11/23/2023

SQL Command

SQL Query Manager for IBM i

Tim Tognazzini
TOGNAZZINI@HOTMAIL.COM

SQL Command Documentation

This document covers the capabilities and options for the SQL command.

Table of Contents

Introduction

The SQL command was originally created as a way to save SQL statements so they could be referenced and run again. The goal was to give the queries a name and descripiton that could be searched for so they could easily be found and selected. Since that initial program was written additional options have been added as needed. The command eventually grew into a query management system that can be used as a query manager, a command line tool and an ad hoc querying tool.

Additional output options have been added to allow the results of a query to be displayed on the screen, printed to a SPLF file, copied to a PDF file, emailed or saved to the IFS as an Excel file, XML file formatted for Excel, CSV File or JSON file. The output can also be saved as a physical file on the IBMi server.

The following is a brief walkthrough of the 3 ways to use the command. More complete documentation will be added in their own sections for each of these options.

Query Manager

Entering SQL on a command line and pressing enter will display a screen with a list of queries saved for the current user. The user can be changed in the upper left corner of the screen to work with queries for another user. Users do not need to be set up on the server. Any name can be used as a user to group queries together for a specific purpose. The user option can be cleared out to see all queries for all users.

The list can be filtered using the search line at the top of the screen. Multiple words can be entered, and the list will be filtered to only entries that include each word. The search uses every column in the list, so it can be searched for a partial match in both the name and the description at the same time.

The option column in front of each entry can be used to edit the entry, run the query or perform several other options. The complete list of options will be covered in the Query Managment section of this document.

Queries can include variables that can be overridden at runtime. They can also have defaults setup for output options as well as have some column formatting added to them.

Command Line Tool

The SQL command can be used as a command line tool. This allows queries to be run in CL programs or in RPG programs using the #$CMD function or the QCMD API.

This is accomplished using the optional parameters for the tool. The first parameter is multi use. It can be the name of a saved query, an SQL statement or a file name. The file name option is only used when running an ad hoc query so it will be covered in the ad hoc section of this document.

The command allows the output from either a saved query or an SQL statement to be printed, viewed, email, saved in the IFS or saved as a physical file on the IBMi server. There are also options to view or print the definition of a saved query.

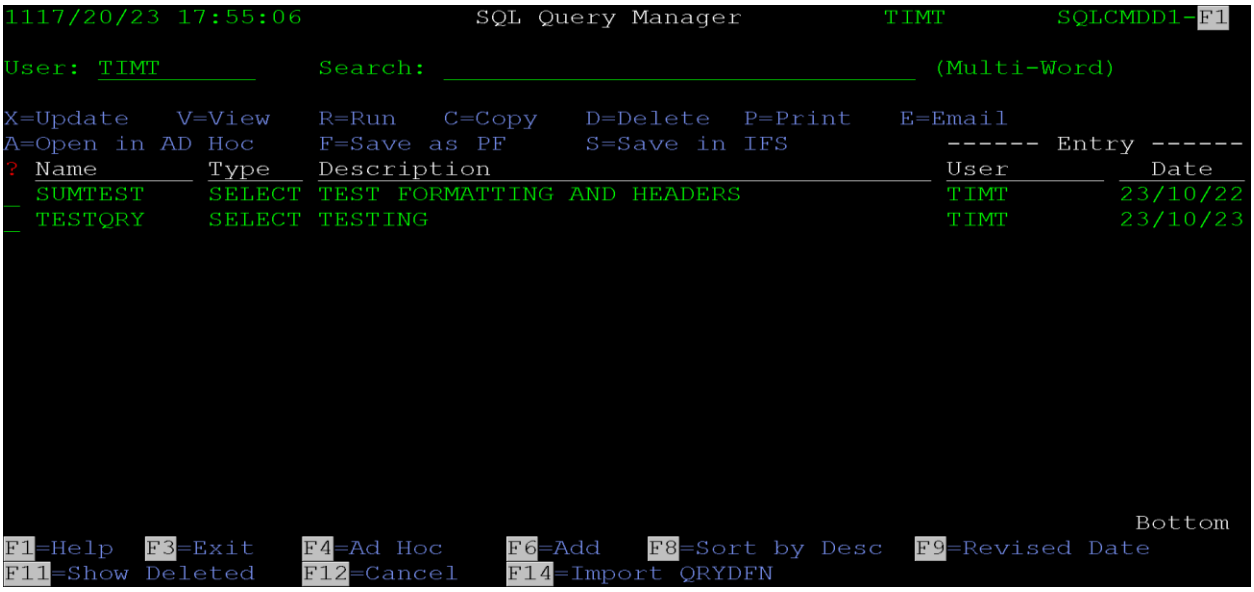
Ad hoc Querying Tool

Ad hoc queries can be run using F4 from the first screen of the program, by passing a filename as the first parameter to the command or by copying a saved query to an ad hoc query using option A in the query list.

The quickest way to start an ad hoc query is by passing a filename as the first parameter of the SQL command. This brings up the ad hoc query entry screen and sets up a basic default query for the file. For example, if `SQL CUSTS` is entered on a command line, the ad hoc query screen will be displayed and a default query of `SELECT * FROM CUSTS` will be entered automatically. The ad hoc screen allows the user to prompt for field names so they will not have to know the columns in the table.

Query Management

The primary method to use the SQL command as a query management tool is to just enter SQL on a command line and press enter. This displays the SQL Query Manager screen. The following is a screen shot of that screen.



This screen lists all queries setup for the user that is signed on. The user field in the upper left-hand corner can be cleared to list queries for all users or can be changed to another user to work with queries saved under their name.

A new query can be added using F6. A query can also be imported from a Query/400 query using F14.

The first column on the screen allows the option listed at the top to be performed for a query in the list. These options are listed below. The initial program was set up to work with alphanumeric characters, but most options work with the standard IBM numeric selection options as well. The numeric equivalent options are listed next to each option in parenthesis.

- **X = Update (2)** - Displays the maintenance screen for a query. This will be covered in the next section.
- **V = View (5)** - displays the queries definition.
- **R = Run (1)** - Runs the query and displays the result on the screen
- **C = Copy (3)** - copies the query to a new name and/or user
- **D = Delete (4)** - Flags the query as deleted. They can still be displayed using F11.
- **P = Print (6)** - Prints the query to a spooled file. A screen is displayed first so parameters can be entered.
- **E = Email (8)** - emails the results of the query, an email screen is displayed first.
- **A = Open in Ad Hoc (9)** - Copies the SQL statement into the ad hoc query display. This is useful for changing the query without updating the save version.
- **F = Save as PF** - Saves the results of the query as a physical file on the server. A screen is displayed first so the parameters can be entered.

- **S = Save in IFS** - Saves the results of the query to a file in the IFS. A screen will be displayed first to enter the desired parameters.
- **I = Reinstate (7)** - This option is only displayed if F11 is used to show deleted queries. It allows a deleted query to be reinstated.

Additional program options are available via function keys. The following list describes what each function key does.

- **F1 = Help** - Displays the help text for the field the cursor is positioned to. This only works if the help text is loaded on the system.
- **F3 = Exit** - Exits the program.
- **F4 = Ad Hoc** - Opens the Ad Hoc query screen. See the Section for Ad Hoc Query Tool for more information.
- **F6 = Add** - Creates a new query. See the section of this document for Adding/Maintaining a Query for more information.
- **F8 = Sort by Desc** - Toggles the sort between sorting by the query description or the date.
- **F9 = Revised/Entry Date** - Toggles the date and user column between displaying the entry or last revised date or user.
- **F11 = Show/Hide Deleted** - Toggles the screen between displaying or hiding deleted queries. Deleted queries are displayed in red if shown. Also if deleted queries are displayed, the reinstate option is also displayed to reinstate them.
- **F12 = Cancel** - Goes to the previous screen. From this screen exits the program.
- **F14 = Import QRYDFN** - Displays the import query screen. See the Import Query/400 Definition section of this document for more information.

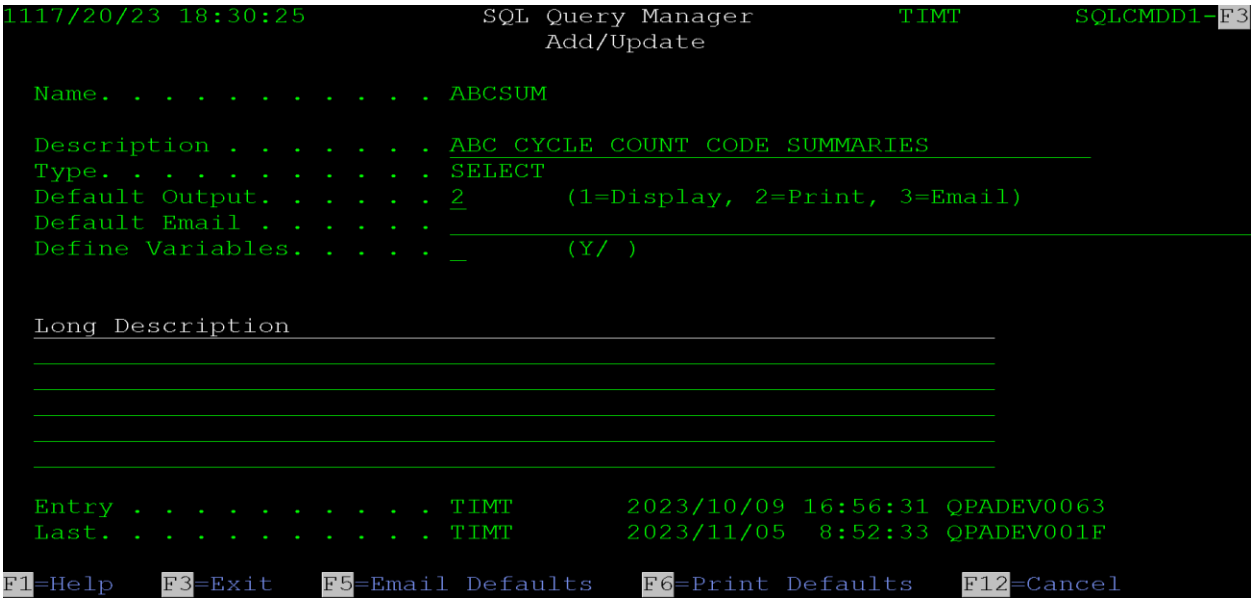
Adding/Maintaining a Query

This section covers both creating and maintaining a query. The only difference between add and update modes is that in add mode the query name can be modified.

Queries can be added using F6 from the main SQL Query Manager Screen. Queries can be updated using option X (2) in front of the query in the list on the SQL Query Manager Screen. Queries can also be added using the save option on the Ad Hoc Query Screen.

Add/update Screen

The first screen that is displayed is the Add/Update screen. The screen looks like this:



When adding a new query, the query name will allow input, and nothing will be filled in. The remainder of this section will show screens for updating an existing query so data will already be populated.

Fields on this screen:

Name - This field is the name the query will be created for or the name of the query being updated. The query user is displayed to the right of the page header. In entry mode, this is the user that the query will be created for.

Description - This field allows maintenance to the description of the query. This is the description that is displayed in the list on the main SQL Query Manager Screen. A good description should be entered to describe the query. This description is normally the primary means of searching for the query if it needs to be found again.

Type - This field is always SELECT. The initial thought of the program was to allow other query types such as update queries to be added, but that functionality has not been created yet.

Default Option - This field is used to enter the default output option of the query. The options are 1=Display, 2=Print and 3=Email. These options are mainly for informational purposes. If the SQL command is used to run the query as a command line tool, the action to perform must be specified.

Options to run the query from the SQL Manger screens are alway output specific, such as run, print or email.

Default Email - This field allows a default email address to be set up for the query. The default email address is used if running the query via the command line tool and specifying *DFT for the email address.

Define Variables - A Y can be entered here to define variables for the query. This causes the program to display the variable definition screen. More information on variables will be included on the variable screen, the SQL Statement Screen and the Command Line Tool sections of this document.

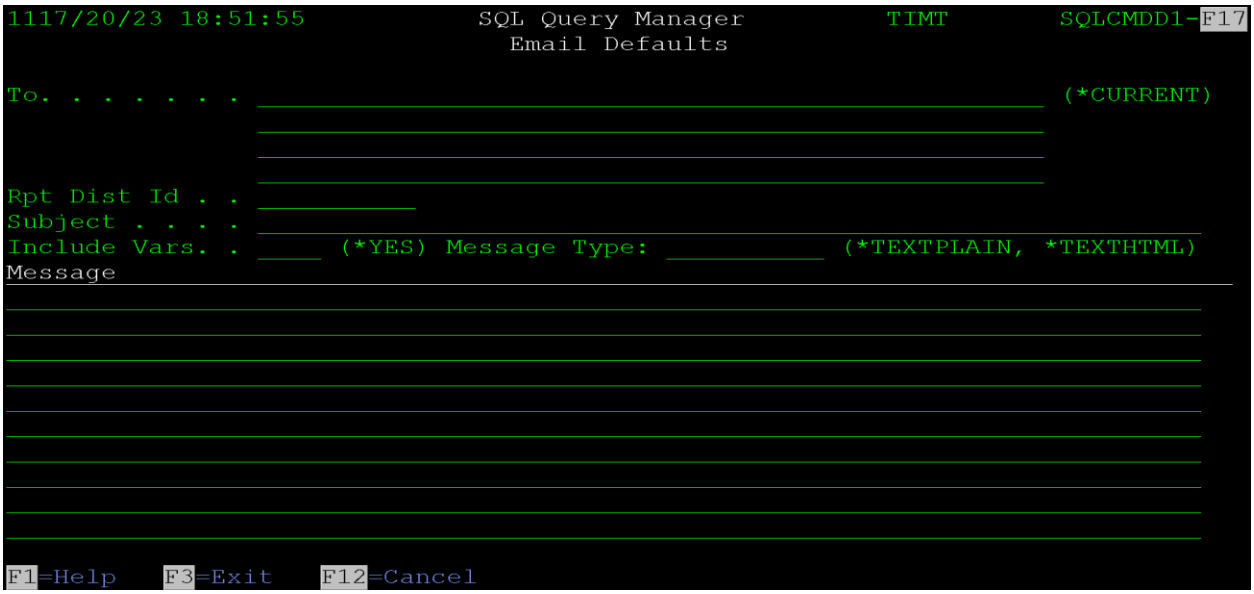
Long Description - These lines allow a long description to be entered for the query. This description is included on the query definition screen or print out. These lines should be used to define what the query is used for or how it works.

This screen includes the following function keys:

- **F1 = Help** - Displays the help text if it is loaded on the system.
- **F3 = Exit** = Exits the Add or Update screens, returns to the main SQL Query Manager list screen.
- **F5 = Email Defaults** - Displays a screen to enter default email options for the query. See the Default Email Options Screen section of this document for details.
- **F6 = Print Defaults** - Displays a screen to enter default print options for the query. See the Default Print Options Screen section of this document for more details
- **F12 = Cancel** - Returns to the previous screen.

Default Email Options Screen

This screen allows entry of the default email options for a saved query.



The screen has these fields:

To - These lines allow for up to 4 default email addresses to be default for the query. These addresses will be used if the query is called from the command line interface and the email addresses are set to default.

Rpt Dist Id - Report Distribution Id. This field is used to specify a default report distribution ID to email the query results to. A report distribution id is part of the Advanced Job Scheduler licensed program from IBM. If that software is not loaded and licensed on the IBM server, this option will not do anything. If entered this option can e used when running the query via the command line tool and specifying *DFT for the report distribution id parameter on the command.

Subject - This field allows a default subject to be entered for the query.

Include Variables - This field states whether the variable values should be included with the email. They will be appended to the email body if this is et to *YES. When emailing the query this can be overridden.

Message Type - This field defaults the type off message sent. This is dependant on the email software used to send the email. The SQL command does send emails itself, instead it must be configured to use third party email software. This option works when using Gumbo products to send the email. If the system is configured to use RJS this option will be ignored, RJS only uses HTML for the body of the email.

If HTML is used, then HTML tags can be included in the message body of the email. For example, “<big>Title</big>

First line” will display like this:

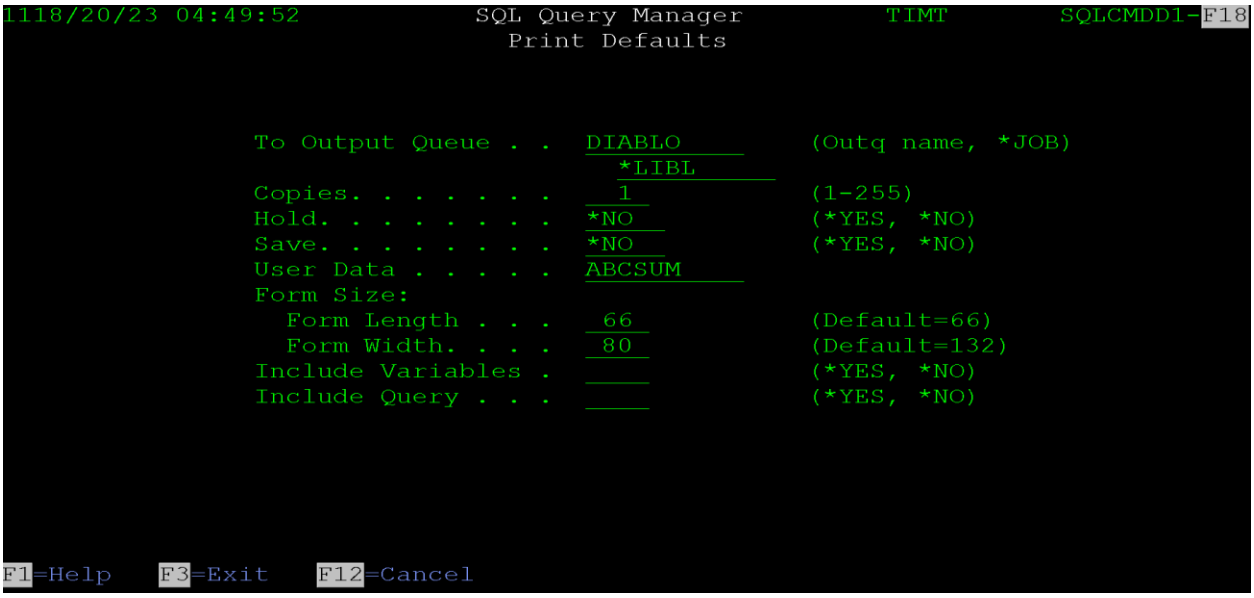
“**Title**

First Line”

Message - This is the default message that will be included as the body of the email.

Default Print Options Screen

This screen allows entry of default print options for the query. It is accessed using F6 from the Add/Update screen.



These print options value default when printing the query via the command line tool or on the screen when printing via the query manager screen options.

Output Queue and Library - These fields allow for entry of the output queue and library. Special values of *LIBL and *CURLIB are allowed. The program will verify that the entered output queue exists.

Copies - The number of copies to print.

Hold – The hold option used on the spooled file.

Save - The save option used on the spooled file.

User Data - The user data used on the spooled file.

Form Size, Length and Width - These override the page length and width used on the spooled file. Any data that does not fit in the width will be truncated.

Include Variables - Entering *YES here will cause the printout to include any variable setup on the query and their overridden values.

Include Query - Entering *YES here will cause the printout to include the SQL statement being run.

Variables Screen

This screen allows entry of user overridable variables. These variables can be used in the query and overridden when running the query via the command line tool. This allows a prompt screen to be written to allow end users to enter selection criteria for the query.

