Visual Query Designer for editing Enesys SharePoint Queries
## Summary

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enesys SharePoint Query Designer Manual</td>
<td>2</td>
</tr>
<tr>
<td>Summary</td>
<td>3</td>
</tr>
<tr>
<td>Query Construction</td>
<td>5</td>
</tr>
<tr>
<td>Enesys SharePoint queries</td>
<td>5</td>
</tr>
<tr>
<td>Conclusion</td>
<td>8</td>
</tr>
<tr>
<td>Using Enesys SharePoint Query Designer</td>
<td>9</td>
</tr>
<tr>
<td>Retrieving data from a SharePoint list</td>
<td>9</td>
</tr>
<tr>
<td>Specifying list columns</td>
<td>11</td>
</tr>
<tr>
<td>Filtering list items</td>
<td>12</td>
</tr>
<tr>
<td>Filtering items based on another statement data</td>
<td>19</td>
</tr>
<tr>
<td>Using column display names or internal names</td>
<td>20</td>
</tr>
<tr>
<td>Using parameters</td>
<td>21</td>
</tr>
<tr>
<td>Using data source variables</td>
<td>24</td>
</tr>
<tr>
<td>Expanding recurring events</td>
<td>30</td>
</tr>
<tr>
<td>Stripping HTML tags</td>
<td>31</td>
</tr>
<tr>
<td>Items and folders</td>
<td>33</td>
</tr>
<tr>
<td>Retrieving the first items of a list</td>
<td>35</td>
</tr>
<tr>
<td>Paging</td>
<td>35</td>
</tr>
<tr>
<td>Custom separator for multiple values columns</td>
<td>37</td>
</tr>
<tr>
<td>Expanding multiple values</td>
<td>38</td>
</tr>
<tr>
<td>Retrieving attachments</td>
<td>40</td>
</tr>
<tr>
<td>Retrieving user information</td>
<td>42</td>
</tr>
<tr>
<td>Retrieving item versions</td>
<td>43</td>
</tr>
<tr>
<td>Retrieving meeting instances</td>
<td>45</td>
</tr>
<tr>
<td>Distinct items</td>
<td>46</td>
</tr>
<tr>
<td>Reporting on Survey lists</td>
<td>47</td>
</tr>
<tr>
<td>Introduction</td>
<td>47</td>
</tr>
<tr>
<td>Adding a Survey statement</td>
<td>49</td>
</tr>
<tr>
<td>Selecting the fields</td>
<td>50</td>
</tr>
<tr>
<td>Filtering data</td>
<td>50</td>
</tr>
<tr>
<td>Survey Properties</td>
<td>51</td>
</tr>
<tr>
<td>Retrieving data from InfoPath forms</td>
<td>55</td>
</tr>
<tr>
<td>Introduction</td>
<td>55</td>
</tr>
<tr>
<td>Adding an InfoPath statement</td>
<td>55</td>
</tr>
<tr>
<td>Selecting the Content Type</td>
<td>56</td>
</tr>
<tr>
<td>Selecting the Fields</td>
<td>57</td>
</tr>
<tr>
<td>Specifying the Field Data Types</td>
<td>59</td>
</tr>
<tr>
<td>Retrieving column choices</td>
<td>59</td>
</tr>
<tr>
<td>Applying operations to lists</td>
<td>66</td>
</tr>
<tr>
<td>Joining SharePoint lists</td>
<td>68</td>
</tr>
<tr>
<td>Merging SharePoint lists</td>
<td>69</td>
</tr>
<tr>
<td>Getting distinct values</td>
<td>70</td>
</tr>
<tr>
<td>Using Intellisense inside SQLQuery statements</td>
<td>70</td>
</tr>
<tr>
<td>Merging a variable number of lists</td>
<td>77</td>
</tr>
</tbody>
</table>
ROLLING UP LIST ITEMS IN AN ENTIRE SITE COLLECTION ................................................................. 82
FILTERING THE SITES .......................................................................................................................... 86
RETRIEVING ADMINISTRATIVE DATA ................................................................................................. 86
USING THE OUTPUT OF A STATEMENT AS AN INPUT OF ANOTHER STATEMENT ....................... 90
JOINING SharePoint lists and external SQL data .............................................................................. 92
SPECIFY THE CONNECTION STRING .............................................................................................. 93
ENTER YOUR QUERY STRING ............................................................................................................. 96
JOINING SQL AND SharePoint DATA ............................................................................................... 97
RETRIEVING XML DATA .................................................................................................................... 97
Query Construction

Enesys SharePoint Query language syntax is based on XML. The syntax is simple and straightforward. However, we are providing a Visual Query Designer that makes it unnecessary to deal with the details of the syntax.

Enesys SharePoint queries

A query is composed of several statements. The following Query Designer screenshot shows a query using 3 statements:

Each statement executes a specific function depending on its type. For instance, the List statement will retrieve data from a specific SharePoint list. The RollUpLists statement will retrieve items in an entire site collection. The SqlQuery statement makes it possible to manipulate the resulting data from other statements using SQL SELECT syntax.

Statements are executed sequentially starting from the first statement. The resulting data from each statement is stored in a temporary table so that it can be further manipulated by a SqlQuery statement. Therefore, in order to join two SharePoint lists, you will need to add a List statement for each list and a SqlQuery statement that will let join both lists using SQL SELECT syntax.
In the Visual Query Designer, you can add a statement by clicking on the “New” menu item inside the Query Statements panel as shown in the following screen shot:

Of course, each statement has its own options and properties available through query designer interface. For instance, for a list statement, you may specify the desired columns or you may decide to filter on a specific folder. You can find more information about the options inside the chapter “Using Enesys SharePoint Query Designer” page “9”.

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Retrieves data from a SharePoint list.</td>
</tr>
<tr>
<td>columnChoices</td>
<td>Retrieves the choices available for a specific list column or site column of type Choice (or derivated).</td>
</tr>
<tr>
<td>survey</td>
<td>Retrieves data from a SharePoint Survey list in a format that makes it possible to build reports (as opposed to the raw data that is normally returned by a survey list).</td>
</tr>
<tr>
<td>infoPath</td>
<td>Retrieves data from an InfoPath form for both promoted and non-promoted fields.</td>
</tr>
<tr>
<td>listsMerge</td>
<td>Merges multiple SharePoint lists at once, based on the data of other statements.</td>
</tr>
<tr>
<td>rollUpLists</td>
<td>Retrieves items from lists in an entire site collection.</td>
</tr>
</tbody>
</table>
### sqlQuery

Uses SQL Select syntax to merge, join and aggregate the data resulting from any other statement.

### xmlQuery

Retrieves XML data from any URL.

### sqlData

Used to retrieve data from any SQL Server or Oracle databases.

### Administrative statements

Various administrative statements that let you retrieve back end information about SharePoint.

As you can see, the `sqlQuery` statement is extremely important in the sense that it will let you apply sophisticated manipulations on SharePoint data from multiple lists and sites. Except for simple needs (or when you just need to roll up list items in an entire site collection using `rollUpLists` statement), your query will often be composed of multiple statements with at least a `sqlQuery` statement that will let you join/merge/aggregate the data resulting from other statements.

