Visual SQL Query Builder

Quick video tutorial

Overview

A query is a request for data or information from a database table or a combination of tables. This data may be generated as results returned by the Structured Query Language (SQL) or as pictorial trend analyses (graphs or complex results, e.g.) from data-mining tools.

Several different query languages may be used to perform a range of simple to complex database queries.

Most database administrators are familiar with SQL since it is the most well-known and widely used query language.

A query can be executed in a project through a couple different methods. Below you will find them listed with a short description.

Method 1: WhereCondition

The first method you can use is the **WhereCondition.** In this case, the data query will be performed in a table, created at **Edit** > **Datasets** > **Tables**. Since the table is already selected, you only need to supply the conditional. You must also run *SelectCommand* to update the query.

Additional information

Check out "Appendix A" for more information

Method 2: Query

The second method is similar to the first one, but it uses a query, created at **Edit**>**Datasets>Queries** and linked to a provider (see image below).

To execute the query, you need to select one of the tables from the database and the condition that you want to filter the data. You need to run *SelectCom mand* to update the query.

```
@Dataset . Query . Query1 . Sql Statement = ” s e l e c t ? from Table1 where
UTCTimestamp  Ticks>=”  +  StartTime . Utc Ticks  +  ”   and   UTCTimestamp  Ticks<=”  +
EndTime . Utc Ticks + ”” ;
@Tag . TableTag = @Dataset . Query . Query1 . SelectCommand () ;
```

Additional information

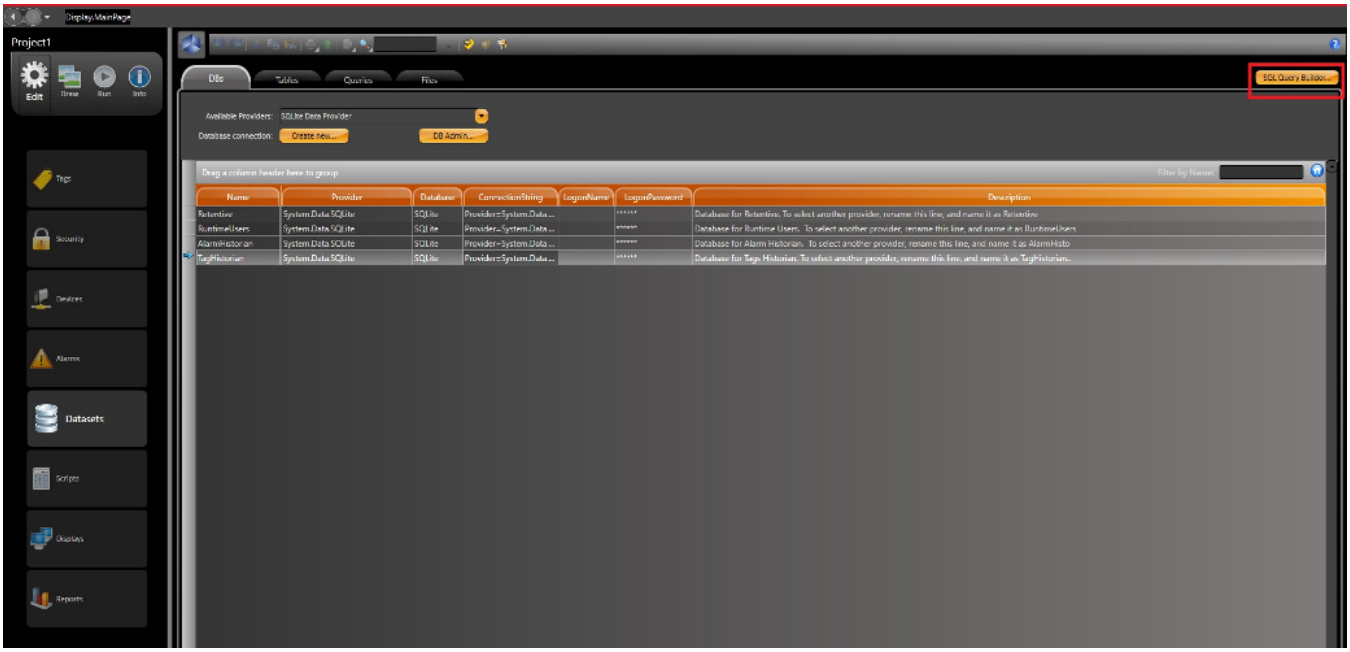
Check out "Appendix A" for more information