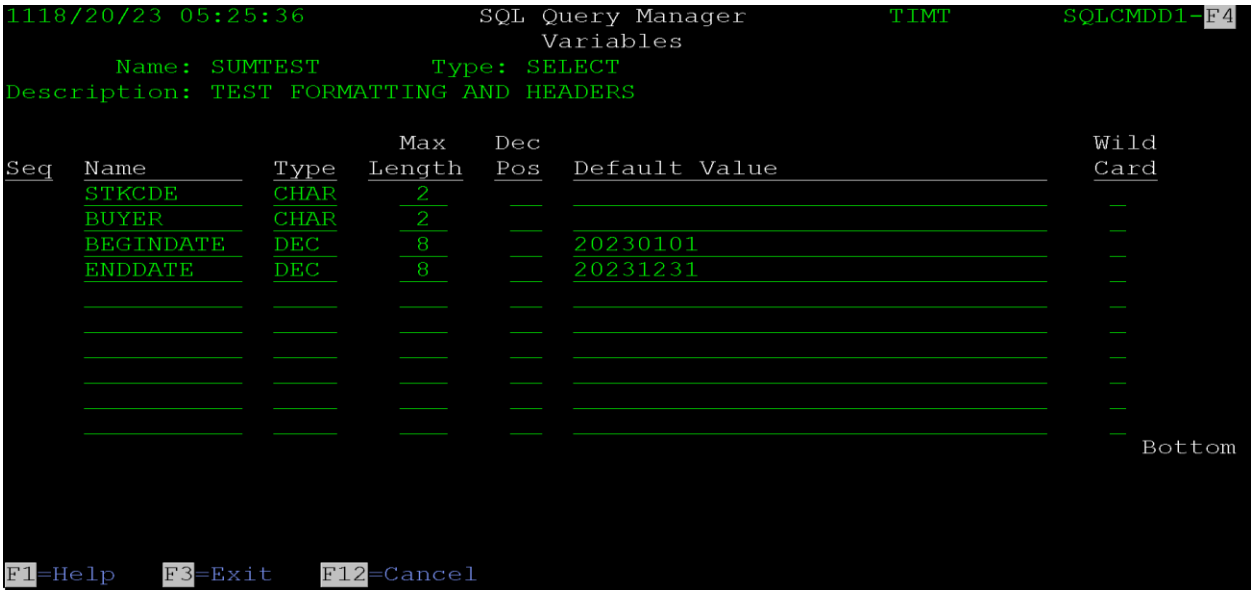
A common example of the use of variables is a date range. The example screen below shows begin and end date variables. In the query these variables are represented by preceding the name with a colon. It would look something like this in the query:

```
where datefield between :begindate and :endate
```

Then when running the query, the dates can be overridden with their correct values, it would look like this:

```
SQL QUERYNAME *PRINT VAROVR((BEGINDATE 20230101) (ENDDATE 20231231))
```

***Suggestion** - Use of variables has mostly been deprecated. It remains in the command for backwards compatibility. If building a prompt screen for a query it is preferred to build the query in the prompt program and pass the full query to the SQL command. This allows for better dynamic selection options. For example, if optional selection criteria is not entered it can just not be included in the SQL statement at all.



Each query allows up to 10 user defined variables to be defined. Each variable requires a type, length and possibly decimal positions. They also should have a default value in case the query is run without the variables being overridden.

Name - This is the variable name used when entering the query. In the query any reference to this name must be preceded by a colon.

Type - This designates whether the variable is numeric or character. The options are CHAR for character or DEC for decimal. Character values are wrapped in apostrophes.

Max Length - This is the maximum length for the variable. It is used to validate that an entered value meets the variables requirements.

Dec Pos - This is the number of decimal positions allowed for the variable. It is only used for DEC type variables.

Default Value - This is the default value for the variable. It is normally only used for testing. For running the query in production, it would be assumed that this value would be overridden.

Wild Card - The wild card option is used if the variable is used in a like statement and should only match the first part of a field. Entering a Y in this column causes the program to concatenate a % sign on the end of the value. Variables flagged this way should only be used in the a like clause in the SQL statement.

Example if a variable is defined as USERNAME CHAR 30 Wild Card = Y, it can be used in a query like this:

```
Select *
from USERS
where user_name like ':USERNAME'
```

When the command is run the name can be passed like this:

```
SQL QUERYNAME *PRINT VAROVR((USERNAME 'TIM'))
```

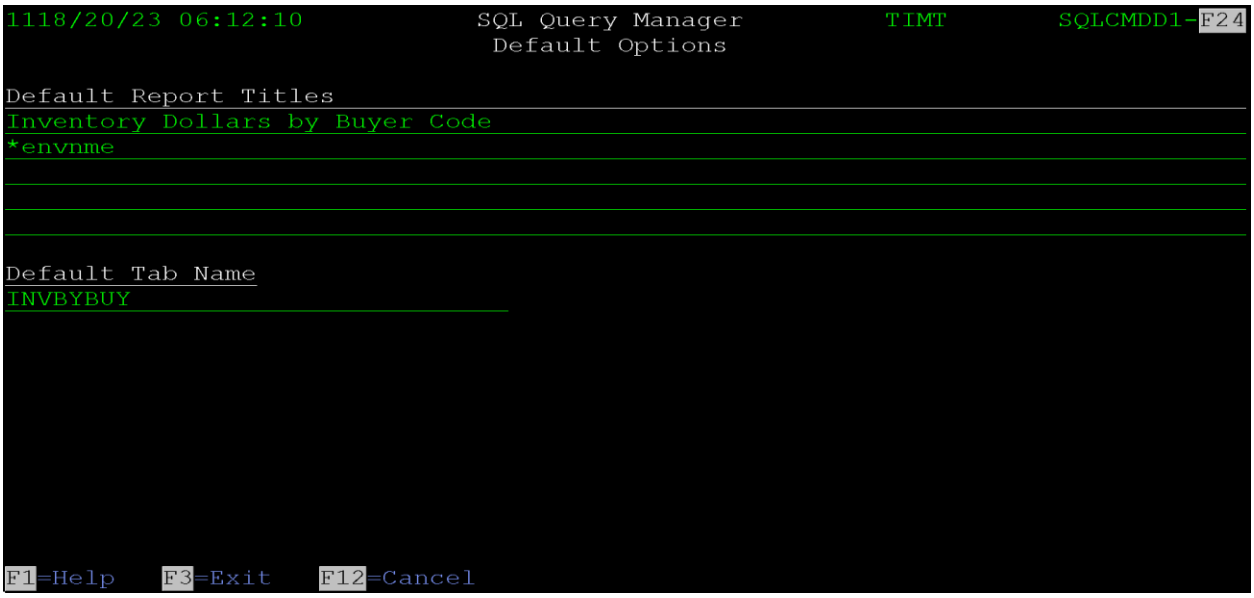
This will cause the query to be run like this:

```
Select *
from USERS
where user_name like 'TIM%'
```

Which will return all users with a name starting with TIM.

Default Options Screen

This screen allows adding default options. These allow the user to specify default report titles and tab name. The screen looks like this:



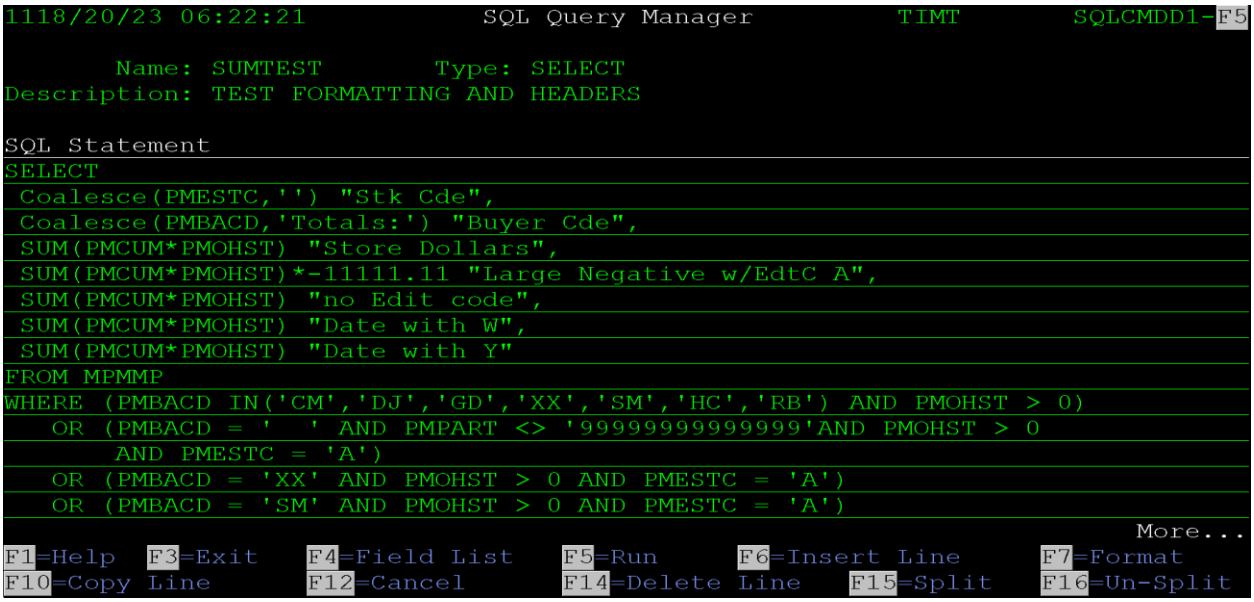
Default Report Titles - These report titles print at the top of a printed or PDF report and are included before the data in an Excel, CSV, XML or JSON file. The values default in when saving the report via one of the Query Manager output screens and in the SQL command line tool. They can be overridden in either place.

***ENVNME** - This special value pulls an environment name from a special message file. The message file must exist in the library list and be named ENVIRONMNT. The message ID pulled is SSI0000. The intent of this special value is to denote which environment the data is from. This is useful when one system has multiple data libraries. For instance, if the system has a test data library, the environment should specify that the data is test data. Without this the output could get confused with live data and misrepresent itself.

Default Tab Name - this option allows the tab name to be defaulted. The tab name is used primarily in the Excel and XML output options. It is considered good practice to make the tab name either, the query name, for a saved query or the program name for a query run from a program. This assists the I/T department in debugging any issues with the data.

SQL Statement Screen

This screen allows entry of an SQL statement. The screen looks like this:



The screen consists of a bunch of lines that the sql statement can be entered into. Page up and down can be used to roll through the lines. The following are some tips for creating useful SQL reports.

Column Data Formatting

- If the field is a numeric date use an SQL UDF to format the date in a correct format for the end use. For instance, if the report will be consumed by a human, an 8 digit numeric date should be formatted as YYYY/MM/DD instead of just the numeric value. However, if the output is going to a physical file, the format should just be a number.
- Numeric data can be overridden on the next screen for printed reports and PDF output. For Excel, XML and CSV output the data should not be formatted because the application that displays the data will format it in an acceptable way.
- If the output is going to a JSON file, the data should not be formatted at all. Adding in dashes, commas or other human readable characters can invalidate JSON data.

Include Correct Report Column Aliases

- Column aliases will be used for column headers in printed output, Excel, CSV and XML output. In JSON output the aliases will be used for the JSON keys. If the output is going to a physical file, then the aliases are used for the field names.
- Column aliases are entered in SQL after the column selection field. Optionally the AS keyword can be used between the selection field and the alias name.
- Include Human Readable Headings if the report is going to be output for human use. This includes reports that are printed, emailed or archived to the IFS in PDF, Excel, or XML format.
- If the report is intended to be used to generate JSON output, use acceptable JSON keys for the column aliases. For example, if the column is a customer account use 'customer_account'

instead of 'Customer Account'. JSON keys are generally lowercase, and words are separated with underscores.

- Don’t Include human readable headings for a query that is intended to be used to create a physical file. Continuing the last example, for a customer account column you might want to use an alias like CUSNBR. This way the field name generated in the DB file/table will be more acceptable to be used in a program. Field names in files are used differently in different programing shops so their exact use will be dictated by the company's programming standards. It is best to avoid column aliases like "Customer Account" because when querying the output file, the column names would have to wrapped in double quotes and correctly match the capitalization.

This screen includes many function keys to assist in entering an SQL statement in a productive manner. These options include things like inserting a line in the middle of a statement, duplicating a line, and prompting for fields. These keys are detailed below. These keys match most of the options in the ad hoc query screen as well, so becoming familiar with them will greatly assist in being productive with this program. Many of the key’s duplicate functions in IBM’s STRSQL command.

F1 = Help - Displays the help text for the screen if it is loaded on the system.

F3 = Exit - Exits updating this query, goes back to the main SQL Query Manager list screen.

F4 = Field List - This option displays a list of fields from all fields in the file. One or more field can be selected to be inserted into the query at the location of the cursor. The field list screen is displayed. Special formatting occurs if the previous word is Select, Where or a join clause. See the Field Selection Screen section of this document for more information. Learning these special cases can make entering SQL statements much more efficient.

F5 = Run - This runs the query to display the output on the screen. it is often very useful when working with a query to see its output. However, it can take a while to run if querying a large file with no selection criteria. One quick solution to this is to add LIMIT 10 to the end of the query while working on it. This will display the output but stop running the query after the first ten lines are found. If used, please ensure that it is removed prior to saving the query. This screen displays the Display Query Output screen. See the Display Query Output section of this document for information on that screen.

F6 = Insert Line - F6 inserts a new line after the line the cursor is one.

F7 = Format - This function is used to format the entire SQL statement. It can be useful when you start entering a query, for instance if F4 is used to select a bunch of fields at once, they get inserted with many fields on one line. F7 will break each field to its own line. However, the format option can do weird things with lists and things in parenthesis. It assumes a comma should start a new line and things in parentheses would be split out. This can wreak havoc on a well formatted SQL statement. If F7 is used and the result is unacceptable, F12 can be pressed to go back a screen and not save the results. However, if any other option is used or the enter key is pressed the modified query will be saved and it will not be possible to recover the previous format of the statement.

F10 = Copy Line - F10 creates a duplicate line of the line the cursor is on. The new line goes directly after the existing line.

***Tip** - Since a line cannot be added before the first line in the list, F10 can be used to duplicate the first line down and then the first line can be replaced with a new value.

F12 = Cancel - This goes back to the previous screen. Any data entered since a function key or the Enter key was last pressed will be lost.

F14 = Delete Line - This deletes the line the cursor is on.

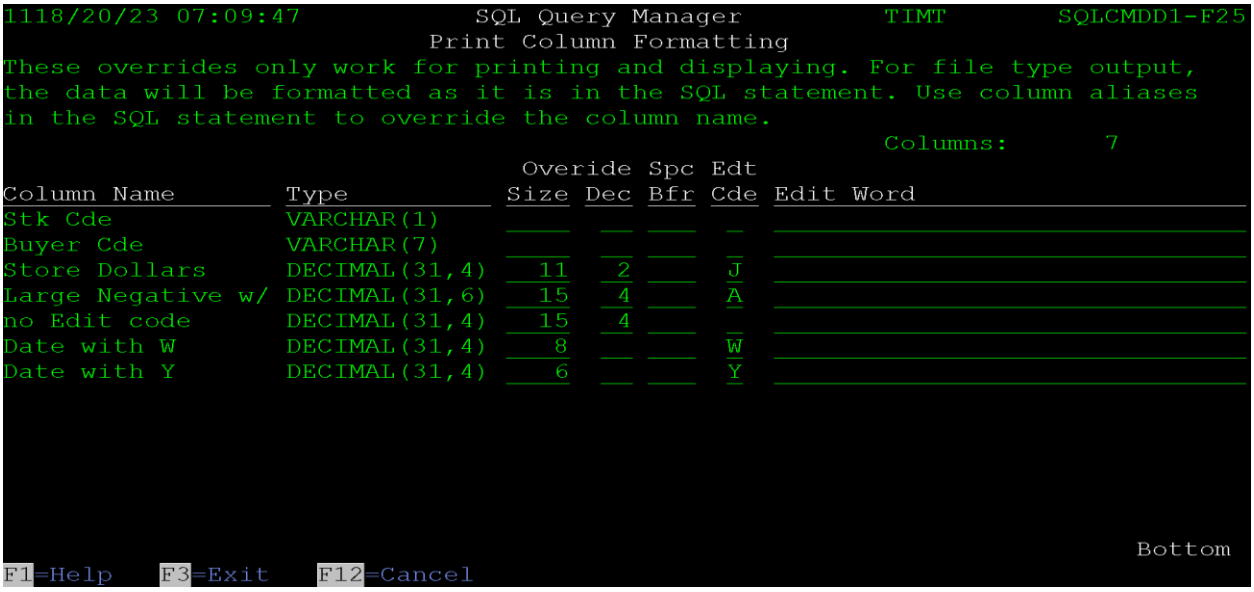
F15 = Split Line - This splits an existing line to a new line. The split is done at the location the cursor is on. This can be useful if a line is getting long.

F16 - Un-Split - This combines two lines together. The data must fit on one line. This can also be useful if F7 was used, and a few lines need to be cleaned up.

Print Column Formatting Screen

This screen allows some formatting of printed output from the query. This is especially useful when working with summary queries, as SQL generally calculates a very large column size for total fields.

The screen looks like this:



It includes each column from the query. Column data is saved by the column name. This means if a new column is added, it will not affect any of the existing column overrides. However, it also means that if a column name is changed, the overrides will be lost and must be re-entered.

Override Size and Decimal positions - These two fields are used to override the largest size and number of decimal positions to be included in the printed or PDF output. These are normally used to make printed output fit on one page. If the value is greater than the field size override, the field will be filled with all nines up to the allowed size. If there are more decimal positions, the data will be rounded to the number of overridden decimal positions.

The example screen included above is of a summary query. It includes extremely large field sizes that the database calculated for the totals. This is a test query, so the values are a little extreme, but it demonstrates the need and usefulness of this feature.

Spc Bfr - Space Before - This column allows the space between columns to be overridden. This is only used in the display, print and PDF output options for the query. If left blank the system will default 0 spaces to the first column and 2 spaces to any column after that. That default is normally acceptable on most queries. Making the space 1 between columns can assist in getting the output to fit on a standard page width in the printed or PDF output.

Edt Cde - Edit Code - This column allows standard IBM edit codes to be applied to the output for displayed, printed and PDF output. User defined edit codes are not allowed.

Edit Word - This was added when edit codes were added. It is currently not functional.

Viewing a Query Definition

This screen displays the information for a saved query definition. It can be accessed via option V = View (5) from the main SQL Query Manager screen.

The following screen show all options entered on a test query. After the screen shots, each section will be discussed in detail.

```
1118/20/23 07:24:38           SQL Query Manager          TIMT       SQLCMDD3-F1
                              Display Definition

User:            TIMT
Name:            SUMTEST
Type:            SELECT
Description:     TEST FORMATTING AND HEADERS
Created:         TIMT 2023/10/22  9:34:48 QPADEV001H
Last Revised:    TIMT 2023/11/18  5:00:20 QPADEV006X
Default Output: DISPLAY

Default Titles
Title 1: Inventory Dollars by Buyer Code
Title 2: *envnme

Long Description
SOME LONG DESCRIPTION THAT DEFINES WHAT THIS QUERY DOES.
THIS QUERY WAS ADDED TO TEST ALL OPTIONS OF THE SQL COMMAND.
IT HAS EVERY OPTION POPULATED SO THAT EVERYTHING CAN BE
TESTED. IT IS ALSO A SUMMARY QUERY WITH COLUMN OVERIDES SO
BOTH THOSEOPTIONS CAN BE TESTED.
                                                                       More...
F1=Help    F3=Exit    F12=Cancel
```

```
1118/20/23 07:24:38           SQL Query Manager          TIMT       SQLCMDD3-F1
                              Display Definition


SQL Statement
SELECT
 Coalesce(PMESTC,'') "Stk Cde",
 Coalesce(PMBACD,'Totals:') "Buyer Cde",
 SUM(PMCUM*PMOHST) "Store Dollars",
 SUM(PMCUM*PMOHST)*-11111.11 "Large Negative w/EdtC A",
 SUM(PMCUM*PMOHST) "no Edit code",
 SUM(PMCUM*PMOHST) "Date with W",
 SUM(PMCUM*PMOHST) "Date with Y"
FROM MPMMP
WHERE (PMBACD IN('CM','DJ','GD','XX','SM','HC','RB') AND PMOHST > 0)
   OR (PMBACD = '  ' AND PMPART <> '999999999999999'AND PMOHST > 0
       AND PMESTC = 'A')
   OR (PMBACD = 'XX' AND PMOHST > 0 AND PMESTC = 'A')
   OR (PMBACD = 'SM' AND PMOHST > 0 AND PMESTC = 'A')
   OR (PMBACD = 'HC' AND PMOHST > 0 AND PMESTC = 'A')
group by grouping sets ( (PMESTC, PMBACD), () )
                                                                       More...
F1=Help    F3=Exit    F12=Cancel
```

```
1118/20/23 07:24:38          SQL Query Manager          TIMT        SQLCMDD3-F1
                             Display Definition

ORDER BY PMESTC, PMBACD

Variable     Type   Length   Dec   Wildcards   Default Value
STKCDE       CHAR      2       0
BUYER        CHAR      2       0
BEGINDATE    DEC       8       0               20230101
ENDDATE      DEC       8       0               20231231

Column Information
Column Count:    7
  Seq  Name                              Type        Len   Dec   EdtC
    1  Stk Cde                           VARCHAR       1
    2  Buyer Cde                         VARCHAR       7
    3  Store Dollars                     DECIMAL      31     4
    4  Large Negative w/EdtC A           DECIMAL      31     6
    5  no Edit code                      DECIMAL      31     4
    6  Date with W                       DECIMAL      31     4
    7  Date with Y                       DECIMAL      31     4
                                                                    More...
F1=Help    F3=Exit    F12=Cancel
```

```
1118/20/23 07:24:38          SQL Query Manager          TIMT        SQLCMDD3-F1
                             Display Definition


Email Defaults
To Email Address:  TOGNAZZINI@HOTMAIL.COM

Print Defaults
Output Queue:      HOLD
OutQ Library:      *LIBL
Copies:            2
Hold Output:       *YES
Save Output:       *YES
User Data:         INVBYBUY
Form Length:       66
Form Width:        80
Include Variable:  *YES
Include Query:     *YES




                                                                    Bottom
F1=Help    F3=Exit    F12=Cancel
```

The first section of the screen lists basic details about the query. The next section lists any defaulted titles if populated. This section will be omitted if there are no title defaults.