The following screenshot shows the `sqlQuery` statement that is used to join both Orders and Customers SharePoint lists:

![SqlQuery screenshot](image)

Note that you can use “Orders” and “Customers” as if they were simple tables.
Conclusion

The approach we have taken with Enesys SharePoint Query makes it possible to come up with sophisticated queries. You are not limited to a single sqlQuery statement per query. You can join the result of a sqlQuery statement with the data resulting from any other statement by using another sqlQuery statement. This specific approach will let you solve most of business needs you are facing regarding SharePoint data stored in multiple lists and sites.
Using Enesys SharePoint Query Designer

Retrieving data from a SharePoint list

You can easily retrieve data from any SharePoint list by using a List Statement.

- Click on the “New” menu item inside the “Query Statements” panel.
- Click on “Add list statement...” to add a new list statement.

- The first step is to enter the relative site URL of your SharePoint site containing the desired list. You may either enter the URL or select an existing data source variable that would contain the correct URL. Then, click on the “Get Lists” button in order to retrieve all the lists of this site. Once retrieved, you can filter them so that you easily find the correct one.
- Select the desired list(s) and click on the “Add” button

The Query Designer will automatically create the query string for you. When adding a List statement, the default behavior is to retrieve all items for all columns from the specified list. As you will see, you can be more specific about the data to retrieve from the list (columns, conditions...).
You only need to run the query and retrieve the list items:

**Specifying list columns**

You can specify the columns you would like to retrieve using the *Columns* tab.
Instead of checking each column box, you can check the ‘*’ box for retrieving all non-hidden fields from the specified SharePoint list.

Leaving all boxes unchecked will return all columns from the SharePoint list, including hidden ones.

**Note:**

It is recommended to specify the exact columns you need in your query. Not only it will perform better but it will also avoid possible unwanted side effects.

Indeed, with both Reporting and Integration Services, the columns will serve as the basis for specifying the data retrieved. The schema definition stored within the report or package is static and all the columns specified need to exist when you execute your report/package.

By specifying all columns or all non-hidden columns, your schema will contain all columns (or non-hidden columns). At a later time, if some unimportant column is deleted from your SharePoint list, your report/package will not work anymore even though you were not using the deleted column in your report layout/package flow.

**Filtering list items**

In order to filter the items retrieved by the list statement, filters can be applied by using the Filters tab.

Filtering at the list level is not your only option. You can also filter using a SqlQuery (see later in the documentation) statement or even at the report level using Reporting Services features. For more complex queries, it is not always possible to apply a filter on a single SharePoint list. You can use a mix of those filtering options as necessary.

As a rule of thumb, you should filter at the List level whenever possible. **Enesys RS Data Extension** will generate CAML filter that SharePoint can use to filter items before returning them. This will improve the performance considerably in most cases.

**Adding Conditions**

In order to add a new condition, click on the menu button next to the “logical operator” (And / Or) inside which you want to add a new condition, and click on the “Add new condition” menu item:
The new condition is added at the end of the specified group.

Setting Condition Properties

Selecting the Field

In order to select the field on which the condition will apply, a drop down list is available with support for auto complete. You can either select the field inside the available fields, or you can start typing the field internal or display name to filter the fields list as shown in the following screen shot:
**Selecting the Operator**

A large range of operators is available to design specific filters as shown in the following screenshot:

- The following condition will retrieve items where the `Category` column is equal to "Condiments":

**Conditions options**

Filtering with the lookup ID

When filtering on lookup columns, you can choose to filter either on the lookup value (the default) or on the lookup ID.
Click on “Lookup value” to change the selection to “Lookup ID”:

Including the time inside date comparisons
When filtering on date columns, you can choose whether to include or not the time as part of the comparison. By default, the comparison only takes the date part into account:

Click on “excluded” to change the selection to “included”:

Adding Groups of Conditions
Groups are used to combine multiple conditions using either And (all conditions apply) or Or (any condition applies) logical operator.
**Adding Inner Groups**

In order to add a new group, click on the menu button next to the Group inside which you want to add a new Group, and click on the “Add new group” menu item:

The new group is added at the end of the specified group, along with a default condition:

*Note that, if the current group is an “And” group, an “Or” group will be created, and similarly, if the current group is an “Or” group, an “And” group will be created.*

You can change the group type (**Or** / **And**) at any time by clicking on the group:

**Adding Parent Groups**

It may be practical to add a parent group. For example, you have the following query, where you want to retrieve the items which Title either begins with “a” or begins with “b”:
But you realize that you also only want items which Category is Beverages. Click on the “Or” group and add a new parent group:

A new “And” group was added as a parent of your existing “Or” group:

Now, you just need to add a new condition inside the “And” group to filter on the Category:

Moving Conditions/Groups
It’s possible to move the conditions and groups inside the query.
The conditions/groups can be moved up and down inside a specific group:

Even if the order of the conditions doesn’t matter, it can be interesting to move up and down conditions if you want to move them left and right just after.

You can also move the conditions/groups left and right:

When moving left a condition, the condition becomes a sibling of its parent:
And, it’s the opposite when moving right a condition/group:

The condition becomes a child of its previous sibling (only if the previous sibling is a Group):

**Filtering items based on another statement data**
Starting with Enesys SharePoint Query 3.8, list statements supports filtering data based on the results of another statement.

The “statement filter” is converted into a CAML query, which is applied before the data is returned from SharePoint. This results in performance improvements in most cases, especially when retrieving few items from a large list.

For example, if you need to retrieve all Issues assigned to some conditionally retrieved users:

- Add a list statement on the UserInfo list that retrieves the users (with necessary filters).
- Select (at least) the ID column from the UserInfo list.
- Add a list statement on the Issues list
- Add a new filter that specifies that the AssignedTo values should be taken from the results of the UserInfo’s ID column:
Items are filtered based on the specified filter:

Note that the “statement filters” are currently limited to a single filter. If you need to filter data from 2 or more result sets, we would recommend you to specify the one that filters out the most data, and then join your lists using a sqlQuery statement.

**Using column display names or internal names**

SharePoint list columns have an internal name and a display name. When a column is initially created, the display name and the internal name of the column are the same (except if the name contains space or accent marks). When you modify the name of a column, it will only modify the display name. The internal name is never modified and you may end up with columns whose internal names no longer have any connections to the display names.
You can choose to use display names when specifying columns to return by setting the `useDisplayName` attribute to true.

Note:

We recommend using internal names rather than display names. It has the advantage of not breaking your queries when a column name is changed. This is something that can easily happen with SharePoint lists.

**Using parameters**

Parameters may be used in various ways in the context of Enesys SharePoint Query (attributes, query...). A parameter is composed of a name surrounded by the characters @ and ! (e.g. @product!).

The following example shows how to return the data from the SharePoint "Products" list whose category is equal to the value of the "cat" parameter. The parameter will be replaced by the corresponding value when the report/package is executed.

When running the query inside Enesys SharePoint Query Designer, you will be asked to enter the value of the “cat” parameter:
Multi-value parameters are supported. However their use only makes sense in the following constructs:

sqlQuery statement with an IN clause:
Pseudo CAML Query in operator:

And

1. Category (Category) is in [Category] using the Lookup value
The value of the filter element from the rollUpLists statement:

Using data source variables
Starting from version 3.2, Enesys SharePoint Query supports variables at the data source level. Variables are limited to the title and relativeSiteUrl attributes inside the list, survey, infoPath, rollUpLists and mergeLists statements, and the connectionString element inside the sqlData statements.

Data source variables are variables that you define in the data source connection string. A data source variable is composed of a name surrounded by the "$" and "!" characters (e.g.: $siteUrl!). The following screenshot shows a shared data source connection string that defines two variables (siteUrl and recEvnsUrl). The values of those variables are respectively "/sites/ersdedemo/" and "/sites/ersdedemo/some recurring event/".
What’s the point of using data source variables?