Method 3: Different Location

This alternative uses the same statement as shown above, but in a different location. Instead of writing it in a task/class or CodeBehind, it can be placed directly into **Edit** > **Datasets** > **Queries** > **SqlStatement Column**.

Method 4: SQL Query Builder

The last option is to use a feature called a **SQL Query Builder**, found at **Edit>Datasets**. It is a graphic interface that facilitates the creation of SQL Statements based on a specific provider. This method is not as usual as the others, so more details regarding its functionalities will be explained below.



Appendix A

For the SQLStatement, which is a property of the Query, and for the WhereCondition, which is the property of the Table of the Dataset, you can customize your query in the database, they are properties of type server, so if you modify this property via Script , either on the server or on the clients, the property value will be synced between all clients.

However, we created the possibility for the user to configure Client Tags in the configuration of these properties, for example:

String1: Select * from {tagTable} where {tagWhereCondition}

Where, "tagTable" and "tagWhereCondition" are Client tags.

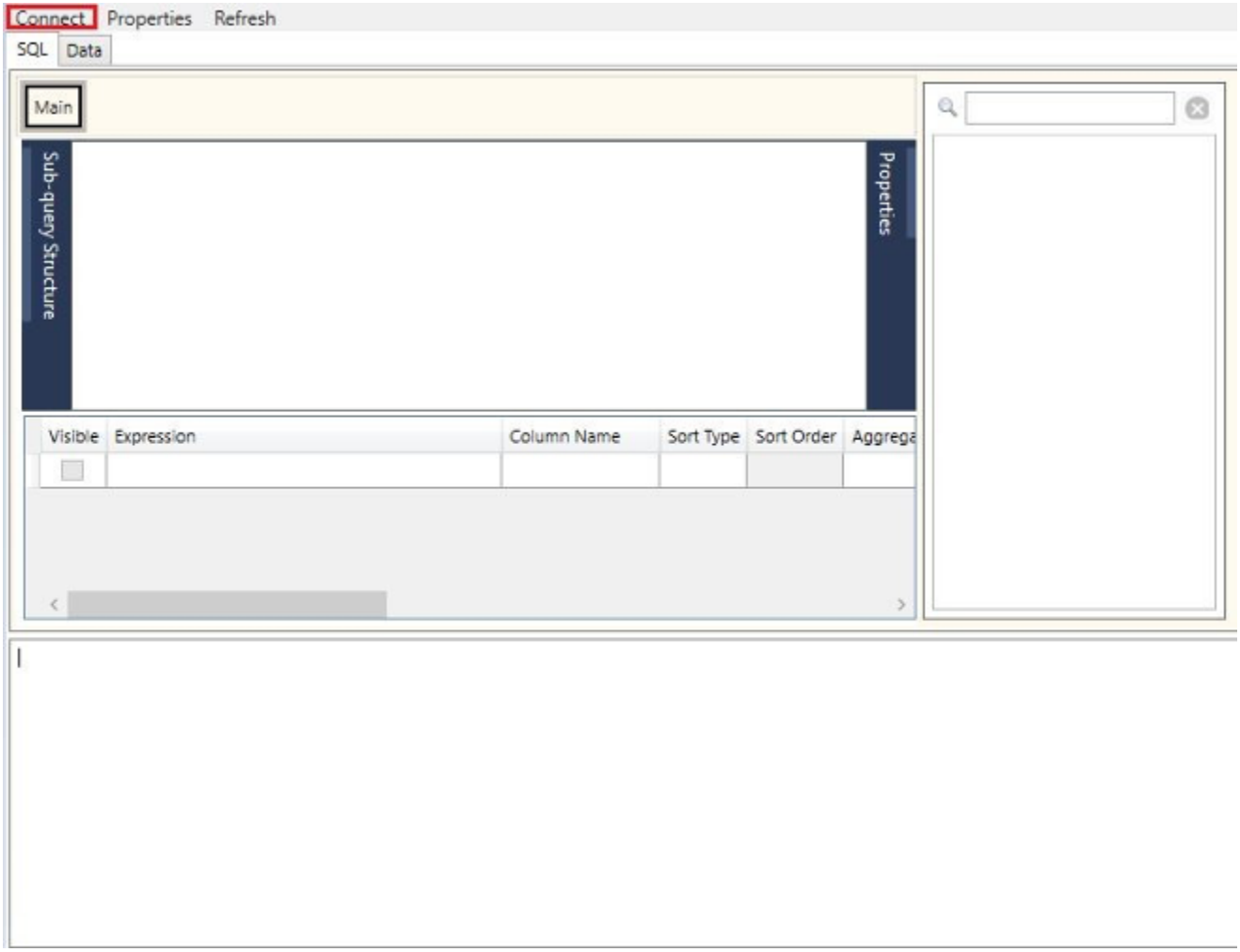
Note that String1 will always be the same, not being modified in the scripts, what will change is the tags of type Client. When we execute the SelectCommand or SelectCommandWithStatus methods, we resolve these tags in the client's context, passing to the server to correct the right query. Multiple clients can use the same query or table without conflict. Although, they will still be entering the same execution queue on the server (this does not mean multithreading).