The long description is listed next. It will be omitted if there is no long description.

The next section lists the SQL statement for the query. Then any declared variables are listed. This section will be omitted if the query does not have any variables defined.

That is followed by the queries columns details. Then the default email address is listed if populated. And finally, the print overrides are listed if entered.

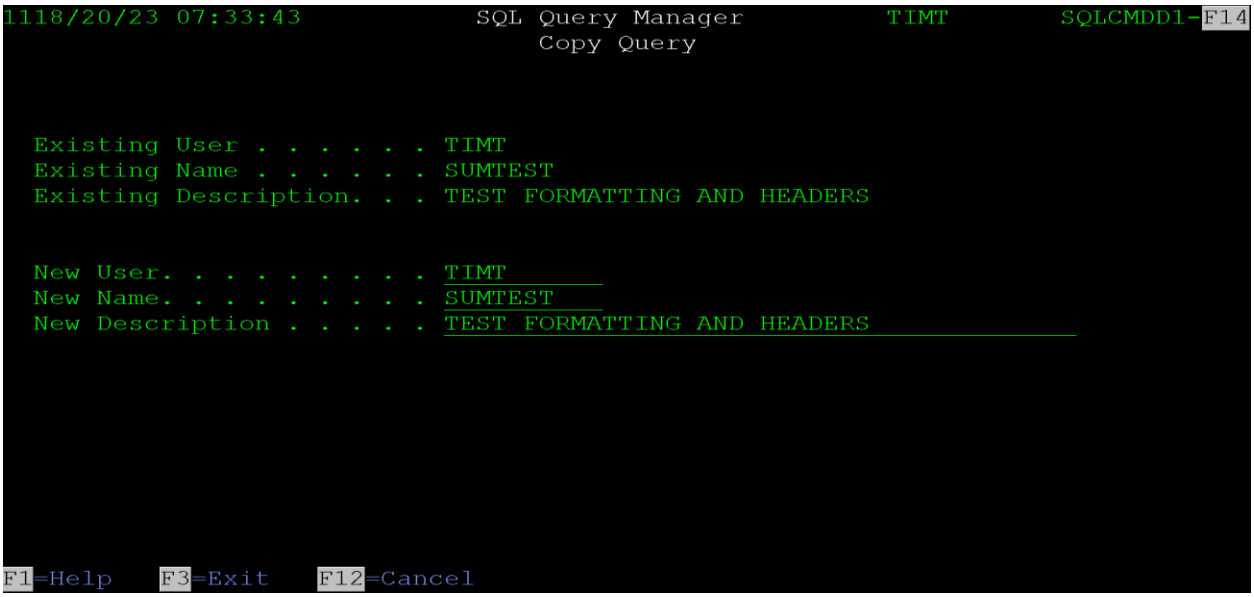
This data can also be printed using the command line tool with an action of *PRTDFT.

Running a Query

Running a query from the SQL Query Manager screen just displays the query output. View the Display Query Output section of this document for more details.

Copying a Query

A query can be copied by using the C = Copy (3) option from the main SQL Query Manager screen. The following screen is displayed:

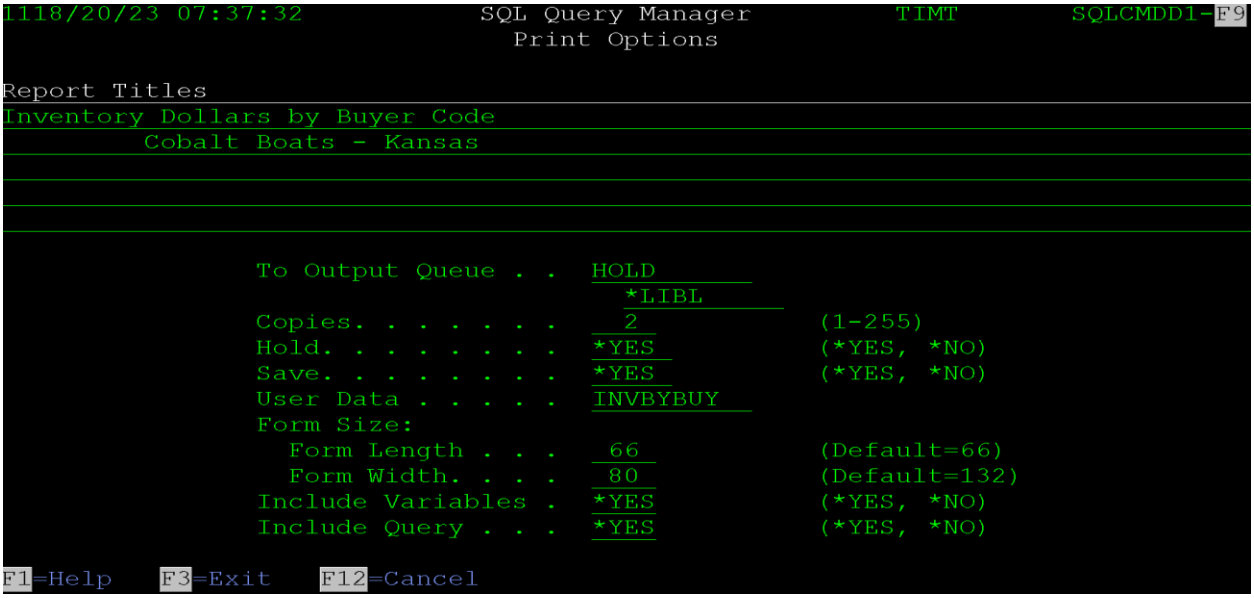


The query can be copied to a new user and/or a new name. The description can also be changed at this time.

Either the name or the user must be changed. An error message will be displayed if there is already a query with the new name for the new user.

Printing a Query

This screen is displayed when using option P = Print (6) from the main SQL Query Manager screen.



All output options are defaulted from the saved queries output options overrides. They can be manually changed before the query is printed. If there are no saved output options, standard defaults will be included.

Report Titles - These titles will be printed centered at the top of the report.

Output Queue and Library - These fields allow for entry of the output queue and library. Special values of *LIBL and *CURLIB are allowed. The program will verify that the entered output queue exists.

Copies - The number of copies to print.

Hold - The hold option used on the spooled file.

Save - The save option used on the spooled file.

User Data - The user data used on the spooled file.

Form Size, Length and Width - These override the page length and width used on the spooled file. Any data that does not fit in the width will be truncated.

Include Variables - Entering *YES here will cause the printout to include any variable setup on the query and their overridden values.

Include Query - Entering *YES here will cause the printout to include the SQL statement being run.

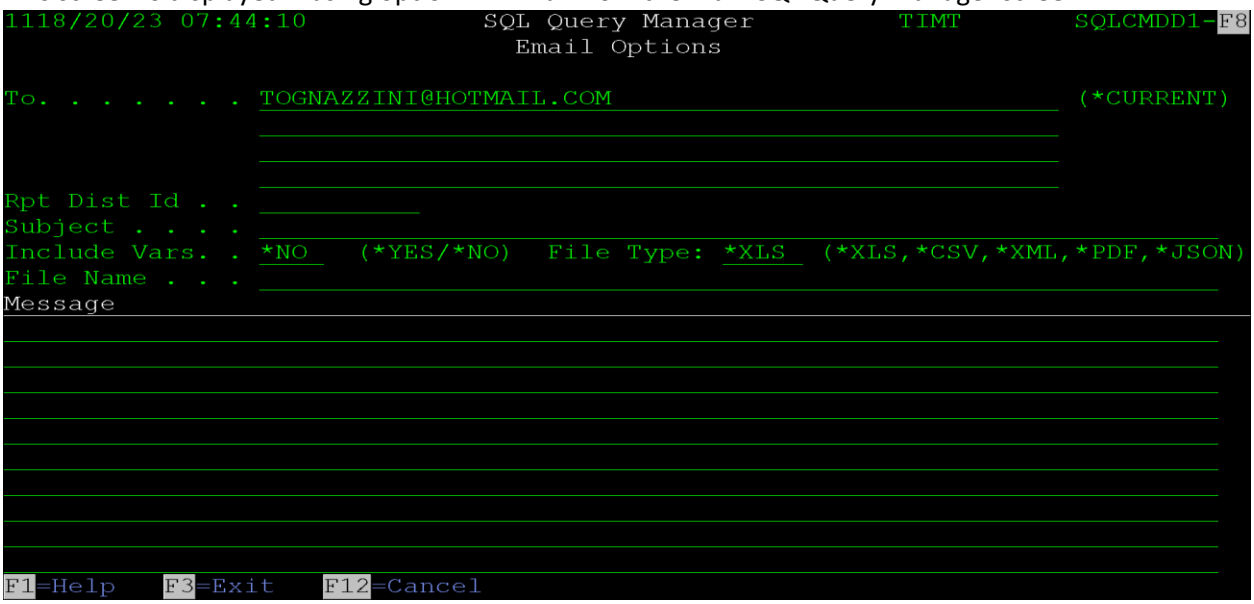
The following is an example of a printed query. The include variables and include query options were both set to *NO to keep the output size down.

2023/11/18 7:40:32 Inventory Dollars by Buyer Code PAGE: 1
Cobalt Boats - Kansas

Stk Cde	Buyer Cde	Store Dollars	Large Negative w/EdtC A	no Edit code
A		556,071.52	6,178,571,817.6983CR	556071.5191
A	HC	265.75	2,952,728.5936CR	265.7456
A	SM	174,993.93	1,944,376,845.5623CR	174993.9335
N	SM	3,443.85	38,265,092.8401CR	3443.8587
S	SM	.00	7.7778CR	.0006
	Totals:	734,775.05	8,164,166,492.4722CR	734775.0577

Emailing a Query

This screen is displayed if using option E = Email from the main SQL Query Manager screen.



To - This allows entry of up to 4 email addresses to email the report to. It has a special value of *CURRENT, this must be configured in the software to pull an address for the current user. It does not work by default. If this is configured the default email address is often already pulled for the current user.

Rpt Dist Id - Report Distribution Id. - This allows entry of a report distribution ID to pull a list of users to email the report to. This only works if IBM's Advanced Job Scheduler licensed program is installed on the server.

Subject - This field allows for entry of the subject on the email being sent.

Include Vars - Set this field to Yes if the query includes variables and you want their values included in the body of the email.

File Type - this field designates the type of file that will be attached to the email the options are:

- ***XLS** - An Excel file will be generated for the query and attached to the email.
- ***CSV** - A CSV (Comma Separated Variable) file will be generated and attached to the email.
- ***XML** - An XML file formatted for use in Excel will be generated and attached to the email. This option was created before there was a way to create a native Excel file and is considered deprecated. It remains active for backwards compatibility.
- ***PDF** - A PDF file will be generated and attached to the email.
- ***JSON** - A JSON file will be generated and attached to the email.

File name - This field allows the file name to be set for the attachment. If it is left blank a default unique file name will be used. This default does not really make sense to anyone so for production emails, the name should be provided. The system will append the correct file extension based on the file type, so it is not necessary to include it in the name.

Message - These lines allow data to be entered for the email body.

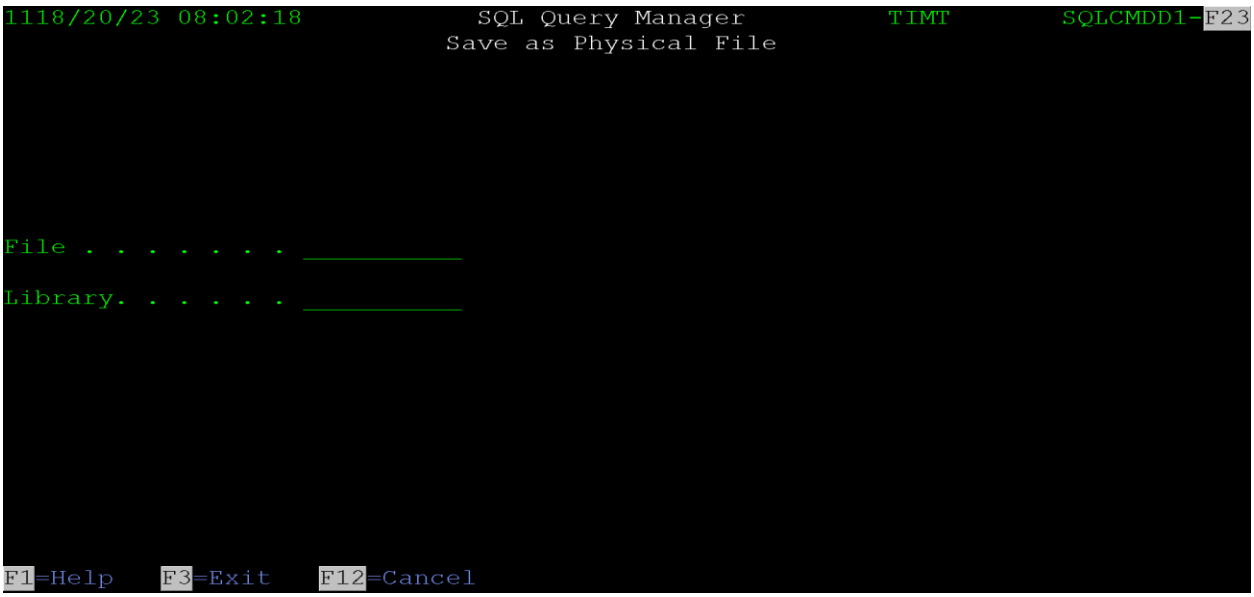
Opening a Query in Ad Hoc

This option is used via option A = Open in Ad Hoc from the main SQL Query Manager screen. It copies the SQL statement from the saved query into the Ad Hoc Query screen. See the Ad Hoc Query Tool section of this document for the options allowed via that system.

This is useful if a user wants to play with a query statement or produce a one-off report based on a saved query definition without augmenting the save definition.

Saving a Query as a Physical File

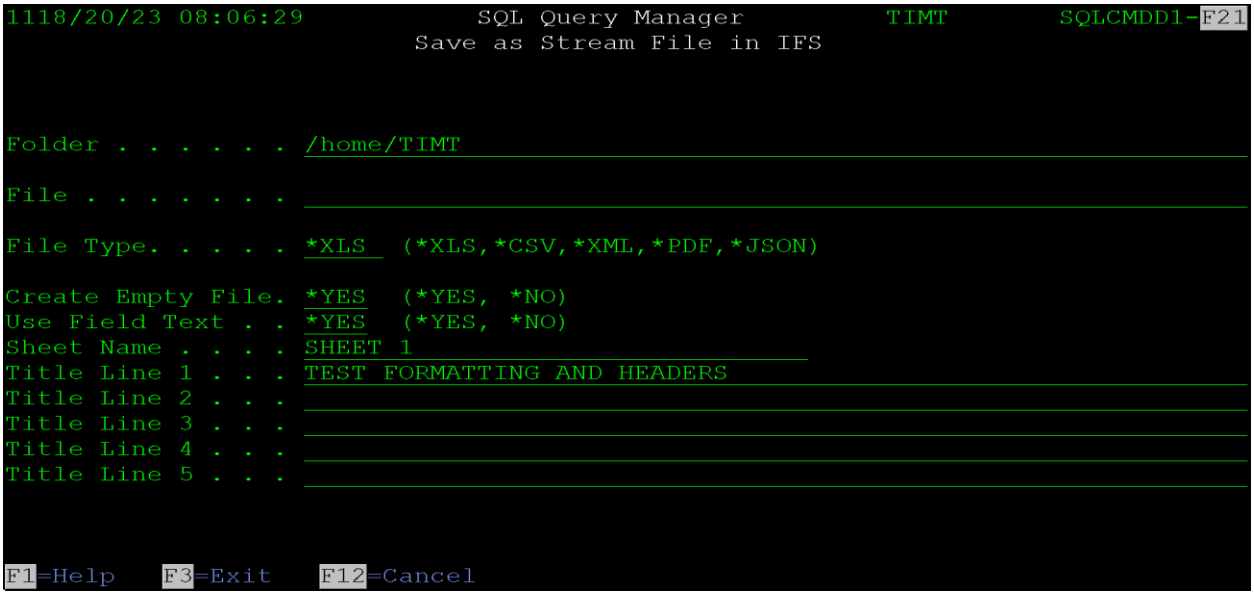
This screen is displayed when using option F = Save as PF from the main SQL Query Manager screen.



Enter the file name and library and press enter. The program will create a physical file in the entered library with the entered name.

Saving a Query in the IFS

This screen is reached by using option S = Save in IFS from the main SQL Query Manager screen. It allows the results of a query to be saved as a stream file in the IFS (IBM's integrated file system). Using the QNTC file system this can also save the output to a shared network folder.



Enter the required data and press enter to generate the file.

Folder - This field allows entry of the folder where the file will be created.

File - This field allows entry of the file name for the created file. The correct file extension will be added based on the file type, so it is not necessary to include it in the file name.

File Type - This designates the type of file that will be created.

Create Empty File - Entering *NO here will not create the file if the query does not return any rows. *YES will create an empty file.

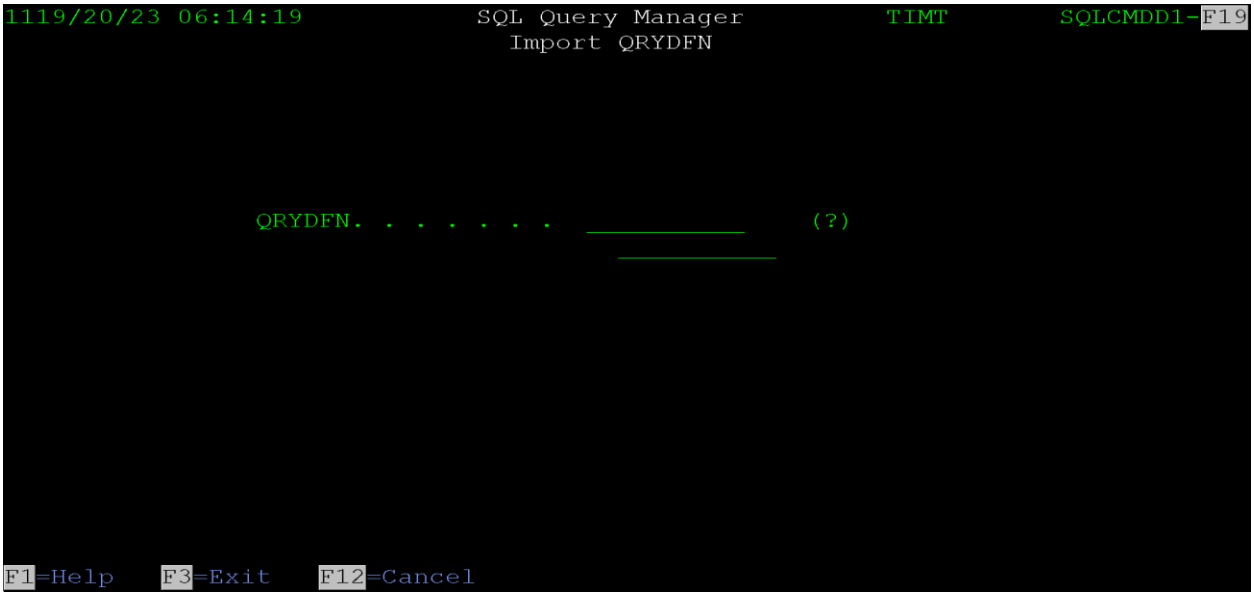
Use Field Text - This option allows using the field text for the column headers. By default the program will field names or aliases if entered in the SQL statement. Entering *YES here changes the program to use the text defined in the file.

