

Using Structured Query Language in gINT

Structured Query Language (SQL), pronounced “*ess-cue-el*” or “*sequel*” is a language used to communicate with relational databases. It is used to retrieve and update data in a database, as well as to build and modify the structure of a database.

gINT insulates you from the details of the SQL queries it generates and, for the vast majority of your usage of the program, you do not need to know anything about SQL. However, a little study of the subject will greatly enhance what you can do with gINT.

In gINT you can use SQL to:

- Create data views in Input
- Modify reports and correspondence files
- Read, add, and alter project data in any table using gINT Rules

Sample Queries

The following samples are selection queries—the most common type of query that you would use in gINT:

Show all boreholes where the sample PID is greater than 100:

```
Select Distinct [PointID]
From [SAMPLE]
Where Val([PID]) > 100
```

Show the total number of boreholes and total length of drilling:

```
Select Count([PointID]) As [Number of BHs],
       Sum([HoleDepth]) As [Total Length Drilling]
From [POINT]
```

The following generates the total thickness of some specified layer in each borehole but only returns the thickness below a specified foundation level. Therefore, those layers above the foundation elevation, and portions of the desired layers above the foundation level, are ignored. The “Parameters” line prompts the operator for the required graphic and foundation elevation.

```
Parameters [Desired Graphic] Text,[Foundation Elev] IEEEDouble;
Select Distinct
  [POINT].[PointID] As [Hole ID],
  (Select Sum([LITHOLOGY].[Bottom] - [LITHOLOGY].[Depth] -
             IIf(([POINT].[Elevation] -
                  [LITHOLOGY].[Depth] -
                  [Foundation Elev]
                 ) > 0,
                 ([POINT].[Elevation] -
                  [LITHOLOGY].[Depth] -
                  [Foundation Elev]
                 )
             ),
       0
    )
From [LITHOLOGY]
```

```

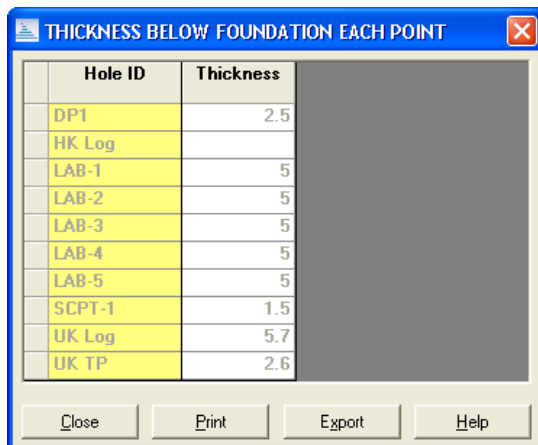
Where (UCase([LITHOLOGY].[Graphic]) = UCCase([Desired Graphic]))
  And (([POINT].[Elevation] - [LITHOLOGY].[Bottom]) <
    [Foundation Elev]
  )
  And [POINT].[PointID] = [LITHOLOGY].[PointID]
) As Thickness
From [POINT] Inner Join [LITHOLOGY]
  ON [POINT].[PointID] = [LITHOLOGY].[PointID]
Where IsNumeric([POINT].[Elevation])

```

Most of the SQL queries that you would find useful in gINT are generally quite simple and are only as complex as the first two samples above. The last sample shows that you can perform sophisticated queries with SQL. Of course, the last sample is by no means the most complex query that could be written.

Queries in Input

You can create your data views in Input via the Queries command (**File ► Queries** in versions 6 and 7; **Tools ► Queries** in version 8). Think of these as quick reports that can be run in Input. The results will be shown in a grid dialog such as the following:



Hole ID	Thickness
DP1	2.5
HK Log	
LAB-1	5
LAB-2	5
LAB-3	5
LAB-4	5
LAB-5	5
SCPT-1	1.5
UK Log	5.7
UK TP	2.6

The data in the dialog can be printed or exported to a Microsoft Excel® or CSV file.