To summarize, you could even just put a { tagSQLStatementClient } in the SQLStatement, and the content of the query would be what was in that tag, specific to each client. It is important to remember that in this way it makes no sense to use the Select and Next properties, as they are only for server execution.

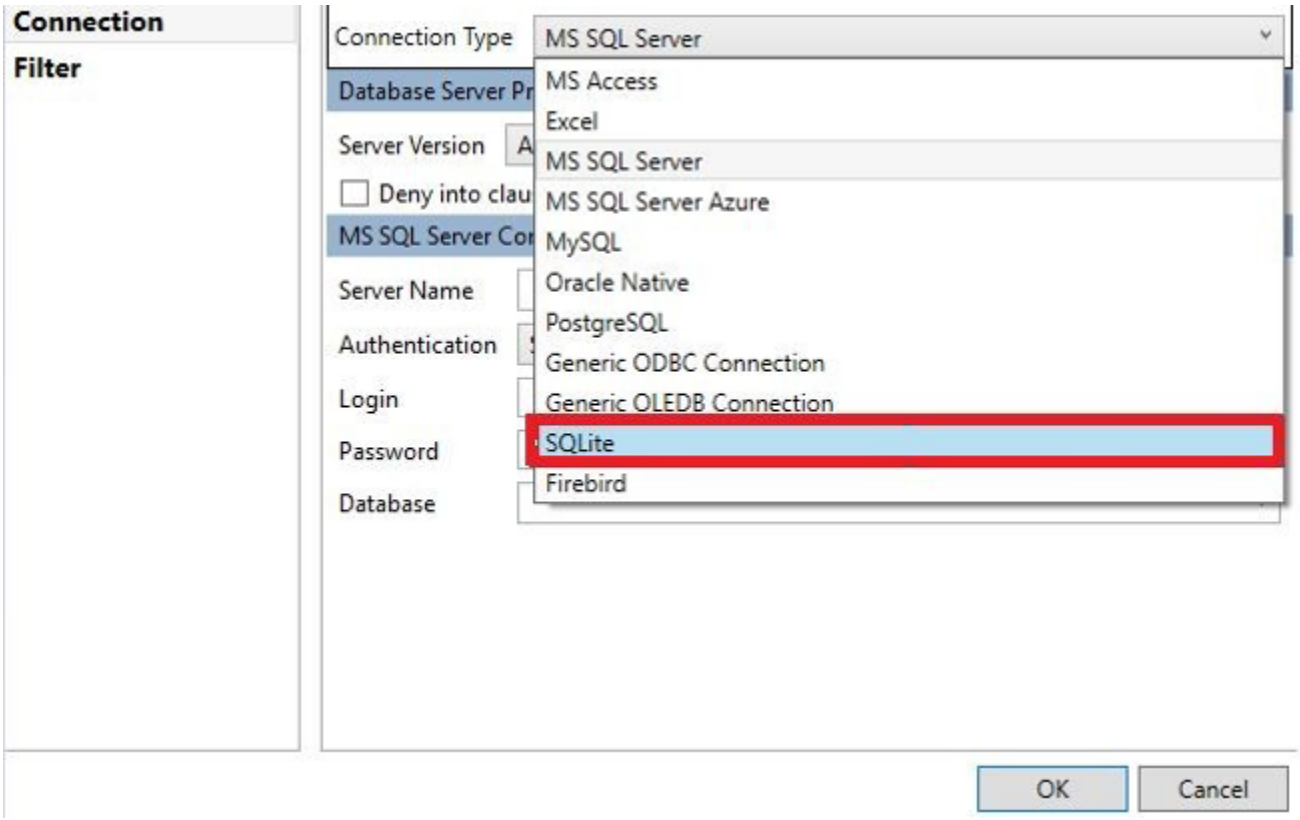