Sheet Name - This option is used when creating an Excel or XML file. It sets the tab name in the spreadsheet.

Title Lines 1 - 5 - These title lines are included at the top of an Excel, XML or CSV file. They are also included as report headers at the top of PDF output. In a JSON file they are included as keys in the root element before the data array.

Import Query/400 Definition

This screen allows a Query/400 query definition to be imported.



Enter the name of the query and the library it is in and press enter. The program will extract the SQL from the query by retrieving the QM Query source. It then formats the query the best it can and loads it into the Ad Hoc Query Tool. From there the query can be edited, run and saved if desired. Please see the Ad Hoc Query Tool section of this document for more information.

Command Line Tool

Using the SQL command as a command line tool creates an easy methadone of accomplishing the following tasks:

- Creating a printed report
- Emailing a report as a PDF, Excel file, CSV file, XML file, or JSON file.
- Displaying the results of an SQL statement
- Saving the results of an SQL statement as a physical file on the server
- Saving the results of an SQL statement as a PDF, Excel file, CSV file, XML file, or JSON file in the IFS or a shared network folder.
- Quickly running an ad hoc query over a file.

The command can also be called from within a CL or RPG program to perform this list of tasks for end users.

Do to the multiple functions the command can be used for, this section of the documentation has been broken into the following sub-sections:

- Full Parameter Reference
- Using SQL to Email a File
- Using SQL to Print a Report
- Using SQL to Create a Physical File
- Using SQL to Create a PC type File in The IFS or a Shared Network Folder
- Integrating the SQL Command into a CL Program
- Integrating the SQL Command into an RPG Program

Full Parameter Reference

Since the command is capable of multiple tasks, it has conditional parameters that are only displayed based on the value of other parameters. The following list lists all parameters. If a parameter is conditional it will state the condition required for it to be displayed.

This list is for reference when working with the command following sections will give information about how to perform specific tasks with the command in more of a how to manor. The following list comes directly from the help text for the command. Prompting the command use F1=Help will display very similar information.

FILE - File/Query Name

This field allows entry of either a file name or a query name. If the action option is left the default of *ADHOC the program expects this to be the name of a file. In this case the program will load to the ad hoc query screen with a default query statement of SELECT * FROM <FILENAME>.

If the action option is changed to any other value, the program will expect a query name to be entered here. The query name needs to already be set up in the SQL Query Manager program.

See the help text for the action field to learn more about how the file field is used with each type of action.

ACTION - Action to take.

This parameter allows entry of the action to take when the program is run. It defaults to *ADHOC which is a dual purpose action. *ADHOC is ignored if no file name is passed so the program just loads to the main maintenance screen. The rest of the values are listed below.

- ***ADHOC** - If a file name is passed opens the ad hoc query screen.
- ***EDIT** - Requires a query name in the FILE parameter, opens the program straight to the update screen for that query.
- ***PRINT** - Requires a query name in the FILE parameter, runs the query output to a spool file.
- ***EMAIL** - Requires a query name in the FILE parameter, emails the query output as an attachment.
- ***DISPLAY** - Requires a query name in the FILE parameter, displays the output from the query.
- ***OUTFILE** - Requires a query name in the FILE parameter, creates a physical file containing the results of the query.
- ***IFSFILE** - Requires a query name in the FILE parameter, creates a PC stream file with the result of the query.
- ***VIEW** - Requires a query name in the FILE parameter, opens displays the queries definition.

USER - User that created the query.

Specifies the user that created the query. This is needed because the system allows the same query name under different users. This means to run a query that someone else created their user profile name needs to be passed to the program.

Hint: If the query is being run from a program that multiple people will be using, the user should always be hard coded so the same query is run regardless of who is signed on.

EMAIL - Email Addresses

Specifies the email address to send to Excel file to. This option is only displayed if the ACTION is *EMAIL.

The field allows email addresses up to 50 characters and allows up to 50 addresses to be included.

The following are the options allowed for the parameter.

- ***CURRENT (default)** - The email address stored in the ACCESSPF for the current user running the job is used.
- ***RPTDSTID** - The RPTDSTID field is displayed and a report distribution ID can be entered. The email addresses will be pulled from that ID.
- ***DFT** - The default email address setup in the query definition will be used.
- **email-address** - Specify the email address of the recipient.

RPTDSTID - Report Distribution ID

Specifies a report distribution ID to pull the email addresses from.

Hint: This parameter will not be displayed unless a special value of *RPTDSTID is entered in the email field.

The email addresses will be pulled from ACCESSPF for all users setup in the report distribution queue for the EMAIL/*ALL/*ALL entry.

FILENAME - File Name

Specifies the name of the file attached to the email. Do not include the file extension, it will be added automatically based on the file type parameter.

This option is only displayed if the ACTION is *EMAIL.

TYPE - File Type

Specifies the type of file to contain the queries output.

This option is only displayed if the ACTION is *EMAIL.

The valid type are:

- ***DFT** - The default file type setup in a saved query is used.
- ***XLS** - Creates an Excel file.
- ***XML** - Creates an Excel formatted XML file.
- ***CSV** - Creates a Comma Separated Variable text file.
- ***PDF** - Creates a PDF file.
- ***JSON** - Creates a file with a JSON object in it.

This option is only displayed if the ACTION is *EMAIL.

SUBJECT - Subject

Specifies the subject for the email.

When prompting, the input field can be expanded by entering an ampersand (&) in the first position of the field, followed by a blank.

This option is only displayed if the ACTION is *EMAIL.

Valid Options:

- *NONE - Default, no subject is included in the email.
- character-value - Specify the subject of the email.

MESSAGE - Message

Specifies a message to include in the email, and its content type.

This option is only displayed if the ACTION is *EMAIL.

Valid Options:

- ***NONE** - This parameter does not place a message in the email.
- **character-value** – Specify the content placed in the email. CHAR(5000)

If a character value is specified, then the second element must be specified.

Element 2: Send as

- ***TEXTPLAIN** - Copy the content into the body of the mail message specifying content type text/plain. Line breaks can be added by placing hex value X'0D25' in the data.
- ***TEXTHTML** - Copy the content into the body of the mail message specifying content type text/html. Line breaks can be added by placing '
' in the data.

Note the SQL command does not send the email directly. It must be configured to use third party email software. The message type depends on the third-party software being used. If Gumbo products are used to send the email both Send As options are valid. If RJS is used to send the email all messages will be sent as HTML regardless of which Send As option is used.

OBJ - Objects

Specifies the path name of additional objects to send. A maximum of 64 path names can be specified. The object must already exist in the IFS.

This option is only displayed if the ACTION is *EMAIL.

This option has only been tested with emails sent using Gumbo products. The attachment types are specific to the email command provided by Gumbo's GSENDMAIL command.

Valid options:

- ***NONE** - Default, No objects are sent.
- **path-name** - Specify an object path name. CHAR(128)

If a path name is specified, then the second element of the parameter must be specified. It's options are:

- ***ATTACH** - Default, Sends the object as an attached file using MIME and specifying "application/octet-stream". This works for most PC type files.
- ***TEXTPLAIN** - Copies the object into the body of the mail message specifying content type text/plain. The CCSID of the file must match the value specified for CHRENC(). Some servers

disregard this request for the second and subsequent body parts. If this is the case, try MSG(*NONE).

- ***TEXTHTML** - A copy the object into the body of the mail message specifying content type text/html. The CCSID of the file must match the value specified for CHRENC(). Some servers disregard this request for the second and subsequent body parts. If this is the case, try MSG(*NONE).
- ***ATTACHPDF** - Sends the object as an attached file using MIME and specifying "application/pdf". Use this value if the attached file contains Adobe's Portable Document Format (pdf) data.
- ***ATTACHPS** - Send the object as an attached file using MIME and specifying "application/postscript". Use this value if the attached file contains postscript data.
- ***NOTE** - Same as *TEXTPLAIN which is preferred.

INCVARS - Include Variable Overrides

Specifies whether to include the values of the variable overrides in the body of the email.

This option is only displayed if the ACTION is *EMAIL.

Valid Options:

- ***NO** - Default - Variable Overrides are not included in the body of the email.
- ***YES** - Variable overrides are included in the body of the email. The message text will be displayed first then a list of all variables and the values they are overridden to are appended to the end of the email body.

PFILE - Physical File

Specifies name of a physical file to create and place the output into. The file cannot exist, or the program will produce an error.

This option is only displayed if the ACTION is *OUTFILE.

Element 1: Output File

- **character-value** - Enter the name of the file to be created. CHAR(10)

Element 2: Library

- ***CURLIB** - Default - The current library of the job is used.
- **character-value** - Enter the library name to create the file in. CHAR(10)

IFSFOLDER - Folder Name

Specifies the name of the folder the output file will be placed in. The name is a path and should start with /.

This option is only displayed if the ACTION is *IFSFILE.

IFSFILE - File Name

Specifies the name of the file created. Do not include the file extension, it will be added automatically based on the file type parameter.

This option is only displayed if the ACTION is *IFSFILE.

IFSTYPE - File Type

Specifies the type of file to contain the queries output.

The valid type are:

- ***DFT** - The default file type setup in a saved query is used.
- ***XLS** - Creates an Excel file.
- ***XML** - Creates an Excel formatted XML file.
- ***CSV** - Creates a Comma Separated Variable text file.
- ***PDF** - Creates a PDF file.
- ***JSON** - Creates a file with a JSON object in it.

This option is only displayed if the ACTION is *IFSFILE.

OUTQ - Output Queue

Specifies the output queue to place the printed output into.

Single values

- ***JOB** - Default - The output will be placed in the output queue from the current job definition.

Element 1: Output Queue

- **character-value** - Specify the output queue to place the output in.

Element 2: Library

- **Library** - Enter the library name of the output queue, if left blank *LIBL will be used.

This option is only displayed if the ACTION is *PRINT.

TITLE - Report Title

Specifies the title to use at the top of the report. The value of *DFT will use the name of a saved query definition. If the query is being passed to the command, then there will not be a title unless it is entered here.

This option is only displayed if the ACTION is *PRINT.

COPIES - Copies

Specifies the number of copies to print. Defaults to 1, cannot be greater than 255.

This option is only displayed if the ACTION is *PRINT.

HOLD - Hold Output

Specifies whether to hold the output in the output queue.

Valid Options:

- ***NO** - Default - The output is not put on hold.
- ***YES** - The output is put on hold.

This option is only displayed if the ACTION is *PRINT.

SAVE - Save Output

Specifies whether to save the output after it has printed.

Valid Options:

- ***NO** - Default - The output is not saved.
- ***YES** - The output is saved.

This option is only displayed if the ACTION is *PRINT.

USRDTA - User Data

Specifies the user data for the spool file.

Valid Options:

- ***DFT** - Default - The user data will contain the query name.
- **character value** - Enter a value to be used for the user data. Char(10)

This option is only displayed if the ACTION is *PRINT.

WIDTH - Width

Specifies the width of the spool file. The default is 132. Any data that does not fit in the width is truncated. The width also determines the CPI. See the chart below for details.

- up to 80 - CPI(10) Portrait
- 81 - 96 - CPI(12) Portrait
- 97 - 132 - CPI(12) Landscape
- 133 - 198 - CPI(15) Landscape
- over 198 - CPI(18) Landscape

This option is only displayed if the ACTION is *PRINT.

LENGTH - Length

Specifies the page length of the spool file. The default is 66.

This option is only displayed if the ACTION is *PRINT.

PINVR - Include Variables

Designates whether variable values are included on the report or not. If a Y is entered in this parameter, the first page of the report will include any variable values before the detail starts to print.

The variables must be defined in a saved query definition, or nothing will be included. If variable overrides are passed to the command, then the values passed will be included. Otherwise, the query will be run with the default values and they will be listed.

This option is only displayed if the ACTION is *PRINT.

PINQY - Include Queries

Designates whether to print the passed query or not. If a Y is entered in this parameter, the report will include a partially formatted printout of the SQL statement passed to the print program.

The formatting is simple because the query statement is passed as a compressed string. This means any formatting done in a saved query definition is stripped and not included in this printout.

This option is only displayed if the ACTION is *PRINT.

PNAME - Program Name

Allows entry of a program name. The name is used when creating a printed report or a PDF file. It is used in the headings to simulate standard report formatting. The name is optional and will not be included if it is not passed.

The name should be passed if a program is creating a report. The name should be the name of the RPG or CL program creating the report.

This option is only displayed if the ACTION is *PRINT or if a PDF file is being created. If creating an Excel or CSV file use the sheet name to show the program name, include it in on a title line or incorporate it into the file name, subject or email message.

VAROVR - Variable overrides

Specifies values to use for specific variables. The entry requires a variable name and a value. The variables are set up in the query definition and the names must match. Each variable setup is also given a type. The value entered for the variable name must match the type the variable is defined for.

This option is only displayed if the ACTION is *PRINT, *EMAIL, *DISPLAY, *OUTFILE OR *IFSFILE.

Element 1: Variable

- **character-value** - Enter the name of the variable to override. CHAR(10)

Element 2: Value

- **character-value** - Enter the value to override the variable to. CHAR(30)

ADDSQL - Additional Queries

This parameter allows additional queries to be added to the command. Additional queries are used to create additional tabs in Excel files. The parameter is a multi-value list. Each entry in the list includes the query to include, sheet name and title lines 1-5. The use of these options is a bit complicated so they will be documented in their own section. See the Multi-Tab Excel File Example section of this document for an example.

Using SQL to Print a Report

The SQL command can be used to print a report to a specified output queue. The command can print a report from a saved query or from an SQL statement passed to the command.

Using a Saved Query

The first parameter of the command is the name of the query, the second parameter is the action to perform. In this case *PRINT is used. The third parameter is the user the query is saved for. If including this command in a program the user should always be included in case a different user is running the program. The simplest command to print is the following:

```
SQL ABCSUM *PRINT COBALT
```

This command runs the query ABCSUM saved for user COBALT and prints the output.

Additional parameters can be used to change the parameters. The following command shows all optional parameters. Refer to the Full Parameter Reference section of this document to see details about each option.

```
SQL FILE(ABCSUM)
    ACTION(*PRINT)
    USER(COBALT)
    TITLE1('Title 1')
    TITLE2('Title 2')
    TITLE3('Title 3')
    TITLE4('Title 4')
    TITLE5('Title 5')
    OUTQ(OUTQLIB/OUTQUE)
    COPIES(2)
    HOLD(*YES)
    SAVE(*YES)
    USRDTA('USER-DATA')
    WIDTH(80)
    LENGTH(66)
    PINVR(*NO)
    PINQY(*NO)
    PNAME(TST001)
    VAROVR((BEGINDATE 20230101) (ENDDATE 20231231))
```

Passing an SQL Statement

Most instances where the SQL command is used to print a report are in programs. If this is the case, it is often best to build a dynamic SQL statement and include the selection and formatting options in the SQL statement and then pass that statement to the command. See the section of this document on Integrating the SQL command into a CL or Integrating the SQL command into an RPG Program for more information and examples.

The following is an example of using the SQL command to print a report by passing a SQL statement to it. Other than the USER parameters, all options from the previous example can be included. Since there are no default titles, they should be manually included so they are in this example.

```
SQL FILE('Select
            csCsNo "Account",
            csName "Customer_Name",
            csStCd "State_Code",
            csZpCd "Zip_Code"
          From CUSTS
          Where csStCd = ''OK'' ')
    ACTION(*PRINT)
    TITLE1('Customers In Oklahoma')
    TITLE2(*ENVNME)
    OUTQ(QGPL/HOLD)
    WIDTH(80)
```

Pay special attention to these facts:

- Each column has been given an alias. This provides column headers for the report.
- Title lines are added so the report has titles. The Special value of *ENVNME was used to add the environment name to the report.
- Quotes in the SQL statement must be doubled. See the Where clause where the state code is being compared to a constant.
- The width was set to 80. Most printers configured on an IBMi will use computer output reduction which will make an 80-column report print in portrait.

Example Output:

```
2023/11/19  7:40:53          Customers In Oklahoma                PAGE:     1
                             Cobalt Boats - Kansas

  Account Customer_Name                  State_Code  Zip_Code
     5300 AGRICO CHEMICAL CO             OK          74101-3166
    26000 ANDY'S MARINE                  OK          73160-0000
    40000 ARROWHEAD MARINA               OK          74349
    95000 BOMAR'S MARINE                 OK          74349-0336
```

Using SQL to Create a Physical File

The SQL command can save the results of a query as a physical file. This is accomplished by using the Action parameter with a value of *OUTFILE. The first parameter can be a save query or a passed SQL statement. If using a saved query, it is best to also include the USER parameter.

The following is an example of creating a physical file from a passed SQL statement.

```
SQL FILE('Select
            csCsNo accNbr,
            csName cusNme,
            csStCd steCde,
            csZpCd zipCde
          From CUSTS
          Where csStCd = ''OK'' ')
    ACTION(*OUTFILE)
    PFILE(TIMLIB/OKCUSTS)
```

This command creates a physical file in TIMLIB called OKCUSTS.

Notice that the column aliases have been changed to be more standardized field names. Do not use pretty column names like would be used in the print option. This would make weird column names in the physical file.

The command tries to pull all attributes from the selected field if possible. If a column is a direct selection column this will work. If a column is generated from a function or equation, this does not work. The following is what the first column definition looks like in the DSPFFD TIMLIB/OKCUSTS command.

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+
             Data       Field  Buffer    Buffer        Field    Column
  Field      Type       Length  Length  Position       Usage    Heading
  ACCNBR     PACKED       9  0       5         1       Both     Customer
                                                                Number
    Field text  . . . . . . . . . . . . . . . :  Customer Number
    Referenced information
      Referenced file . . . . . . . . . . . . :  CUSTS
        Library . . . . . . . . . . . . . . . :  DATA_TEST
      Referenced record format  . . . . . . . :  CUSTSR
      Referenced field  . . . . . . . . . . . :  CSCSNO
      Attributes changed  . . . . . . . . . . :  None
    Default value . . . . . . . . . . . . . . :  None
```

Notice that the field is named ACCNBR, from the alias used in the query, but all other attributes are pulled from the file that the field was pulled from.

Using SQL to Create a PC type File in The IFS or a Shared Network Folder

The SQL command is capable of converting the results of an SQL query into a PDF, Excel, CSV, XML or JSON file. These files can be saved to the IFS or emailed. When creating a PDF file, most of the same options for creating a report are used to format the output of the spool file used to create the PDF.

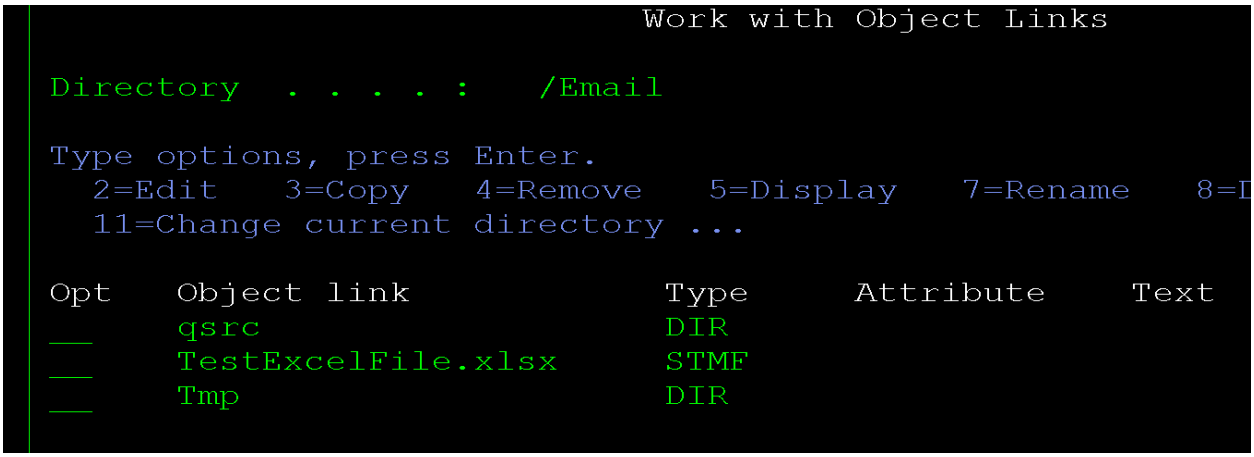
To save a file in the IFS, the action code of *IFSFILE is used. This causes the SQL command to include the output folder and output file parameters.

The following is an example of creating an Excel file in the IFS.

