

Type SQL statement, press Enter.
> select * from blank_flag
SELECT statement run complete.
===> drop table blank_flag

F3=Exit    F4=Prompt    F6=Insert line    F9=Retrieve    F10=Copy line    Bottom
F12=Cancel    F13=Services    F24=More keys

```

Query Manager

Query manager provides a facility for storing SQL statements so they may be executed at a later time and repetitively. This is useful for producing daily, weekly, or quarterly reports. In a non-reporting, data manipulation application, QM/400 has been used successfully in an end of year routine to take data from an active transaction file containing production data and move the old to a historical file. To work with this powerful tool also requires some knowledge of Command Language (CL).

If you have trouble entering statements other than SELECT, check with your security officer. QM/400 has a separate security area to control who has access to what kinds of statements. This is in addition to the normal object level security you have on your system which controls what a user can do to a file.

Prior to beginning a QM session, it may be necessary to perform file overrides so that you are working with the proper company data. At a command line, key:

```
OVRDBF FILE(MSPMP100) MBR(m025) OVRSCOPE(*JOB)
```

It may be necessary to substitute a different company member in place of M025. This can also be done prior to entering Dynamic SQL to apply to that session. In either event, it is important to specify the override scope of *JOB for it to apply. As always, if something is done frequently, it can be incorporated into a CL program or command.

There are 2 major phases of QM/400. The first is the interactive definition of QM procedures and the second is executing them. Execution is performed with the command STRQMQRV as shown later. The definition phase is entered by keying:

```
STRQM
```

The following screen will come up:

```

                                DB2 Query Manager for AS/400
                                System:  NEXGEN2

Select one of the following:

    1. Work with Query Manager queries
    2. Work with Query Manager report forms
    3. Work with Query Manager tables

    10. Work with Query Manager profiles

Selection

F3=Exit  F12=Cancel  F22=QM Statement
(C) COPYRIGHT IBM CORP. 1982, 2000.

```

Option 1 is where we will spend most of our time. Report forms are usually accessed through option 1, however it is beyond the scope of this article. Option 3 allows changing data in tables subject to system security. This also will not be covered here.

Menu item 10 is for setting the environment to make it easier to use and to suit your working style. The author works with the following settings changed from the default which are highlighted:

```

                                Change Query Manager Profile

User profile . . . . . :  WERNER
Description . . . . . :  Bernard Werner

Type choices, press Enter.
Default library for QM objects . . . .  WERNER      Name, *CURLIB
Default object creation authority . . .  *LIBCRTAUT    *CHANGE, *ALL, *USE
                                           *EXCLUDE, *LIBCRTAUT
                                           Authorization list
Run query mode . . . . . 1                          1=Interactive
                                           2=Batch
Display run options . . . . . Y                      Y=Yes, N=No
Confirmation messages in QM . . . . . Y              Y=Yes, N=No
Naming convention . . . . . *SYS                     *SYS, *SAA
RDB Connection method . . . . . *RUW                *RUW, *DUW
Default library for QM tables . . . . . WERNER      Name, *NONE, *CURLIB
Query data output . . . . . 1                        1=Display, 2=Printer
                                           3=File
                                           More...

F3=Exit    F4=Prompt  F5=Refresh  F10=Display additional details
F12=Cancel F22=QM Statement

```

It is suggested that QM procedures and tables be stored in your personal library as a work area. Only if they are used in production should they be in a production library. And on the next screen the default creation mode is changed:

```

Change Query Manager Profile

User profile . . . . . : WERNER
Description . . . . . : Bernard Werner

Type choices, press Enter.
Printer to use for output . . . . . *JOB      Name, *JOB
                                           F4 for list
File to use for output . . . . . *NONE     Name, *NONE
                                           F4 for list
Library . . . . .                      Name, *LIBL, *CURLIB
Job description for batch run . . . . *USRPRF Name, *USRPRF
                                           F4 for list
Library . . . . .                      Name, *LIBL, *CURLIB
Commitment control level . . . . . 1       1=None, 2=Change
                                           3=Cursor stability
                                           4=All
                                           5=Repeatable Read
Default query creation mode . . . . . 1     1=SQL, 2=Prompted
                                           More...

