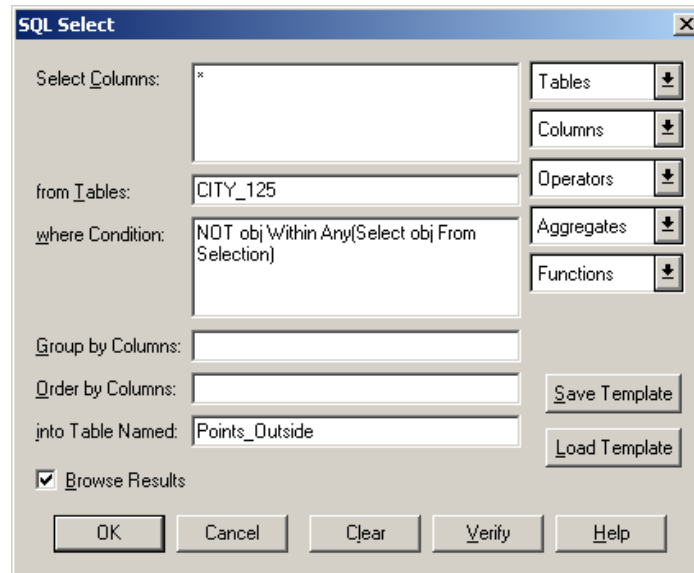


Better Living with SQL

Ok, you've seen *SQL Select* in the MapInfo Professional Menu Bar, but you're still confused with all this "sequel" stuff! **Structured Query Language (SQL)**, sometimes pronounced "sequel", is the basis for ALL MapInfo Professional queries. Even if you've delegated yourself to only use the *Query* → *Select* menu item, MapInfo still uses SQL to drive the filtering process behind the scenes.

SQL is nothing more than a sentence structure that controls the filtering, sorting, computation, even joining of tabular data. Consider the following *SQL Select* window:



If you were to write the SQL sentence that corresponds to the above dialog, it would look something like the following:

```
Select * From CITY_125 Where NOT obj Within Any(Select obj From Selection)
```

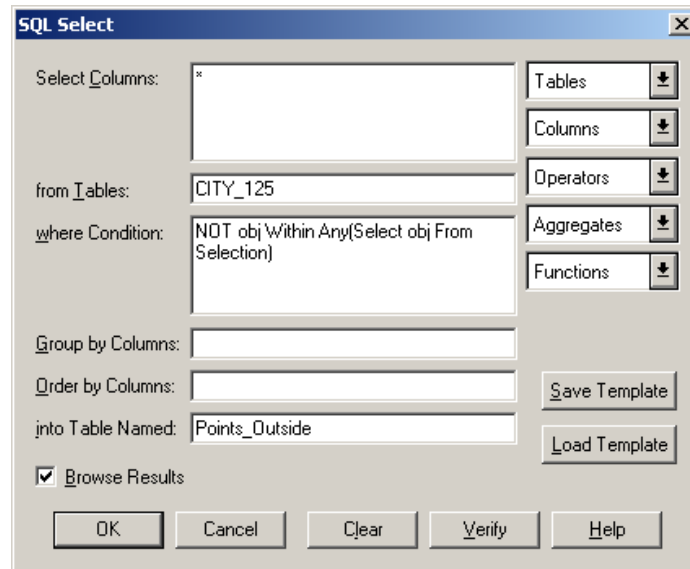
This notation is possible with any *SQL Select* dialog window, and in fact, MapInfo will reduce the contents of these dialog boxes to a SQL statement prior to processing them. So, armed with this knowledge, let's take a look at a few useful SQL statements.

Finding Points Outside a Region

Finding points that fall within a region is a fairly straightforward task in MapInfo Professional. The following SQL statement will select all the points from the CITY_125 table that fall within the State of Arizona:

```
Select * From CITY_125, STATES Where CITY_125.Obj Within STATES.Obj AND STATES.State = "AZ"
```

But what if you want to find points that fall outside the State of Arizona? The following dialog illustrates how to write a SQL query to find all the points from the CITY_125 table that fall outside the boundary of the currently selected object.