How to use SQL Query Builder

Loading Data

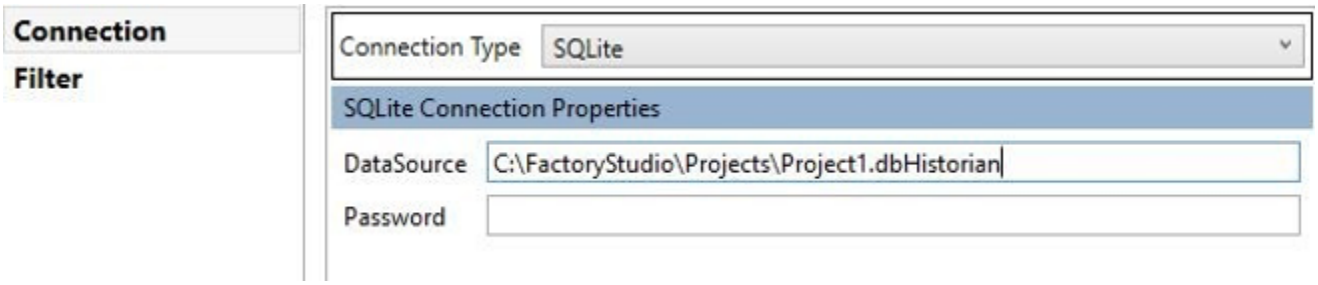
Before you can begin, you need to make sure your databases and providers are configured correctly in the project. The first thing you need to do is load the data into the Query Builder. To do so, open the builder and click on the *Connect* button in the top-left corner.