F3=Exit      F4=Prompt  F5=Refresh  F10=Display additional details
F12=Cancel   F22=QM Statement

```

The default query creation mode controls whether you key SQL statements under your control (SQL) or go through a prompted session more like Query/400. Note that it is possible to specify a custom job description if you want to run QM procedures in a different job queue or with a different library list than the one specified in the default job description.

In SQL mode, when option 1 is entered to work with query manager queries, we are met with a screen that looks like this:

```

Work with Query Manager Queries

Library . . . . . WERNER      Name, F4 for list
Query creation mode . . . . . : SQL

Type options, press Enter.
1=Create  2=Change  3=Copy  4=Delete  5=Display  6=Print  7=Rename
9=Run     10=Convert to SQL

Opt Query      Type      Description

(Cannot find object to match specified name.)

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F11=Display query only  F12=Cancel
F16=Repeat position to  F17=Position to  F24=More keys

```

The library that shows in the top center is the one specified for QM object creation under option 10 as is the default creation mode.

To create a QM procedure, key in a 1 followed by a valid system name. This is an alphanumeric up to 10 characters, beginning with an alphabetic character. Hit enter and the display invites keying:


```

                                Work with Query Manager Queries

Library . . . . . WERNER      Name, F4 for list
Query creation mode . . . . : SQL

Type options, press Enter.
  1=Create   2=Change   3=Copy   4=Delete   5=Display   6=Print   7=Rename
  9=Run      10=Convert to SQL

Opt  Query      Type      Description
    MS0001Q
    MS0001Q      SQL      Select items of a specific demand type

                                                                Bottom
F3=Exit   F4=Prompt   F5=Refresh   F11=Display query only   F12=Cancel
F16=Repeat position to   F17=Position to   F24=More keys

```

From this screen, the procedure can be modified with option 2, copied to a different name or library with option 3, deleted with option 4, executed with option 9, and so on. Here, option 2 is used to change the procedure. A technique not available in dynamic SQL is used to turn it into an interactive procedure:

```

                                Edit Query
Columns . . . . : 1 70      Query . . . . : MS0001Q
QM . .

Type SQL Statement
***** Beginning of Data *****
0001.00 SELECT prdno,msflg,descp,utmes
0002.00 FROM mspmp100
0003.00 WHERE msflg = &Demand_Type
***** End of Data *****

F2=Alternate keys   F3=Exit   F4=Prompt   F5=Run report   F6=Run sample
F9=Retrieve         F15=Check syntax   F24=More keys
(C) COPYRIGHT IBM CORP.

```

The blank in single quotes is replaced by the ampersand (shift - 7) followed by up to 30 characters which are used as a prompt when executed. From this screen, the procedure can be run with an F5. In the following screen, the blank is entered by surrounding it with single quotes:

```

Display Program Messages

Job 403978/WERNER/QPADEV0008 started on 05/17/01 at 21:48:22 in subsystem QI
Type a value for variable "Demand_Type" and press Enter.

Type reply, press Enter.
Reply . . .

F3=Exit   F12=Cancel

```

Note that using a prompting variable like the above but for numeric input would not require quotes. After keying the variable and hitting ENTER results in a screen that requests a number of items as to how to run the procedure:

```

                                Run Query

Query . . . . . : *
Library . . . . . :

Type choices, press Enter.
Run query mode . . . . . 1          1=Interactive
                                   2=Batch
Run sample only . . . . . N        Y=Yes, N=No
Form . . . . . *SYSDFT           Name, *, *SYSDFT, F4 for list
Library . . . . .                Name, *CURLIB, *LIBL
Output . . . . . 1                 1=Display, 2=Printer
                                   3=File

                                Bottom

F3=Exit  F4=Prompt  F12=Cancel

```

We'll run this interactively, outputting to the display using the system default format for the data by just hitting ENTER. Had we created a form, it could be specified here to print a report in precisely the desired format. Instead, the system's default for the report form will suffice:

```

                                Display Report
Query . . . . . : WERNER/MS0001Q      Width . . . . . : 142
Form . . . . . : *SYSDFT              Column . . . . . : 1
Control . . . . .