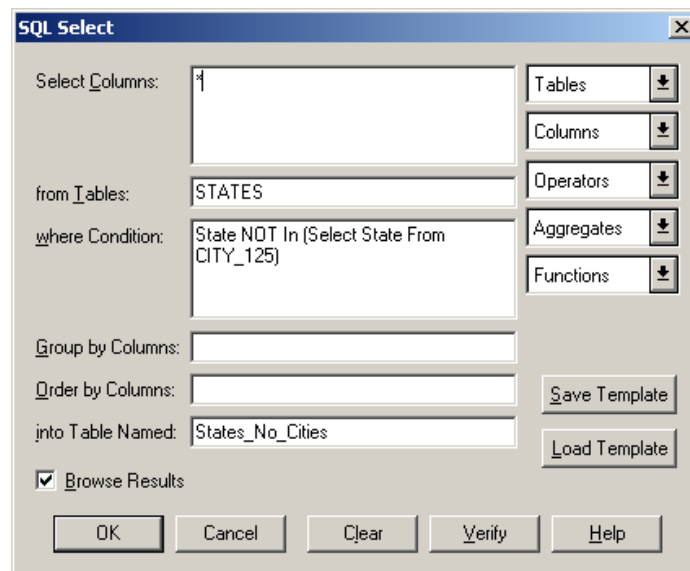


If you wanted to find all the points from the CITY_125 table that fall outside the State of Arizona you could issue the following SQL statement:

```
Select * From CITY_125 Where NOT obj Within Any(Select obj From STATES Where State = "AZ")
```

Selecting Items Not in a Table

Sometimes it proves to be useful to find records that exist in one table that are not present in another. Take a look at the following *SQL Select* window:

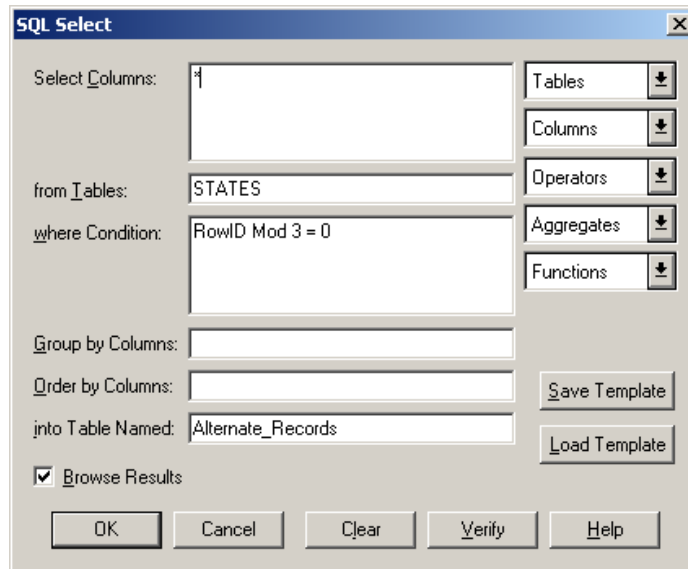


This SQL statement will list all the states that are not in a list of states in the CITY_125 table. You might notice that the preceding two SQL windows included a SQL statement within the Where Condition box. This technique of placing a SQL statement in the Where Condition is called a Sub-Select. Sub-Selects are useful because they allow us to compare a single column in a table to a list of values created by a second SQL statement. One note however, MapInfo Professional

only allows one Sub-Select statement in the Where Condition. Once you learn how to use them, they can prove to be invaluable in writing simple, effective and fast queries.

Selecting Alternate Records

In rare cases, it might be useful to select a subset of records not based on values in the table, but rather their position in the table. To select records based on their position in the table, or their RowID use the following query:



The screenshot shows the 'SQL Select' dialog box with the following fields and values:

- Select Columns: (empty)
- from Tables: STATES
- where Condition: RowID Mod 3 = 0
- Group by Columns: (empty)
- Order by Columns: (empty)
- into Table Named: Alternate_Records
- Browse Results

Buttons at the bottom: OK, Cancel, Clear, Verify, Help. On the right side, there are dropdown menus for Tables, Columns, Operators, Aggregates, and Functions, along with Save Template and Load Template buttons.