Once queries are written, they can be named and described and run from a list dialog. The end user doesn't need to know the details of the SQL queries. They just run the desired query by double-clicking the name or clicking the Execute button at the lower left.

Name	Description	Date
AVERAGE ELEVATION		1/31/2004 2:37:54 PM
CHISELLING LENGTH AND TIME EACH POINT	For each borehole	9/19/2001 3:15:06 PM
CHISELLING LENGTH AND TIME TOTAL	Sum of all boreholes	9/19/2001 3:15:06 PM
LEN DRILLING IN A MONTH		12/17/2003 7:20:56 PM
MIN MAX EAST NORTH ELEV		4/4/2005 8:33:01 AM
NUM BH & LEN DRILLING	For each borehole	9/19/2001 3:15:06 PM
THICKNESS BELOW FOUNDATION	Sum of all boreholes	4/13/2002 2:12:23 PM
THICKNESS BELOW FOUNDATION EACH POINT	For each borehole	4/13/2002 2:12:23 PM
THICKNESS EACH POINT	For each borehole	1/15/2006 9:13:24 AM
TOP RECORDS IN UNCONFINED READINGS		12/19/2003 5:16:09 PM
TOTAL LAYER THICKNESS	Sum of all boreholes	4/13/2002 2:09:55 PM
*		

This feature is only available for gINT Logs Plus and gINT Professional. Search for “Queries” in Help and look at the “Queries (Commands)” topic for further details.

SQL Functions in Reports and Correspondence Files

At output time, gINT generates at least one SQL query for the report and one for every entity on the report. You do not need to know the details of this process; you simply specify the appropriate field references and use the functions appropriate to the output you need to generate. Most of gINT’s functions are simplified front-ends to SQL functions. For example, in the header or footer of a log form, the following text expression

```
Number of samples: <<Count(<<SAMPLE.Type>>)>>
```

generates the following SQL

```
Select Count([SAMPLE].[Type])
From [SAMPLE]
Where [SAMPLE].[PointID] = 'xxx'
```

where “xxx” is the name of the PointID being output. If you wrote the same expression in a text document that had a “Project” key set, the “Where” clause would not be included the SQL query and the function would return the total number of samples for the entire project. The Output engine knows when and how to use a Where clause appropriately, depending on the report.

gINT has a rich library of functions, but it is impossible to create all the functions needed for all clients in all situations. Instead, gINT provides four functions that allow you to write your own functions with SQL:

- Sql
- SqlCount
- SqlList
- SqlRange

For example, let's say on your log output you wished to show the types of samplers used in each hole. There is no dedicated function to perform this task. Instead, you can write your own:

```
Samplers used: _
<<SqlList(", ", _
  Select Distinct [SAMPLE].[Type] _
  From [SAMPLE] _
  Where [SAMPLE].[PointID] = 'xxx' _
  Order By [SAMPLE].[Type]_
)>>
```

The above will generate a nonduplicated list of sample types used on the current borehole ("xxx"), separated by a comma and a space, and order the types alphabetically.

As far as gINT is concerned, correspondence files are just another report style, and you can use the same type of expressions as on reports, including writing your own SQL queries.

Search Help for the above functions for details and samples of usage.

SQL in gINT Rules

gINT Rules uses a dialect of VBA (Visual Basic for Applications) that gives you full access to the same level of SQL functionality in our gINT code. You must first add the reference to the Microsoft DAO 3.6 object Library (5.0) (in the gINT Rules dialog Code tab, select **Edit > References**). Using SQL in the gINT Rules programming environment, you can read, add, and alter data in the project in any table. Many of the gINT Rules samples available on our Web site have samples of SQL usage (www.gintsoftware.com/support_gintrules.html).

A little investment in time studying the samples in gINT Help and gINT Rules can go a long way to extending the functionality of gINT significantly. There are thousands of books and articles devoted to the subject. You will also find extensive support for SQL on the Web, just perform a search for "SQL" or "Structured Query Language."