Line  . . . . .1 . . . . .2 . . . . .3 . . . . .4 . . . . .5 . . . . .6 . . . . .7.
      Product Number  Dmd Type  Product Description          Unit of Me
-----
000001  BIKES          Bicycle Planner Item          EA
000002  BIKES01          Bicycle Planner Item          EA
000003  FRAMECOLOR        Frame Color Feature           EA
000004  SEATS             feature                        EA
000005  SOUNDSYSTEM        feature                        EA
000006  TRANSMISSION        feature                        EA
000007  DECALFEATURE        Decal Feature                 EA
000008  GEARFEATURE        Gear Feature                   EA
*****  * * * * *  E N D  O F  D A T A  * * * * *

                                Bottom

F3=Exit  F12=Cancel  F19=Left  F20=Right  F21=Split

```

Keying F3 to leave the display brings up the exit dialogue:

```

                                Display Report
Query . . . . .:  WERNER/MS0001Q                               Width . . . .:  142
Form .....:
Contr :                               Exit                               :
Line  :                               :                               : ..7.
      :  Type choice, press Enter.   :                               : f Me
      :                               :                               : ----
00000 :  Option . . . . .  2          1=Exit saving active data       :
00000 :                               2=Exit without saving active data :
00000 :                               3=Resume displayed report        :
00000 :                               :                               :
00000 :  F12=Cancel                    :                               :
00000 :                               :                               :
00000 : .....:                               :
000008  GEARFEATURE                Gear Feature                      EA
*****  * * * * *  E N D  O F  D A T A  * * * * *

                                Bottom

F3=Exit   F12=Cancel   F19=Left   F20=Right   F21=Split

```

Exiting is done with option 1 or 2. The difference at this stage is unimportant. We can now take our knowledge and construct an interactive CL program to print this report:

```

Columns . . . :  1  71                Edit                               WERNER/QCLSRC
SEU==>
FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
***** Beginning of data *****
0001.00          OVRDBF          FILE(MSPMP100) MBR(M025)
0002.00          OVRPRTF        FILE(QPQXPRTF) PAGESIZE(*N 132)
0003.00          STRQMQR        QMQR(MS0001Q) OUTPUT(*PRINT)
***** End of data *****

F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor   F11=Toggle
F16=Repeat find   F17=Repeat change   F24=More keys

(C) COPYRIGHT IBM CORP. 1981, 1999.

```

The OVRDBF is unnecessary if you are working with a single member implementation of PRMS or if you are wanting to access the first member. QM reports are 80 columns by default and the OVRPRTF statement forces the report to 132 columns if that is needed. The STRQMQR statement executes the procedure "MS0001Q", directing output to the job's printer, again using the system's default report form. The program is executed from a command line. Only the screen requesting the demand type needs to be entered. No other interaction is required.

Ordinarily, the report would print immediately. On the Nexgen system, all output is held so that when one goes to look at the report using WRKSPLF, this is the result:

Display Spooled File			
File	QPQXPRTF	Page/Line	1/3
Control		Columns	1 - 78
Find			
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...			
Product Number	Dmd Type	Product Description	Unit of Measure
-----	-----	-----	-----
BIKES		Bicycle Planner Item	EA
BIKES01		Bicycle Planner Item	EA
FRAMECOLOR		Frame Color Feature	EA
SEATS		feature	EA
SOUNDSYSTEM		feature	EA
TRANSMISSION		feature	EA
DECALFEATURE		Decal Feature	EA
GEARFEATURE		Gear Feature	EA
05/17/01 23:21:24			
F3=Exit F12=Cancel F19=Left F20=Right F24=More keys			Bottom

Additional reports for the other demand types could be created by rerunning MS0001C and entering other alpha codes in single quotes (e.g., 'C' for combined or 'D' for dependent).

Summary

In short order, this article has presented a number of basic SQL statements. When saved in a Query Manager/400 procedure used with CL, it is possible to quickly put together simple, yet useful procedures which can create a report or manipulate data. >end<