Basically, it will let you parameterized the relative site URLs used for any number of reports and makes it easy to point your reports/queries to a new server/site by simply changing the value of the data source variables instead of going through each report and change the relative site URL. Here are some scenarios that are a good fit:

- You would like to move your reports to another server and the site structure is not exactly the same (the relative site URL used in your queries need to be changed).
- You would like to change the URL of your site or move your site (the relative site URL used in your queries need to be changed).
- You would like to execute your reports over some test site/server that does not match the hierarchy of your current execution context.

Do not confuse data source variables and report parameters. Report parameters are resolved at run time. Data source variables are resolved at design time. Their intent is quite different.

The rest of this chapter will describe how you can use the data source variables in combination with Enesys SharePoint Query Designer.
Using a variable in the relativeSiteUrl attribute

When adding new list, survey, infoPath, mergeLists or rollUpLists statements, you can select a variable for specifying the relative site URL:

By doing this, your query will use the "$siteUrl!" variable in the relativeSiteUrl attribute instead of the relative site URL specified as a literal value:  

```xml
<root xmlns="http://eneysoft.com/schemas">
  <list title="Products" relativeSiteUrl="$/sites/erscdemo!" tableName="Products">
    <fields />
    <query />
  </list>
  <resultSet>Products</resultSet>
</root>
```
Using a variable in the title attribute
You can also use variables instead of literal values in title attributes.

Considering that I already have a variable \$list\ that is equal to “Products”, we will show how to add a list named “Products” and replace its title by \$list\.

- In the “Add List Statement” form, select the “Products” list and add it to your project.
- Right click on the Products list and click on the “Properties” menu item.

- Go to the “Url and Title” tab, and inside the “List Title” group, you can choose that a variable is used for specifying the title:
Once validating, the query uses "$list!" in the list **title**:

As you can see on the "Url and Title" tab page inside the Properties form, you can easily revert to a literal value, or even specify that you want to use a parameter for your value.
Using variables inside the sqlData’s Connection String

Starting from version 3.5, you can specify data source variables inside the sqlData Connection String.

Given the fact that you have specified a `sqlservername` variable with a correct SQL Server name, you will be able to use it inside the "Connection Properties" form instead of specifying the server directly:

![Connection Properties Form]

Note:

The variables are not limited to the “Data Source” property; you can use it in any property. You can even assign a single variable (that contains the necessary properties and their values) to the connection string. However, in this case, you won’t be able to edit the connection string using the “Connection Properties” form.

For more information, please read the "sqlData statement" inside the Enesys SharePoint Query manual.
Expanding recurring events

The `expandRecurrent` attribute lets you expand recurring events for an event list.

The `expandRecurrent` attribute goes along with `expandFirstDate` and `expandLastDate` required attributes that lets you define the range of dates for which recurring events will be expanded. Parameters may be used to set `expandFirstDate` and `expandLastDate` values.

The `expandRecurrent` attribute possible values are:

- **Automatic**: retrieves both recurring and non-recurring events for the date range specified by the `expandFirstDate` and `expandLastDate` attributes
- **Standard**: retrieves all events according to the CAML query and expands the recurring events for the date range specified by the `expandFirstDate` and `expandLastDate` attributes.
- **None**: retrieves events without expanding recurring events

In order to obtain events between two dates including the recurring events, you would set up the following query, using the “Automatic” mode:
The "Automatic" mode is the easiest way to retrieve events (both recurring and non-recurring events) for a given date range. Unless you have a very specific scenario, you will mostly use this mode when retrieving events from an event list.

Note that all events overlapping the specified date range are taken into account. If you only want to discard the overlapping events, you must add a sqlQuery statement as shown in the following picture.

**Query:**

```sql
SELECT * 
FROM Events
WHERE datetime(EventDate) >= datetime(@startDate!) 
AND datetime(EndDate) <= datetime(@endDate!)
```

### Stripping HTML Tags

It is a well-known limitation that Reporting Services 2005 (this is not the case anymore with SSRS 2008) is not able to handle HTML tags within a specific field. Thus, HTML data within a field will be displayed as plain text as shown in the following image:
Though, you may use your own approach to remove HTML tags by using Reporting Services embedded code features, we have added the ability to strip HTML tags for a specific SharePoint list using the `stripHtml` attribute:
Though not really appealing, stripping HTML makes text at least readable:

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
</table>
| Html Test | - Bold text  
- Italic text  
- Reporting Services does not support HTML data. It will display HTML tags as plain text. |
| Essential Resources for SharePoint Developers | - Some interesting resources for SharePoint Developers  
- Microsoft Office SharePoint Server (MOSS) SDK and ECM Starter Kit  
- Windows SharePoint Services (WSS) SDK and Workflow Starter Kit  
- Visual Studio Extensions for SharePoint Services (November CTP)  
- Customizing and Branding Web Content Management-Enabled SharePoint Sites  
- MOSS for Content Management Server Developers (Beta)  
- Office Developer Screencasts (applies to all of Office)  
- SharePoint Developer Map (also includes InfoPath and 2007 Office System posters)  
- MOSS and WSS Online Clinics  
- MOSS portal on the Office Developer Center  
- SharePoint Developer Center  
- 7 Development Projects for SharePoint – online book  
- MSDN Community Content (WSS & MOSS)  
- F1 Help from Visual Studio Document Explorer  
- Project SDK Download |

Note:

SSRS 2008 is able to render HTML tags (a subset). In most cases, this feature should not be necessary if you are using SSRS 2008 or later.

**Items and folders**

The default behavior when retrieving items using a `list` statement is to return all items within all folders and subfolders. You can override this behavior by specifying a value for the optional `folder` and `viewScope` attributes.

The `folder` attribute lets you specify the folder name from which the items will be retrieved. When not specified, the root folder is the default value.

The `viewScope` attribute lets you specify which items will be retrieved. Valid values are:

- **Default**: retrieves only the files and subfolders in a specific folder (default value when a folder is specified)
- **FilesOnly**: retrieves only the files of a specific folder.
- **Recursive**: retrieves all files of all folders starting from the specified folder (default value when no folder is set)
- **RecursiveAll**: retrieves all files and all subfolders of all folders starting from the specified folder

When specifying a folder attribute, you will retrieve items located in the specified folder as well as subfolders as shown in the following screenshot:

![Screenshot](image)

To retrieve root items and folders, set the `folder` attribute to a "/" (slash) value.

Instead of specifying the folder using a literal value, you can use a report parameter (e.g.: `folder="@someReportParameter!"`).

If you want to retrieve all items and all folders, just leave the `folder` attribute empty and set the `viewScope` attribute to `RecursiveAll`. 
Retrieving the first n items of a list

The `rowLimit` attribute lets you specify the number of items to retrieve.

Paging

The `paging` attribute lets you specify how many items at once should be retrieved from a SharePoint list. Enesys SharePoint Query will retrieve list items in chunks of the number of specified items by the `paging` attribute.