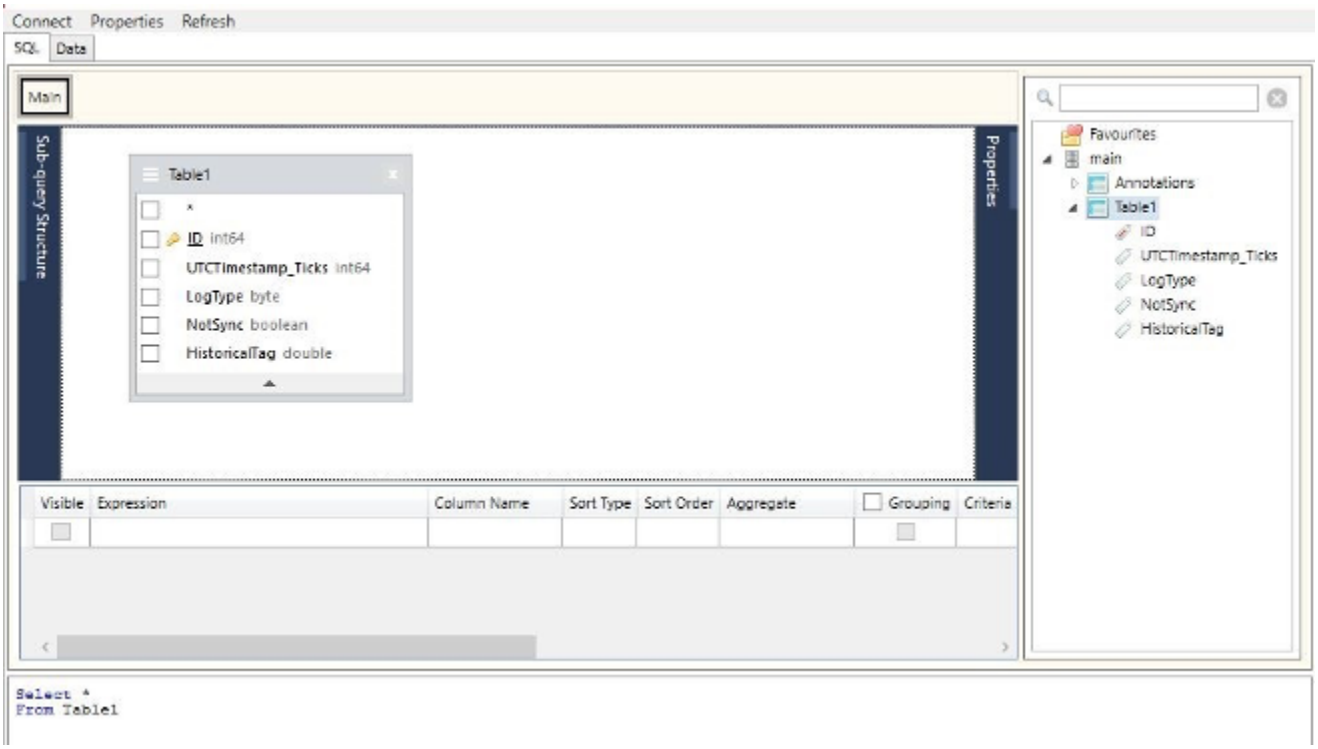
A popup will appear with a combobox containing various types of providers. Select the one you will work with. In this example, we will set up a connection to a SQLite DB.



Now, we need to configure the path to the database. The path is the same one seen below in the Connection String's DataSource field.