This query will select every third record from the STATES table starting with the third record. If instead you wanted to start with the first record, use the following SQL statement:

```
Select * From STATES Where RowID Mod 3 = 1
```

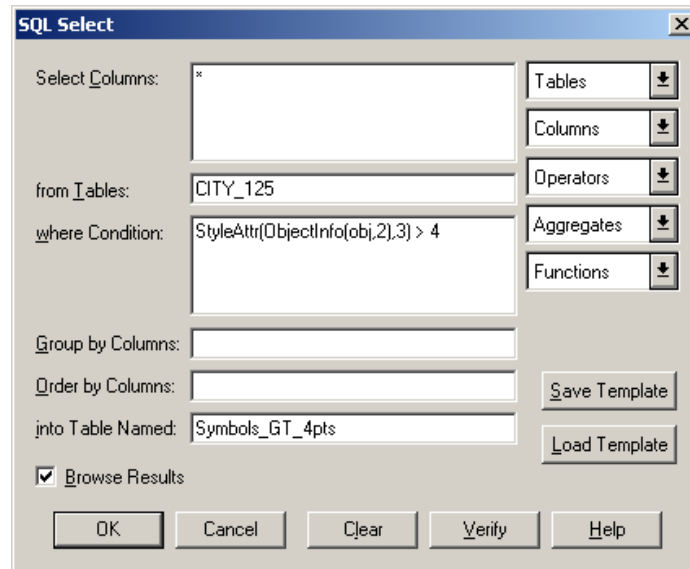
What about every fifteen (15) rows, starting with row number 7...

```
Select * From STATES Where RowID Mod 15 = 7
```

Once you get the hang of it, you can try any combination of records and starting points. For those of you who are curious, Mod is a computational operator that returns the remainder from the division operator. For example, $4 \text{ Mod } 3 = 1$, or $9 \text{ Mod } 9 = 0$.

Selecting By Color

Just like it might be useful to select alternating records, it might also be useful to use criteria such as an object's color, size or type to filter tabular data. Consider the following *SQL Select* window:



There are a couple of things going on here, so let's break the Where Condition down to it's components. First there is the function `StyleAttr()`. The `StyleAttr()` function returns any number of things depending on what is provided. Let's reference the **MapBasic User's Guide** to get more information on `StyleAttr()`.

StyleAttr() function

Purpose:

Returns one attribute of a Pen, Brush, Font or Symbol style.

Syntax:

`StyleAttr(style , attribute)`

Return Value:

String or Integer, depending on the *attribute* parameter.

Attribute Values:

Attribute Name	Attribute Value	Description
PEN_WIDTH	1	The width of a line in points or pixels.
PEN_PATTERN	2	The style pattern of a line such as dashed or solid.
PEN_COLOR	4	The color of a line object.
BRUSH_FORECOLOR	2	The foreground color of a region object.
BRUSH_BACKCOLOR	3	The background color of a region object.
FONT_POINTSIZE	3	The point size of a text object.
SYMBOL_COLOR	2	The color of a symbol (point) object.
SYMBOL_POINTSIZE	3	The point size of a symbol object.

When querying values from a MapInfo table, use the *Attribute Value* NOT the *Attribute Name*. The second function used in the above SQL Select dialog, is the `ObjectInfo()` function. `ObjectInfo()` returns an object style based on the type of object submitted and the attribute selected. Again we'll defer to the MapBasic User's Guide to get information about the `ObjectInfo()` function.

ObjectInfo() function

Purpose:

Returns Pen, Brush or other values describing a graphical object.

Syntax:

ObjectInfo(*object* , *attribute*)

Return Value:

SmallInt, Integer, String, Float, Pen, Brush, Symbol, or Font, depending on the *attribute* parameter.