By default, Enesys SharePoint Query will retrieve items from a SharePoint list in chunks of 2000 items. However, if you want to put less pressure on your SharePoint server, especially if you are retrieving a large number of columns, you may want to specify a lower number.
Please note that the **paging** attribute overrides the **rowLimit** attribute. Therefore, it is not possible to specify a **rowLimit** attribute when specifying a paging attribute. It is however possible to mimic the **rowLimit** feature at the list level by adding a **sqlQuery** statement as shown in the following sample that shows a query returning 1500 items from the product list:
Custom separator for multiple values columns

The following type of SharePoint columns may have multiple values:

- Choice
- Lookup
- User
- Attachments

By default, multiple values will be separated by a comma. You may specify your own separator by setting the `multiValuesSeparator` attribute value like shown in the following screenshot:
By running this query, you will get each student skill and spoken language separated by two slashes as shown in the following screen shot:

Expanding multiple values

Retrieving multiple values separated by some separator will not help if you would like to group items based on the possible values of such a column.

To address this scenario, it is possible to duplicate items for each value stored in the desired multiple values column as if you were doing a "Join" with those values. The desired multiple values column is specified using the `expandMultiValuesColumn` attribute as shown in the following screenshots:
By running this query, the "Skills" column (which is a lookup column accepting multiple values) will only get one value for each student. On the other hand, students are duplicated accordingly to reflect each skill as shown in the following screenshot:
Without using `expandMultiValuesColumn` attribute, you would retrieve items in the following form:

```
<table>
<thead>
<tr>
<th>Title</th>
<th>SkillsID</th>
<th>Skills</th>
<th>Spoken_language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buchanan</td>
<td></td>
<td>Asp.Net,C#</td>
<td>English,French</td>
</tr>
<tr>
<td>Callahan</td>
<td></td>
<td>Moss 2007,Wss V3</td>
<td>English,German</td>
</tr>
<tr>
<td>Davolio</td>
<td></td>
<td>Asp.Net,C#,SQL Server 2005</td>
<td>English,Spanish</td>
</tr>
<tr>
<td>Dodsworth</td>
<td></td>
<td>Asp.Net,C#,Moss 2007,SGL Server</td>
<td>English</td>
</tr>
<tr>
<td>Fuller</td>
<td></td>
<td>Asp.Net,C#</td>
<td>English,Italian</td>
</tr>
<tr>
<td>King</td>
<td></td>
<td>Asp.Net,C#</td>
<td>English,French,German</td>
</tr>
<tr>
<td>Leverling</td>
<td></td>
<td>Asp.Net,C#</td>
<td>English,French,German,Spanish,Italian</td>
</tr>
</tbody>
</table>
```

Rather than specifying the column to expand as a literal, you may use a report parameter (e.g.: `expandMultiValuesColumn="@ChoosenColumn!"`).

**Important notice:**

Be aware that items, being duplicated (somewhat like a join operation), some calculations may not be appropriate when building reports using this approach.

**Retrieving attachments**

Starting from version 2.1, the URLs to the files that have been attached to a specific list item are available through the “Attachments” column. Prior to the version 2.1, the “Attachments” column was only returning the number of attachments.

Multiple attachments will be separated using the separator specified by `multiValuesSeparator` attribute or a comma if the attribute is not specified.
When you would like to display individual attachments in a report and possibly allow the user to click on the attachment for displaying the associate file, you may set the `expandMultiValuesColumn` attribute value to “Attachments” as shown in the following screenshot:
Retrieving user information

It is possible to retrieve information about users in a site collection by using the **UserInfo** list:

One interesting use of the “UserInfo” list is to join it to some other list in order to get more details about users specified using a “Person or Group” column.
The query shown in the following screenshot will join an Issue list and the “UserInfo” list based on the user assigned to the issue:

Retrieving item versions
Starting from Enesys SharePoint Query version 2.1, it is possible to retrieve item versions using the itemVersions attribute.
You can either retrieve all versions, or a specific number of versions. The following screenshot shows how to retrieve all versions of items from SharePoint "Issues" list.

**Specifying the item version fields**
Starting from version 3.5, you can specify the fields for which you want to retrieve item versions. This lets you improve the overall performances when retrieving item versions.
Open the Properties form and go to the “Item Versions” tab. You can now select the fields for which you want to retrieve item versions:

![Image of Change item versions options window]

Please note that only currently selected fields are displayed inside the “Available fields”. If a field doesn’t appear, you need to select it before being able to select it for item versions retrieval.

Important notice:

Due to SharePoint Web Services limitations, retrieving item versions is a very time consuming process. In order to improve performance, it is recommended that you explicitly specify the fields for which you need to retrieve item versions instead of relying on the default behavior which is to return item versions for all the selected fields.

**Retrieving meeting instances**

When retrieving items from lists within a meeting workspace, you may specify the meeting instance for which you would like to retrieve items.
To retrieve items for all meeting occurrences, you can set the `meetingInstanceId` attribute value to “-1” as shown in the following screenshot:

**Distinct items**

When retrieving items to fill the possible values of a parameter, you will probably need to retrieve distinct items from your list.

While it's possible to add a "SELECT DISTINCT ..." sqlQuery statement to your query, starting from the version 3.8, Enesys SharePoint Query allows you to directly return distinct items from your SharePoint list.
When doing a case sensitive distinct, “Word” is different from “word” or “WORD”, and similarly with non-ascii letters, “бука” is considered different to “Буква”. Several items are returned for each different case.

When doing a case insensitive distinct, “Word” is considered equal to “word” and “WORD”, and similarly, “бука” is considered equal to “Буква”. A single item is returned for all different cases.

**Reporting on Survey lists**

**Introduction**

The raw data returned from a SharePoint Survey List can’t be used as-is for building reports. This is due to the fact that each answer is represented by a single column in a survey list. This makes it impossible to use for most reporting scenarios.

To overcome this limitation, the *survey* statement applies a transformation to the initial survey list data in order to provide you with data that can be used for building reports.
The schema returned by the *survey* statement is static and does not depend on how many questions you may have in your Survey. The following table describes the schema returned by the *survey* statement:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuestionID</td>
<td>An auto-increment ID, each question having a different ID.</td>
</tr>
<tr>
<td>FieldType</td>
<td>The type of the SharePoint field (Text, Number, Choice, GridChoice...).</td>
</tr>
<tr>
<td>FieldName</td>
<td>The display name of the field (the question title).</td>
</tr>
<tr>
<td>GridStartNum</td>
<td>Only used with GridChoices. Represents the first available number.</td>
</tr>
<tr>
<td>GridEndNum</td>
<td>Only used with GridChoices. Represents the last available number.</td>
</tr>
<tr>
<td>GridTxtRng1</td>
<td>Only used with GridChoices. Contains the text representing the first number.</td>
</tr>
<tr>
<td>GridTxtRng2</td>
<td>Only used with GridChoices. Contains the text representing the middle number.</td>
</tr>
<tr>
<td>GridTxtRng3</td>
<td>Only used with GridChoices. Contains the text representing the last number.</td>
</tr>
<tr>
<td>GridNATxt</td>
<td>Only used with GridChoices. Contains the text used for the “N/A” option. It may be null if the “N/A” option is not available for this GridChoice.</td>
</tr>
<tr>
<td>SubQuestionTitle</td>
<td>Only used with GridChoices. Contains the name of the GridChoice option.</td>
</tr>
<tr>
<td>Choice</td>
<td>Contains the possible values of this question.</td>
</tr>
<tr>
<td>AnswerID</td>
<td>Contains the ID of the user answer.</td>
</tr>
<tr>
<td>Value</td>
<td>Contains the user answer to this particular question.</td>
</tr>
</tbody>
</table>

For example, if your Survey is composed of 10 questions, you will have about 10 lines returned for each user answer (the number varies depending on the question types and the options you have selected – see later in the documentation).

