Abstract
This talk describes work in-progress on an interactive SQL Query building environment for the SAS® system. It uses full screen menus and point-and-shoot concepts for selecting tables and variables as it leads you through the construction of a query. The constructed query can be run to preview the data selected, saved as program text for inclusion in other jobs, or saved as an SQL View for use at a later time by other SAS procedures.

Background
SQL (Structured Query Language) is a language for accessing data stored in tables. It is a non-procedural language that is implemented by many database systems on mainframes, minicomputers and personal workstations. The SQL language enjoys far greater applications portability than proprietary query languages due to its high degree of standardisation.

Interactive Interfaces
SQL is an "English like" language, but it still has many stylistic conventions that must be followed to compose a valid query. To the uninitiated, SQL is just another programming tool that must be learned before one can achieve useful results.

There is good potential for "front-ending" SQL with other tools. The SAS/EQL English language parser is one such tool. QBE® (Query By Example), implemented by IBM for the SQL/DS® database is another — it allows one to pose queries from a full-screen set of menus and panels.

Objectives of the QUERY window
• An interactive query building tool should allow a user to specify, run, save and reuse database queries. It should operate in a manner that will allow people without any knowledge of SQL to generate queries.
• The tool should generate SQL that is free of syntax errors, so that it can be processed directly by a SQL processor like the SAS Systems PROC SQL.
• It would not hurt if the tool was fun to use, and presented the generated SQL. Users would learn SQL concepts by osmosis!

The “Painting” Metaphor
Many point-and-click systems use and present their screens as a canvas and a few palettes of tools. The user selects from a palette with the mouse, and "paints" on the canvas. The palettes give access to the raw building blocks of the system, and the canvas displays how the user has chosen to combine them. Similar to the way an artist might mask a section of his painting, a user can select a region of the canvas and operate on that selected region only.

The prototype QUERY window uses this model of end-user interaction. Instead of a series of computer-generated questions that the user must respond to, the QUERY window allows the user to control the direction that the session takes. In other words, the QUERY window is non-modal — all the functions presented by the interface are available without the user having to change modes.

The QUERY Canvas
The QUERY window is invoked from the SAS Display Manager with the "QUERY" command, or by selecting it from the GLOBALS pull-down menu. The first window that is displayed is the main canvas, where the user will paint his query.

The window is organised as a template for an SQL query. Various palettes are available from the pull-down menus. The fields on the left-hand side of the template correspond to the clauses of an SQL query. Just as a painter will focus in on one aspect of a painting, the user focuses in on some aspect of the query he is building by selecting one of these areas.

The issue of focus relates to how later selections are interpreted. For example, to specify sorting details, one selects the "Order By" area of the canvas (giving it the focus), and then selects items from a variable or expression palette.

The QUERY Palettes
Tables — This palette is selected from the Palette menu button, or by focussing on the "From" clause in the main canvas. It presents a list of available SAS datasets for the user to select from. When a table is selected from this palette, its name is added to the "From" section of the main canvas, and a variables palette is created for that table.

If more than one table is selected, the QUERY window will construct a default match clause, using all common variables. A possible refinement for this would be to look-up the match specification in some global repository containing these details.

It is sometimes useful to select a table twice, when the query involves combining a table with itself. In this case the query window creates two variable palettes, one for each "instance" of the table used in the query.
In this example, I have selected two tables, and the QUERY window has generated the default match clause, and created a variable palette for each selected table. Each window is sizeable and moveable.

**Variables** — This palette is created when the user selects a table. It lists the variables in the dataset, and is highlighted according to the current focus in the main canvas. All the Variable palettes are hot-linked to the main canvas. If you change focus in the main canvas, then the highlighting in each variable window changes to reflect those variables that have been selected in the clause that has the new focus.

In this next screen, I have selected variables from each table. As each variable is selected, it is added into the clause that has focus in the main canvas — the "Select" clause in this case.

**Expressions** — This is a subsidiary canvas, used for painting expressions, (which are in turn used in the various clauses of a SQL query). It is selected from the canvas pulldown in the main window. This screen image shows the expression canvas after building an expression that computes the total price for an item sold.

When the expression canvas is active, all selections in variable palettes are directed to the expression canvas. The user builds the expression by selecting operators and operands until satisfied. Selecting "END" transfers the variable to the expression palette where it can be selected in the same fashion as other variables — the expression will be added to the clause that has the focus in the main canvas.

In a sense, the expression just painted has become a new "variable" that can be selected into any of the areas of the main canvas. This next screenshot shows the effect of selecting EXPR1, while the "Order By" clause has focus in the main canvas. Selections while "Order By" has focus cycle through three states — selected, selected descending, and unselected. In the example below, EXPR1 has been selected twice.

