



## Using Structured Query Language (SQL) in gINT

---

### *Introduction*

---

Structured Query Language (SQL), pronounced "see-quill," is an interface standard used to communicate with relational databases. It is used to retrieve data from a database, to update data in a database, and to build and modify the structure of a database.

gINT uses SQL throughout the program. For example, in Input, when you move to a SAMPLE table you are seeing just the data for the currently selected PointID. This is a "data view" specified by the internal gINT code. If you select the **Tables ► View Entire Table** menu you will see a read-only data view of all the data in the table. Both views are dictated by SQL commands issued by gINT. In producing reports, gINT gathers all your references in your reports to fields in your database and creates many SQLs, both simple and extremely complex, to extract the necessary information.

gINT insulates you from the details of the SQLs 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 the program.

gINT provides three facilities that allow you to write your own SQLs.

### *Sample Queries*

---

Following are samples of some selection queries, the most common type that would be used 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]
```

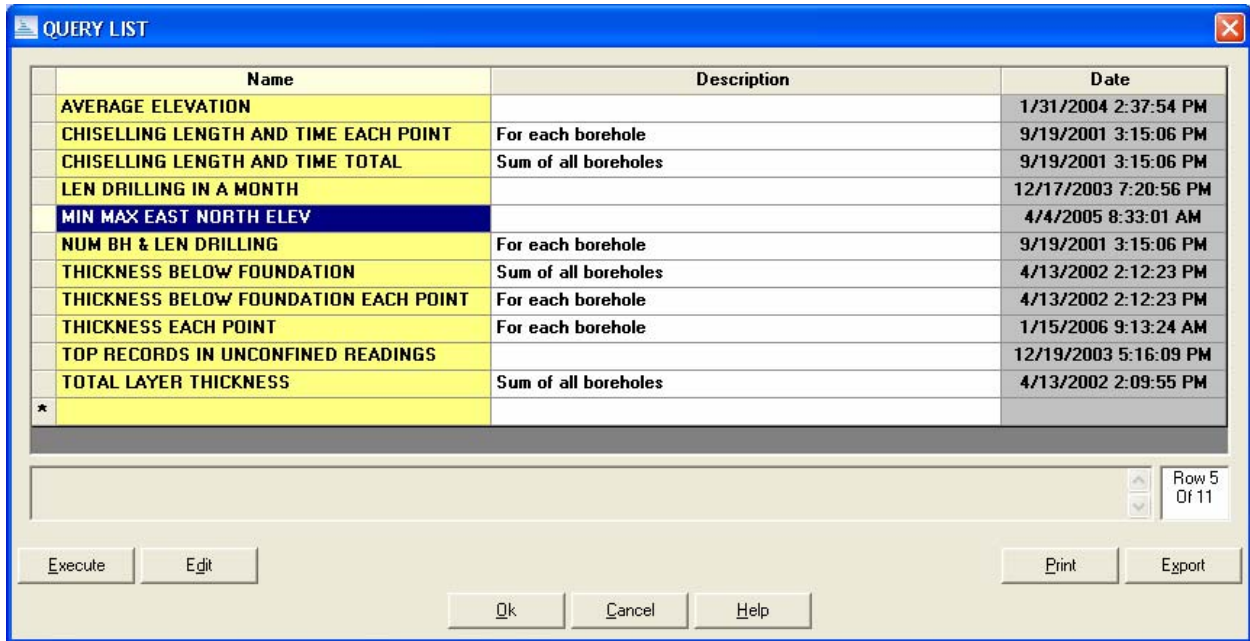
The following generates the total thickness of some specified layer in each borehole but only returns the thickness below a specified foundation level. Therefore, those layers above the foundation elevation, and portions of the desired layers above the foundation level, are ignored. The "Parameters" 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] -
```





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 SQLs. They just run the desired query by double clicking on its name or clicking the Execute button at the lower left.

This facility 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 for the report and least one for every entity on the report. You do not need to know the details of this process. You just 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 in the SQL 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 SQLs:

- 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 SQLs.

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 functionality concerning SQLs as we have in our gINT code. You must first add the reference to the Microsoft DAO 3.6 object Library (5.0) (**Edit ► References** in the gINT Rules dialog "Code" window). Using SQLs in this 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](http://www.gintsoftware.com/support_gintrules.html)).

## ***Summary***

---

SQL, an interface standard used to communicate with relational databases, is used by many programs including gINT. 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 "Structured Query Language."