Here is a sample data for 3 questions and a single user answer:

- The user answered “Google Ads” to the "How did you hear about us?" question
- For the question "How would you rate our website" with a scale from 1 to 5, the user gave a note of 4 to "Overall content", a note of 4 to "Ease of navigation" and a note of 3 to "Overall look".
For the question “How much do you agree with each of the following statements about the website” with a scale from 1 to 5, the user gave a note of 3 to “The information on the website is clearly written and easy to understand”, and a note of 4 to “The website contains no grammar or spelling errors”

Adding a Survey statement

- Add a new Survey statement to your query

- Enter the relative site URL or select an existing data source variable that points to the Survey list and click on the “Get Lists” button. Only lists of type “Survey” are displayed.

- Select the Survey list(s) you want to query and click on Add.
Selecting the fields
It’s not possible to select any field with the `survey` statement. It will always return all the columns available in the static schema described previously.

Please keep in mind that the columns that are displayed in the query designer Columns panel represent the schema of the initial SharePoint survey list and not the schema that is returned after transformation by the survey statement:

Filtering data
Because the `survey` statement returns a schema that is completely different than the initial SharePoint survey list schema, filtering data can be confusing.

Basically, you have two possible approaches for filtering the Survey data:

- Filtering at the list level. In that case, conditions will rely on the initial survey list schema. In that schema, each question is represented by a column.
- Filtering on the transformed data by using an additional `SqlQuery` statement and apply a `SQL Select` operation.

Filtering before transformation
Similarly to the `list` statement, you can filter at the statement/list level.

When filtering at the statement/list level, `Enesys RS Data Extension` will transform your conditions into CAML query so that the filtering take place at the SharePoint level. Therefore, your condition will rely on the initial SharePoint survey list schema which is completely different than the resulting schema returned by the `survey` statement.
Though, this can be confusing, just keep in mind that there are certainly very few scenarios that would make it necessary to filter at this level.

Here are some filtering examples:

- Retrieving a specific user answer identified by its ID:

  ![Filtering example 1](image1)

- Retrieving all the answers of a specific user:

  ![Filtering example 2](image2)

You can find more information inside the "Filtering list items" chapter of the list statement, page 12.

**Filtering on the transformed data**

This is certainly the most natural approach for filtering your survey data. It is obviously much more natural to filter data based on the schema returned (that is the data you are manipulating in the end) rather than doing the necessary “mental shift” between the initial and target schema.

For filtering using this approach, you will add a `sqlQuery` statement that will let you apply a SQL `SELECT` operation to your survey data.

For example, if you don't want to include the answers to a particular question, you can add a `sqlQuery` statement with the following query:

```sql
SELECT *
FROM [MySurveyList]
WHERE [FieldName] != 'How did you hear about us?'
```

**Survey Properties**

Most of the survey properties work exactly the same way as for the list statement. You may refer to the associated documentation for list statements.
However, two new properties are specific to the survey statement: includeAllOptions and expandMultiValues:

![Image of the Properties of the “Survey On ERSDE” statement]

**Removing the options that are not selected**

The includeAllOptions allows you to indicate whether you want to include the options for which no user selected this option. For example, the users may not have selected all the options of your “Choice” question.

If includeAllOptions is true (default behavior), even the choices that are not selected are retrieved. On the contrary, if includeAllOptions is false, the choices that were not selected are not retrieved.

Please note that all the possible values for the “GridChoice” (Rating scale) columns are always returned, even if the includeAllOptions option is false.

This option may be particularly useful to set to false for two scenarios:

1. **You are retrieving the data of a particular item:**
   
   Generally, you don't care about the choices that the user didn't select: you only want the choice the user selected.
With **includeAllOptions=true**, you have several items for each question of type Choice, MultiChoice or Boolean:

<table>
<thead>
<tr>
<th>QuestionID</th>
<th>FieldName</th>
<th>FieldType</th>
<th>AnswerID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Your last name:</td>
<td>Text</td>
<td>Smith</td>
<td>3 Smith</td>
</tr>
<tr>
<td>2</td>
<td>Your first name:</td>
<td>Text</td>
<td>John</td>
<td>3 John</td>
</tr>
<tr>
<td>3</td>
<td>Your age:</td>
<td>Number</td>
<td>75</td>
<td>3 75</td>
</tr>
<tr>
<td>4</td>
<td>Your birthdate:</td>
<td>DateTime</td>
<td>11/2/1935</td>
<td>3 11/2/1935</td>
</tr>
<tr>
<td>5</td>
<td>Revenue per month:</td>
<td>Currency</td>
<td>$4,242.00</td>
<td>3 $4,242.00</td>
</tr>
<tr>
<td>6</td>
<td>Gender</td>
<td>Choice</td>
<td>Male</td>
<td>3 Male</td>
</tr>
<tr>
<td>7</td>
<td>Address</td>
<td>Note</td>
<td>Jamestown England</td>
<td>3 Jamestown England</td>
</tr>
<tr>
<td>8</td>
<td>Which of those OS is your favorite one:</td>
<td>Choice</td>
<td>Windows 7</td>
<td>3 Windows 7</td>
</tr>
<tr>
<td>8</td>
<td>Which of those OS is your favorite one:</td>
<td>Choice</td>
<td>Windows Vista</td>
<td>3 Windows Vista</td>
</tr>
<tr>
<td>8</td>
<td>Which of those OS is your favorite one:</td>
<td>Choice</td>
<td>Windows XP</td>
<td>3 Windows XP</td>
</tr>
<tr>
<td>8</td>
<td>Which of those OS is your favorite one:</td>
<td>Choice</td>
<td>Mac OS X</td>
<td>3 Mac OS X</td>
</tr>
<tr>
<td>8</td>
<td>Which of those OS is your favorite one:</td>
<td>Choice</td>
<td>Ubuntu</td>
<td>3 Ubuntu</td>
</tr>
</tbody>
</table>

With **includeAllOptions=false**, you have only a single item for each question, so your report will be easier to build.
2. You are retrieving all the data and you don’t want to display the choices that are not selected:
   a. With **includeAllOptions=true**, all the choices are retrieved.

   ![Example Table]

   b. With **includeAllOptions=false**, only the selected choices are retrieved.

**Expanding the multi-values**

By default, the multi-values answers are not expanded, and each selected value is separated with a semicolon and a space ("; "), to match the default behavior of SharePoint. (You can change this separator by editing the `multiValuesSeparator` value)

For example, if the `expandMultiValues` option is **false** (default behavior):

![Example Table]
And if the `expandMultiValues` option is true:

<table>
<thead>
<tr>
<th>Qu...</th>
<th>FieldTypeName</th>
<th>FieldType</th>
<th>FieldName</th>
<th>G</th>
<th>G</th>
<th>G</th>
<th>G</th>
<th>Su</th>
<th>Choice</th>
<th>Ans...</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>MultiChoice</td>
<td>Select the features...</td>
<td>Querying Survey lists</td>
<td>Querying Survey lists</td>
<td>Querying Survey lists</td>
<td>Querying Oracle servers</td>
<td>Querying Oracle servers</td>
<td>Querying Oracle servers</td>
<td>Parameterizing the list title</td>
<td>Merging lists enumerated in a result set</td>
<td>Intellisense for SQL Server</td>
</tr>
</tbody>
</table>

**Retrieving data from InfoPath forms**

Starting from the version 3.7, Enesys SharePoint Query lets you retrieve data from both promoted and non-promoted fields of your InfoPath forms.