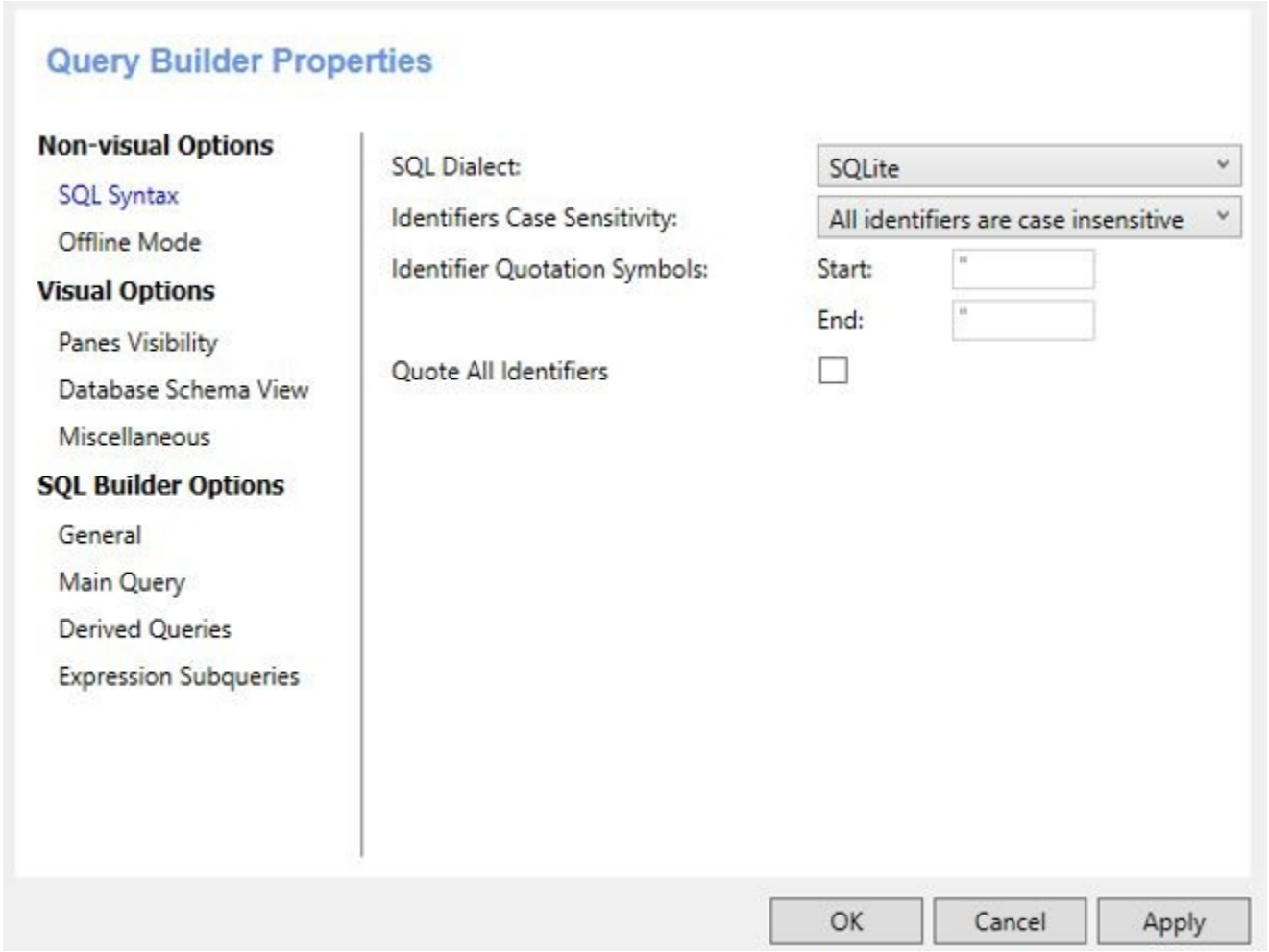
If you wrote the correct path, you should be able to see all of the available tables and their elements in the right corner. Double-click on one of the tables to load its elements into the Sub-Query Structure.



Properties

A Properties button is located in the top left corner. When you select it, a popup will open that contains the Query Builder's customizable properties.