Attribute Values:

Attribute Name	Attribute Value	Description
OBJ_INFO_PEN	2	Returns the object Pen style.
OBJ_INFO_SYMBOL	2	Returns the object Symbol style.
OBJ_INFO_TEXTFONT	2	Returns the object Text font style.
OBJ_INFO_BRUSH	3	Returns the object Brush style.

Using these two functions together, we can select any number of object style parameters to create a new MapInfo table. Here's how it goes together:

```
Select * From CITY_125 Where StyleAttr( ObjectInfo ( obj , 2 ) , 3 ) > 4
```

This query will select all records with Symbol objects with a point size greater than 4. There is another option however. This same query will also select Text objects with a point size greater than 4 points. Because of the interchangeability of attribute values for the `StyleAttr()` function and the `ObjectInfo()` function, the same query can yield different results based on the type of objects that are submitted.

Let's take a look at one other example and this time we'll use the `StyleAttr()` and `ObjectInfo()` functions to select records based on their foreground color. Take a look at the following SQL Select dialog:

The screenshot shows the 'SQL Select' dialog box. The 'Select Columns:' field contains an asterisk (*). The 'from Tables:' field contains 'STATES'. The 'where Condition:' field contains the SQL query: `StyleAttr(ObjectInfo(obj,3),2) = RGB(192,255,192)`. The 'into Table Named:' field contains 'Green_States'. The 'Browse Results' checkbox is checked. The dialog also features buttons for 'Tables', 'Columns', 'Operators', 'Aggregates', 'Functions', 'Save Template', 'Load Template', 'OK', 'Cancel', 'Clear', 'Verify', and 'Help'.

In this query we're using the `ObjectInfo(obj , 3)` function to return a Brush style, and then subsequently using this style to query the BRUSH_FOREGROUND (2) color property of the `StyleAttr()` function. What is returned is an Integer number that represents a color. The color number is computed using the following formula:

```
Color = (65536 * Red) + (256 * Green) + Blue
```

But, there's an easier way. Rather than compute this for each color you want to query, MapInfo has a function to return this number for you: `RGB()`. The `RGB()` function has the following structure:

```
Color = RGB( red component , green component , blue component )
```

So the complete query to return all those states that match that awful default green color is as follows:

```
Select * From STATES Where StyleAttr( ObjectInfo ( obj , 3 ) , 2 ) = RGB(192,255,192)
```

With a little practice you can combine Style attributes with other selection criteria to create increasingly complex queries.

Finding Duplicate Records

Sometimes it becomes necessary to identify records within a table that are not unique based on a column of data in the table. Again, we'll make use of a sub-select query to select records that are present in a list of duplicate values. Here's the first part of the query:

The screenshot shows the 'SQL Select' dialog box with the following configuration:

- Select Columns: Zip, Count(*)
- from Tables: US_CUSTG
- where Condition: (empty)
- Group by Columns: Zip
- Order by Columns: 1
- into Table Named: Zip_Count
- Browse Results

Buttons on the right: Tables, Columns, Operators, Aggregates, Functions, Save Template, Load Template.

Buttons at the bottom: OK, Cancel, Clear, Verify, Help.

The above query will create a list of ZIPCodes and a count of the number of times that ZIPCode appears in the table.

We then need to take the resulting table from this query and run it through a subsequent query which will list all records that do not have unique ZIPCodes. Here's the second part:

The screenshot shows the 'SQL Select' dialog box with the following configuration:

- Select Columns: *
- from Tables: US_CUSTG
- where Condition: ZIP In (Select ZIP From Zip_Count Where Col2 > 1)
- Group by Columns: (empty)
- Order by Columns: (empty)
- into Table Named: Dup_Records
- Browse Results

Buttons at the bottom: OK, Cancel, Clear, Verify, Help.

Right-side controls: Tables, Columns, Operators, Aggregates, Functions (dropdown menus); Save Template, Load Template (buttons).

This query will yield all the records in the US_CUSTG table that do not have unique ZIPCodes. The reference to Col2 corresponds to the Count column from the Zip_Count query previously run.

All the queries we have run here have a practical use in everyday life -- the hard part is identifying when a particular query is called for. Happy querying!