**Introduction**

SharePoint lets you promote InfoPath fields as columns inside your SharePoint list. While it may be sufficient in some cases, in other cases, you may need to retrieve data directly from the XML file, for example, if you want to do a 1-n relation or if you don’t want to promote lots of fields into your SharePoint List.

With the version 3.6, Enesys SharePoint Query added support for XML Namespaces inside the `xmlQuery` statement that allowed you to query Infopath forms. However, this approach involves using sub-reports and does not allow manipulating both data as a single dataset.

With the version 3.7, the brand new `InfoPath` statement allows you to directly retrieve data from promoted fields and any data from the InfoPath form.

**Adding an InfoPath statement**

Adding a new `InfoPath` statement is similar to any other statement retrieving SharePoint List data:
• Add a new InfoPath statement to your query:

![InfoPath statement dialog box]

• Enter the relative site URL or select an existing data source variable that points to the InfoPath library and click on the “Get Lists” button. Only InfoPath libraries are displayed.

![SharePoint Site dialog box]

• Select the InfoPath library you want to query and click on the Add button.

**Selecting the Content Type**

This step is optional and depends whether your SharePoint Library allows the management of Content Types and whether several InfoPath Content Types are available inside the Library.

Indeed, the InfoPath statement only allows you to query a single InfoPath Content Type: all the files that are not InfoPath forms or InfoPath forms associated to another Content Type will not be retrieved.
If the InfoPath Library allows the management of Content Types and several InfoPath Content Types are available inside the Library, you will need to select the Content Type you want to retrieve data from:

Once selected, the XML schema of this Content Type will be parsed and displayed inside a tree view for allowing you to select the non-promoted fields to retrieve.

**Selecting the Fields**

The InfoPath statement allows you to retrieve data from both promoted and non-promoted fields. The Columns tab is divided into two parts to differentiate them:

Selecting promoted fields is similar to the selection of List columns, so no further details are given here. You can find more information inside the “Specifying list columns” chapter.

Selecting the non-promoted fields can be a little tricky. Indeed, the InfoPath statement allows you to handle a single 1-n relation inside your InfoPath Form. This relation represents the “main path” of your query, and is highlighted in bold:
In the previous screenshot, if there are several “leaves” inside the Form, they will all be returned. Whereas, all the selected fields that are not part of this main path will be managed as a 1-1 relation, and only the first item will be retrieved even if there are several items.

For example, in the following screenshot, all the “leaves” will be returned, being part of the main path, while only the first “item” will be retrieved:

It is for this reason that it is important to always select your 1-n relation first. Once you have selected your 1-n relation, you can select all the additional fields you want.
In the previous sample, if we selected the “items” node first, this would have been our main path: all the “items” would be retrieved and only the first “leave” would have been retrieved.

Note:

Even if you are only interested in InfoPath Form Fields, we recommend you to select at least one column from the Promoted Fields. Or else all the SharePoint columns will be returned too.

**Specifying the Field Data Types**

Enesys SharePoint Query relies on the XML schema of the InfoPath form to retrieve the data type of each available field. But it happens that the XML data type is not the type you want to retrieve in the end.

You can change the field data type by clicking on the link just after its name and select another data type:

![Field Data Type Selection](image)

**Retrieving column choices**

Starting from version 3.8, Enesys SharePoint Query allows you to easily retrieve a list of all the possible choices from any column of type Choice (or derivated).

It is handy when you need to quickly fill the possible values for a parameter, or if you need to display the string representation of some workflow columns.
Add a new `columnChoices` statement:

The `columnChoices` statement allows you to retrieve the choices from a list or a site column. In the first screen, you will need to choose whether you want to retrieve data from a list or a site column.

If you want to retrieve choices from a site column, check the correct radio button and enter the relative URL to the SharePoint site.
If you want to retrieve choices from a list column, check the correct radio button and click on the Add button. It will now display a form for selecting the list containing the column.

Once you have selected your list or your site, you can select the column to query.
The designer only displays the columns of type “Choice” or derivated:

Note that you can choose to show all columns by unchecking the “Show only Choices” checkbox, but those columns won’t be selectable.
Once the column is selected, you can execute the statement and see the results:

As you can see on the previous screen shot, 2 columns are returned:

- **ID**: either contains the lookup ID or an auto-incremented integer
- **Choice**: contains the possible choices

Having an auto-incremented integer allows you to join the `columnChoices` statement with a list statement to have string representations for some integer values.
For example, columns of type “WorkflowStatus” return integer values (Approval column):

By joining the list results with a `columnChoices`, you can have the string representation of those values.
Using a `columnChoices` statement, we query the same list, and select the Approval column of type `WorkflowStatus`:

![Query Designer](image)

We now need to join the Approval column from the list statement, with the ID column from the `columnChoices` statement using a `sqlQuery` statement:
A string representation for the Approval column is now available.

**Applying operations to lists**

You are not limited to query a single SharePoint list. **Enesys SharePoint Query** lets you apply specific operations between SharePoint lists.

You can define as many list statements as necessary and apply as many operations (**sqlQuery** statements) as you like in order to obtain the desired Dataset from which you will build your report.

Each list statement defined within a query returns a set of data items from a SharePoint list for which we will use the generic term "result set". The **tableName** attribute is used to give a unique name to this "result set". The unique name of the result set will serve as the basis for specifying result sets involved in operations.
You can manipulate "result sets" by using `sqlQuery` statements. The data resulting from a `sqlQuery` statement is considered a "result set" as well and it can be further manipulated using other `sqlQuery` statements.

The following query shows three `list` statements used to retrieve data from three specific SharePoint lists as well as a `sqlQuery` statement used to further manipulate (join and group) the data retrieved from the SharePoint lists.
Joining SharePoint lists

The SQL JOIN operation lets you join matching items between two result sets based on their joining columns. The tableName attribute lets you give a unique name to the set of data resulting from the inner join operation.

The following image displays a JOIN statement between the Products and the Suppliers SharePoint lists. Rather than using the supplier value pointed to by the lookup column, the query relies on the supplier ID.
You will notice that the “SupplierID” column has not been selected in the “Products” list. It has been automatically created by Enesys SharePoint Query for holding the ID of the “Supplier” pointed to by the lookup column.

Joining lists is useful (though not limited to that usage) when dealing with SharePoint lists linked with a lookup field.

Note that a SharePoint lookup column will return 2 columns in the resulting data for holding both the actual value and the target list item’s ID. It is recommended to join lists using those ID rather than the actual value.

**Merging SharePoint lists**

You can merge two result sets by using an SQL **UNION** operation:
**Getting distinct values**

The **DISTINCT** keyword is especially useful for creating a report dataset used to set the available values of a report parameter.

Say you have a SharePoint Products list with a Category column and you would like to let the user of your report select at run time the category of products that will be displayed in the report. You can easily achieve this by creating a specific report dataset as shown in the following image:

![Sql Query: DistinctCategories Properties](image)

```
SELECT DISTINCT Category
FROM Products
ORDER BY Category DESC
```

**Using Intellisense inside sqlQuery statements**

Starting from the version 3.6, Intellisense was greatly enhanced inside **sqlQuery** statements.

**Intellisense Items**

Intellisense is provided for the following items:

- **Statement names**

  The names of all the statements inside the current query (statements located before the current **sqlQuery** statement):

  ![Query Statements](image)

  The **Products list**, **RssFeed xmlQuery** and the **SqlCustomers sqlData** statements located before the current **sqlQuery** have Intellisense provided for their names. Contrary to the **Employees list** that is located after, that can’t be used inside the **sqlQuery** statement.
Field names

The names of all the available fields inside the selected statements:

The only selected statement inside the query is the Products list; the only fields available are the ones from this statement.