```
SQL FILE('Select
            csCsNo "Account",
            csName "Customer_Name",
            csStCd "State_Code",
            csZpCd "Zip_Code"
          From CUSTS
          Where csStCd = ''OK'' ')
    ACTION(*IFSFILE)
    TITLE1('Customers In Oklahoma')
    TITLE2(*ENVNME)
    IFSFOLDER('/Email')
    IFSFILE('TestExcelFile')
    IFSTYPE(*XLS)
```

The file can be displayed in the ifs using the WRKLNK command, it looks like this:

Command: WRKLNK '/Email/*'



Notice the SQL command added the appropriate PC file extension to the file. Since an Excel file is a PC stream file, and a zipped file at that, using option 5 to view the file will just show a bunch of gobbledygook. Like this:

```
Browse : /Email/TestExcelFile.xlsx
Record :      1   of      25 by  18                    Column :    1   1258 by 131
Control :

....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9...
 ***********Beginning of data**************
&.           Qäû M   ^       Ì¾ Ï?Ê,ËÇÁÁÈË ËÇÁÁÈ  Ì_¾¸ª)Ëº  f «f¸R GçSC+TÈ    "À{+ #L~¶æ §a
¡¦ÈÏ   Û pZ  Sò *£é¯:ù£ ÿ ñR H9JÅ /kÙ¡¶$¸ê> uÿ;ló ú
¯DPø W -s-5N U( Å v¾ & @Ñº¢7³¡ Ì R#OQã`0NÅ$¶-Rª¤á  XX, " Î[Ü nï<bh~ =B$ "lGId_Êº_¼¯ é¿^-C Î
```

Viewing a CSV file via the work link command is ok, but PC file types are best viewed using the PC applications they were created for. This file can be opened in Excel by creating a network share to the email folder, pointing windows explorer at it and opening the file in windows (all this can be done on a MAC or Linux system as well using their appropriate equivalent options).

Opening this file in Excel looks like this:

	A	B	C	D
1	**Customers In Oklahoma**			
2	**Cobalt Boats - Kansas**			
3				
4	**Account**	**Customer_Name**	**State_Code**	**Zip_Code**
5	5300	AGRICO CHEMICAL CO	OK	74101-3166
6	26000	ANDY'S MARINE	OK	73160-0000
7	40000	ARROWHEAD MARINA	OK	74349
8	95000	BOMAR'S MARINE	OK	74349-0336
9	103000	CAMERON CORP	OK	74145
10	108000	CATFISH BAY MARINE	OK	73439

Using the SQL Command to email a PC File

The SQL command can also email a PC file. The options for generating the file are the same as creating a file in the IFS, the action parameter just gets set to *EMAIL instead of *IFSFILE. This causes the command to include the mail options parameters. The following is an example of emailing the same excel that was saved in the IFS.

```
SQL FILE('Select
            csCsNo "Account",
            csName "Customer_Name",
            csStCd "State_Code",
            csZpCd "Zip_Code"
          From CUSTS
          Where csStCd = ''OK'' ')
    ACTION(*EMAIL)
    TITLE1('Customers In Oklahoma')
    TITLE2(*ENVNME)
    EMAIL(TOGNAZZINI@HOTMAIL.COM)
    FILENAME('Test Excel File')
    SHEET('OK_Customers')
    SUBJECT('Test Emailed Excel File')
    MESSAGE('Attached is a test Excel file emailed form the SQL
command.' *TEXTHTML)
```

This emails the same file that was saved to the IFS.

The email, file name, sheet name, subject and message parameters were added to the command to fill in those properties on the email. Most of these options are obvious, but the Full Parameter Reference Section of this document can be referenced for specific on those parameters.

Integrating the SQL Command into a CL Program

Since the SQL command is a command line tool, it is easy to integrate it into a CL program. The SQL command can just be entered and prompted to fill in the parameters. This can be useful in both interactive and batch programs. In interactive programs a prompt screen can be displayed to prompt for parameters while creating an SQL statement or for variable overrides when using a saved query. In batch programs the parameters can be passed into the program or pulled from anywhere in the system.

The following is an example of using the command from within a CL program to email the same file used in the rest of the examples. The state code and email address are hard coded for the example. They could be passed into the program or the user could be prompted for them if desired.

```
0001.00 PGM
0002.00
0003.00   DCL &STATE  *CHAR   2 VALUE('OK')
0004.00   DCL &EMAIL  *CHAR  50 VALUE('TOGNAZZINI@HOTMAIL.COM')
0005.00   DCL &SQLSTM *CHAR 500
0006.00
0007.00   CHGVAR &SQLSTM +
0008.00     'Select +
0009.00        csCsNo "Account", +
0010.00        csName "Customer_Name", +
0011.00        csStCd "State_Code", +
0012.00        csZpCd "Zip_Code" +
0013.00      From CUSTS'
0014.00
0015.00   IF (&STATE *NE ' ') (CHGVAR &SQLSTM (&SQLSTM *TCAT +
0016.00          ' Where csStCd = ''' *CAT &STATE *CAT ''''))
0017.00
0018.00   SQL FILE(&SQLSTM) +
0019.00       ACTION(*EMAIL) +
0020.00       TITLE1('Customers In ' *CAT &STATE) +
0021.00       TITLE2(*ENVNME) +
0022.00       EMAIL(TOGNAZZINI@HOTMAIL.COM) +
0023.00       FILENAME('Customer List') +
0024.00       SHEET('SQLCMDC2') +
0025.00       SUBJECT('Customer List') +
0026.00       MESSAGE('Attached is a list of customers.' *TEXTHTML)
0027.00
0028.00 ENDPGM
```

Notice that the SQL statement was built dynamically to allow for all states if one was not selected. The subject, file name and message could have also been built dynamically based on the whether or not a state was selected. This was not done to keep the example simple.

Line 7 shows the start of the SQL statement. On Line 15 the state selection is added, only if it is populated. Line 18 is the SQL command being run with the built query.

Building complex dynamic queries is often easier in an RPG program. The next section shows using the SQL command in an RPG program with more dynamic options.

Integrating the SQL Command into an RPG Program

Using the SQL command in an RPG program is slightly more complicated because the QCMD API program must be used to run the command. Systems that have the #$ base functions added make this a little easier. In the following example the #$CMD procedure will be used to run the command.

In the test system that procedure comes from the CBTFNCV1 service program which is included in the CBT binding directory in the CTL-OPT's. Prototypes for the #$ functions are copied in the in the #$CBTFNCV1PR source member. The #$DBLQ function is also used. This function just doubles any quotes in a string. #$DBLQ is provided in the same service program #$CMD.

Like the CL program example this program hard codes the selected state and email address. These could be prompted from the user or passed to the program.

The program is written in **free because all new programs should be. All of this works in fixed format RPG as well.

```
**free
Ctl-Opt datedit(*ymd) Option(*NoDebugIO:*NoShowCpy:*SrcStmt) DftActGrp(*No)
BndDir('CBT') Main(Main);

// SQL command RPG Example

/Copy qsrc,CBTFNCV1PR  // prototypes for all #$ procedures

Dcl-Proc Main;
  Dcl-S sqlStm varchar(500);
  Dcl-S message varchar(100);
  Dcl-S subject varchar(100);
  Dcl-S fileName varchar(100);
  Dcl-S title3 varchar(100);
  Dcl-S state char(2) inz('OK');
  Dcl-S where varchar(5);
  Dcl-S email varchar(50) inz('tognazzini@hotmail.com');

  // used in case there are multiple optional selection criteria,
  // the first one will be where then each one after that is and
  where = 'Where';

  // Dynamically build the sql statement
  sqlStm = '+
    Select +
      csCsNo "Account", +
      csName "Customer_Name", +
      csStCd "State_Code", +
```

```
      csZpCd "Zip_Code" +
    From CUSTS';

  // add state selection if populated
  If state <> '';
    sqlStm += ' ' + where + '  csStCd = ''OK''';
    where = 'and';
  EndIf;

  // build the subject, message, optional third title and file name,
  // they are different if a state is selected or not
  If state <> '';
    subject = 'Customer list for state code ' + state;
    message = 'Attached is the customer list for customer in state ' + state;
    fileName = 'Customers in ' + state;
    title3 = 'For State: ' + state;
  Else;
    subject = 'Customer list' + state;
    message = 'Attached is a customer list';
    fileName = 'Customer List';
    title3 = '';
  EndIf;

  // email the customer list
  #$CMD('+
  SQL FILE(''' + #$DBLQ(sqlStm) + ''') +
      ACTION(*EMAIL) +
      TITLE1(''Customer List'') +
      TITLE2(*ENVNME) +
      TITLE3(''' + #$DBLQ(title3) + ''') +
      EMAIL(' + email + ') +
      FILENAME(''' + #$DBLQ(fileName)+''') +
      SUBJECT(''' + #$DBLQ(subject) + ''') +
      MESSAGE(''' + #$DBLQ(message) + ''' *TEXTHTML)');

End-Proc;
```

The emailed file attachment looks like this:

	A	B	C	D
1	**Customer List**			
2	**Cobalt Boats - Kansas**			
3	**For State: OK**			
4				
5	**Account**	**Customer_Name**	**State_Code**	**Zip_Code**
6	5300	AGRICO CHEMICAL CO	OK	74101-3166
7	26000	ANDY'S MARINE	OK	73160-0000
8	40000	ARROWHEAD MARINA	OK	74349

Ad Hoc Query Tool

Using the SQL command as an ad hoc query tool provides several benefits over the standard IBM provided query tools. IBM provides Query/400, QM query, STRSQL and GUI SQL as standard querying tools. The SQL command has these features that are lacking from most of IBM's tools:

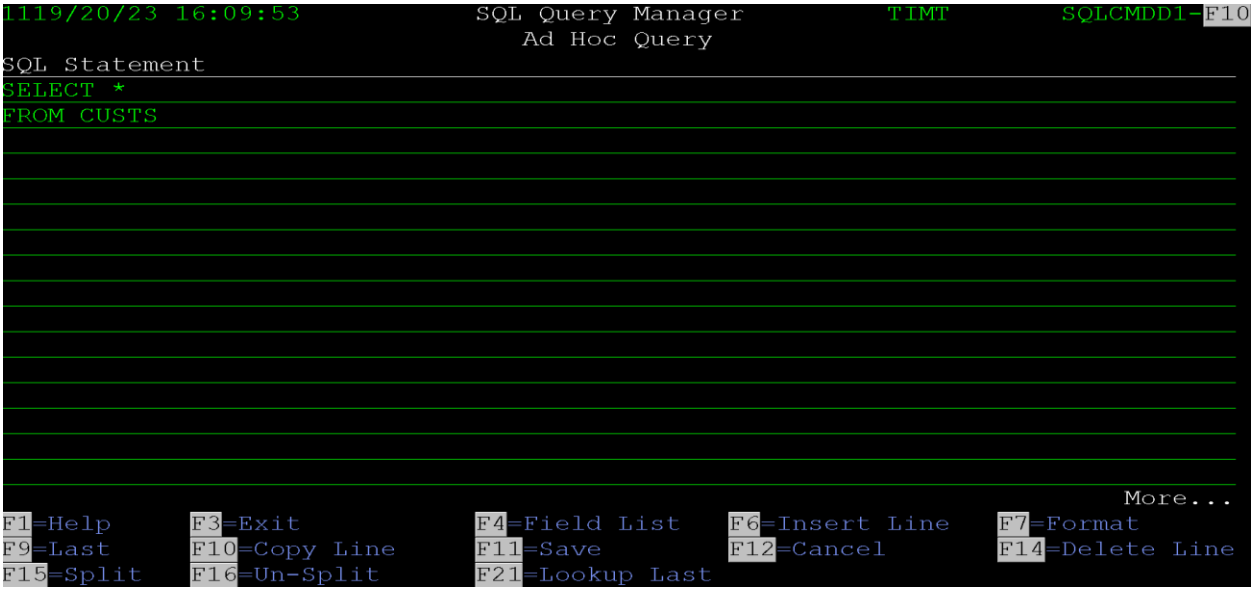
- **Output Options** - the output from the query tool can be printed, emailed, saved to the IFS or saved to a physical file on the server. While some of these options are available in some of the IBM tools, the SQL command is unique in the fact that email and archive options can produce several file types including Excel, CSV, XML, JSON and PDF.
- **Standard SQL Interface** - The SQL command uses a standard SQL interface to pull the data for the query. IBM's STRSQL command and GUI SQL have this feature as well, but STRQRY cannot save the query for latter use. GUI SQL can save the query to a PC file, but it does not create a searchable list to easily find the query later.
- **All Features in One Program** - The SQL command is the only option that incorporates all options from the list of commands.
- **Statement Validation** - The SQL command interface includes SQL Statement formatting and validation. If there is an error in the statement it displays the SQL error message and positions the cursor to the location of the error. GUI SQL does this as well.
- **Quick Query** - The SQL command can be used from a command line to quickly query a file. Just typing SQL and the file name on a command will bring up the query.
- **Searchable Results** - The SQL command is also the only option that allows searching within the returned result set.
- **Convert *QRYDFN** - The SQL command also has the ability to convert a Query/400 query to an SQL statement. QM query provides this feature via an external command, but the others do not.

The quickest way to get started with an ad hoc query is to simply enter SQL and a file name on the command line. This opens the SQL Ad Hoc query screen with a default SQL statement populated to select all fields from the entered file.

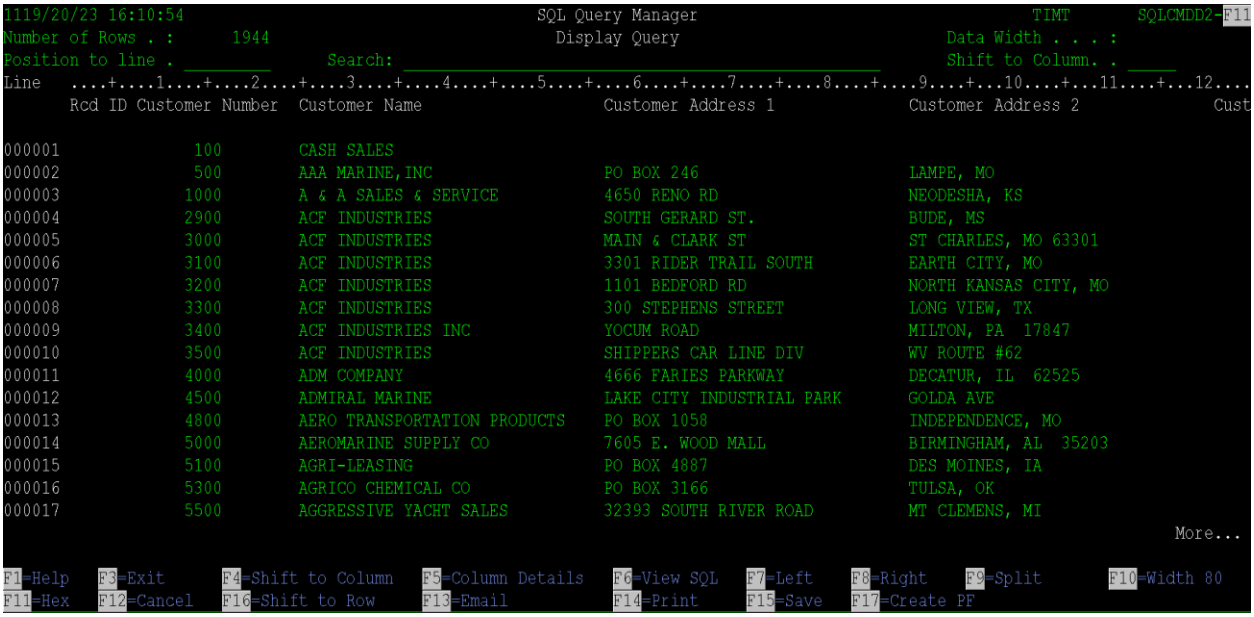
The following command is an example of this:

```
SQL CUSTS
```

This command displays the following screen:



Just pressing enter will display the results of the query. The display query screen looks like this:



This screen allows searching within the data, emailing the result set, saving the result set in a variety of ways and more. See the Display Query Output Screen section of this document for all options available.

The Ad Hoc Query screen has many options that make it a powerful querying tool. The following is a list of the functions that can be performed via command keys. Some of these functions will be discussed more in their own section as they bring up additional screens.

F1 = Help - Displays help text for the field the cursor is on if the help text is loaded on the system.

F3 = Exit - Exits the program.

F4 = Field List - This option displays a list of fields from all fields in the file. One or more fields can be selected to be inserted into the query at the location of the cursor. The field list screen is displayed.

Special formatting occurs if the previous word is Select, where or a join clause. See the Field Selection Screen of this document for more information. Learning these special cases can make entering an SQL statement much more efficient.

F6 = Insert Line - F6 inserts a new line after the line the cursor is one.

F7 = Format - This function is used to format the entire SQL statement. It can be useful when you start entering a query, for instance if F4 is used to select a bunch of fields at once, they get inserted with many fields on one line. F7 will break each field to its own line. However, the format option can do really weird things with lists and things in parenthesis. It assumes a comma should start a new line and things in parentheses would be split out. This can wreak havoc on a well formatted SQL statement. If F7 is used and the result is unacceptable, F12 can be pressed to go back a screen and not save the results. However, if any other option is used or the enter key is pressed the modified query will be saved and it will not be possible to recover the previous format of the statement.

F9 = Last - Loads the last run query onto the screen.

F10 = Copy Line - F10 creates a duplicate line of the line the cursor is on. The new line goes directly after the existing line.

***Tip** - Since a line cannot be added before the first line in the list, F10 can be used to duplicate the first line down and then the first line can be replaced with a new value.

F11 = Save - Displays a screen to save the current query as an SQL Query Manager query.

F12 = Cancel - This goes back to the previous screen. Any data entered since a function key or the Enter key was last pressed will be lost.

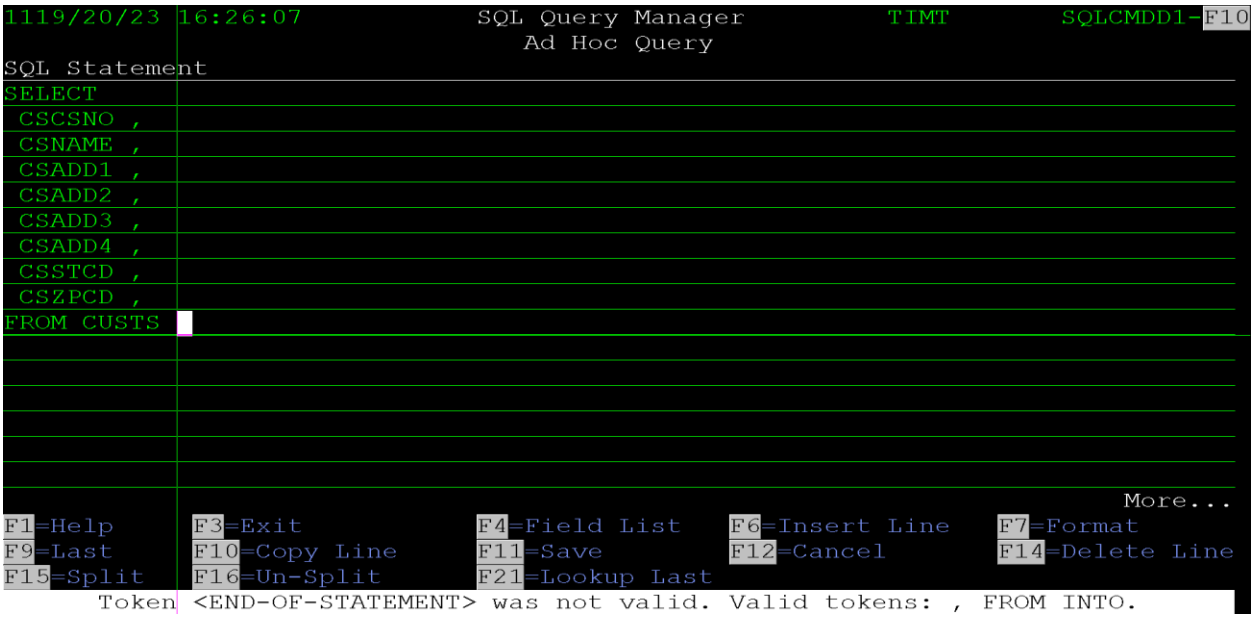
F14 = Delete Line - This deletes the line the cursor is on.

F15 = Split Line - This splits an existing line to a new line. The split is done at the location the cursor is on. This can be useful if a line is getting long.

F16 = Un-Split - This combines two lines together. The data must fit on one line. This can also be useful if F7 was used, and a few lines need to be cleaned up.

F21 = Lookup Last - Displays a list of the last run queries and allows selection of one. Selecting a previously run query loads it to the Ad Hoc Query screen.

If an SQL statement is entered with an error the error is displayed at the bottom of the screen and the cursor is positioned to the error. The following screen shot shows an example of this.

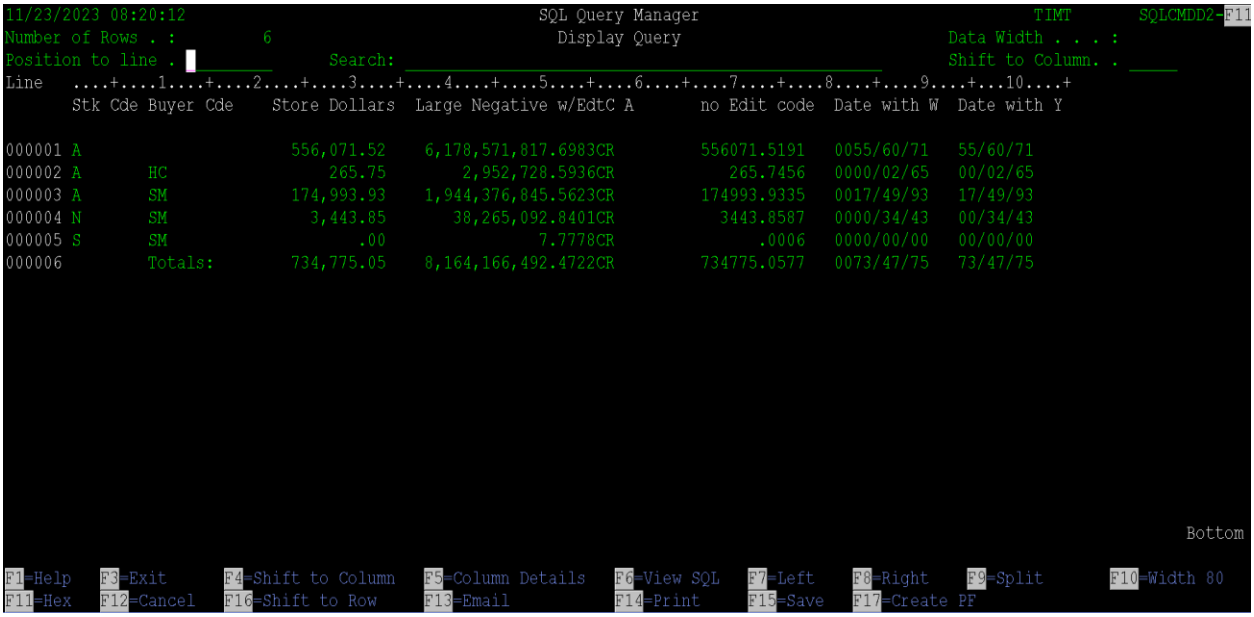


This common error is produced because the last field in the list has a coma after it. In this instance the SQL engine assumes that FROM is a field in the file and CUSTS is an alias for that field, so it thinks that the statement is missing the From clause. That is why the cursor is positioned at the end of the statement.

Display Query Output Screen

The display query output screen is displayed anywhere that the output from a query is displayed on the screen. This can be from running a saved query or ad hoc query, via the command line interface with an action of display or from with the SQL Manager tool.

The display screen looks like this:



The screen displays the results of a query. It shows the number of rows selected in the upper left-hand corner. It allows positioning and searching within the data. The data can be emailed, printed, saved to the IFS or saved in a physical file.

The following are the options on the screen:

Position to line - This field allows the query to be positioned to one of the line numbers on the left-hand side of the screen. It allows the following options to be entered:

- ***TOP, TOP, or T** - These positions to the top of the data. It is the same as entering a 1 to position to the first line.
- ***BOTTOM, *BOT, BOTTOM, BOT, or B** - Any of these position to the bottom of the list.
- **-nnn** - A negative sign followed by a number positions the list back that many rows. For example, if the top row displayed on the screen is 100, then -20 would position the screen to line 80.
- **+nnn** - A plus sign followed by a number positions the list forward that many rows. For example, if the top row displayed on the screen is 100, then +20 would position the screen to line 120.
- **nnn** - A number entered here positions the screen to start the data at the entered line number.

Search - The search field allows filtering the data displayed. The filter works on all fields in the data. The search string also allows multiple words to be entered at once. This means that you can search using more than one selection criteria.

For example, if a query displays a list of customers and their addresses, you can search for both a city and a state by entering both on the search line separated by a space.

Since the search field works on all columns, more data may be displayed than is desired. For instance, if you have a customer list with a city and state column and you want to only include customers in Miami Florida, you can enter 'MIAMI FL' in the search field. However, if a customer exists in Miami Oklahoma and their name is PAM'S FLOWER SHOP, they will still show up. This is because the city will contain MIAMI and the customer's name contains the FL so it meets the entered search criteria.

If the query contains any columns that have a space in the column header, the search field will not be displayed. This is because the query select will not work on a column with a space in the name. The WHERE clause would look something like this: WHERE ORDER NUMBER=1234, the space in the name makes the statement contain invalid SQL.

If the query uses a CTE that cannot be made into a view the search string will not be displayed. This is because a CTE cannot be used as a subquery. To get around this, the system tries to make a temporary view out of the SQL statement, but not all queries can be made into a view. For example, views cannot reference logical files or have an order by clause.

Shift to Column -This field allows for positioning to a specific column in the data.

Type a number 1 through 99999 that identifies the number of the column position you want to appear as the first column on the left side of the report display. Type a number preceded by a plus or minus sign (+n or -n) for relative positioning to the right or left on the display.

For example, if you:

- Type 50 and press Enter, column 50 will become the first column on the left side of the display.
- Type +50 and press Enter, the report is moved 50 columns toward the right.
- Type -50 and press Enter, the report is moved 50 columns toward the left.

The column indicators in the scale line at the top of the display provide column position information. Pluses (+) indicate the fives position, and the number (for example, 1, 2, 3, and so on) identifies the tens position.

The F4 function key can also be used to position the screen to the column the cursor is on.

The command can display a maximum of 99,999 column positions of information.

Most other options on this screen are provided via function keys. The following list provides details about what each function key does.

F1 = Help - This key displays the online help text for the field the cursor is on. This only works if the help text is loaded on the system.

F3 = Exit - F3 exits the program and returns to the screen that the data was displayed from.

F4 = Shift to Column - This shifts the screen to the column position the cursor is on. It is useful if a column is cut off at the end of the screen. Putting the cursor on the first position of the column and pressing F4 will make that column the first column on the screen.

F5 = Column Details - Pressing F5 will display a screen that lists details about each column in the query. The screen displays the column name, type length, decimal positions, width and offset within the displayed data.

F6 = View SQL - Pressing F6 will display the SQL statement passed to the display program.

F7 = Left – This shifts the screen horizontally one screen width to the left.

F8 = Right – This shifts the screen horizontally one screen width to the right.

F9 = Split - This causes the list of data to be split between the right and left side of the screen. The split happens at the location of the cursor when F9 is pressed. Once the data is split the horizontal positioning only works for the data on the right side of the split.

This is useful if the data contains a key field as the first column and multiple screens of data to the right of that. For example, if a customer list includes a lot of columns, the data can be split to freeze the account number on the left-hand side of the screen while scrolling horizontally through the data on the right side of the screen.

Pressing F9 again will remove the split from the data.

F10 = Width 80/132 – F10 toggles between an 80 column and 132 column screen. This option will only be allowed if the terminal device allows displaying 132 columns.

F11 = Hex - F11 toggles displaying the hex value of columns. This is only useful if the data contains any binary fields. When pressed each row of the data is split to 3 rows, with a space between them. The first row shows the character value of the data, the second row shows the first character of the hex value of the character and the third row shows the second character of the hex value of the character.

F16 = Shift to Row - This causes the screen to be positioned vertically to the row that the cursor is on. The row the cursor is on becomes the first row on the screen.

F13 = Email - This option displays a new screen that allows the data set to be emailed. See the Email Options subsection below for more details.

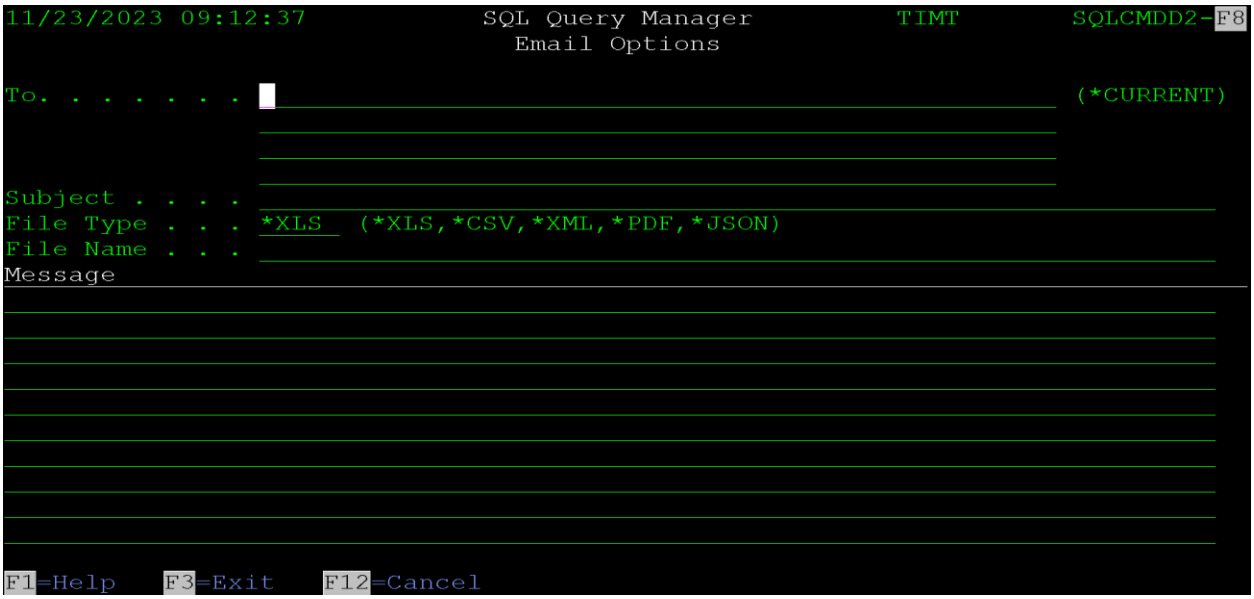
F14 = Print - Allows the results of the query to be printed. A new screen is displayed that allows entry of the printing parameters. See the Print Options sub section below.

F15 = Save - Allows the data set to be saved to a folder in the IFS or a shared network folder. A new screen will be displayed that allows entry of the save options. See the Save as Stream File in IFS subsection below for more information.

F17 = Create PF - This option displays a new screen that will allow the results of the query to be saved to a new physical file on the server. A screen will be displayed to enter the file and library name.

Email Options Screen

The results of a query can be emailed via F13 from the Display Query screen. The following screen is displayed so the parameters can be entered for the email.



The screen allows these parameters to be specified:

To - The To section allows up to 4 email addresses to be specified. If the system has a way to try to pull the email address for a user, *CURRENT will send the email to the email address of the user signed in. Otherwise enter the email address to send the email to.

Subject - Allows a subject to be entered for the email.

File Type - Allows the type of output to be changed. The options are:

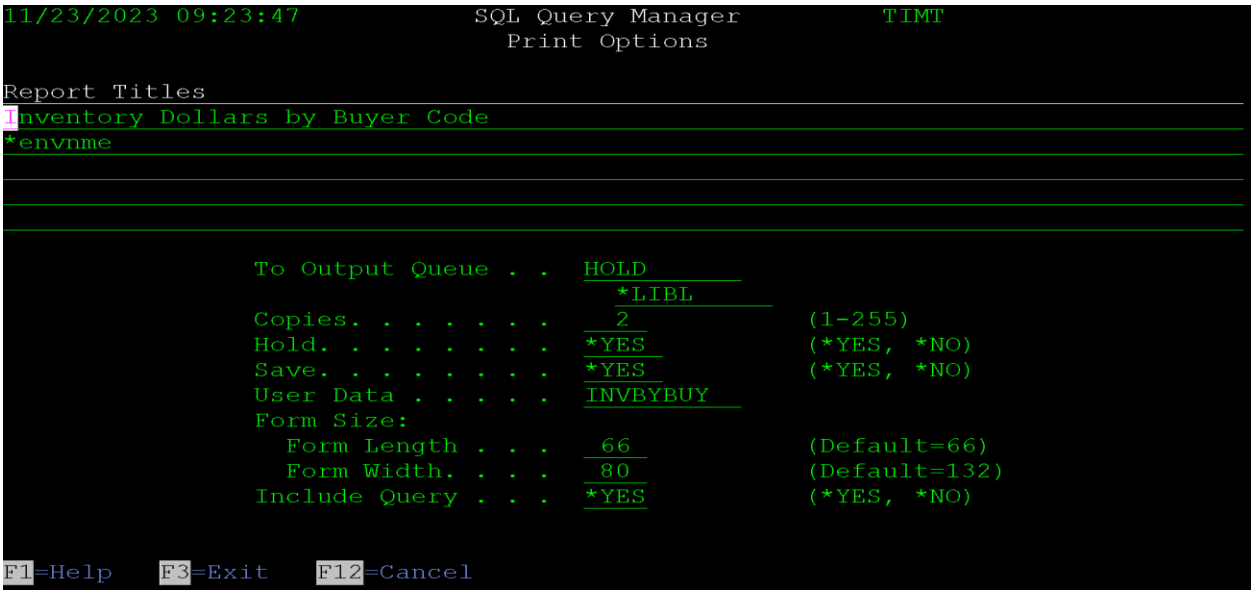
- ***XLS** - An Excel file will be created from the query results and attached to the email.
- ***CSV** - The data will be attached to the email as a CSV file (Comma Separated Variable).
- ***XML** - An XML file formatted for Excel will be created from the data. This has been replaced with the Excel option and was left in the command for backwards compatibility. It is best to use the native Excel option going forward.
- ***PDF** - A PDF file will be created from the data. If the data does not fit on one line of the generated report, it will be truncated.
- ***JSON** - A file containing JSON data will be created from the query results and attached to the email.

File Name - This allows the file name of the attachment to be entered. If left blank the system will generate a unique temporary name for the file. The system will add the appropriate PC file extension to the file, so it is not needed in the name.

Message - These lines of data allow for a message to be added to the body of the email.

Print Options Screen

The following screen is displayed when using F14 = Print form the query Display Query Screen. This screen allows the output from a query to be printed.



The screen allows these options to be entered:

Report Titles - These titles will be printed centered at the top of the report. If the query being displayed is from a saved query in the SQL Query Manager program, then default values may already be populated here. Otherwise, the user can enter up to 5 title lines for the report. The special value of *ENVNME will cause the program to attempt to pull the environment name from a message file called EINVRONMNT with a message ID of SSI0000 from the library list.

Output Queue and Library - These fields allow for entry of the output queue and library. Special values of *LIBL and *CURLIB are allowed. The program will verify that the entered output queue exists.

Copies - The number of copies to print.

Hold - The hold option used on the spooled file.

Save - The save option used on the spooled file.

User Data - The user data used on the spooled file.

Form Size, Length and Width - These override the page length and width used on the spooled file. Any data that does not fit in the width will be truncated.

Include Variables - Entering *YES here will cause the printout to include any variable setup on the query and their overridden values.

Include Query - Entering *YES here will cause the printout to include the SQL statement being run.

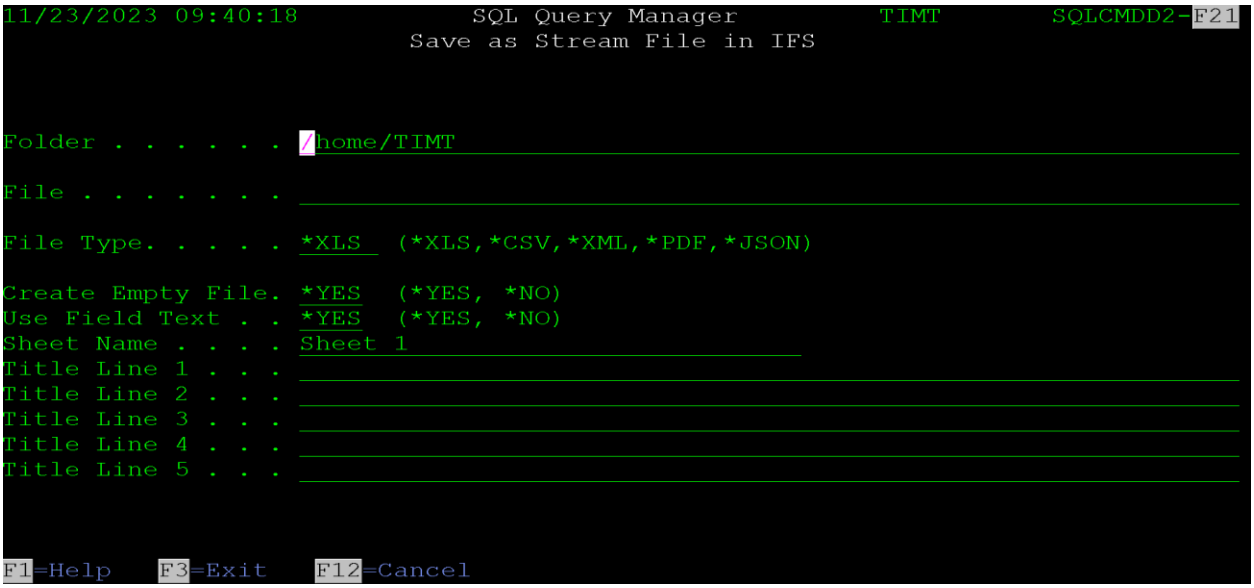
The following is an example of a printed query. The include query options was set to *NO to keep the output size down.