The tables palette has been removed from the display by selecting "end" from the "file" pulldown. It can be recalled at any time.

**User Interface Issues**

**Character Screen compatibility** — The QUERY window should remain compatible with character-cell terminals (3270 and ASCII). While it would be nice to implement variable selection by "dragging" a variable to some clause in the main canvas, it would preclude running the application on the majority of displays in use at customer sites.

**Variable Selection** — The variables are arranged in the list in the same order that they occur on the dataset initially. Selecting the "SORT" menu item rearranges the list into alphabetical order (or back to dataset order if selected a second time).
Selecting variables individually from a list becomes tedious for large tables. The "MANY" button presents a dialog allowing one to select and/or deselect variables. It allows the user to specify a pattern for those names that should match, using the same pattern matching specification as the LIKE operator in SQL. This example shows a user selecting all variables that begin with 'cusr'.

ク Click to select the criteria:

Pattern: %cusr%

Select the effect on those variables matched:

Select All
Deselect All

Zoom in — Eventually the main canvas becomes too small to display all the selected items for some area. In the prototype, three dots appear on the main canvas indicating that some information is not displayed. Selecting these dots with the mouse will display another window listing elements in a vertical list, that is scrollable. Like the variable palettes, this list is hot-linked to the main canvas, so that changes are immediately reflected in all appropriate windows.

One can also "Zoom in" on individual items in the various clauses. Zooming in on an item in the select clause will present a panel showing format, informat and label attributes for that item — these can be modified.

Preserving Queries

The "File" menu item in the main window will allow the user to:

- save the current canvas to a catalog
- retrieve a canvas from a catalog
- initialise the canvas from a SQL View definition
- create a SQL View definition from the canvas

Parameterising Queries

I plan to extend the expression dialog to allow the user to reserve placeholders in expressions. Placeholders would be associated with a "prompt string", or a SAS/AF® program name to elicit the value for the placeholder from the user at runtime.

Using this facility, one could build a library of canned queries, that could be run by someone "filling in the blanks".

Towards more complicated SQL

The SQL I have presented this far have involved a single query expression. Many real-world queries can be answered in this simple fashion, but the QUERY window should provide access to the more powerful features of SQL. All SQL constructs revolve around the query expression — an expression that returns a table as its result. In some special cases (like the subquery) the table returned has only one row and column.

The QUERY window will allow one to encapsulate query expressions into objects, and then combine these objects.

Set Operators — The set operators (UNION, INTERSECT and EXCEPT) operate on SQL query expressions. I plan to implement a canvas that operates on encapsulated query objects described above. In a sense it is just another expression canvas, whose operators are UNION, INTERSECT and EXCEPT, and whose operands are query objects.

Subqueries and Correlated Subqueries — A subquery can be represented by an encapsulated query object. A correlated subquery can be thought of as a parameterised query, the parameters being the correlated values that are supplied by the outer query.

For example, given this query that returns the supplier of the most expensive part, in each region.

```
SELECT REGION, SNAME
FROM SUPPLIERS S
WHERE SNUMBER = (SELECT SNUMBER
                  FROM INVENTORY
                  WHERE REGION=S.REGION
                  HAVING PRICE=MAX(PRICE))
GROUP BY REGION;
```

One could answer this query with the QUERY window. All though I begin with the inner subquery here, the user might start with the outer one, and then realise that he needed the inner one. This is not a problem, as he can save the uncompleted canvas of the outer query, compose the inner one, and then resume the outer one.

Composing the inner query, with a parameter for the S.REGION correlated reference would result in an object I have chosen to name it MAXPRICE, that requires one parameter.

```
SELECT SNUMBER
FROM INVENTORY
WHERE REGION=<parameter 1>
HAVING PRICE=MAX(PRICE);  
```

This object is then referenced in the outer query as follows. The use of curly braces distinguishes the reference from a regular function call — this may be too subtle, I am open to suggestions for better method of referring to objects.

```
SELECT REGION, SNAME
FROM SUPPLIERS
WHERE SNUMBER = MAXPRICE{REGION}
GROUP BY REGION;
```

Disclaimer

In this paper, I present some of my thoughts on an implementation offering these features. This project is still in its very formative stages, and I would welcome any ideas on the issues being addressed (or not addressed as the case may be).

I intend to provide a Full-Screen Query building window in some future version of the SAS System. It may not be implemented as described here — the goal of this paper is to stimulate discussion.
and hopefully get feedback on users requirements for such a window. Please send your comments to: Paul Kent, SAS Institute, Inc.

Trademarks
QBE and SQL/DS are registered trademark of IBM Corporation.
SAS and SAS/AF are registered trademarks of SAS Institute, Inc.