A join is made between the Products and the SqlSuppliers statements; fields from both statements are available for selection.

Code Snippets

Code snippets for quickly adding some SQL clauses:

“SELECT” code snippet

A “SELECT” snippet for inserting a SELECT clause with a FROM clause that selects a single statement.

By validating (with the Tab key), the snippet is inserted:
The statement names are available for replacement:

```
SELECT
FROM TableName
```

Replace with the name of a statement.

Enter/select the correct statement and click on Enter to validate the snippet modification. The caret is then moved just after the `SELECT` so that you can enter your selected fields:

```
SELECT
FROM Products
```

"Join" code snippet

A "JOIN" snippet for inserting a JOIN clause with an equality condition:

```
SELECT
FROM Products INNER JOIN {}
JOIN {}
```

Snippet for inserting a JOIN clause with a condition.

Similarly, you have to enter/select the name of the joined statement:

```
SELECT
FROM Products INNER JOIN TableName ON column1 = column2
```

Replace with the name of the joined statement.

Once chosen, click on the Tab key to move to the next placeholder that asks you for the first column used in the join:

```
SELECT
FROM Products INNER JOIN [Product categories] ON column1 = column2
```

The name of the column from the first table used in the join.

Once chosen, click on the Tab key to move to the next placeholder that asks you for the second column:

```
SELECT
FROM Products INNER JOIN [Product categories] ON Products. CategoryID = column2
```

The name of the column from the second table used in the join.

You can move to any placeholder by clicking on the Tab key to move forward or by clicking on the Shift+Tab keys to move backward.

Click on Enter to validate the modification.
SQLite functions

All the SQLite functions and their parameters:

Parameter description is available inside the functions when typing the characters "(" and ",".

Note:
As there is a large amount of SQLite functions, Intellisense for the SQLite functions is disabled by default as it may be difficult to find your fields within those functions. You can enable Intellisense for the functions by clicking on the following button:

Query Parsing
The query is parsed in real-time in order to provide some “context” to the Intellisense provider, so that it can provide only meaningful data.
Fields from selected statements
Only fields from selected statements are available:

Two statements are available ("Product categories" and "Products"), but none is selected inside the current SQL query, no fields are available for selection.

The same two statements are available but this time, "Products" is selected, its fields are available.

Both statements are joined, all the fields are available.
Table aliases

Table aliases are fully supported:

The name “p” is recognized as an alias for the Products statement, fields from the Products statement are available for completion.

The name “pc” is recognized as an alias for the “Product categories” statement, its fields are available.

Meaningful data

Intellisense data is only available at places where it makes sense to use it.

Some examples:

The query being empty, only a SELECT clause makes sense right here.

Only statements are provided, as neither functions nor column names make sense within the FROM clause.
At this point, you can specify neither a table name, nor a column name, nor a function name, only the JOIN code snippet is available.

Frequently Asked Questions

1. I don't have Intellisense at all.

There are several things to check:

- Ensure that your version of Enesys RS Data Extension or Enesys IS Data Extension supports enhanced Intellisense (at least v3.6 for Enesys RS Data Extension and v1.2 for Enesys IS Data Extension).
- Ensure that the "Enable Intellisense" button is checked.
- Ensure that some statements are located before the sqlQuery statement.

You may also read the following question if none of those checks resolved the problem.

2. I don’t have Intellisense for one of my statements.

It may be for several reasons:

- The statement is located after the current sqlQuery statement; therefore it can’t be used inside this particular sqlQuery statement.
- For statements that retrieve external data (such as sqlData and xmlQuery), it is necessary to silently execute the statement in order to retrieve the schema. If the execution failed, no error message is displayed but Intellisense is not available for those statements. Ensure that the statement is well configured and Intellisense should be available once corrected.
- For other sqlQuery statements, it needs the query to be valid and the statements that it uses to be properly configured. Please ensure both points.

3. No Intellisense is provided inside the sqlData statement?
The Intellisense feature is currently only available inside the sqlQuery statement. If you want Intellisense for SQL Server or Oracle data, we would advise you to use tools that provide Intellisense for such providers (e.g.: "SQL Server Management Studio" for SQL Server) for writing your query, then copy it back to our Query Designer.

**Known limitations**

There are some known limitations to our current Intellisense implementation, especially about subqueries.

**Subqueries inside the FROM clause**

You can use subqueries inside the FROM clause of your SQL statement.

**For example:**

```sql
SELECT * FROM (SELECT DISTINCT Category FROM Products) AS DistinctCategories
```

In this case, Intellisense will be provided within the subquery, but no Intellisense will be provided for the DistinctCategories table inside the outer query.

You can overcome this limitation by using an intermediate sqlQuery statement that executes the subquery.

**Subqueries inside the WHERE clause**

Subqueries inside the WHERE clause can use data from the outer query, as you can see in the following sample that retrieves the minimum unit price, grouped by category:

**For example:**

```sql
SELECT Category, UnitPrice FROM Products p
WHERE UnitPrice = (SELECT MIN(UnitPrice) FROM Products minP WHERE p.CategoryID = minP.CategoryID)
```

Intellisense for the "p" table and its fields will not be prompted inside the subquery.

**Merging a variable number of lists**

Starting from version 3.8, Enesys SharePoint Query allows you to merge a variable number of lists in a single operation using the listsMerge statements. All the lists to merge should be enumerated in another statement. The statement may be of any type (an aListCollection statement, a sqlQuery statement...), as long as it returns the two mandatory columns for merging the lists:

- **Title**: contains the list name
- **RelativeSiteUrl**: contains the relative URL to the SharePoint site
In the following sample, we will merge the items from all the document libraries of a specific site.

We will consider that an aListCollection statement is already configured to retrieve the list of all the document libraries of our site (/sites/ersdedemo/ in the sample):

Add a new listsMerge statement:
Inside the “Add listsMerge statement” form, select the statement that will be used as the data source. In our sample we only have a single statement:

After a quick verification to ensure that the statement has the 2 mandatory columns (Title & RelativeSiteUrl), you are now able to select the reference list that will be used to infer the schema.
The reference list is directly stored inside the statement representation, and you need to specify a list at the statement creation. In order to select this list, you have two possibilities:

1. Either you can click on the “Select List” button which will display a form for selecting a list, similar to the classic “Add list statement” form:

   ![Select List Form]

   - Enter the URL to the SharePoint site, relative to http://sps2010demo (e.g.:/sites/project/):
   - Enter the Relative Site URL:
   - Select the URL from an existing data source variable:
   - Get Lists

2. Or, you can click on the “Get First List” button, which will execute the statement in order to retrieve the first available list from this statement.

   Please note that this method may take a long time depending on your current query.

Once you have selected the list or retrieved the first one, its title / relative site URL will be displayed:
You are now able to add your new `listsMerge` statement, by clicking on the OK button.
The schema displayed is the one of your reference list; you can select all the desired columns (ID and FileLeafRef in our sample) and execute your statement:

Our query merged the documents of all our document libraries.