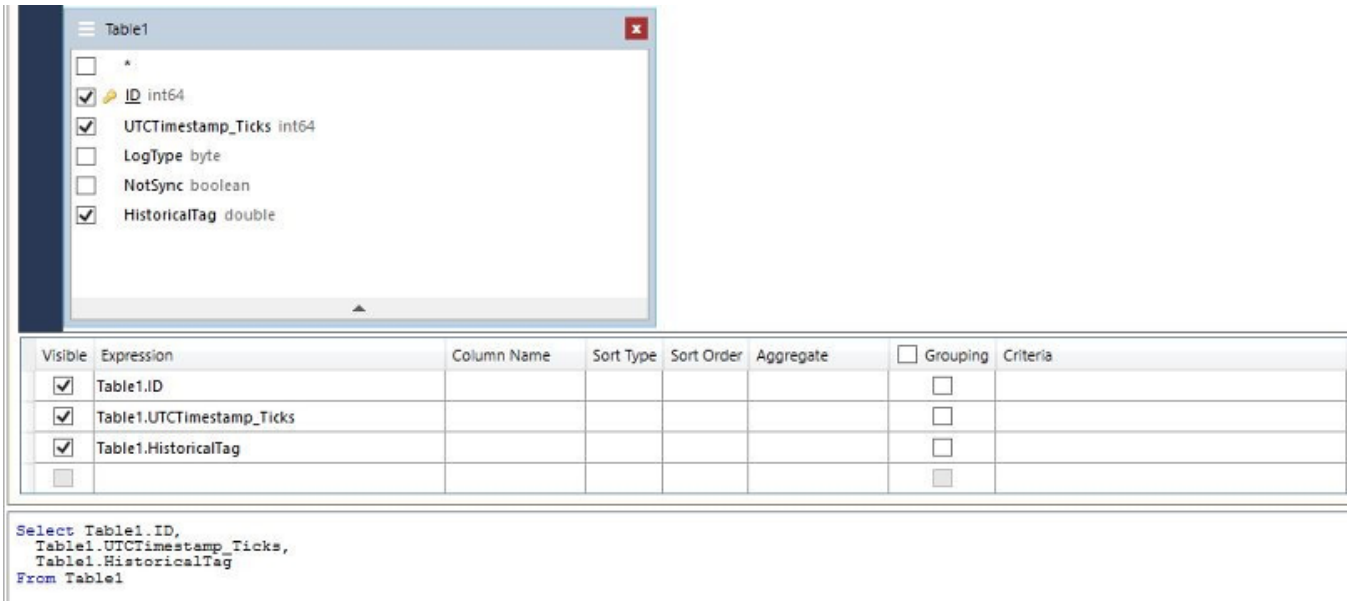
The image below shows all of the properties that are available to be customized.



Creating Statements

After the selected table is loaded into the sub-query structure, a statement will be initialized at the bottom of the page.

You can filter individual columns from the table by selecting specific checkboxes. When you do, the system will filter the table results based on your selection. If you do not select any checkboxes, the table will have not have any filters.



The columns allow you to add conditions which filter values from the table.

The column options are:

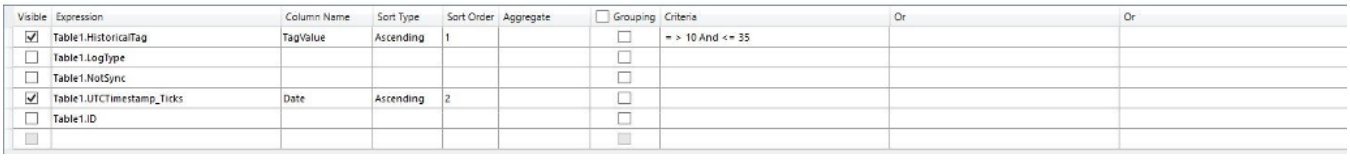
- **Visible**: Remove the entire column from the query results
- **Expression**: The original column name
- **Column Name**: Give a table, or a column in a table, a temporary name. Aliases are often used to make column names more readable. An alias only exists for the duration of the query.
- **Sort Type**: Sort the results in ascending or descending order
- **Sort Order**: Sort the order of the columns in the results
- **Aggregate**: The values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning. E.g.: *Avg*, *Count*, *Sum*.
- **Grouping**: Group column elements. Enable creating filter conditions for groups
- **Criteria**: Criteria for the selection condition E.g.: =, >, <, ! =.
- **Or**: Same as the Criteria

Statement Example

To better illustrate the query builder feature, let's create an example which assumes the following requirements for our query result:

- Only the UTCTimestamp Ticks and HistoricalTag columns are required
- All column names need to be easily understandable
- Elements will be sorted in ascending order
- We only want the HistoricaTag values between 10 and 35

Fill in the columns with these requirements as seen in the image below.



If everything was filled in correctly, the final SQL Statement generated by the Query Builder should be:

```
Select Table1 . Historical Tag As TagValue , Table1 . UTCTimestamp Ticks As Date
From Table1
Where Table1 . Historical Tag = Table1 . Historical Tag > 10 And Table1 . Historical Tag <= 35
Order By TagValue , Date
```