```
2023/11/18  7:40:32      Inventory Dollars by Buyer Code            PAGE:     1
                              Cobalt Boats - Kansas

Stk Cde Buyer Cde    Store Dollars  Large Negative w/EdtC A       no Edit code
A                      556,071.52    6,178,571,817.6983CR         556071.5191
A       HC                 265.75        2,952,728.5936CR            265.7456
A       SM             174,993.93    1,944,376,845.5623CR         174993.9335
N       SM               3,443.85       38,265,092.8401CR           3443.8587
S       SM                    .00                7.7778CR                .0006
        Totals:        734,775.05    8,164,166,492.4722CR         734775.0577
```

Save as Stream file in IFS Screen

The following screen is displayed when F15=Save is pressed from the Display Query screen. This screen allows the results from a query to be saved to an IFS folder or shared network folder as a PC stream file.



The screen allows the following options to be entered:

Folder - This field allows entry of the folder where the file will be created. Using the QNTC file system allows saving the file to a shared network folder.

File - This field allows entry of the file name for the created file. The correct file extension will be added based on the file type, so it is not necessary to include it in the file name.

File Type - This designates the type of file that will be created. The options are:

- ***XLS** - An Excel file will be created from the query results and attached to the email.
- ***CSV** - The data will be attached to the email as a CSV file (Comma Separated Variable).
- ***XML** - An XML file formatted for Excel will be created from the data. This has been replaced with the Excel option and was left in the command for backwards compatibility. It is best to use the native Excel option going forward.
- ***PDF** - A PDF file will be created from the data. If the data does not fit on one line of the generated report, it will be truncated.
- ***JSON** - A file containing JSON data will be created from the query results and attached to the email.

Create Empty File - Entering *NO here will not create the file if the query does not return any rows. *YES will create an empty file.

Use Field Text - This option allows using the field text for the column headers. By default the program will use field names or aliases if entered in the SQL statement. Entering *YES here changes the program to use the text defined in the file.

Sheet Name - This option is used when creating an Excel or XML file. It sets the tab name in the spreadsheet.

Title Lines 1 - 5 - These title lines are included at the top of an Excel, XML or CSV file. They are also included as report headers at the top of PDF output. In a JSON file they are included as keys in the root element before the data array.

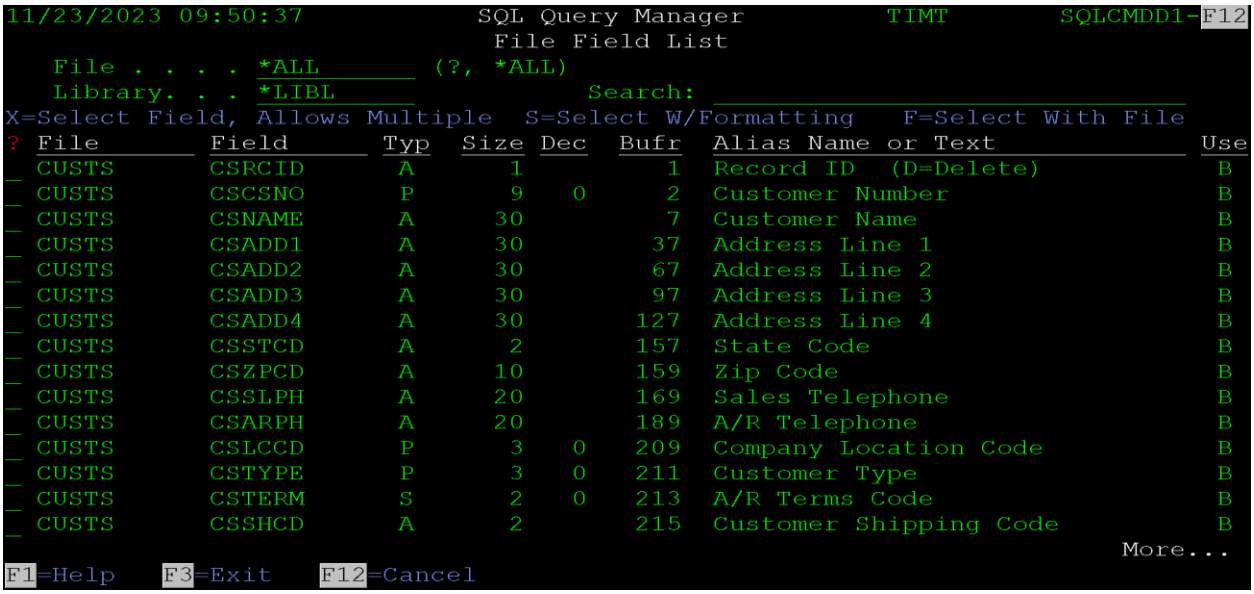
Enter these options and press enter to generate the file.

Field Selection Screen

The field selection screen allows the user to select a field or a list of fields from one of the files used in the query. It can be called from the Ad Hoc query entry screen or the saved query entry screen using option F4=Field List.

The screen lists all fields in the files used in the query. It figures out what the files are by inspecting each word in the query and seeing if there is a file named that in the library list. This can have the side effect of sometimes pulling in extra files if a file is named after one of the reserved words used in SQL. For example, one system this is run on has a file named CASE, so if a CASE statement is used in the query, the field selection screen includes all fields from the CASE file as well.

The screen looks like this:



This screen was displayed by pressing F4 on a query that includes the CUSTS file. Since that was the only file in the query, only fields for the CUSTS file are displayed.

The main function used on this screen is just selecting fields with the X option from the SFL. Option 1 works the same as X for those used to using standard IBM option numbers. This returns a comma separated field list to the calling program. The calling program will insert that list into the SQL statement at the location of the cursor.

The F option works similarly, but each field in the list is qualified with the file name. For example, if the CSCSNO field is selected with an S the field returned to the calling program will be CUSTS.CSCSNO instead of just CSCSNO.

The S option is used for creating pretty formatted columns from the selected fields. It displays another screen that allows field aliases to set up for each field and applies some custom SQL functions to format the data in the column. The SQL UDF's it uses are not loaded by default when the SQL command is created on the system, so some custom work may need to be done to use these functions. This functionality will be displayed in the example usage below.

The file filed can be used to display the fields for only one file in the query. It defaults to *ALL which causes the list to include all fields in all files in the query.

The search field allows the list to be filter based on entered search criteria. This is useful when looking for a specific field. For example, in the list on the screen above one might want to find the date customer was entered on. If they did not know the name of the field, they could enter date in the search field and the list would be filtered like this:

```
11/23/2023 15:42:58            SQL Query Manager          TIMT        SQLCMDD1
                                File Field List
   File . . . . *ALL        (?, *ALL)
   Library. . . *LIBL                 Search: DATE
X=Select Field, Allows Multiple  S=Select W/Formatting   F=Select With File
? File        Field       Typ   Size  Dec   Bufr   Alias Name or Text
_ CUSTS       CSLSCR       S       8    0    328   Last Payment Date
_ CUSTS       CSLSAD       S       8    0    336   Last Adjustment Date
_ CUSTS       CSLSIN       S       8    0    344   Last Invoice Date
_ CUSTS       CSLSCM       S       8    0    352   Last Credit Date
_ CUSTS       CSSUDT       S       8    0    360   Set up Date
```

This shorter list of fields makes it obvious that the CSSUDT field is the setup date for the account.

Since the field list behaves differently based on where the fields will be inserted, the following documentation will show how they work by examples.

Simple Field Selection - Example

In its simplest form, the field selection screen is used to pull a list of fields and insert them as a comma separated list in an SQL command. This example will walk though doing an ad hoc query on the file named CUSTS. This file contains customer information, and we are going to assume that we are wanting to create an address list from the file.

To quickly get to the ad hoc query screen with a default query already generated, use the command `SQL CUSTS`. This displays the following statement in the ad hoc screen of the SQL command:

Command entry example:

```
Selection or command
===> SQL CUSTS
```

Displays this screen:

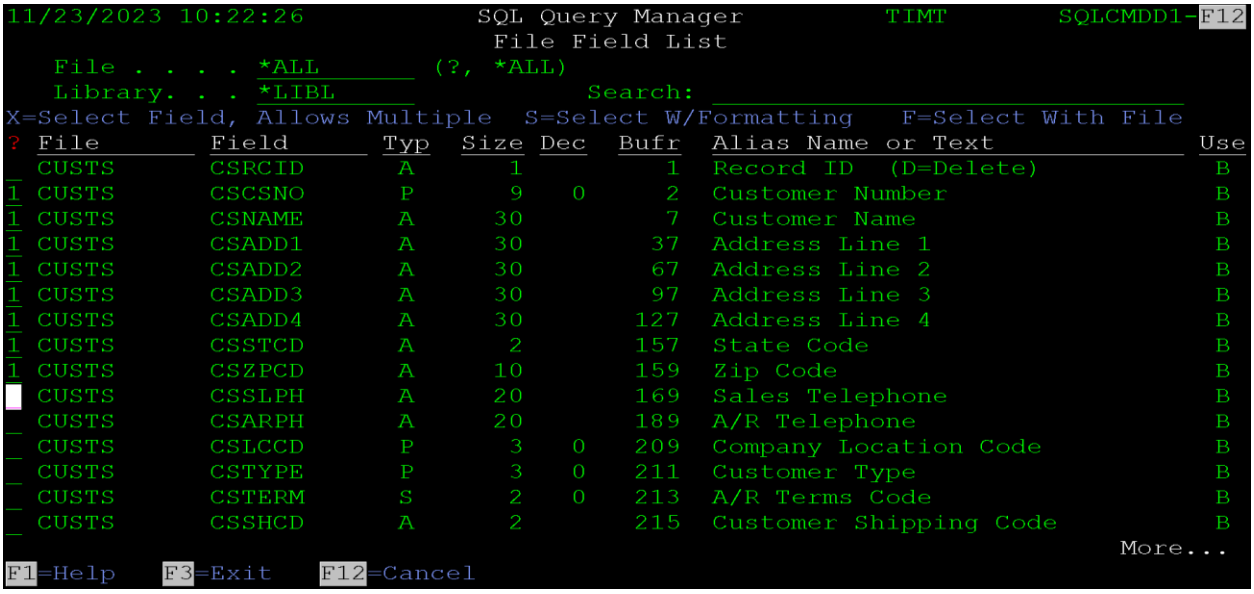
```
11/23/2023 10:08:06                SQL Query Manager
                                      Ad Hoc Query
SQL Statement
SELECT *
FROM CUSTS
```

The program defaults to pulling all fields. This is useful for a quick look at the data in a file, but we do not want to list all fields in the file, instead we want to list just the customer number, name and address fields.

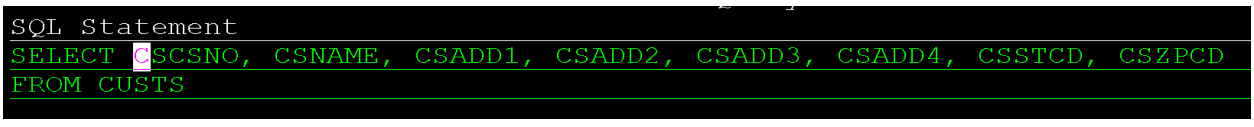
To do this we will use the field selection option to select the fields we want to list. Before pressing F4, it is best to remove the asterisk. This is because the command will append the field list after where the cursor is positioned, and we do not want the command to be `SELECT * CSCSNO, ....` We just want it to be `SELECT CSCSNO, . . .`. So arrow over and remove the asterisk and leave the cursor one space after the SELECT before pressing F4.

The screen should look like this before pressing F4.

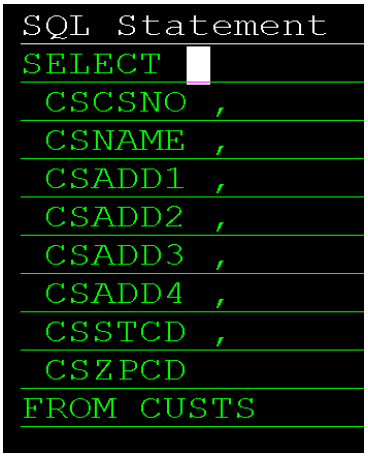
Then the field selection screen is displayed. Enter a 1 in each column that we want to list in the query. It should look like this:



Pressing enter will return the field list to the query. The query will now look like this:



The fields are added as a column separated list. Often it is desirable to include each field on its own line. This can be accomplished using the F7=Format option. Pressing F7 now will change the query to look like this:



At this point the query can have column names (aliases), selection clause (WHERE) and ordering (ORDER BY) clauses added to it.

The field list can be used to select multiple where conditions at once and to build a formatted list of fields for the ON clause of a join. See the Field Selection for a Where Clause and Field Selection for an On Clause sections of this document for examples.

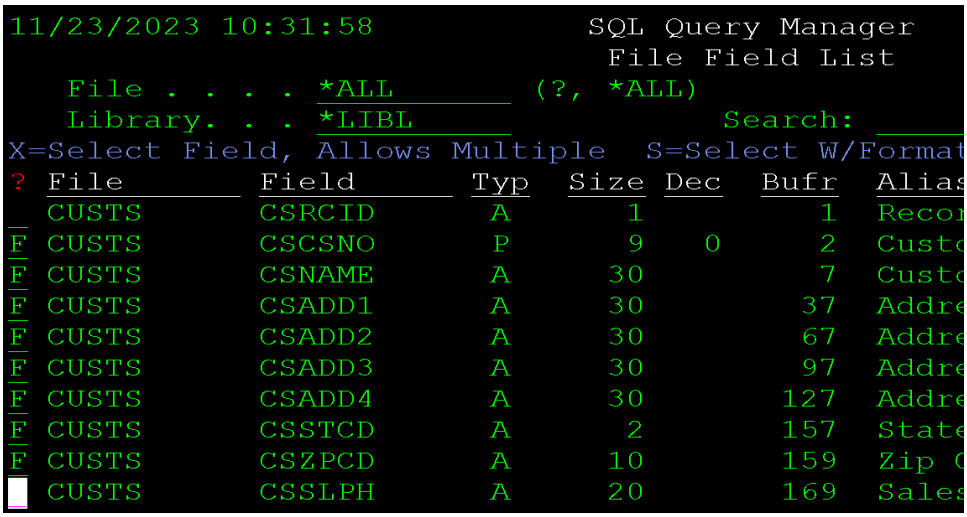
The fields listed for the columns in a query can also include the file name qualifier and a formatted fields list. See the next 2 sections of this document for examples of those uses

Simple Field Selection with File Names

This section shows how to select fields with file name qualifiers. This works very similar to the previous simple field selection section of this document. So, this section will only include the differences.

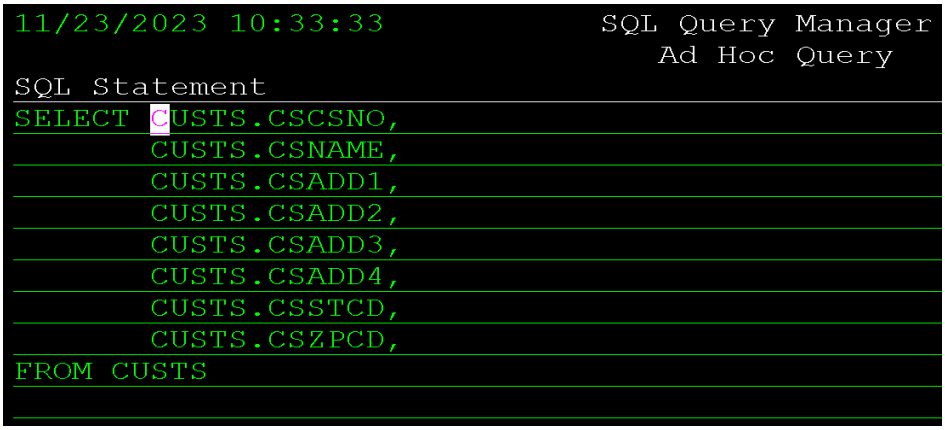
Starting with a query like this:

Press F4 and the field selection screen will be displayed. Enter an F in the selection column for each field that is desired. The screen should look like this:



Press enter and a screen will be displayed that shows the columns selected. This screen can just be ignored, so press enter on it.

The selected fields will be inserted into the original query, one line per field and formatted like this:



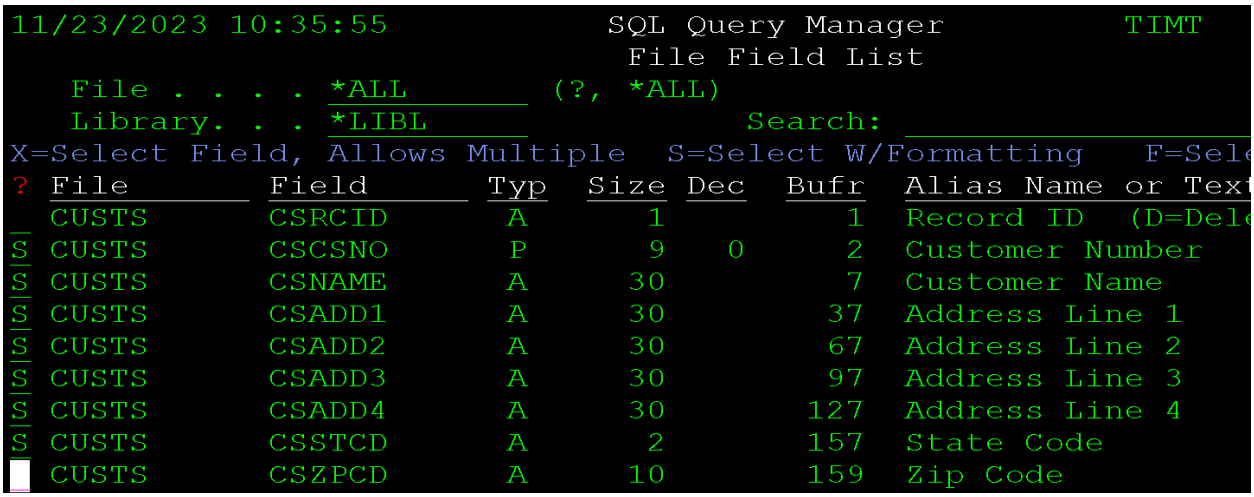
Field Selection with Formatting

When selecting fields for a query it often desirable to format the columns with column headers and format the data to display in a format more familiar to humans. The field selection screen provides the S=Select W/Formatting option to make this task easier.

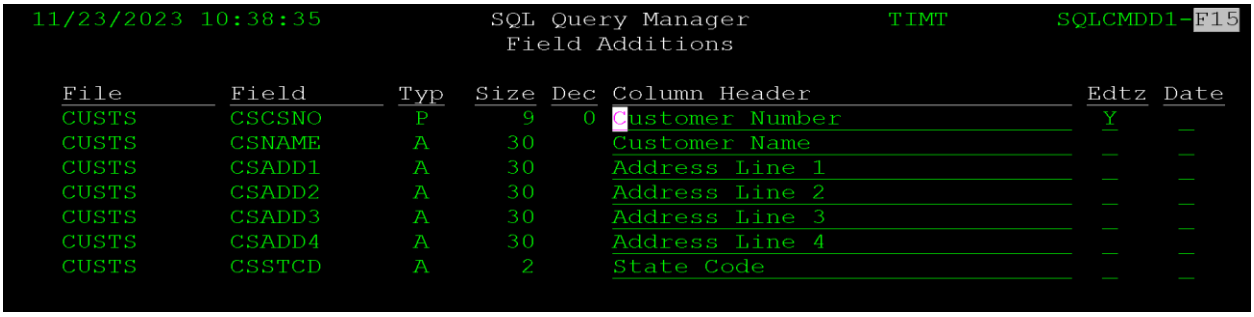
The following example extends the previous 2 examples by selecting the fields and adding formatting automatically.

Just like the previous 2 examples, start with a query like this and press the F4 key.

The same list of fields is displayed, but this time they will be selected with the S option like this:



Pressing enter on this screen will display the field addition screen that allows column aliases to be changed as well as some formatting options to be added to each field. The screen looks like this:



The Column Header column allows the alias name for each column to be modified.

The EDTZ column allows the field to be wrapped in an SQL UDF called EDTZ. This attempts to format the data as integer-like data with no decimal position. It also limits the width of the column to the width of the column selected from the table. Do not use this option if the SQL UDF EDTZ has not been set up on your system.

The date column wraps the column in a function called FDATE. This attempts to format the date as a human readable date by insert slashes. This is intended to be used on numeric columns containing date information. For example, in the customer file above, the last payment date is stored in column CSLSCR which is a Decimal(8) column with the data formatted as a YYYYMMDD date. So, 20231130 represents November 30th 2023. The FDATE function will display the value as 2023/11/30. This only works if the FDATE function has been loaded on the system.

Pressing enter on the screen above will return to the query entry screen and insert the selected columns with formatting. With the columns from the screen above the query will now look like this:

```
11/23/2023 10:49:05                SQL Query Manager                TIMT
                                      Ad Hoc Query
SQL Statement
SELECT SUBSTR(EDTZ(CUSTS.CSCSNO,9),1,9) AS "Customer Number",
       CUSTS.CSNAME AS "Customer Name",
       CUSTS.CSADD1 AS "Address Line 1",
       CUSTS.CSADD2 AS "Address Line 2",
       CUSTS.CSADD3 AS "Address Line 3",
       CUSTS.CSADD4 AS "Address Line 4",
       CUSTS.CSSTCD AS "State Code",
FROM CUSTS
```

Since fields are often inserted in an existing query, the last column still includes a comma. Since this example doesn't have any more fields that comma needs to be removed.

Notice that each column has an alias with the column header populated. The customer number is also formatted with the EDTZ function and a length. The system this is run on does not have the EDTZ function so that needed to be removed. Then when the query is run the data looks like this:

```
11/23/2023 10:51:42                              SQL Query Manager
Number of Rows . :      1944                       Display Query
Position to line . _________      Search: ____________________________________________
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....
       Customer_Number Customer_Name                Address_Line_1                Address_L

000001       100       CASH SALES
000002       500       AAA MARINE,INC             PO BOX 246                    LAMPE, MO
000003      1000       A & A SALES & SERVICE      4650 RENO RD                  NEODESHA,
000004      2900       ACF INDUSTRIES             SOUTH GERARD ST.              BUDE, MS
000005      3000       ACF INDUSTRIES             MAIN & CLARK ST               ST CHARLE
000006      3100       ACF INDUSTRIES             3301 RIDER TRAIL SOUTH        EARTH CIT
000007      3200       ACF INDUSTRIES             1101 BEDFORD RD               NORTH KAN
```

Field Selection for a Where Clause

When Selecting fields for a where clause the program tries to format the data in a manner consistent with the entry of a where clause.

To show an example of this use, we are going to select customers in Oklahoma that start with an A from the CUSTS file used in previous examples. To keep the example small, we will just select all fields with * option of the select clause. See the previous sections for column selection assistance.

Starting with an SQL command like this:

```
SQL Statement
SELECT *
FROM CUSTS
where
```

With the cursor directly after the where clause, press F4. The field selection screen will be displayed. Enter an X or 1 in the column in front of the Customer Name (CSNAME) and State Code (CSSTCD) columns and press enter. The query will be redisplayed like this:

```
SQL Statement
SELECT *
FROM CUSTS
where CSNAME = ''
  AND CSSTCD = ''
```

The program added the two selection fields on their own lines with an AND between each one. This is intended to speed the entry of selection criteria. It has no way of knowing if you want an = comparison or a like or what you want to compare the data to. If a field is numeric, it adds fieldName = 0 to the where clause. If the field is alphanumeric, it adds fieldName = '' to the clause.

To accomplish our goal of only selecting customer in Oklahoma that start with an A we would manually modify the statement to look like this:

```
SQL Statement
SELECT *
FROM CUSTS
where CSNAME like 'A%'
  AND CSSTCD = 'OK'
```

Field Selection for a On Clause

When joining another file in the SQL command, it helps to be able to select the fields to join on from a list. If the F4=File Selection screen is used after the keyword on, the format of the returned fields is changed to assist in the creation of the on the on clause.

Starting with a query like this:

```
SQL Statement
SELECT *
FROM SLHDR
left join CUSTS on
```

Pressing F4 will display the same field list selection screen. However, when using option X or 1 to selection fields the program will format the returned field list like this:

```
SQL Statement
SELECT *
FROM SLHDR
left join CUSTS on CSCSNO=CSCSNO AND
                   CSNAME=CSNAME
```

For example purposes both the customer number (CSCSNO) and customer name (CSNAME) columns were selected for the join.

The query entry screen added the fields as a list of fileName=fieldname, added the and keyword between each one, and added each additional field as it's one line. It is formatted this way to simplify the creation of join statements. The second field on the equals clauses still needs to be changed to the corresponding field names in the SLHDR file. The program cannot match those fields up automatically.

Multi-Tab Excel File Example

The following example demonstrates how to create a multi tab-excel file using the SQL command line tool. This example uses an RPG program, a custom view and the #$CMD RPG procedure to create and email an Excel file with multiple tabs in it.

The example is going to create an excel file of a customer aging. It will have a tab for all open balances, past due account, and zero balance accounts.

To make the RPG program shorter (less complicated), the program will use an SQL View that handles the selection and formatting of the file columns.

To apply standard naming conventions the objects created will be:

- **AGERPTB1VW** - The view used by program AGERTPB1
- **AGERPTB1** - The program that created the Excel file and emails it.

The view will be created with these SQL commands:

```
Create or replace view TIMLIB/AGERPTB1VW as
Select
  CSCSNO "Customer Number",
  CSNAME "Cuztomer Name",
  CSSTCD "State",
  CSCRBL "Total Balance",
  CSBAL1 "Current Balance",
  CSBAL2 "1-30 Days",
  CSBAL3 "31-60 Days",
  CSBAL4 "61-90 Days",
  CSBAL5 "Over 91 Days"
From CUSTS;

Label on Table TIMLIB/AGERPTB1VW is 'Customer Aging View';
```

These SQL statements will be put in source member AGERPTB1VW for source code management reasons.

The RPG program will be called AGERPTB1. It looks like this:

```
0001.00 **free
0002.00 Ctl-Opt datedit(*ymd) Option(*NoDebugIO:*NoShowCpy:*SrcStmt) DftActGrp(*No) BndDir('CBT') Main(Main);
0003.00
0004.00 // SQL command RPG Example - Creating a Multi-Tab Spreadsheet
0005.00 // this program is an example of using the SQL command to create a multiple tab spreadsheet
0006.00 // and email email to someone.
0007.00
0008.00 /Copy qsrc,CBTFNCV1PR  // prototypes for all #$ procedures
0009.00
0010.00 Dcl-Proc Main;
0011.00   Dcl-S sqlStm1 varchar(500);
0012.00   Dcl-S sqlStm2 varchar(500);
0013.00   Dcl-S sqlStm3 varchar(500);
0014.00   Dcl-S subject varchar(100);
0015.00   Dcl-S title3 varchar(100);
0016.00   Dcl-S email varchar(50) inz('tognazzini@hotmail.com');
0017.00
0018.00   // Build the SQL statements for each tab. They will all pull from the view created for
0019.00   // this program. The only difference will be the selection criteria.
```

```
0020.00   // notice that the field names need to use the aliases in the correct case in double quotes.
0021.00   // this is because the view formats the columns that way.
0022.00   sqlStm1 = 'Select * from AGERPTB1VW where "Total Balance" <> 0';
0023.00   sqlStm2 = 'Select * from AGERPTB1VW +
0024.00             where "1-30 Days" <> 0 +
0025.00                or "31-60 Days" <> 0 +
0026.00                or "61-90 Days" <> 0 +
0027.00                or "Over 91 Days" <> 0';
0028.00   sqlStm3 = 'Select * from AGERPTB1VW where "Total Balance" = 0';
0029.00
0030.00   // These variables just create some the values included in the SQL command
0031.00   // below. It often makes the program less cluttered to include the creation of special
0032.00   // values outside the command call, as opposed to inserting all the logic into the
0033.00   // call. The message parameter of the command below is an example of building the values
0034.00   // inside the command instead of outside of it. It appends the data directly into the
0035.00   // message parameter instead of building it in a separate variable and then including it.
0036.00   subject = 'Customer aging as of ' + %char(%date():*usa);
0037.00   title3 = 'Run Date: ' + %char(%date():*usa);
0038.00
0039.00   // The #$CMD procedure just runs an AS400 command from within an RPG Program.
0040.00   // the following statement uses the SQL command to create an Excel file with
0041.00   // a tab for each of the SQL statements above. It sets the tab name and titles
0042.00   // for each of the tabs. The titles include the Tab Title, environment name, and
0043.00   // the date the report was run on. Each tab is named something helpful.
0044.00   #$CMD('+
0045.00   SQL FILE(''' + #$DBLQ(sqlStm1) + ''') +
0046.00       ACTION(*EMAIL) +
0047.00       EMAIL(' + email + ') +
0048.00       FILENAME(''Customer Aging Report'') +
0049.00       TYPE(*XLS) +
0050.00       SHEET(''Open Accounts'') +
0051.00       TITLE1(''Customer Aging - All Open Balances'') +
0052.00       TITLE2(*ENVNME) +
0053.00       TITLE3(''' + #$DBLQ(title3) + ''') +
0054.00       SUBJECT(''' + #$DBLQ(subject) + ''') +
0055.00       MESSAGE(''Attached is the customer aging report.<br><br>It +
0056.00               was ran for:' + %char(%date():*USA) + ''' *TEXTHTML) +
0057.00       ADDSQL(+
0058.00         ('''+#$DBLQ(SQLSTM2)+''' +
0059.00          ''Past Due Accounts'' +
0060.00          ''Past Due Customer Aging'' +
0061.00          *ENVNME +
0062.00          '''+ TITLE3 + ''') +
0063.00          ('''+#$DBLQ(SQLSTM3)+ ''' +
0064.00           ''Zero Balance Accounts'' +
0065.00           ''Zero Balance Accounts'' +
0066.00           *ENVNME +
0067.00           '''+#$DBLQ(TITLE3)+''' +
0068.00           ) +
0069.00       )');
0070.00
0071.00 End-Proc;
```

Most of the work is done in the SQL command. The ADDSQL parameter is used to add the second and third tab. Despite the length of this program, there are only 7 actual calculation lines included. Most of the code is comments and declarations. In fact, the entire program could have been written as one call to #$CMD, with the full command built there. Spreading it out some can make the code more clear.

The ADDSQL parameter is a multi-element parameter that allows up to 300 additional SQL statements to be added as additional tabs when creating an Excel file. It is best not to use more than about 10 tabs. Navigating an Excel file with a bunch of tabs can get complicated quickly as the tabs span well past the end of the page and scrolling through them is not always user friendly.

Each instance of the ADDSQL parameter contain these elements:

- **SQL Statement** - This is the SQL statement used to generate the data displayed on the tab.
- **Sheet Name** - This allows entry of the name used on the tab.

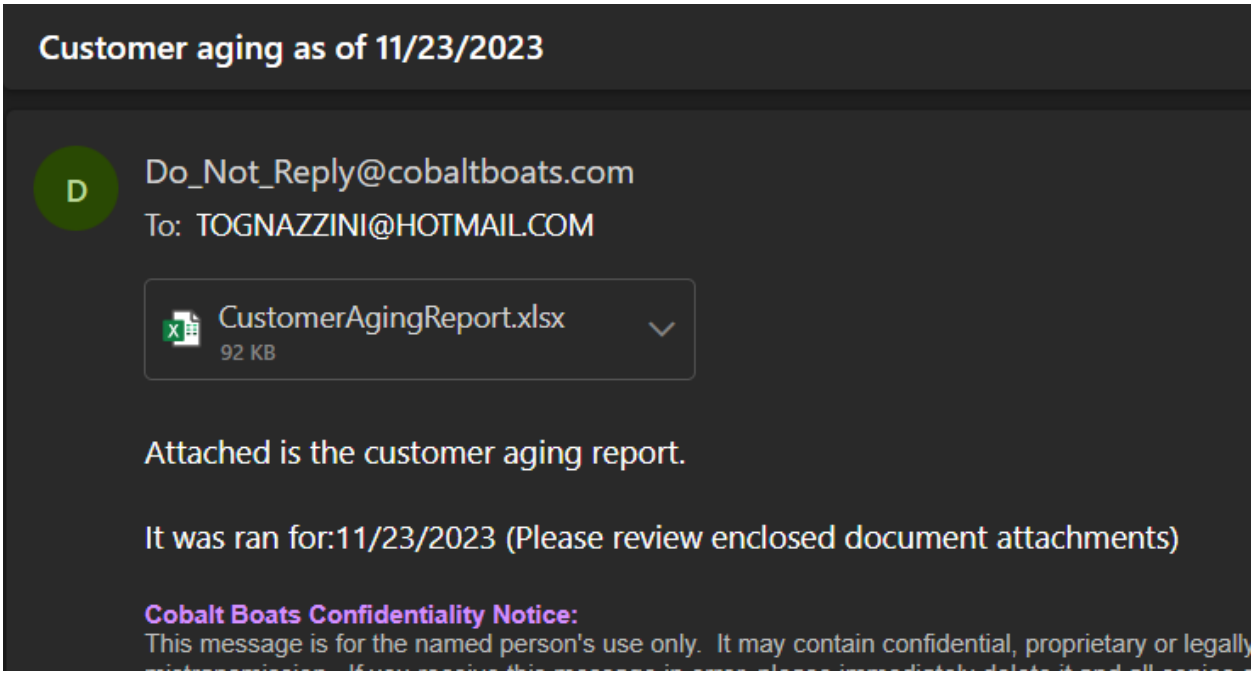
- **Title 1 - 5** - This allows adding custom titles to each tab. Since each tab is showing different data, it is a good idea to include at least one title that states what data is being displayed on the tab.

This is often easier to visualize in a CL program so here the exact program from above written in CL instead of RPG:

```
0001.00 PGM
0002.00
0003.00   DCL &SQLSTM1 *CHAR 500
0004.00   DCL &SQLSTM2 *CHAR 500
0005.00   DCL &SQLSTM3 *CHAR 500
0006.00   DCL &subject *CHAR 100
0007.00   DCL &title3  *CHAR 50
0008.00   DCL &date    *CHAR 6
0009.00   DCL &fmtdate *CHAR 10
0010.00
0011.00   /* get the date in a pretty format */
0012.00   RTVJOBA    DATE(&DATE)
0013.00   CVTDAT     DATE(&DATE) TOVAR(&FMTDATE) TOFMT(*YYMD) TOSEP('/')
0014.00
0015.00   /* Build the SQL statements for each tab. They will all pull from the view created for       +
0016.00      this program. The only difference will be the selection criteria.                         +
0017.00      notice that the field names need to use the aliases in the correct case in double quotes.  +
0018.00      this is because the view formats the columns that way.                                    */
0019.00   chgvar &sqlStm1 ('Select * from AGERPTB1VW where "Total Balance" <> 0')
0020.00   chgvar &sqlStm2 ('Select * from AGERPTB1VW +
0021.00                     where "1-30 Days" <> 0 +
0022.00                        or "31-60 Days" <> 0 +
0023.00                        or "61-90 Days" <> 0 +
0024.00                        or "Over 91 Days" <> 0')
0025.00   chgvar &sqlStm3 ('Select * from AGERPTB1VW where "Total Balance" = 0')
0026.00
0027.00   /* These variables just create some the values included in the SQL command               +
0028.00      below. It often makes the program less cluttered to include the creation of special    +
0029.00      values outside the command call, as opposed to inserting all the logic into the        +
0030.00      call. The message parameter of the command below is an example of building the values  +
0031.00      inside the command instead of outside of it. It appends the data directly into the     +
0032.00      message parameter instead of building it in a separate variable and then including it. */
0033.00   chgvar &subject ('Customer aging as of ' *cat &fmtdate)
0034.00   chgvar &title3  ('Run Date: ' || &fmtDate)
0035.00
0036.00   SQL FILE(&SQLSTM1) +
0037.00       ACTION(*EMAIL) +
0038.00       TITLE1('Customer Aging - All Open Balances') +
0039.00       TITLE2(*ENVNME) +
0040.00       EMAIL(TOGNAZZINI@HOTMAIL.COM) +
0041.00       FILENAME('Customer Aging Report') +
0042.00       SHEET('Open Accounts') +
0043.00       SUBJECT(&subject) +
0044.00       MESSAGE(('Attached is the customer aging report.<br><br>It +
0045.00                was ran for:' || &fmtDate)  *TEXTHTML) +
0046.00       ADDSQL(+
0047.00         (&SQLSTM2 'Past Due Accounts' 'Past Due Customer Aging' *envnme &title3) +
0048.00         (&SQLSTM3 'Zero Balance Accounts' 'Zero Balance Accounts' *envnme &title3) +
0049.00         )
0050.00
0051.00 ENDPGM
```

Note that lines 47 and 48 show 2 instances of the ADDSQL parameter being passed. Each instance contains the SQL statement, the tab name and the first 3 title lines. Since title lines 4 and 5 are not being used, they are just excluded.

Running either program will produce an email that looks like this:



The attached file looks like this:

	A	B	C	D	E	F	G	H	I
1	**Customer Aging - All Open Balances**								
2	**Cobalt Boats - Kansas**								
3									
4	**Customer Number**	**Cuztomer Name**	**State**	**Total Balance**	**Current Balance**	**1-30 Days**	**31-60 Days**	**61-90 Days**	**Over 91 Days**
5	100	CASH SALES	00	10,000.00	10,000.00	0.00	0.00	0.00	0.00
6	72500	BILL'S MARINE SERVICE	MD	229,302.15	228,989.60	312.55	0.00	0.00	0.00
7	85500	BOATS BY GEORGE, INC.	NY	250,612.62	233,235.94	1,067.19	16,309.49	0.00	0.00
8	89000	BOAT TOWN, INC.	TX	-878.30	-878.30	0.00	0.00	0.00	0.00
9	203000	AQUA MARINA YACHTS LTD EXP	00	-1,740.37	5,562.49	-4,571.23	-2,399.02	-332.61	0.00
10	207200	GERMAN BROTHERS MARINA	NY	115,331.78	112,864.56	2,537.13	526.62	-1,013.20	416.67
11	294200	LAKE NORMAN MARINA	NC	-23,995.33	-23,995.33	0.00	0.00	0.00	0.00

OPEN_ACCOUNTS PAST_DUE_ACCOUNTS ZERO_BALANCE_ACCOUNTS

The second tab only includes accounts with balances in the last 4 columns. It looks like this:

	A	B	C	D	E	F	G	H	I	J
1	**Past Due Customer Aging**									
2	**Cobalt Boats - Kansas**									
3	**Run Date: 2023/11/23**									
4										
5	**Customer Number**	**Cuztomer Name**	**State**	**Total Balance**	**Current Balance**	**1-30 Days**	**31-60 Days**	**61-90 Days**	**Over 91 Days**	
6	72500	BILL'S MARINE SERVICE	MD	229,302.15	228,989.60	312.55	0.00	0.00	0.00	
7	85500	BOATS BY GEORGE, INC.	NY	250,612.62	233,235.94	1,067.19	16,309.49	0.00	0.00	
8	203000	AQUA MARINA YACHTS LTD EXP	00	-1,740.37	5,562.49	-4,571.23	-2,399.02	-332.61	0.00	
9	207200	GERMAN BROTHERS MARINA	NY	115,331.78	112,864.56	2,537.13	526.62	-1,013.20	416.67	
10	61000	BELGIAN BOAT SERVICE/B -D- EXP	00	-1,274.75	1,856.17	0.00	0.00	-3,000.00	-130.92	
11	294700	LAKESHORE SERVICE	TX	-323.16	0.00	0.00	0.00	0.00	-323.16	

OPEN_ACCOUNTS PAST_DUE_ACCOUNTS ZERO_BALANCE_ACCOUNTS

Finally, the third tab only includes accounts with a zero balance. It looks like this:

	A	B	C	D	E	F	G	H	I	J
1	**Zero Balance Accounts**									
2	**Cobalt Boats - Kansas**									
3	**Run Date: 2023/11/23**									
4										
5	**Customer Number**	**Cuztomer Name**	**State**	**Total Balance**	**Current Balance**	**1-30 Days**	**31-60 Days**	**61-90 Days**	**Over 91 Days**	
6	500	AAA MARINE,INC	MO	0.00	0.00	0.00	0.00	0.00	0.00	
7	1000	A & A SALES & SERVICE	KS	0.00	0.00	0.00	0.00	0.00	0.00	
8	2900	ACF INDUSTRIES	MS	0.00	0.00	0.00	0.00	0.00	0.00	
9	3000	ACF INDUSTRIES	MO	0.00	0.00	0.00	0.00	0.00	0.00	
10	3100	ACF INDUSTRIES	MO	0.00	0.00	0.00	0.00	0.00	0.00	
11	3200	ACF INDUSTRIES	MO	0.00	0.00	0.00	0.00	0.00	0.00	

OPEN_ACCOUNTS PAST_DUE_ACCOUNTS ZERO_BALANCE_ACCOUNTS