As you can see on the previous screen shot, 2 columns are automatically added to the result set:

- **Source_Title**: contains the title of the list from which this item is retrieved
- **Source_Url**: contains the relative site url of the list

**Rolling up list items in an entire site collection**

You can use the `rollUpLists` statement for merging lists items within an entire site collection based on a specific list’s name.
When using the `rollUpLists` statement, the lists from the site you specified are retrieved, as well as all the subsites of the specified site:

- If the list you want to rollUp is already available at this point, select the list and click on the Add button.
- If the list is located inside a subsite, click on its name inside the Tree View. Its lists will be loaded so that you can select the correct list.

Note:

Starting from v3.7, Enesys SharePoint Query supports a new `referenceSite` attribute that contains the URL to the site containing the list that will be used as the schema. You can specify the `referenceSite` by selecting the correct sub-site inside the sites tree view.

You are not bind to merge lists starting from the root of the site collection. It is possible to specify a child site as the starting point (i.e.: `relativeSiteUrl="/sites/ersdedemo/demosubsite01/"`).
When rolling up items using the `rollUpList` statement, two columns will be added to the result set. The first column “relativeSiteUrl” contains the relative site URL of the list while the second, "rstLabel" contains the name of the list and the relative site URL of the list from which the items have been retrieved as shown in the following screenshot:

Rather than specifying the starting site in the query, you may use a report parameter (e.g.: `relativeSiteUrl="@StartingSiteUrl"`).
You will be able to edit the method for specifying the URL inside the Properties form of the statement once created.
**Filtering the sites**

Starting from version 3.2, you can filter the sites that will be used to roll up items.

The *filter* property lets you specify the sites to include. Multiple sites may be specified by separating them with a comma. You can also use a report parameter instead.

Note that the site names are relative to starting site (*relativeSiteUrl*) specified.

For more information about the other properties (*includeRoot*, *recursive* and *strict*), see the reference manual.

**Retrieving administrative data**

Starting from version 3.1, *Enesys SharePoint Query* (Standard and Enterprise editions only) provides specific query statements for retrieving sites and lists information such as list collection, list permissions, site permissions, etc.
Those statement share a common approach to specify the sites or lists you are interested in using the `from` properties.
The following image shows an **aListCollection** statement which retrieves information about all lists available in the site “/site/ersdedemo/”:
You can filter the resulting data by using the `where` property. This property can contain a SQL WHERE clause:

The following table describes the various statements available in this category.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List collection</td>
<td>The <code>aListCollection</code> query statement lets you retrieve information about all the lists in a site or a set of sites.</td>
</tr>
<tr>
<td>List columns</td>
<td>The <code>aListColumns</code> query statement lets you retrieve the columns schema for one or more lists.</td>
</tr>
<tr>
<td>List permissions</td>
<td>The <code>aListPermissions</code> query statement lets you retrieve permissions from one or multiple lists.</td>
</tr>
<tr>
<td>Site collection</td>
<td>The <code>aSiteCollection</code> query statement lets you retrieve all the sites that are contained in a site collection.</td>
</tr>
<tr>
<td>SharePoint groups</td>
<td>The <code>aSiteCollectionGroups</code> query statement lets you retrieve all the SharePoint groups that are available in one or more site collections.</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Site columns</td>
<td>The <code>aSiteColumns</code> query statement lets you retrieve the column definitions for each column available on the specified sites.</td>
</tr>
<tr>
<td>Site content types</td>
<td>The <code>aSiteContentTypes</code> query statement lets you retrieve the content type definitions for each content type available on the specified sites.</td>
</tr>
<tr>
<td>Site permissions</td>
<td>The <code>aSitePermissions</code> query statement lets you retrieve the permissions given at the site level for a specific site or a set of sites.</td>
</tr>
<tr>
<td>User collection</td>
<td>The <code>aUserCollection</code> query statement lets you retrieve the information for the specified users, and the information for the users in the specified groups.</td>
</tr>
</tbody>
</table>

**Using the output of a statement as an input of another statement**

Because it makes sense for instance to get list collection for all sites in a collection or list permissions for all lists in a site, it is possible to use the result of a specific statement as the input of another statement.

The following example explains how to get the list collection from all sites in a specific site collection.

You will add two administrative statements: `aSiteCollection` and `aListCollection`. 
Modify the “From” property of the **aSiteCollection** statement in order to retrieve all sites in a specific site collection (/sites/ersdedemo/ in our example):
In order to retrieve information about all lists available in all sites of your site collection, you will need to specify the `from` type of `aListCollection` to “ResultSet” and then choose the name of your previous statement (`aSiteCollection1`) as value when `type="ResultSet"` as shown in the following screenshot:

![Properties of the “aListCollection1” statement](image)

**Joining SharePoint lists and external SQL data**

Starting from version 3.4, Enesys SharePoint Query (Standard and Enterprise editions only) makes it possible to join SharePoint data and external SQL data by using the `sqlData` statement.

You can use the `sqlData` statement to retrieve data from SQL Server databases or Oracle databases. In the following sample, we will retrieve data from SQL Server, but the process is similar for Oracle.
The `sqlData` statement lets you query any SQL Server database; the data can be further joined with SharePoint data using `sqlQuery` statement.

Specify the Connection String
Once you added the `sqlData` statement, you will be asked to choose the Data Source. You can choose between Microsoft SQL Server and Oracle.

Select the Microsoft SQL Server data source and click on OK to validate.

Then, you need to specify the Connection String used to connect to the SQL Server database.
This Connection Properties editor lets you specify the elements of your connection string to use to connect to the SQL Server.

- Enter the name of your SQL server or choose one inside the “Server name” list.
- Choose the credentials to use to connect to the SQL server. You can choose between:
  - Integrated Security (Windows Authentication)
  - Specified SQL account (SQL Server Authentication)
  - Credentials of the current data source (Data Source Credentials)

The last option allows you to always use the credentials specified inside the current data source. If Integrated Security was used at the data source level, the connection to the SQL Server will be made using Integrated Security, if an account was specified at the data source level, this account will be passed to the SQL connection. It also avoids having to specify a login/password as plain text when using SQL authentication.

- Select the database name
- If you need to specify advanced option, click on the “Advanced” button to edit any available option of the SQL Server Connection String.
Click on “Test Connection” to ensure that the connection is possible.

Note that you can further edit this connection string by following these steps:

- Go to the Properties tab of your sqlData statement
Click on the “…” button next to the `connectionString` property in order to display the Connection Properties editor.

**Enter your query string**

Once the connection to the SQL Server is done, specify your query string. In this example, we will retrieve data from the Categories table (from the well-known Northwind database).
**Joining SQL and SharePoint Data**

In order to complete the query you will need to add a `list` statement to retrieve data from a SharePoint list and add a `SqlQuery` statement for joining both the SharePoint list and the SQL data retrieved using a `SqlData` statement:

![Query Designer](image)

Run the query and you get your data connected together.

**Retrieving XML Data**

Enesys SharePoint Query provides an `xmlQuery` statement for retrieving XML data. Similarly to the other statements, you will be able to merge, join... the XML data with any other data retrieved.
In order to query an XML file (or directly specify inline XML data), add a new `XmlQuery` statement:

The “Add xmlQuery statement” form lets you specify the data source of your statement. You can choose between a URL and some inline XML data:
When directly entering XML data, click on the Edit button to display an XML editor:

Once your Data Source is entered, you need to specify the type of formula to use to select the data inside the XML source.

You can use an XPath formula that allows you to retrieve repeating fields. By clicking on the Edit button, a schema is generated from the XML data, to help you enter your XPath formula:
Or, you can use a custom ElementPath formula that allows you to retrieve both repeating and non-repeating fields at the same time, and also allows you to change the resulting data type: